Computer Intensive Statistics
STAT:7400

Luke Tierney

Spring 2019

Introduction

Syllabus and Background

Basics

- Review the course syllabus

  `http://www.stat.uiowa.edu/~luke/classes/STAT7400/syllabus.pdf`

- Fill out info sheets.
  - name
  - field
  - statistics background
  - computing background

Homework

- some problems will cover ideas not covered in class
- working together is OK
- try to work on your own
- write-up must be your own
- do not use solutions from previous years
- submission by GitHub at `http://github.uiowa.edu` or by **Icon** at `http://icon.uiowa.edu/`.

Project

- Find a topic you are interested in.
- Written report plus possibly some form of presentation.

Ask Questions

- Ask questions if you are confused or think a point needs more discussion.
- Questions can lead to interesting discussions.

Computational Tools

Computers and Operating Systems

- We will use software available on the Linux workstations in the Mathematical Sciences labs (Schaeffer 346 in particular).
- Most things we will do can be done remotely by using `ssh` to log into one of the machines in Schaeffer 346 using `ssh`. These machines are

  `l-lnx2`*xy*`.stat.uiowa.edu`

  with $xy = 00, 01, 02, \dots, 19$.
- You can also access the CLAS Linux systems using a browser at

  `http://fastx.divms.uiowa.edu/`

  - This connects you to one of several servers.
  - It is OK to run small jobs on these servers.
  - For larger jobs you should log into one of the lab machines.
- Most of the software we will use is available free for installing on any Linux, Mac OS X, or Windows computer.
- You are free to use any computer you like, but I will be more likely to be able to help you resolve problems you run into if you are using the lab computers.

Git and GitHub

- Git is a *version control system* that is very useful for keeping track of revision history and collaboration.
- We will be using the University's GitHub server.
- *Today* you should log into the page `http://github.uiowa.edu` with your HawkID.
- I will then create a repository for you to use within the class organization at `https://github.uiowa.edu/STAT7400-Spring-2019`.
- A brief introduction to Git is available at `http://www.stat.uiowa.edu/~luke/classes/STAT7400/git.html`.

What You Will Need

- You will need to know how to
  - run R
  - Compile and run C programs
- Other Tools you may need:
  - text editor
  - command shell
  - `make`, `grep`, etc.

Class Web Pages

The class web page

```
http://www.stat.uiowa.edu/~luke/classes/STAT7400/
```

contains some pointers to available tools and documentation. It will be updated throughout the semester.

Reading assignments and homework will be posted on the class web pages.

Computing Account Setup: Do This Today!

- Make sure you are able to log into the CLAS Linux systems with your HawkID and password. The resources page at

  ```
  http://www.stat.uiowa.edu/~luke/classes/
  STAT7400/resources.html
  ```

  provides some pointers on how to do this. If you cannot, please let me know immediately.

- If you have not done so already, log into the page

  ```
  http://github.uiowa.edu
  ```

  with your HawkID to activate your GitHub account.

Computational Statistics, Statistical Computing, and Data Science

Computational Statistics: Statistical procedures that depend heavily on computation.

- Statistical graphics
- Bootstrap
- MCMC
- Smoothing
- Machine lerning
- . . .

Statistical Computing: Computational tools for data analysis.

- Numerical analysis
- Optimization
- Design of statistical languages
- Graphical tools and methods
- ...

Data Science: A more recent term, covering areas like

- Accessing and cleaning data
- Working with big data
- Working with complex and non-standard data
- Machine learning methods
- Graphics and visualization
- . . .

Overlap: The division is not sharp; some consider the these terms to be equivalent.

Course Topics

- The course will cover, in varying levels of detail, a selection from these topics in *Computational Statistics*, *Statistical Computing*, and *Data Science*:
  - basics of computer organization
  - data technologies
  - graphical methods and visualization
  - random variate generation
  - design and analysis of simulation experiments
  - bootstrap
  - Markov chain Monte Carlo
  - basics of computer arithmetic
  - numerical linear algebra
  - optimization algorithms for model fitting
  - smoothing
  - machine learning and data mining
  - parallel computing in statistics
  - symbolic computation
  - use and design of high level languages
- Some topics will be explored in class, some in homework assignments.
- Many could fill an entire course; we will only scratch the surface.
- Your project is an opportunity to go into more depth on one or more of these areas.
- The course will interleave statistical computing with computational statistics and data science; progression through the topics covered will not be linear.
- Working computer assignments and working on the project are the most important part.

- Class discussions of issues that arise in working problems can be very valuable, so raise issues for discussion.
- Class objectives:
  - Become familiar with some ideas from computational statistics, statistical computing, and data science.
  - Develop skills and experience in using the computer as a research tool.

Thumbnail Sketch of R

- R is a language for statistical computing and graphics.
- Related to the S language developed at Bell Labs.
- High level language
  - somewhat functional in nature
  - has some object-oriented features
  - interactive
  - can use compiled C or FORTRAN code
- many built-in features and tools
- well developed extension mechanism (packages)
  - tools for writing packages
  - many contributed packages available.

Some examples:

- Fitting a linear regression to simulated data:

```
> x <- c(1,2,3,4,3,2,1)
> y <- rnorm(length(x), x + 2, 0.2)
> lm(y ~ x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      1.887        1.019
```

- A function to sum the values in a vector

```
> mysum <- function(x) {
+     s <- 0
+     for (y in x) s <- s + y
+     s
+ }
> mysum(1:10)
[1] 55
```

Thumbnail Sketch of C

- C is a low level language originally developed for systems programming
- Originally developed at Bell Labs for programming UNIX
- Can be used to write very efficient code
- Can call libraries written in C, FORTRAN, etc. on most systems
- A reasonable book on C is *Practical C Programming, 3rd Edition*, By Steve Oualline. The publisher's web site is

  `http://www.oreilly.com/catalog/pcp3/`

  There are many other good books available.
- A simple example program is available at

  `http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/hello`.

Example: summing the numbers in a vector:

```
#include <stdio.h>

#define N 1000000
#define REPS 1000

double x[N];

double sum(int n, double *x)
{
    double s;
    int i;

    s = 0.0;
    for (i = 0; i < N; i++) {
        s = s + x[i];
    }
    return s;
}

int main()
{
    double s;
    int i, j;

    for (i = 0; i < N; i++)
        x[i] = i + 1;

    for (j = 0; j < REPS; j++)
        s = sum(N, x);

    printf("sum = %f\n", s);
    return 0;
}
```

Speed Comparisons

Consider two simple problems:

- computing the sum of a vector of numbers
- computing the dot product of two vectors

The directory

```
http://www.stat.uiowa.edu/~luke/classes/STAT7400/
                    examples/speed
```

contains code for these problems in C, Lisp-Stat, and R. Timings are obtained with commands like

```
time ddot
```

for the C versions, and

```
x<-as.double(1:1000000)
system.time(for (i in 1:1000) ddot(x,x))
```

for R.

The results:

Sum	Time (sec)	base = C	base = C -O2
C sum	2.33	1.00	2.21
C sum -O2	1.05	0.45	1.00
R sum	0.81	0.35	0.77
R mysum	21.42	9.21	20.40
C sumk	7.92	3.41	7.54
C sumk -O2	4.21	1.81	4.00
R mysumk	83.15	35.76	79.19

Dot Product	Time (sec)	base = C	base = C -O2
C ddot	2.34	1.00	2.25
C ddot -O2	1.04	0.45	1.00
R ddot	47.85	20.47	46.01
R crossp	1.46	0.63	1.40

Notes:

- R sum means built-in `sum`; R crossp means `crossprod`
- `sumk` and `mysumk` use Kahan summation.

Some conclusions and comments:

- Low level languages like C *can* produce much faster code.
- It is much easier to develop code in an interactive, high level language.
- Usually the difference is *much* less.
- Improvements in high level language runtime systems (e.g. byte compilation, runtime code generation) can make a big difference.
- Using the right high level language function (e.g. `sum`) can eliminate the difference.
- High level language functions may be able to take advantage of multiple cores.
- Speed isn't everything: accuracy is most important!

Basic Computer Architecture

Typical Machine Layout

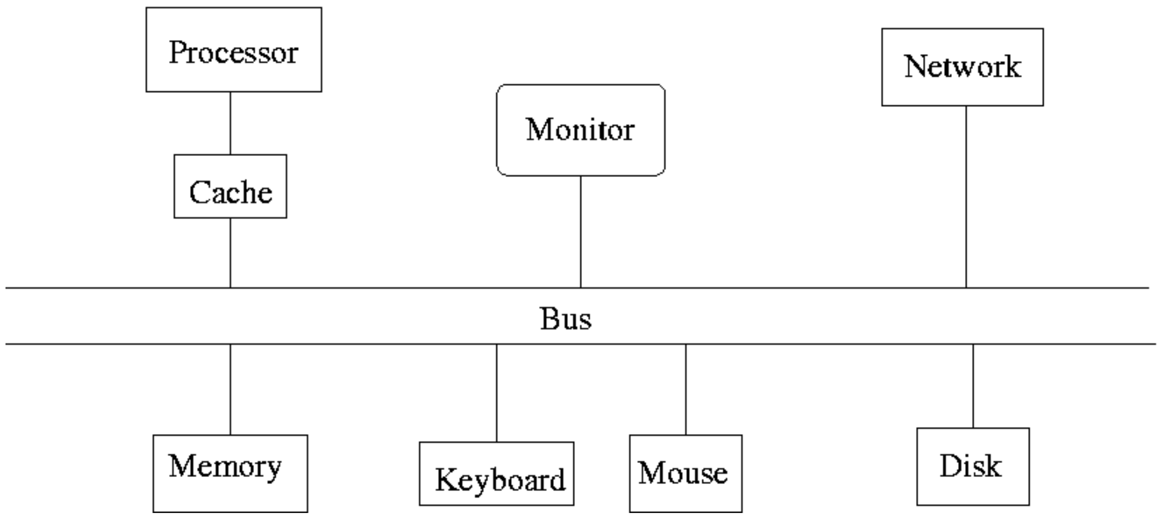


Figure based on M. L. Scott, *Programming Language Pragmatics*, Figure 5.1, p. 205

Structure of Lab Workstations

Processor and Cache

```
luke@l-lnx200 ~% lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  2
Core(s) per socket:  4
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               94
Model name:          Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
Stepping:            3
CPU MHz:             3895.093
CPU max MHz:         4000.0000
CPU min MHz:         800.0000
BogoMIPS:            6816.00
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            8192K
NUMA node0 CPU(s):   0-7
Flags: ...
```

- There is a single *quad-core* processor with *hyperthreading* that acts like eight separate processors
- Each has 8Mb of L3 cache

Memory and Swap Space

```
luke@l-lnx200 ~% free
              total        used        free      shared  buff/cache   available
Mem:       32866464      396876    27076056       33620     5393532    31905476
Swap:      16449532           0    16449532
```

- The workstations have about 32G of memory.
- The swap space is about 16G.

Disk Space

Using the `df` command produces:

```
luke@l-lnx200 ~% df
luke@l-lnx200 ~% df
Filesystem                     1K-blocks      Used Available Use% Mounted on
...
/dev/mapper/vg00-root           65924860  48668880  13884156  78% /
/dev/mapper/vg00-tmp             8125880     28976   7661092   1% /tmp
/dev/mapper/vg00-var            75439224  13591304  57992768  19% /var
/dev/mapper/vg00-scratch       622877536     33068 622844468   1% /var/scratch
...
netapp2:/vol/grad              553648128 319715584 233932544  58% /mnt/nfs/netapp2/grad
...
netapp2:/vol/students          235929600  72504448 163425152  31% /mnt/nfs/netapp2/students
...
```

- Local disks are large but mostly unused
- Space in `/var/scratch` can be used for temporary storage.
- User space is on network disks.
- Network speed can be a bottle neck.

Performance Monitoring

- Using the `top` command produces:

```
top - 11:06:34 up  4:06,  1 user,  load average: 0.00, 0.01, 0.05
Tasks: 127 total,   1 running, 126 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni, 99.8%id,  0.2%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  16393524k total,   898048k used, 15495476k free,   268200k buffers
Swap: 18481148k total,        0k used, 18481148k free,   217412k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1445 root      20   0  445m  59m  23m S  2.0  0.4   0:11.48 kdm_greet
    1 root      20   0 39544 4680 2036 S  0.0  0.0   0:01.01 systemd
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    5 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kworker/u:0
    7 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/u:0H
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    9 root      RT   0     0    0    0 S  0.0  0.0   0:00.07 watchdog/0
   10 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1
   12 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/1:0H
   13 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/1
   14 root      RT   0     0    0    0 S  0.0  0.0   0:00.10 watchdog/1
   15 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/2
   17 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/2:0H
   18 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/2
...
```

- Interactive options allow you to kill or *renice* (change the priority of) processes you own.
- The command `htop` may be a little nicer to work with.
- A GUI tool, **System Monitor**, is available from one of the menus. From the command line this can be run as `gnome-system-monitor`.
- Another useful command is `ps` (process status)

```
luke@l-lnx200 ~% ps -u luke
  PID TTY          TIME CMD
 4618 ?        00:00:00 sshd
 4620 pts/0    00:00:00 tcsh
 4651 pts/0    00:00:00 ps
```

  There are many options; see `man ps` for details.

Processors

Basics

- Processors execute a sequence of instructions
- Each instruction requires some of
  - decoding instruction
  - fetching operands from memory
  - performing an operation (add, multiply, ...)
  - etc.
- Older processors would carry out one of these steps per clock cycle and then move to the next.
- most modern processors use *pipelining* to carry out some operations in parallel.

Pipelining

A simple example:

```
$s \leftarrow 0$
**for** $i = 1$ to $n$ **do**
    $s \leftarrow s + x_i y_i$
**end**
```

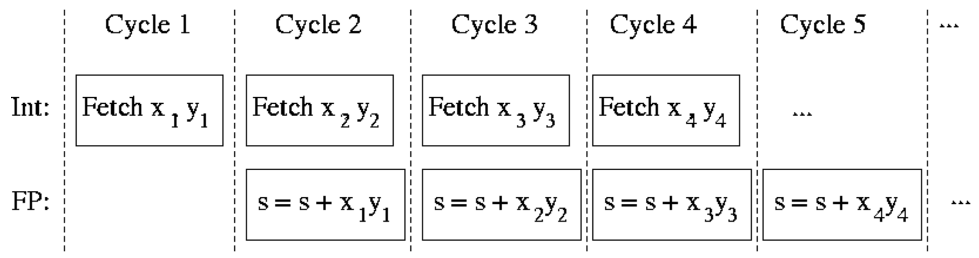
Simplified view: Each step has two parts,

- Fetch x_i and y_i from memory
- Compute $s = s + x_i y_i$

Suppose the computer has two functional units that can operate in parallel,

- An *Integer* unit that can fetch from memory
- A *Floating Point* unit that can add and multiply

If each step takes roughly the same amount of time, a pipeline can speed the computation by a factor of two:



- Floating point operations are much slower than this.
- Modern chips contain many more separate functional units.
- Pipelines can have 10 or more stages.
- Some operations take more than one clock cycle.
- The compiler or the processor orders operations to keep the pipeline busy.
- If this fails, then the pipeline *stalls*.

Superscalar Processors, Hyper-Threading, and Multiple Cores

- Some processors have enough functional units to have more than one pipeline running in parallel.
- Such processors are called *superscalar*
- In some cases there are enough functional units per processor to allow one physical processor to pretend like it is two (somewhat simpler) logical processors. This approach is called *hyper-threading*.
  - Hyper-threaded processors on a single physical chip share some resources, in particular cache.
  - Benchmarks suggest that hyper-threading produces about a 20% speed-up in cases where dual physical processors would produce a factor of 2 speed-up
- Recent advances allow full replication of processors within one chip; these are *multi core* processors.
  - Multi-core machines are effectively full multi-processor machines (at least for most purposes).
  - Dual core processors are now ubiquitous.
  - The machines in the department research cluster have two dual core processors, or four effective processors.
  - Our lab machines have a single quad core processor.
  - Processors with 6 or 8 or even more cores are available.
- Many processors support some form of vectorized operations, e.g. SSE2 (Single Instruction, Multiple Data, Extensions 2) on Intel and AMD processors.

Implications

- Modern processors achieve high speed though a collection of clever tricks.
- Most of the time these tricks work extremely well.
- Every so often a small change in code may cause pipelining heuristics to fail, resulting in a pipeline stall.
- These small changes can then cause large differences in performance.
- The chances are that a “small change” in code that causes a large change in performance was not in fact such a small change after all.
- Processor speeds have not been increasing very much recently.
- Many believe that speed improvements will need to come from increased use of explicit parallel programming.
- More details are available in a talk at

  `http://www.infoq.com/presentations/`
  `click-crash-course-modern-hardware`

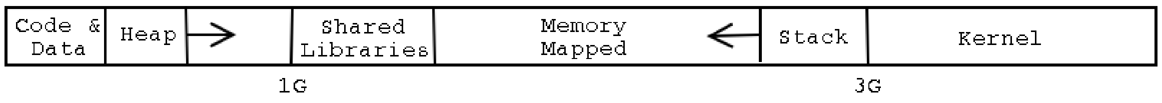
Memory

Basics

- Data and program code are stored in memory.
- Memory consists of *bits* (binary integers)
- On most computers
  - bits are collected into groups of eight, called *bytes*
  - there is a natural *word size* of W bits
  - the most common value of W is still 32; 64 is becoming more common; 16 also occurs
  - bytes are numbered consecutively, $0, 1, 2, \ldots, N = 2^W$
  - an *address* for code or data is a number between 0 and N representing a location in memory, usually in bytes.
  - $2^{32} = 4,294,967,296 = 4\text{GB}$
  - The maximum amount of memory a 32-bit process can address is 4 Gigabytes.
  - Some 32-bit machines can use more than 4G of memory, but each process gets at most 4G.
  - Most hard disks are *much* larger than 4G.

Memory Layout

- A process can conceptually access up to 2^W bytes of address space.
- The operating system usually reserves some of the address space for things it does on behalf of the process.
- On 32-bit Linux the upper 1GB is reserved for the operating system kernel.
- Only a portion of the usable address space has memory allocated to it.
- Standard 32-bit Linux memory layout:

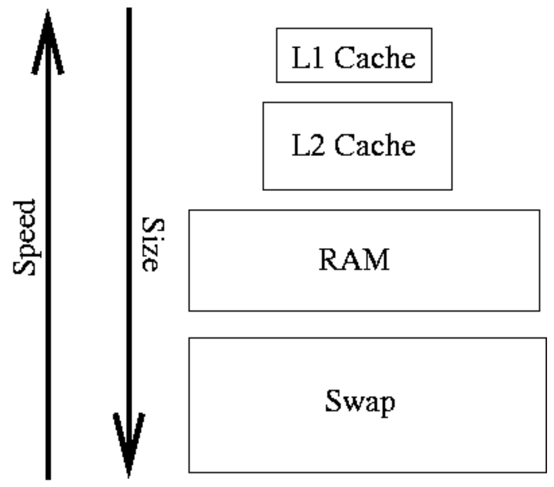


- Standard heap can only grow to 1G.
- `malloc` implementations can allocate more using memory mapping.
- Obtaining large amounts of contiguous address space can be hard.
- Memory allocation can slow down when memory mapping is needed.
- Other operating systems differ in detail only.
- 64-bit machines are much less limited.
- The design matrix for n cases and p variables stored in double precision needs $8np$ bytes of memory.

	$p = 10$	$p = 100$	$p = 1000$
n = 100	8,000	80,000	800,000
n = 1,000	80,000	800,000	8,000,000
n = 10,000	800,000	8,000,000	80,000,000
n = 100,000	8,000,000	80,000,000	800,000,000

Virtual and Physical Memory

- To use address space, a process must ask the kernel to map physical space to the address space.
- There is a hierarchy of physical memory:



- Hardware/OS hides the distinction.
- Caches are usually on or very near the processor chip and very fast.
- RAM usually needs to be accessed via the bus
- The hardware/OS try to keep recently accessed memory and locations nearby in cache.

- A simple example:

```
msum <- function(x) {
    nr <- nrow(x)
    nc <- ncol(x)
    s <- 0
    for (i in 1 : nr)
        for (j in 1 : nc)
            s <- s + x[i, j]
    s
}
m <- matrix(0, nrow = 5000000, 2)
system.time(msum(m))
##   user  system elapsed
##  1.712   0.000   1.712
fix(msum) ## reverse the order of the sums
system.time(msum(m))
##   user  system elapsed
##  0.836   0.000   0.835
```

- Matrices are stored in *column major order*.
- This effect is more pronounced in low level code.
- Careful code tries to preserve *locality of reference*.

Registers

- Registers are storage locations on the processor that can be accessed very fast.
- Most basic processor operations operate on registers.
- Most processors have separate sets of registers for integer and floating point data.
- On some processors, including i386, the floating point registers have *extended precision*.
- The i386 architecture has few registers, 8 floating point, 8 integer data, 8 address; some of these have dedicated purposes. Not sure about x86_64 (our lab computers).
- RISC processors usually have 32 or more of each kind.
- Optimizing compilers work hard to keep data in registers.
- Small code changes can cause dramatic speed changes in optimized code because they make it easier or harder for the compiler to keep data in registers.
- If enough registers are available, then some function arguments can be passed in registers.
- Vector support facilities, like SSE2, provide additional registers that compilers may use to improve performance.

Processes and Shells

- A *shell* is a command line interface to the computer's operating system.
- Common shells on Linux and MacOS are `bash` and `tcsh`.
- You can now set your default Linix shell at `https://hawkid.uiowa.edu/`
- Shells are used to interact with the file system and to start processes that run programs.
- You can set process limits and environment variables the shell.
- Programs run from shells take command line arguments.

Some Basic `bash`/`tcsh` Commands

- `hostname` prints the name of the computer the shell is running on.
- `pwd` prints the current working directory.
- `ls` lists files a directory
  - `ls` lists files in the current directory.
  - `ls foo` lists files in a sub-directory `foo`.
- `cd` changes the working directory:
  - `cd` or `cd`   moves to your home directory;
  - `cd foo` moves to the sub-directory `foo`;
  - `cd ..` moves up to the parent directory;
- `mkdir foo` creates a new sub-directory `foo` in your current working directory;
- `rm`, `rmdir` can be used to remove files and directories; **BE VERY CAREFUL WITH THESE!!!**

Standard Input, Standard Output, and Pipes

- Programs can also be designed to read from *standard input* and write to *standard output*.
- Shells can redirect standard input and standard output.
- Shells can also connect processes into *pipelines*.
- On multi-core systems pipelines can run in parallel.
- A simple example using the `bash` shell script `P1.sh`

  ```
  #!/bin/bash

  while true; do echo $1; done
  ```

  and the `rev` program can be run as

  ```
  bash P1.sh fox
  bash P1.sh fox > /dev/null
  bash P1.sh fox | rev
  bash P1.sh fox | rev > /dev/null
  bash P1.sh fox | rev | rev > /dev/null
  ```

The `proc` File System

- The `proc` file system allows you to view many aspects of a process.

Computer Arithmetic

Computer Arithmetic in Hardware

- Computer hardware supports two kinds of numbers:
  - fixed precision integers
  - floating point numbers
- Computer integers have a limited range
- Floating point numbers are a finite subset of the (extended) real line.

Overflow

- Calculations with native computer integers can overflow.
- Low level languages usually do not detect this.
- Calculations with floating point numbers can also overflow to $\pm\infty$.

Underflow

- Floating point operations can also underflow (be rounded to zero).

A Simple Example

A simple C program, available in

```
http://www.stat.uiowa.edu/~luke/classes/STAT7400/
                    examples/fact
```

that calculates *n*! using integer and double precision floating point produces

```
luke@itasca2 notes% ./fact 10
ifac = 3628800, dfac = 3628800.000000
luke@itasca2 notes% ./fact 15
ifac = 2004310016, dfac = 1307674368000.000000
luke@itasca2 notes% ./fact 20
ifac = -2102132736, dfac = 2432902008176640000.000000
luke@itasca2 notes% ./fact 30
ifac = 1409286144, dfac = 265252859812191032188804700045312.000000
luke@itasca2 notes% ./fact 40
ifac = 0, dfac = 815915283247897683795548521301193790359984930816.00000
luke@itasca2 fact% ./fact 200
ifac = 0, dfac = inf
```

- Most current computers include $\pm\infty$ among the finite set of representable real numbers.
- How this is used may vary:
  - On our x86_64 Linux workstations:

    ```
    > exp(1000)
    [1] Inf
    ```
  - On a PA-RISC machine running HP-UX:

    ```
    > exp(1000)
    [1] 1.797693e+308
    ```

    This is the largest finite floating point value.

Arithmetic in R

Higher level languages may at least detect integer overflow. In R,

```
> typeof(1:100)
[1] "integer"
> p<-as.integer(1)   # or p <- 1L
> for (i in 1:100) p <- p * i
Warning message:
NAs produced by integer overflow in: p * i
> p
[1] NA
```

Floating point calculations behave much like the C version:

```
> p <- 1
> for (i in 1:100) p <- p * i
> p
[1] 9.332622e+157
> p <- 1
> for (i in 1:200) p <- p * i
> p
[1] Inf
```

The `prod` function converts its argument to double precision floating point before computing its result:

```
> prod(1:100)
[1] 9.332622e+157
> prod(1:200)
[1] Inf
```

Bignum and Arbitrary Precision Arithmetic

Other high-level languages may provide

- arbitrarily large integers(often called *bignums*)
- rationals (ratios of arbitrarily large integers)

Some also provide arbitrary precision floating point arithmetic.

In Mathematica:

```
In[3]:= Factorial[100]

Out[3]= 93326215443944152681699238856266700490715968264381621468592963895217 5\
>    99999322991560894146397615651828625369792082722375825118521091686400000 00\
>    0000000000000000
```

In R we can use the `gmp` package available from CRAN:

```
> prod(as.bigz(1:100))
[1]    "93326215443944152681699238856266700490715968264381621468592963895217 5
     99999322991560894146397615651828625369792082722375825118521091686400000 00
     0000000000000000"
```

- The output of these examples is slightly edited to make comparison easier.
- These calculations are *much* slower than floating point calculations.
- C now supports `long double` variables, which are often (but not always!) slower than `double` but usually provide more accuracy.
- Some FORTRAN compilers also support *quadruple precision* variables.

Rounding Errors

A simple, doubly stochastic 2×2 Markov transition matrix:

```
> p <- matrix(c(1/3, 2/3, 2/3,1/3),nrow=2)
> p
          [,1]      [,2]
[1,] 0.3333333 0.6666667
[2,] 0.6666667 0.3333333
```

Theory says:

$$P^n \to \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$$

Let's try it:

```
> q <- p
> for (i in 1:10) q <- q %*% q
> q
     [,1] [,2]
[1,]  0.5  0.5
[2,]  0.5  0.5
```

The values aren't exactly equal to 0.5 though:

```
> q - 0.5
              [,1]          [,2]
[1,] -1.776357e-15 -1.776357e-15
[2,] -1.776357e-15 -1.776357e-15
```

We can continue:

```
> for (i in 1:10) q <- q %*% q
> q
     [,1] [,2]
[1,]  0.5  0.5
[2,]  0.5  0.5
> for (i in 1:10) q <- q %*% q
> for (i in 1:10) q <- q %*% q
> q
          [,1]      [,2]
[1,] 0.4999733 0.4999733
[2,] 0.4999733 0.4999733
```

Rounding error has built up.

Continuing further:

```
> for (i in 1:10) q <- q %*% q
> q
          [,1]      [,2]
[1,] 0.4733905 0.4733905
[2,] 0.4733905 0.4733905
> for (i in 1:10) q <- q %*% q
> q
             [,1]         [,2]
[1,] 2.390445e-25 2.390445e-25
[2,] 2.390445e-25 2.390445e-25
> for (i in 1:10) q <- q %*% q
> for (i in 1:10) q <- q %*% q
> for (i in 1:10) q <- q %*% q
> q
     [,1] [,2]
[1,]    0    0
[2,]    0    0
```

As another example, the log-likelihood for right-censored data includes terms of the form $\log(1-F(x))$. For the normal distribution, this can be computed as

```
log(1 - pnorm(x))
```

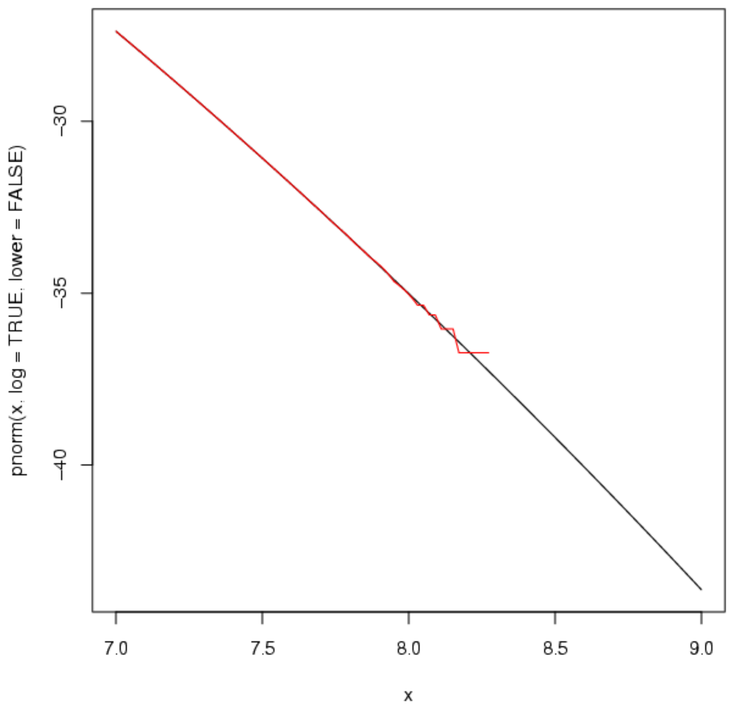
An alternative is

```
pnorm(x, log = TRUE, lower = FALSE)
```

The expressions

```
x <- seq(7,9,len=100)
plot(x, pnorm(x, log = TRUE,lower = FALSE), type = "l")
lines(x, log(1 - pnorm(x)), col = "red")
```

produce the plot



Some notes:

- The problem is called *catastrophic cancellation*.
- Floating point arithmetic is not associative or distributive.
- The range considered here is quite extreme, but can be important in some cases.
- The expression `log(1 - pnorm(x))` produces invalid results ($-\infty$) for `x` above roughly 8.3.
- Most R cdf functions allow `lower.tail` and `log.p` arguments (shortened to `log` and `lower` here)
- The functions `expm1` and `log1p` can also be useful.

  $$\begin{aligned}\texttt{expm1(x)} &= e^x - 1\\ \texttt{log1p(x)} &= \log(1+x)\end{aligned}$$

  These functions also exist in the standard C math library.

Another illustration is provided by the behavior of the expression

$$e^{-2x^2} - e^{-8x^2}$$

near the origin:

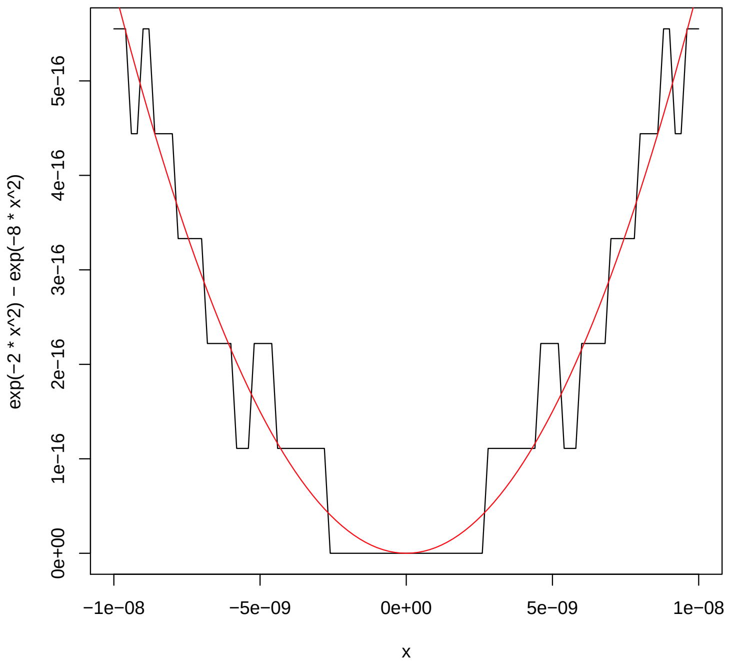
```
x <- seq(-1e-8, 1e-8, len = 101)
plot(x, exp(-2 * x ^ 2) - exp(-8 * x ^ 2), type = "l")
```

Rewriting the expression as

$$e^{-2x^2}\left(1 - e^{-6x^2}\right) = -e^{-2x^2}\mathrm{expm1}(-6x^2)$$

produces a more stable result:

```
lines(x, -exp(-2 * x ^ 2) * expm1(-6 * x ^ 2), col = "red")
```



Example: Sample Standard Deviations

```
> x <- 100000000000000 + rep(c(1,2), 5)
> x
 [1] 1e+14 1e+14 1e+14 1e+14 1e+14 1e+14 1e+14 1e+14 1e+14 1e+14
> print(x, digits = 16)
 [1] 100000000000001 100000000000002 100000000000001 100000000000002
 [5] 100000000000001 100000000000002 100000000000001 100000000000002
 [9] 100000000000001 100000000000002
> n <- length(x)
> s <- sqrt((sum(x^2) - n * mean(x)^2) / (n - 1))
> s
[1] 0
> s == 0
[1] TRUE
> y <- rep(c(1,2), 5)
> y
 [1] 1 2 1 2 1 2 1 2 1 2
> sqrt((sum(y^2) - n * mean(y)^2) / (n - 1))
[1] 0.5270463
> sd(x)
[1] 0.5270463
> sd(y)
[1] 0.5270463
```

- The "computing formula" $\sum x_i^2 - n\overline{x}^2$ is not numerically stable.
- A two-pass algorithm that first computes the mean and then computes $\sum(x_i - \overline{x})^2$ works much better.
- There are also reasonably stable one-pass algorithms.

Example: Truncated Normal Distribution

- Sometimes it is useful to simulate from a standard normal distribution conditioned to be at least a, or truncated from below at a.
- The CDF is
$$F(x|a) = \frac{\Phi(x) - \Phi(a)}{1 - \Phi(a)}$$
for $x \geq a$.
- The inverse CDF is
$$F^{-1}(u|a) = \Phi^{-1}(\Phi(a) + u(1 - \Phi(a)))$$
- This can be computed using

```
Finv0 <- function(u, a) {
    p <- pnorm(a)
    qnorm(p + u * (1 - p))
}
```

- Some plots:

```
u <- (1:100) / 101
plot(u, Finv0(u, 0), type = "l")
plot(u, Finv0(u, 2), type = "l")
plot(u, Finv0(u, 4), type = "l")
plot(u, Finv0(u, 8), type = "l")
```

- An improved version:

```
Finv1 <- function(u, a) {
    q <- pnorm(a, lower.tail = FALSE)
    qnorm(q * (1 - u), lower.tail = FALSE)
}

lines(u, Finv1(u, 8), col = "red")
```

- This could be further improved if the tails need to be more accurate.

Integer Arithmetic

- Integer data types can be signed or unsigned; they have a finite range.
- Almost all computers now use *binary place-value* for unsigned integers and *two's complement* for signed integers.
- Ranges are

$$\begin{array}{ll} \text{unsigned:} & 0,1,\dots,2^n-1 \\ \text{signed:} & -2^{n-1},\dots,2^{n-1}-1. \end{array}$$

  For C `int` and Fortran `integer` the value $n = 32$ is almost universal.
- If the result of `+`, `*`, or `-` is representable, then the operation is exact; otherwise it *overflows*.
- The result of `/` is typically truncated; some combinations can overflow.
- Typically overflow is silent.
- Integer division by zero signals an error; on Linux a SIGFPE (floating point error signal!) is sent.
- statistical calculations rarely need to use integers directly except
  - as dimension sizes and indices for arrays
  - as sizes in storage allocation requests
  - as frequencies for categorical data
- Storing scaled floating point values as small integers (e.g. single bytes) can save space.
- As data sets get larger, being able to represent integers larger than $2^{31}-1$ is becoming important.
- Double precision floating point numbers can represent integers up to 2^{53} exactly.

- Detecting integer overflow portably is hard; one possible strategy: use double precision floating point for calculation and check whether the result fits.
  - This works if integers are 32-bit and double precision is 64-bit IEEE
  - These assumptions are almost universally true but should be tested at compile time.

  Other strategies may be faster, in particular for addition, but are harder to implement.

- You can find out how R detects integer overflow by looking in the file

  ```
  src/main/arithmetic.c
  ```

  The R sources are available at

  ```
  https://svn.r-project.org/R/
  ```

Floating Point Arithmetic

- Floating point numbers are represented by a sign s, a *significand* or *mantissa* sig, and an *exponent* exp; the value of the number is

$$(-1)^s \times sig \times 2^{exp}$$

  The significand and the exponent are represented as binary integers.

- Bases other than 2 were used in the past, but virtually all computers now follow the IEEE standard number 754 (IEEE 754 for short; the corresponding ISO standard is ISO/IEC/IEEE 60559:2011).

- IEEE 754 specifies the number of bits to use:

	sign	significand	exponent	total
single precision	1	23	8	32
double precision	1	52	11	64
extended precision	1	64	15	80

- A number is *normalized* if $1 \leq sig < 2$. Since this means it looks like

$$1.something \times 2^{exp}$$

  we can use all bits of the mantissa for the *something* and get an extra bit of precision from the implicit leading one.

- Numbers smaller in magnitude than $1.0 \times 2^{exp_{min}}$ can be represented with reduced precision as

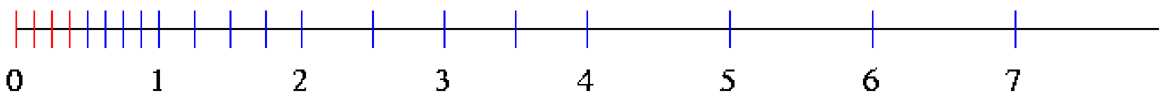
$$0.something \times 2^{exp_{min}}$$

  These are *denormalized* numbers.

- Denormalized numbers allow for *gradual underflow*. IEEE 745 includes them; many older approaches did not.

- Some GPUs set denormalized numbers to zero.

For a significand with three bits, $exp_{min} = -1$, and $exp_{max} = 2$ the available nonnegative floating point numbers look like this:



Normalized numbers are blue, denormalized numbers are red.

- Zero is not a normalized number (but all representations include it).
- Without denormalized numbers, the gap between zero and the first positive number is larger than the gap between the first and second positive numbers.

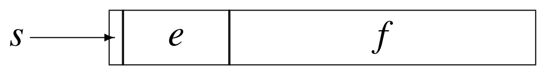
There are actually two zeros in this framework: $+0$ and -0. One way to see this in R:

```
> zp <- 0        ## this is read as +0
> zn <- -1 * 0   ## or zn <- -0; this produces -0
> zn == zp
[1] TRUE
> 1 / zp
[1] Inf
> 1 / zn
[1] -Inf
```

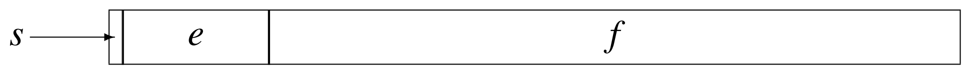
This can identify the direction from which underflow occurred.

- The IEEE 754 representation of floating point numbers looks like

Single precision, exponent bias $b = 127$



Double precision, exponent bias $b = 1023$



- The exponent is represented by a nonnegative integer e from which a *bias* b is subtracted.
- The fractional part is a nonnegative integer f.
- The representation includes several special values: $\pm\infty$, NaN (*Not a Number*) values:

	e	f	Value
Normalized	$1 \leq e \leq 2b$	*any*	$\pm 1.f \times 2^{e-b}$
Denormalized	0	$\neq 0$	$\pm 0.f \times 2^{-b+1}$
Zero	0	0	± 0
Infinity	$2b+1$	0	$\pm\infty$
NaN	$2b+1$	$\neq 0$	NaN

- $1.0/0.0$ will produce $+\infty$; $0.0/0.0$ will produce NaN.
- On some systems a flag needs to be set so $0.0/0.0$ does not produce an error.
- Library functions like exp, log will behave predictably on most systems, but there are still some where they do not.
- Comparisons like `x <= y` or `x == y` should produce `FALSE` if one of the operands is NaN; most Windows C compilers violate this.
- Range of exactly representable integers in double precision:

$$\pm(2^{53} - 1) \approx \pm 9.0072 \times 10^{15}$$

- Smallest positive (denormalized) double precision number:

$$2^{-b+1} \times 2^{-52} = 2^{-1074} \approx 4.940656 \times 10^{-324}$$

Machine Characteristics

Machine Characteristics in R

The variable `.Machine` contains values for the characteristics of the current machine:

```
> .Machine
$double.eps
[1] 2.220446e-16
$double.neg.eps
[1] 1.110223e-16
$double.xmin
[1] 2.225074e-308
$double.xmax
[1] 1.797693e+308
$double.base
[1] 2
$double.digits
[1] 53
$double.rounding
[1] 5
$double.guard
[1] 0
$double.ulp.digits
[1] -52
$double.neg.ulp.digits
[1] -53
$double.exponent
[1] 11
$double.min.exp
[1] -1022
$double.max.exp
[1] 1024
$integer.max
[1] 2147483647
$sizeof.long
[1] 8
$sizeof.longlong
[1] 8
$sizeof.longdouble
[1] 16
$sizeof.pointer
[1] 8
```

The help page gives details.

Machine Epsilon and Machine Unit

Let m be the smallest and M the largest positive finite normalized floating point numbers.

Let $\text{fl}(x)$ be the closest floating point number to x.

Machine Unit

The *machine unit* is the smallest number $\mathbf{u}$ such that

$$|\text{fl}(x) - x| \leq \mathbf{u}\,|x|$$

for all $x \in [m, M]$; this implies that for every $x \in [m, M]$

$$\text{fl}(x) = x(1+u)$$

for some u with $|u| \leq \mathbf{u}$. For double precision IEEE arithmetic,

$$\mathbf{u} = \frac{1}{2}2^{1-53} = 2^{-53} \approx 1.110223 \times 10^{-16}$$

Machine Epsilon

The *machine epsilon* ε_m is the smallest number x such that

$$\text{fl}(1+x) \neq 1$$

For double precision IEEE arithmetic,

$$\varepsilon_m = 2^{-52} = 2.220446 \times 10^{-16} = 2\mathbf{u}$$

$\mathbf{u}$ and ε_m are very close; they are sometimes used interchangeably.

Computing Machine Constants

A standard set of routines for computing machine information is provided by

> Cody, W. J. (1988) MACHAR: A subroutine to dynamically determine machine parameters. Transactions on Mathematical Software, 14, 4, 303-311.

Simple code for computing machine epsilon is in

```
http://www.stat.uiowa.edu/~luke/classes/STAT7400/
                  examples/macheps
```

The R code looks like

```
eps <- 2
neweps <- eps / 2
while (1 + neweps != 1) {
    eps <- neweps
    neweps <- neweps / 2.0
}
eps
```

and produces

```
> eps
[1] 2.220446e-16
> .Machine$double.eps
[1] 2.220446e-16
> eps == .Machine$double.eps
[1] TRUE
```

Analogous C code compiled with

```
cc -Wall -pedantic -o eps eps.c
```

produces

```
luke@itasca2 macheps% ./eps
epsilon = 2.22045e-16
```

The *same* C code compiled with optimization (and an older gcc compiler) on a i386 system

```
cc -Wall -pedantic -o epsO2 eps.c -O2
```

produces

```
luke@itasca2 macheps% ./epsO2
epsilon = 1.0842e-19
```

Why does this happen?

Here is a hint:

```
> log2(.Machine$double.eps)
[1] -52
> log2(1.0842e-19)
[1] -63
```

Some Notes

- Use equality tests `x == y` for floating point numbers with caution
- Multiplies can overflow—use logs (log likelihoods)
- Cases where care is needed:
  - survival likelihoods
  - mixture likelihoods.
- Double precision helps a lot

Floating Point Equality

- R FAQ 7.31: Why doesn't R think these numbers are equal?

  ```
  > b <- 1 - 0.8
  > b
  [1] 0.2
  > b == 0.2
  [1] FALSE
  > b - 0.2
  [1] -5.551115e-17
  ```

- Answer from FAQ:

  The only numbers that can be represented exactly in R's numeric type are integers and fractions whose denominator is a power of 2. Other numbers have to be rounded to (typically) 53 binary digits accuracy. As a result, two floating point numbers will not reliably be equal unless they have been computed by the same algorithm, and not always even then. For example

  ```
  > a <- sqrt(2)
  > a * a == 2
  [1] FALSE
  > a * a - 2
  [1] 4.440892e-16
  ```

  The function `all.equal()` compares two objects using a numeric tolerance of `.Machine$double.eps ^ 0.5`. If you want much greater accuracy than this you will need to consider error propagation carefully.

- The function `all.equal()` returns either `TRUE` or a string describing the failure. To use it in code you would use something like

  ```
  if (identical(all.equal(x, y), TRUE)) ...
  else ...
  ```

  but using an explicit tolerance test is probably clearer.

- Bottom line: be **VERY CAREFUL** about using equality comparisons with floating point numbers.

Numerical Linear Algebra

Preliminaries

Conditioning and Stability

- Some problems are inherently difficult: no algorithm involving rounding of inputs can be expected to work well. Such problems are called *ill-conditioned.*
- A numerical measure of conditioning, called a *condition number*, can sometimes be defined:
  - Suppose the objective is to compute $y = f(x)$.
  - If x is perturbed by Δx then the result is changed by
  $$\Delta y = f(x+\Delta x) - f(x).$$
  - If
  $$\frac{|\Delta y|}{|y|} \approx \kappa \frac{|\Delta x|}{|x|}$$
  for small perturbations Δx then κ is the *condition number* for the problem of computing $f(x)$.
- A particular algorithm for computing an approximation $\tilde{f}(x)$ to $f(x)$ is *numerically stable* if for small perturbations Δx of the input the result is close to $f(x)$.

Error Analysis

- Analyzing how errors accumulate and propagate through a computation, called *forward error analysis*, is sometimes possible but often very difficult.
- *Backward error analysis* tries to show that the computed result

  $$\tilde{y} = \tilde{f}(x)$$

  is the exact solution to a slightly perturbed problem, i.e.

  $$\tilde{y} = f(\tilde{x})$$

  for some $\tilde{x} \approx x$.
- If
  - the problem of computing $f(x)$ is well conditioned, and
  - the algorithm $\tilde{f}$ is stable,

  then

  $$\begin{aligned} \tilde{y} &= \tilde{f}(x) && \text{computed result} \\ &= f(\tilde{x}) && \text{exact result for some } \tilde{x} \approx x \\ &\approx f(x) && \text{since } f \text{ is well-conditioned} \end{aligned}$$
- Backward error analysis is used heavily in numerical linear algebra.

Solving Linear Systems

Many problems involve solving linear systems of the form

$$Ax = b$$

- least squares normal equations:

$$X^T X\beta = X^T y$$

- stationary distribution of a Markov chain:

$$\begin{aligned} \pi P &= \pi \\ \sum \pi_i &= 1 \end{aligned}$$

If A is $n \times n$ and non-singular then in principle the solution is

$$x = A^{-1}b$$

This is not usually a good numerical approach because

- it can be numerically inaccurate;
- it is inefficient except for very small n.

Triangular Systems

- Triangular systems are easy to solve.
- The upper triangular system

$$\begin{bmatrix} 5 & 3 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \end{bmatrix}$$

  has solution

$$\begin{aligned} x_2 &= 4/2 = 2 \\ x_1 &= (16 - 3x_2)/5 = 10/5 = 2 \end{aligned}$$

- This is called *back substitution*
- Lower triangular systems are solved by *forward substitution*.
- If one of the diagonal elements in a triangular matrix is zero, then the matrix is singular.
- If one of the diagonal elements in a triangular matrix is close to zero, then the solution is very sensitive to other inputs:

$$\begin{bmatrix} 1 & a \\ 0 & \varepsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

  has solution

$$\begin{aligned} x_2 &= \frac{b_2}{\varepsilon} \\ x_1 &= b_1 - a\frac{b_2}{\varepsilon} \end{aligned}$$

- This sensitivity for small ε is inherent in the problem: For small values of ε the problem of finding the solution x is ill-conditioned.

Gaussian Elimination

- The system
$$\begin{bmatrix} 5 & 3 \\ 10 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 16 \\ 36 \end{bmatrix}$$
can be reduced to triangular form by subtracting two times the first equation from the second.

- In matrix form:
$$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 5 & 3 \\ 10 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 16 \\ 36 \end{bmatrix}$$
or
$$\begin{bmatrix} 5 & 3 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \end{bmatrix}$$
which is the previous triangular system.

- For a general 2×2 matrix A the lower triangular matrix used for the reduction is
$$\begin{bmatrix} 1 & 0 \\ -\frac{a_{21}}{a_{11}} & 1 \end{bmatrix}$$

- The ratio $\frac{a_{21}}{a_{11}}$ is a called a *multiplier*.

- This strategy works as long as $a_{11} \neq 0$.

- If $a_{11} \approx 0$, say
$$A = \begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix}$$
for small ε, then the multiplier $1/\varepsilon$ is large and this does not work very well, even though A is very well behaved.

- Using this approach would result in a numerically unstable algorithm for a well-conditioned problem.

Partial Pivoting

- We can ensure that the multiplier is less than or equal to one in magnitude by switching rows before eliminating:

$$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}5 & 3\\ 10 & 8\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}16\\ 36\end{bmatrix}$$

  or

$$\begin{bmatrix}10 & 8\\ 5 & 3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}36\\ 16\end{bmatrix}$$

- The matrix to reduce this system to triangular form is now

$$\begin{bmatrix}1 & 0\\ -0.5 & 1\end{bmatrix}$$

- So the final triangular system is constructed as

$$\begin{bmatrix}1 & 0\\ -0.5 & 1\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}5 & 3\\ 10 & 8\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ -0.5 & 1\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}16\\ 36\end{bmatrix}$$

  or

$$\begin{bmatrix}10 & 8\\ 0 & -1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}36\\ -2\end{bmatrix}$$

- Equivalently, we can think of our original system as

$$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}1 & 0\\ 0.5 & 1\end{bmatrix}\begin{bmatrix}10 & 8\\ 0 & -1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}16\\ 36\end{bmatrix}$$

- The decomposition of A as

$$A = PLU$$

  with P a permutation matrix, L lower trianbular with ones on the diagonal, and U upper triangular is called a *PLU decomposition*.

PLU Decomposition

- In general, we can write a square matrix A as

$$A = PLU$$

  where

  - P is a *permutation matrix*, i.e.
    - it is an identity matrix with some rows switched
    - it satisfies $PP^T = P^TP = I$, i.e. it is an *orthogonal matrix*
  - L is a *unit lower triangular matrix*, i.e.
    - it is lower triangular
    - it has ones on the diagonal
  - U is upper triangular

- The permutation matrix P can be chosen so that the multipliers used in forming L all have magnitude at most one.

- A is non-singular if and only if the diagonal entries in U are all non-zero.

- If A is non-singular, then we can solve

$$Ax = b$$

  in three steps:

  1. Solve $Pz = b$ for $z = P^Tb$ (permute the right hand side)
  2. Solve $Ly = z$ for y (forward solve lower triangular system)
  3. Solve $Ux = y$ for x (back solve upper triangular system)

- Computational complexity:

  - Computing the PLU decomposition takes $O(n^3)$ operations.
  - Computing a solution from a PLU decomposition takes $O(n^2)$ operations.

Condition Number

- Linear systems $Ax = b$ have unique solutions if A is non-singular.
- Solutions are sensitive to small perturbations if A is close to singular.
- We need a useful measure of closeness to singularity
- The *condition number* is a useful measure:

$$\begin{aligned}\kappa(A) &= \frac{\max_{x\neq 0}\frac{\|Ax\|}{\|x\|}}{\min_{x\neq 0}\frac{\|Ax\|}{\|x\|}}\\ &= \left(\max_{x\neq 0}\frac{\|Ax\|}{\|x\|}\right)\left(\max_{x\neq 0}\frac{\|A^{-1}x\|}{\|x\|}\right)\\ &= \|A\|\|A^{-1}\|\end{aligned}$$

  where $\|y\|$ is a *vector norm* (i.e. a measure of length) of y and

$$\|B\| = \max_{x\neq 0}\frac{\|Bx\|}{\|x\|}$$

  is the corresponding *matrix norm* of B.

- Some common vector norms:

$$\begin{aligned}\|x\|_2 &= \sqrt{\sum_{i=1}^n x_i^2} && \text{Euclidean norm}\\ \|x\|_1 &= \sum_{i=1}^n |x_i| && L_1 \text{ norm, Manhattan norm}\\ \|x\|_\infty &= \max_i |x_i| && L_\infty \text{ norm}\end{aligned}$$

Some Properties of Condition Numbers

- $\kappa(A) \geq 1$ for all A.
- $\kappa(A) = \infty$ if A is singular
- If A is diagonal, then
$$\kappa(A) = \frac{\max |a_{ii}|}{\min |a_{ii}|}$$
- Different norms produce different values; the values are usually qualitatively similar

Sensitivity of Linear Systems

Suppose x solves the original system and x^* solves a slightly perturbed system,

$$(A + \Delta A)x^* = b + \Delta b$$

and suppose that

$$\delta \kappa(A) \leq \frac{1}{2}$$
$$\frac{\|\Delta A\|}{\|A\|} \leq \delta$$
$$\frac{\|\Delta b\|}{\|b\|} \leq \delta$$

Then

$$\frac{\|x - x^*\|}{\|x\|} \leq 4\delta\kappa(A)$$

Stability of Gaussian Elimination with Partial Pivoting

Backward error analysis: The numerical solution $\hat{x}$ to the system

$$Ax = b$$

produced by Gaussian elimination with partial pivoting is the exact solution for a perturbed system

$$(A + \Delta A)\hat{x} = b$$

with

$$\frac{\|\Delta A\|_\infty}{\|A\|_\infty} \leq 8n^3 \rho \mathbf{u} + O(\mathbf{u}^2)$$

- The value of ρ is not *guaranteed* to be small, but is rarely larger than 10
- The algorithm would be considered numerically stable if ρ were guaranteed to be bounded.
- *Complete pivoting* is a bit more stable, but much more work.
- The algorithm is considered very good for practical purposes.

General Linear Systems in R

R provides

- `solve` for general systems, based on LAPACK's DGESV.
- DGESV uses the *PLU* decomposition.
- `forwardsolve`, `backsolve` for triangular systems.
- `kappa` computes an estimate of the condition number or the exact condition number based on the Euclidean norm.

Cholesky Factorization

Suppose A is symmetric and (strictly) positive definite, i.e.

$$x^T Ax > 0$$

for all $x \neq 0$. Examples:

- If X is the $n \times p$ design matrix for a linear model and X is of rank p, then $A = X^T X$ is strictly positive definite.

  If X is not of full rank then $A = X^T X$ is non-negative definite or positive semi-definite, i.e. $x^T Ax \geq 0$ for all x.

- If A is the covariance matrix of a random vector X then A is positive semidefinite:

  $$\begin{aligned} c^T Ac &= c^T E[(X-\mu)(X-\mu)^T]c \\ &= E[((X-\mu)^T c)^T (X-\mu)^T c] \\ &= \text{Var}((X-\mu)^T c) \geq 0 \end{aligned}$$

  The covariance matrix is strictly positive definite unless $P(c^T X = c^T \mu) = 1$ for some $c \neq 0$, i.e. unless there is a perfect linear relation between some of the components of X.

Theorem

If A is strictly positive definite, then there exists a unique lower triangular matrix L with positive diagonal entries such that

$$A = LL^T$$

This is called the *Cholesky factorization*.

Properties of the Cholesky Factorization Algorithm

- It uses the symmetry to produce an efficient algorithm.
- The algorithm needs to take square roots to find the diagonal entries.
- An alternative that avoids square roots factors A as

  $$A = LDL^T$$

  with D diagonal and L unit lower triangular.
- The algorithm is numerically stable, and is guaranteed not to attempt square roots of negative numbers if

  $$q_n \mathbf{u} \kappa_2(A) \leq 1$$

  where q_n is a small constant depending on the dimension n.
- The algorithm will fail if the matrix is not (numerically) strictly positive definite.
- Modifications using pivoting are available that can be used for nonnegative definite matrices.
- Another option is to factor $A_\lambda = A + \lambda I$ with $\lambda > 0$ chosen large enough to make A_λ numerically strictly positive definite. This is often used in optimization.

Some Applications of the Cholesky Factorization

- Solving the normal equations in least squares. This requires that the predictors be linearly independent
- Generating multivariate normal random vectors.
- Parameterizing strictly positive definite matrices: Any lower triangular matrix L with arbitrary values below the diagonal and positive diagonal entries determines and is uniquely determined by the positive definite matrix $A = LL^T$

Cholesky Factorization in R

- The function `chol` computes the Cholesky factorization.
- The returned value is the upper triangular matrix $R = L^T$.
- LAPACK is used.

QR Factorization

An $m \times n$ matrix A with $m \geq n$ can be written as

$$A = QR$$

where

- Q is $m \times n$ with orthonormal columns, i.e. $Q^T Q = I_n$
- R is upper triangular
- Several algorithms are available for computing the QR decomposition:
  - Modified Gram-Schmidt
  - Householder transformations (reflections)
  - Givens transformations (rotations)

  Each has advantages and disadvantages.
- LINPACK `dqrdc` and LAPACK `DGEQP3` use Householder transformations.
- The QR decomposition exists regardless of the rank of A.
- The rank of A is n if and only if the diagonal elements of R are all non-zero.

Householder Transformations

- A Householder transformation is a matrix of the form

$$P = I - 2vv^T/v^Tv$$

  where v is a nonzero vector.

- Px is the reflection of x in the hyperplane orthogonal to v.

- Given a vector $x \neq 0$, choosing $v = x + \alpha e_1$ with

$$\alpha = \pm\|x\|_2$$

  and e_1 the first unit vector (first column of the identity) produces

$$Px = \mp\|x\|_2 e_1$$

  This can be used to zero all but the first element of the first column of a matrix:

$$P\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}$$

  This is the first step in computing the *QR* factorization.

- The denominator v^Tv can be written as

$$v^Tv = x^Tx + 2\alpha x_1 + \alpha^2$$

- Choosing $\alpha = \text{sign}(x_1)\|x\|_2$ ensures that all terms are non-negative and avoids cancellation.

- With the right choice of sign Householder transformations are very stable.

Givens Rotations

- A Givens rotation is a matrix G that is equal to the identity except for elements $G_{ii}, G_{ij}, G_{ji}, G_{jj}$, which are

  $$\begin{bmatrix} G_{ii} & G_{ij} \\ G_{ji} & G_{jj} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

  with $c = \cos(\theta)$ and $s = \sin(\theta)$ for some θ.

- Premultiplication by G^T is a clockwise rotation by θ radians in the (i, j) coordinate plane.

- Given scalars a, b one can compute c, s so that

  $$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

  This allows G to zero one element while changing only one other element.

- A stable way to choose c, s:

```
if $b = 0$
  $c = 1$; $s = 0$
else
  if $|b| > |a|$
    $\tau = -a/b$; $s = 1/\sqrt{1+\tau^2}$; $c = s\tau$
  else
    $\tau = -b/a$; $c = 1/\sqrt{1+\tau^2}$; $s = c\tau$
  end
end
```

- A sequence of Givens rotations can be used to compute the QR factorization.
  - The zeroing can be done working down columns or across rows.
  - Working across rows is useful for incrementally adding more observations.

Applications

- The QR decomposition can be used for solving $n \times n$ systems of equations
$$Ax = b$$
since $Q^{-1} = Q^T$ and so
$$Ax = QRx = b$$
is equivalent to the upper triangular system
$$Rx = Q^T b$$
- The QR decomposition can also be used to solve the normal equations in linear regression: If X is the $n \times p$ design matrix then the normal equations are
$$X^T X b = X^T y$$
If $X = QR$ is the QR decomposition of X, then
$$\begin{aligned} X^T X &= R^T Q^T Q R = R^T R \\ X^T y &= R^T Q^T y \end{aligned}$$
If X is of full rank then R^T is invertible, and the normal equations are equivalent to the upper triangular system
$$Rb = Q^T y$$
This approach avoids computing $X^T X$.
- If X is of full rank then $R^T R$ is the Cholesky factorization of $X^T X$ (up to multiplications of rows of R by ± 1).

QR with Column Pivoting

Sometimes the columns of X are linearly dependent or nearly so.

By permuting columns we can produce a factorization
$$A = QRP$$
where

- P is a permutation matrix
- R is upper triangular and the diagonal elements of R have non-increasing magnitudes, i.e.
$$|r_{ii}| \geq |r_{jj}|$$
if $i \leq j$
- If some of the diagonal entries of R are zero, then R will be of the form
$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}$$
where R_{11} is upper triangular with non-zero diagonal elements non-increasing in magnitude.
- The rank of the matrix is the number of non-zero rows in R.
- The *numerical rank* of a matrix can be determined by
  - computing its QR factorization with column pivoting
  - specifying a tolerance level ε such that all diagonal entries $|r_{ii}| < \varepsilon$ are considered numerically zero.
  - Modifying the computed QR factorization to zero all rows corresponding to numerically zero r_{ii} values.

Some Regression Diagnostics

The projection matrix, or hat matrix, is

$$H = X(X^T X)^{-1} X^T = QR(R^T R)^{-1} R^T Q^T = QQ^T$$

The diagonal elements of the hat matrix are therefore

$$h_i = \sum_{j=1}^{p} q_{ij}^2$$

If $\hat{e}_i = y_i - \hat{y}_i$ is the residual, then

$$\begin{aligned}
s_{-i}^2 &= \frac{\text{SSE} - \hat{e}_i^2/(1-h_i)}{n-p-1} = \text{ estimate of variance without obs. } i \\
t_i &= \frac{\hat{e}_i}{s_{-i}\sqrt{1-h_i}} = \text{externally studentized residual} \\
D_i &= \frac{\hat{e}_i^2 h_i}{(1-h_i)^2 s^2 p} = \text{Cook's distance}
\end{aligned}$$

QR Decomposition and Least Squares in R

- The R function `qr` uses either LINPACK or LAPACK to compute QR factorizations.
- LINPACK is the default.
- The core linear model fitting function `lm.fit` uses QR factorization with column pivoting.

Singular Value Decomposition

An $m \times n$ matrix A with $m \geq n$ can be factored as

$$A = UDV^T$$

where

- U is $m \times n$ with orthonormal columns, i.e. $U^T U = I_n$.
- V is $n \times n$ orthogonal, i.e. $VV^T = V^T V = I_n$.
- $D = \text{diag}(d_1, \ldots, d_n)$ is $n \times n$ diagonal with $d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$.

This is the *singular value decomposition*, or *SVD* of A.

- The values $d_1, \ldots, d_n$ are the *singular values* of A.
- The columns of U are the *right singular vectors* of A.
- The columns of V are the *left singular vectors* of A.
- If the columns of A have been centered so the column sums of A are zero, then the columns of UD are the *principal components* of A.
- Excellent algorithms are available for computing the SVD.
- These algorithms are usually several times slower than the QR algorithms.

Some Properties of the SVD

- The Euclidean matrix norm of A is defined as
$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}$$
with $\|x\|_2 = \sqrt{x^T x}$ the Euclidean vector norm.
- If A has SVD $A = UDV^T$, then
$$\|A\|_2 = d_1$$
- If $k < \text{rank}(A)$ and
$$A_k = \sum_{i=1}^{k} d_i u_i v_i^T$$
then
$$\min_{B:\text{rank}(B) \leq k} \|A - B\|_2 = \|A - A_k\| = d_{k+1}$$
In particular,
  - $d_1 u_1 v_1^T$ is the best rank one approximation to A (in the Euclidean matrix norm).
  - A_k is the best rank k approximation to A.
  - If $m = n$ then $d_n = \min\{d_1, \ldots . d_n\}$ is the distance between A and the set of singular matrices.
- If A is square then the condition number based on the Euclidean norm is
$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{d_1}{d_n}$$
- For an $n \times p$ matrix with $n > p$ we also have
$$\kappa_2(A) = \frac{\max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}}{\min_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}} = \frac{d_1}{d_n}$$
  - This can be used to relate $\kappa_2(A^T A)$ to $\kappa_2(A)$.
  - This has implications for regression computations.
- The singular values are the non-negative square roots of the eigenvalues of $A^T A$ and the columns of V are the corresponding eigenvectors.

Moore-Penrose Generalized Inverse

Suppose A has rank $r \leq n$ and SVD $A = UDV^T$. Then

$$d_{r+1} = \cdots = d_n = 0$$

Let

$$D^+ = \text{diag}\left(\frac{1}{d_1}, \ldots, \frac{1}{d_r}, 0, \ldots, 0\right)$$

and

$$A^+ = VD^+U^T$$

Then A^+ satisfies

$$\begin{aligned} AA^+A &= A \\ A^+AA^+ &= A^+ \\ (AA^+)^T &= AA^+ \\ (A^+A)^T &= A^+A \end{aligned}$$

A^+ is the unique matrix with these properties and is called the Moore-Penrose *generalized inverse* or *pseudo-inverse*.

SVD and Least Squares

If X is an $n \times p$ design matrix of less than full rank, then there are infinitely many values of b that minimize

$$\|y - Xb\|_2^2$$

Among these solutions,

$$b = (X^T X)^+ X^T y$$

minimizes $\|b\|_2$.

This is related to *penalized regression* where one might choose b to minimize

$$\|y - Xb\|_2^2 + \lambda \|b\|_2^2$$

for some choice of $\lambda > 0$.

SVD and Principal Components Analysis

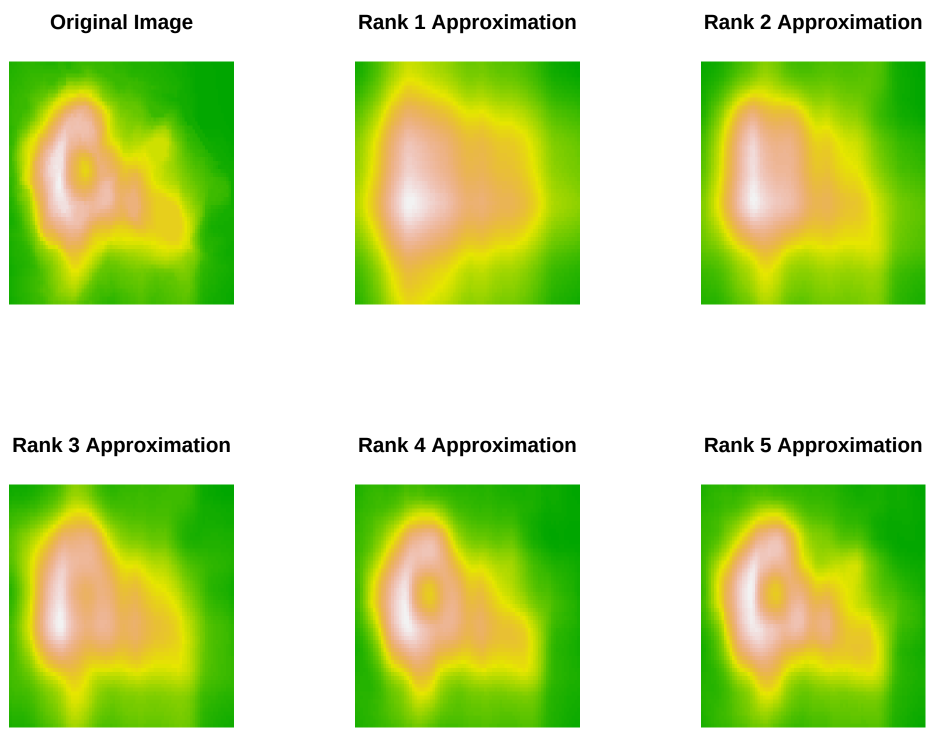
- Let X be an $n \times p$ matrix of n observations on p variables.
- Principal components analysis involves estimating the eigenvectors and eigenvalues of the covariance matrix.
- Let $\widetilde{X}$ be the data matrix with columns centered at zero by subtracting the column means.
- The sample covariance matrix is
$$S = \frac{1}{n-1}\widetilde{X}^T\widetilde{X}$$
- Let $\widetilde{X} = UDV^T$ be the SVD of the centered data matrix $\widetilde{X}$.
- Then
$$S = \frac{1}{n-1}VDU^TUDV^T = \frac{1}{n-1}VD^2V^T$$
- So
  - The diagonal elements of $\frac{1}{n-1}D^2$ are the eigenvalues of S.
  - The columns of V are the eigenvectors of S.
- Using the SVD of $\widetilde{X}$ is more numerically stable than
  - forming $\widetilde{X}^T\widetilde{X}$
  - computing the eigenvalues and eigenvectors.

SVD and Numerical Rank

- The rank of a matrix A is equal to the number of non-zero singular values.
- Exact zeros may not occur in the SVD due to rounding.
- Numerical rank determination can be based on the SVD. All $d_i \leq \delta$ can be set to zero for some choice of δ. Golub and van Loan recommend using
$$\delta = \mathbf{u}\|A\|_\infty$$
- If the entries of A are only accurate to d decimal digits, then Golub and van Loan recommend
$$\delta = 10^{-d}\|A\|_\infty$$
- If the numerical rank of A is $\hat{r}$ and $d_{\hat{r}} \gg \delta$ then $\hat{r}$ can be used with some confidence; otherwise caution is needed.

Other Applications

- The SVD is used in many areas of numerical analysis.
- It is also often useful as a theoretical tool.
- Some approaches to compressing $m \times n$ images are based on the SVD.
- A simple example using the `volcano` data:



```
s$d
 [1] 9644.2878216  488.6099163  341.1835791  298.7660207  141.8336254
 [6]   72.1244275   43.5569839   33.5231852   27.3837593   19.9762196
 ...
[61]    0.9545092
```

SVD in R

- R provides the function `svd` to compute the SVD.
- Implementation used to use LINPACK but now can use LINPACK or LAPACK, with LAPACK the default.
- You can ask for the singular values only—this is will be faster for larger problems.

Eigenvalues and Eigenvectors

Let A be an $n \times n$ matrix. λ is an *eigenvalue* of A if

$$Av = \lambda v$$

for some $v \neq 0$; v is an *eigenvector* or A.

- If A is a real $n \times n$ matrix then it has n eigenvalues.
  - Several eigenvalues may be identical
  - Some eigenvalues may be complex; if so, then they come in conjugate pairs.
  - The set of eigenvalues is called the *spectrum*
- If A is symmetric then the eigenvalues are real
- If A is symmetric then
  - A is strictly positive definite if and only if all eigenvalues are positive.
  - A is positive semi-definite if and only if all eigenvalues are nonnegative.
  - There exists an orthogonal matrix V such that
$$A = V\Lambda V^T$$
    with $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$; the columns of V are the corresponding normalized eigenvectors.
  - This is called the *spectral decomposition* of A.
- Some problems require only the largest eigenvalue or the largest few, sometimes the corresponding eigenvectors are also needed.
  - The stationary distribution of an irreducible finite state-space Markov chain is the unique eigenvector, normalized to sum to one, corresponding to the largest eigenvalue $\lambda = 1$.
  - The speed of convergence to the stationary distribution depends on the magnitude of the second largest eigenvalue.
- The R function `eigen` can be used to compute eigenvalues and, optionally, eigenvectors.

Determinants

- Theoretically the determinant can be computed as the product of
  - the diagonals of U in the PLU decomposition
  - the squares of the diagonals of L in the Cholesky factorization
  - the diagonals of R in the QR decomposition
  - the eigenvalues
- Numerically these are almost always bad ideas.
- It is almost always better to work out the sign and compute the sum of the logarithms of the magnitudes of the factors.
- The R functions `det` and `determinant` compute the determinant.
  - `determinant` is more complicated to use, but has a `logarithm` option.
- Likelihood and Bayesian analyses often involve a determinant;
  - usually the log likelihood and log determinant should be used.
  - usually the log determinant can be computed from a decomposition needed elsewhere in the log likelihood calculation, e.g. a Cholesky factorization

Non-Negative Matrix Factorization

A number of problems lead to the desire to approximate a non-negative matrix X by a product

$$X \approx WH$$

where W, H are non-negative matricies of low rank, i.e. with few columns.

There are a number of algorithms available, most of the form

$$\min_{W,H}[D(X,WH) + R(W,H)]$$

where D is a loss function and R is a possible penalty for encouraging desirable characteristics of W, H, such as smoothness or sparseness.

The R package `NMF` provides one approach, and a vignette in the package provides some background and references.

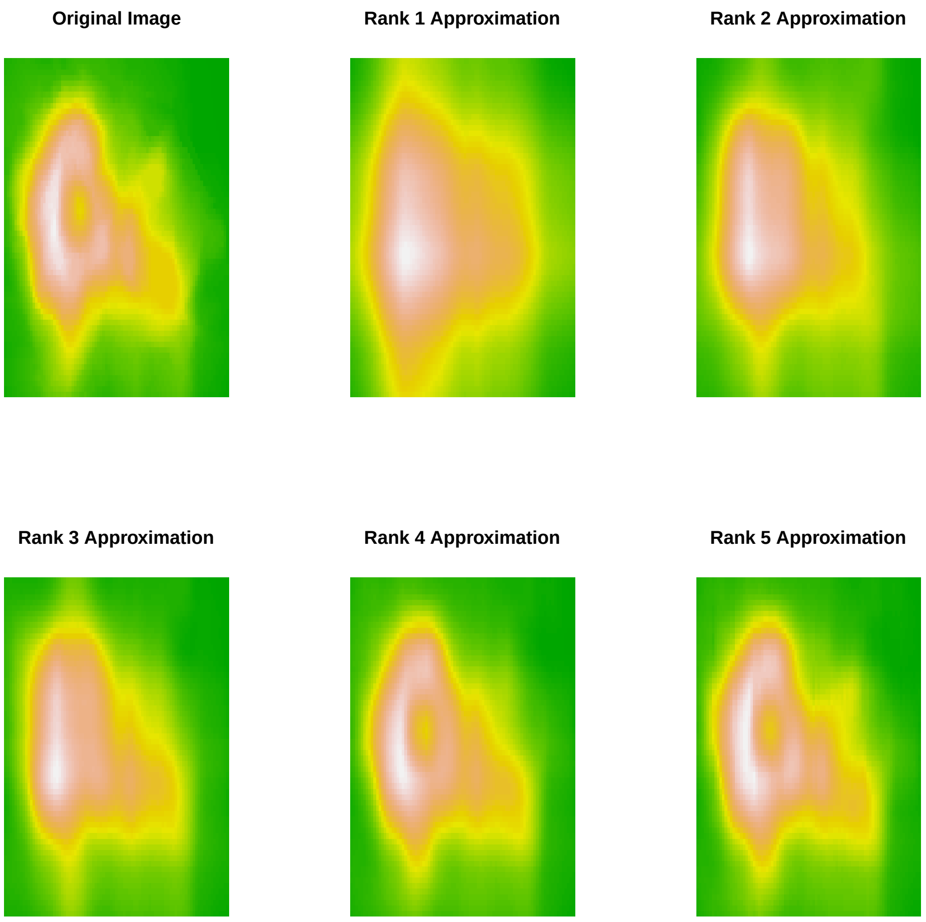
As an example, using default settings in the NMF package the `volcano` image can be approximated with factorizations of rank $1,\dots,5$ by

```
library(NMF)
nmf1 = nmf(volcano, 1); V1 <- nmf1@fit@W %*% nmf1@fit@H
nmf2 = nmf(volcano, 2); V2 <- nmf2@fit@W %*% nmf2@fit@H
nmf3 = nmf(volcano, 3); V3 <- nmf3@fit@W %*% nmf3@fit@H
nmf4 = nmf(volcano, 4); V4 <- nmf4@fit@W %*% nmf4@fit@H
nmf5 = nmf(volcano, 5); V5 <- nmf5@fit@W %*% nmf5@fit@H
```

The relative error for the final image is

```
> max(abs(volcano - V5)) / max(abs(volcano))
[1] 0.03273659
```

The images:



Another application is *recommender systems*.

- For example, X might be ratings of movies (columns) by viewers (rows).
- The set of actual values would be very sparse as each viewer will typically rate only a small subset of all movies.
- W would be a user preference matrix, H a corresponding movie feature matrix.
- The product WH would provide predicted ratings for movies the users have not yet seen.

Other Factorizations

Many other factorizations of matrices are available and being developed. Some examples are

- Robust variants of the SVD
- Sparse variants, e.g. Dan Yang, Zongming Ma, and Andreas Buja (2014), "A Sparse Singular Value Decomposition Method for High-Dimensional Data," Journal of Computational and Graphical Statistics 23(4), 923–942.
- Constrained factorizations, e.g. C. Ding, T. Li, and M. I. Jordan (2010), "Convex and Semi-Nonnegative Matrix Factorizations," IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(1), 45–55.

Exploiting Special Structure

Specialized algorithms can sometimes be used for matrices with special structure.

Toeplitz Systems

- Stationary time series have covariance matrices that look like

$$\begin{bmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \sigma_3 & \dots \\ \sigma_1 & \sigma_0 & \sigma_1 & \sigma_2 & \dots \\ \sigma_2 & \sigma_1 & \sigma_0 & \sigma_1 & \dots \\ \sigma_3 & \sigma_2 & \sigma_1 & \sigma_0 & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

- This is a *Toeplitz* matrix.
- This matrix is also symmetric — this is not required for a Toeplitz matrix.
- Special algorithms requiring $O(n^2)$ operations are available for Toeplitz systems.
- General Cholesky factorization requires $O(n^3)$ operations.
- The R function `toeplitz` creates Toeplitz matrices.

Circulant Systems

- Some problems give rise to matrices that look like

$$C_n = \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ a_n & a_1 & a_2 & \dots & a_{n-1} \\ a_{n-1} & a_n & a_1 & \dots & a_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_2 & a_3 & a_4 & \dots & a_1 \end{bmatrix}$$

- This is a *circulant* matrix, a subclass of Toeplitz matrices.
- Circulant matrices satisfy

$$C_n = F_n^* \operatorname{diag}(\sqrt{n} F_n a) F_n$$

  where F_n is the *Fourier matrix* with

$$F_n(j,k) = \frac{1}{\sqrt{n}} e^{-(j-1)(k-1)2\pi\sqrt{-1}/n}$$

  and F_n^* is the *conjugate transpose*, *Hermitian transpose*, or *adjoint matrix* of F_n.

- The eigen values are the elements of $\sqrt{n} F_n a$.
- Products $F_n x$ and $F_n^* x$ can be computed with the *fast Fourier transform (FFT)*.
- In R

$$\begin{aligned} \sqrt{n} F_n x &= \texttt{fft(x)} \\ \sqrt{n} F_n^* x &= \texttt{fft(x, inverse = TRUE)} \end{aligned}$$

- These computations are generally $O(n \log n)$ in complexity.
- Circulant systems can be used to approximate other systems.
- Multi-dimensional analogs exist as well.
- A simple example is available on line.

Sparse Systems

- Many problems lead to large systems in which only a small fraction of coefficients are non-zero.
- Some methods are available for general sparse systems.
- Specialized methods are available for structured sparse systems such as
  - tri-diagonal systems
  - block diagonal systems
  - banded systems
- Careful choice of row and column permutations can often turn general sparse systems into banded ones.

Sparse and Structured Systems in R

- Sparse matrix support in R is improving.
- Some packages, like `nlme`, provide utilities they need.
- One basic package available on CRAN is `sparseM`
- A more extensive package is `Matrix`
- `Matrix` is the engine for mixed effects/multi-level model fitting in `lme4`

Update Formulas

- Update formulas are available for most decompositions that allow for efficient adding or dropping of rows or columns.
- These can be useful for example in cross-validation and variable selection computations.
- They can also be useful for fitting linear models to very large data sets; the package `biglm` uses this approach.
- I am not aware of any convenient implementations in R at this point but they may exist.

Iterative Methods

- Iterative methods can be useful in large, sparse problems.
- Iterative methods for sparse problems can also often be parallelized effectively.
- Iterative methods are also useful when
  - *Ax* can be computed efficiently for any given *x*
  - It is expensive or impossible to compute *A* explicitly

Gauss-Seidel Iteration

Choose an initial solution $x^{(0)}$ to

$$Ax = b$$

and then update from $x^{(k)}$ to $x^{(k+1)}$ by

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

for $i = 1, \ldots, n$.

This is similar in spirit to Gibbs sampling.

This can be written in matrix form as

$$x^{(k+1)} = (D+L)^{-1}(-Ux^{(k)}+b)$$

with

$$L = \begin{bmatrix} 0 & 0 & \dots & \dots & 0 \\ a_{21} & 0 & \dots & & \vdots \\ a_{31} & a_{32} & \ddots & & 0 \\ \vdots & & & 0 & 0 \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & 0 \end{bmatrix}$$

$$D = \operatorname{diag}(a_{11}, \dots, a_{nn})$$

$$U = \begin{bmatrix} 0 & a_{12} & \dots & \dots & a_{1n} \\ 0 & 0 & \dots & & \vdots \\ 0 & 0 & \ddots & & a_{n-2,n} \\ \vdots & & & & a_{n-1,n} \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Splitting Methods

The Gauss-Seidel method is a member of a class of *splitting methods* where

$$Mx^{(k+1)} = Nx^{(k)} + b$$

with $A = M - N$.

For the Gauss-Seidel method

$$\begin{aligned} M &= D + L \\ N &= -U. \end{aligned}$$

Other members include Jacobi iterations with

$$\begin{aligned} M_J &= D \\ N_J &= -(L + U) \end{aligned}$$

Splitting methods are practical if solving linear systems with matrix M is easy.

Convergence

A splitting method for a non-singular matrix A will converge to the unique solution of $Ax = b$ if

$$\rho(M^{-1}N) < 1$$

where

$$\rho(G) = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } G\}$$

is the *spectral radius* of G.

This is true, for example, for the Gauss-Seidel method if A is strictly positive definite.

Convergence can be very slow if $\rho(M^{-1}N)$ is close to one.

Successive Over-Relaxation

A variation is to define

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) + (1-\omega)x_i^{(k)}$$

or, in matrix form,

$$M_\omega x^{(k+1)} = N_\omega x^{(k)} + \omega b$$

with

$$\begin{aligned} M_\omega &= D + \omega L \\ N_\omega &= (1-\omega)D - \omega U \end{aligned}$$

for some ω, usually with $0 < \omega < 1$.

- This is called *successive over-relaxation (SOR)*, from its first application in a structural engineering problem.
- For some choices of ω we can have

  $$\rho(M_\omega^{-1}N_\omega) \ll \rho(M^{-1}N)$$

  and thus faster convergence.
- For some special but important problems the value of ω that minimizes $\rho(M_\omega^{-1}N_\omega)$ is known or can be computed.

Conjugate Gradient Method

If A is symmetric and strictly positive definite then the unique solution to $Ax = b$ is the unique minimizer of the quadratic function

$$f(x) = \frac{1}{2}x^T Ax - x^T b$$

Any nonlinear or quadratic optimization method can be used to find the minimum; the most common one used in this context is the conjugate gradient method.

Choose an initial x_0, set $d_0 = -g_0 = b - Ax_0$, and then, while $g_k \neq 0$, for $k = 0, 1, \ldots$ compute

$$\begin{aligned}
\alpha_k &= -\frac{g_k^T d_k}{d_k^T A d_k} \\
x_{k+1} &= x_k + \alpha_k d_k \\
g_{k+1} &= Ax_{k+1} - b \\
\beta_{k+1} &= \frac{g_{k+1}^T A d_k}{d_k^T A d_k} \\
d_{k+1} &= -g_{k+1} + \beta_{k+1} d_k
\end{aligned}$$

Some properties:

- An alternate form of g_{k+1} is

  $$g_{k+1} = g_k + \alpha_k A d_k$$

  This means only one matrix-vector multiplication is needed per iteration.
- The vector g_k is the gradient of f at x_k.
- The initial direction $d_0 = -g_0$ is the *direction of steepest descent* from x_0
- The directions $d_0, d_1, \ldots$ are A-*conjugate*, i.e. $d_i^T A d_j = 0$ for $i \neq j$.
- The directions $d_0, d_1, \ldots, d_{n-1}$ are linearly independent.

Convergence

- With exact arithmetic,
$$Ax_n = b$$
That is, the conjugate gradient algorithm terminates with the exact solution in n steps.
- Numerically this does not happen.
- Numerically, the directions will not be exactly A-conjugate.
- A convergence tolerance is used for termination; this can be based on the relative change in the solution
$$\frac{\|x_{k+1} - x_k\|}{\|x_k\|}$$
or the residual or gradient
$$g_k = Ax_k - b$$
or some combination; an iteration count limit is also a good idea.
- If the algorithm does not terminate within n steps it is a good idea to restart it with a steepest descent step from the current x_k.
- In many sparse and structured problems the algorithm will terminate in far fewer than n steps for reasonable tolerances.
- Convergence is faster if the condition number of A is closer to one. The error can be bounded as
$$\|x - x_k\|_A \le 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^k$$
with $\|x\|_A = \sqrt{x^T A x}$.
- *Preconditioning* strategies can improve convergence; these transform the original problem to one with $\tilde{A} = C^{-1}AC^{-1}$ for some symmetric strictly positive definite C, and then use the conjugate gradient method for $\tilde{A}$
- Simple choices of C are most useful; sometimes a diagonal matrix will do.
- Good preconditioners can sometimes be designed for specific problems.

A Simple Implementation

```
cg <- function(A, b, x, done) {
    dot <- function(x, y) crossprod(x, y)[1]

    n <- 0
    g <- A(x) - b
    d <- -g

    repeat {
        h <- A(d)
        u <- dot(d, h)
        a <- -dot(g, d) / u

        n <- n + 1
        x.old <- x
        x <- x + a * d
        g <- g + a * h

        b <- dot(h, g) / u
        d <- -g + b * d
        if (done(g, x, x.old, n))
            return(list(x = as.vector(x),
                        g = as.vector(g),
                        n = n))
    }
}
```

- The linear transformation and the termination condition are specified as functions.
- The termination condition can use a combination of the gradient, current solution, previous solution, or iteration count.
- A simple example:

```
> X <- crossprod(matrix(rnorm(25), 5))
> y <- rnorm(5)
> cg(function(x) X %*% x, y, rep(0, 5), function(g, x, x.old, n) n >= 5)
$x
[1] 11.461061 -7.774344  1.067511 87.276967 -8.151556

$g
[1] -9.219292e-13  2.566836e-13 -1.104117e-12  1.690870e-13  1.150191e-13

$n
[1] 5

> solve(X, y)
[1] 11.461061 -7.774344  1.067511 87.276967 -8.151556
```

Linear Algebra Software

Some Standard Packages

Open source packages developed at national laboratories:

- LINPACK for linear equations and least squares
- EISPACK for eigenvalue problems
- LAPACK newer package for linear equations and eigenvalues

Designed for high performance. Available from Netlib at

`http://www.netlib.org/`

Commercial packages:

- IMSL used more in US
- NAG used more in UK
- ...

BLAS: Basic Linear Algebra Subroutines

Modern BLAS has three levels:

Level 1: Vector and vector/vector operations such as

- dot product $x^T y$
- scalar multiply and add (axpy): $\alpha x + y$
- Givens rotations

Level 2: Matrix/vector operations, such as Ax

Level 3: Matrix/matrix operations, such as AB

- LINPACK uses only Level 1; LAPACK uses all three levels.
- BLAS defines the interface.
- Standard reference implementations are available from Netlib.
- Highly optimized versions are available from hardware vendors and research organizations.

Cholesky Factorization in LAPACK

The core of the DPOTRF routine:

```
*
*        Compute the Cholesky factorization A = L*L'.
*
         DO 20 J = 1, N
*
*           Compute L(J,J) and test for non-positive-definiteness.
*
            AJJ = A( J, J ) - DDOT( J-1, A( J, 1 ), LDA, A( J, 1 ),
     $            LDA )
            IF( AJJ.LE.ZERO ) THEN
               A( J, J ) = AJJ
               GO TO 30
            END IF
            AJJ = SQRT( AJJ )
            A( J, J ) = AJJ
*
*           Compute elements J+1:N of column J.
*
            IF( J.LT.N ) THEN
               CALL DGEMV( 'No transpose', N-J, J-1, -ONE, A( J+1, 1 ),
     $                     LDA, A( J, 1 ), LDA, ONE, A( J+1, J ), 1 )
               CALL DSCAL( N-J, ONE / AJJ, A( J+1, J ), 1 )
            END IF
   20    CONTINUE
```

- DDOT and DSCAL are Level 1 BLAS routines
- DGEMV is a Level 2 BLAS routine

ATLAS: Automatically Tuned Linear Algebra Software

Available at

`http://math-atlas.sourceforge.net/`

- Analyzes machine for properties such as cache characteristics.
- Runs extensive tests to determine performance trade-offs.
- Creates Fortran and C versions of BLAS and some LAPACK routines tailored to the particular machine.
- Provides some routines that take advantage of multiple processors using *worker threads*.

OpenBLAS

- Another high-performance BLAS library was developed and maintained by Kazushige Goto.
- This is now being developed and maintained as the OpenBLAS project, available from

  `http://xianyi.github.com/OpenBLAS/`

- Also provides versions that take advantage of multiple processors.

Vendor Libraries

- Intel provides the Math Kernel Libraries (MKL)
- AMD has a similar library.

Using a High-Performance BLAS with R

- R comes with a basic default BLAS.
- R can be built to use a specified BLAS.
- Once built one can change the BLAS R uses by replacing the shared library R uses.
- Some simple computations using the default and MKL vendor BLAS for the data

  ```
  N <- 1000
  X <- matrix(rnorm(N^2), N)
  XX <- crossprod(X)
  ```

  Results:

Timing Expression	Default/ Reference	MKL SEQ	MKL THR
`system.time(for (i in 1:5) crossprod(X))`	2.107	0.405	0.145
`system.time(for (i in 1:5) X %*% X)`	3.401	0.742	0.237
`system.time(svd(X))`	3.273	0.990	0.542
`system.time(for (i in 1:5) qr(X))`	2.290	1.094	1.107
`system.time(for (i in 1:5) qr(X, LAPACK=TRUE))`	2.629	0.834	0.689
`system.time(for (i in 1:20) chol(XX))`	2.824	0.556	0.186

- These results are based on the non-threaded and threaded Intel Math Kernel Library (MKL) using the development version of R.
- Versions of the current R using MKL for BLAS are available as

  ```
  /group/statsoft/R-patched/build-MKL-seq/bin/R
  /group/statsoft/R-patched/build-MKL-thr/bin/R
  ```

- Currently the standard version of R on our Linux systems seems to be using OpenBLAS with multi-threading disabled.

Final Notes

- Most reasonable approaches will be accurate for reasonable problems.
- Choosing good scaling and centering can make problems more reasonable (both numerically and statistically)
- Most methods are efficient enough for our purposes.
- In some problems worrying about efficiency is important if reasonable problem sizes are to be handled.
- Making sure you are using the right approach to the right problem is much more important than efficiency.
- Some quotes:
  - D. E. Knuth, restating a comment by C. A. R. Hoare:

    > We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

  - W. A. Wulf:

    > More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason — including blind stupidity.

Optimization

Preliminaries

Many statistical problems involve minimizing (or maximizing) a function $f : \mathscr{X} \to \mathbb{R}$, i.e. finding

$$x^* = \underset{x \in \mathscr{X}}{\operatorname{argmin}} f(x)$$

- Maximizing $f(x)$ is equivalent to mimimizing $-f(x)$.
- The domain $\mathscr{X}$ can be
  - a finite set — combinatorial optimization
  - a continuous, usually connected, subset of $\mathbb{R}^n$
- The function can be
  - continuous
  - twice continuously differentiable
  - unrestricted
- The function can
  - have a single local minimum that is also a global minimum
  - have one or more local minima but no global minimum
  - have many local minima and a global minimum

- Algorithms can be designed to
  - find a local minimum from some specified starting point
  - find the global minimum
- The objective can be
  - to find a global minimum
  - to find a local minimum
  - to improve on an initial guess
- Very good software is available for many problems
  - general purpose optimizers will often do well
  - occasionally it is useful to write your own, usually to exploit special structure
  - understanding general strategies is useful even when using canned software
- Optimization problems often require some tuning
  - proper scaling is often critical

One-Dimensional Optimization

Given an initial point $x^{(k)}$ and a direction d we can define

$$x(t) = x^{(k)} + td$$

and set

$$x^{(k+1)} = x^{(k)} + t^* d$$

where

$$t^* = \text{argmin}_t f(x(t))$$

with $t \in \mathbb{R}$ restricted to satisfy $x(t) \in \mathscr{X}$.

- This is often called a *line search*.
- Many one-dimensional optimization methods are available.
- Many are based on finding a root of the derivative.
- Newton's method approximates $f(x(t))$ by a quadratic.
- Once the function f has been minimized in the direction d a new direction can be tried.
- It is usually not necessary to find the minimizer t^* exactly — a few steps in an iterative algorithm are often sufficient.
- R provides
  - `optimize` for minimizing a function over an interval
  - `uniroot` for finding the root of a function in an interval

Choosing Search Directions

Several methods are available for choosing search directions:

- *Cyclic descent, or coordinate descent*: if $x = (x_1, \dots, x_n)$ then minimize first with respect to x_1, then x_2, and so on through x_n, and repeat until convergence
  - This method is also called *non-liner Gauss-Seidel iteration* or *backfitting*. It is similar in spirit to Gibbs sampling.
  - This may take more iterations than other methods, but often the individual steps are very fast.
  - A recent paper and the associated `sparsenet` package takes advantage of this for fitting sparse linear models.
- *Steepest descent*: Take $d = -\nabla f(x^{(k)})$.
  - Steepest descent can be slow to converge due to zig-zagging.
  - It can work well for problems with nearly circular contours.
  - Preconditioning to improve the shape of the contours can help.
- *Conjugate gradient*: Start with the steepest descent direction and then choose directions conjugate to a strictly positive definite matrix A, often a Hessian matrix.
  - This can reduce the zig-zagging.
  - It needs to be restarted at least every n steps.
  - Unless the objective function f is quadratic a new matrix A is typically computed at each restart.

Most of these methods will be *linearly convergent*: under suitable regularity conditions and if $x^{(0)}$ is close enough to a local minimizer x^* then

$$\lim_{k\to\infty} \frac{\|x^{(k+1)} - x^*\|}{\|x^{(k)} - x^*\|} = a$$

with $0 < a < 1$; a is the *rate of convergence*. A method for which $a = 0$ is said to converge *super-linearly*.

Newton's Method

Newton's method approximates f by the quadratic

$$f^{(k)}(x) = f(x^{(k)}) + \nabla f(x^{(k)})(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T \nabla^2 f(x^{(k)})(x - x^{(k)})$$

and computes $x^{(k+1)}$ to minimize the quadratic approximation $f^{(k)}$:

$$x^{(k+1)} = x^{(k)} - \left(\nabla^2 f(x^{(k)})\right)^{-1} \nabla f(x^{(k)})$$

- Convergence can be very fast: if
  - f is twice continuously differentiable near a local minimizer x^*
  - $\nabla^2 f(x^*)$ is strictly positive definite
  - $x^{(0)}$ is sufficiently close to x^*

  then
  $$\lim_{k\to\infty} \frac{\|x^{(k+1)} - x^*\|}{\|x^{(k)} - x^*\|^2} = a$$
  with $0 < a < \infty$. This is called *quadratic convergence*.

- If these conditions fail then Newton's method may not converge, even for a convex unimodal function.
- If the new value $x^{(k+1)}$ does not improve f then one can use a line search in the direction of the Newton step.
- Newton's method requires first and second derivatives:
  - Numerical derivatives can be used.
  - Computing the derivatives can be very expensive if n is large.
- Many implementations
  - modify the second derivative matrix if it is not strictly positive definite.
  - switch to an alternate method, such as steepest descent, if the Newton direction does not produce sufficient improvement
- Computation of the Newton step is usually done using a Cholesky factorization of the Hessian.
- A modified factorization is often used if the Hessian is not numerically strictly positive definite.
- If $\widetilde{\theta}$ is a consistent estimate of θ and $\widehat{\theta}$ is computed as a single Newton step from $\widetilde{\theta}$, then, under mild regularity conditions, $\widehat{\theta}$ will have the same asymptotic normal distribution as the MLE.

Quasi-Newton Methods

Quasi-Newton methods compute the next step as

$$x^{(k+1)} = x^{(k)} - B_k^{-1} \nabla f(x^{(k)})$$

where B_k is an approximation to the Hessian matrix $\nabla^2 f(x^{(k)})$.

- B_{k+1} is usually chosen so that

$$\nabla f(x^{(k+1)}) = \nabla f(x^{(k)}) + B_{k+1}(x^{(k+1)} - x^{(k)})$$

- For $n = 1$ this leads to the *secant method*; for $n > 1$ this is under-determined.
- For $n > 1$ various strategies for low rank updating of B are available; often these are based on successive gradient values.
- The inverse can be updated using the *Sherman-Morrison-Woodbury formula*: If A is invertible then

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u}$$

  as long as the denominator is not zero.
- Commonly used methods include
  - Davidon-Fletcher-Powell (DFP)
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS)
- Methods generally ensure that
  - B_k is symmetric
  - B_k is strictly positive definite
- Convergence of Quasi-Newton methods is generally super-linear but not quadratic.

Fisher Scoring

Maximum likelihood estimates can be computed by minimizing the negative log likelihood

$$f(\theta) = -\log L(\theta)$$

- The Hessian matrix $\nabla^2 f(\theta^{(k)})$ is the *observed information matrix* at $\theta^{(k)}$.
- Sometimes the *expected information matrix* $I(\theta^{(k)})$ is easy to compute.
- The Fisher scoring method uses a Newton step with the observed information matrix replaced by the expected information matrix:

$$\theta^{(k+1)} = \theta^{(k)} + I(\theta^{(k)})^{-1} \nabla \log L(\theta^{(k)})$$

Example: Logistic Regression

Suppose $y_i \sim \text{Binomial}(m_i, \pi_i)$ with

$$\pi_i = \frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)}$$

The negative log likelihood, up to additive constants, is

$$f(\beta) = -\sum (y_i x_i^T \beta - m_i \log(1 + \exp(x_i^T \beta))$$

with gradient

$$\begin{aligned} \nabla f(\beta) &= -\sum \left(y_i x_i - m_i \frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)} x_i \right) \\ &= -\sum (y_i x_i - m_i \pi_i x_i) \\ &= -\sum (y_i - m_i \pi_i) x_i \end{aligned}$$

and Hessian

$$\nabla^2 f(\beta) = \sum m_i \pi_i (1 - \pi_i) x_i x_i^T$$

The Hessian is non-stochastic, so Fisher scoring and Newton's method are identical and produce the update

$$\beta^{(k+1)} = \beta^{(k)} + \left(\sum m_i \pi_i^{(k)} (1 - \pi_i^{(k)}) x_i x_i^T \right)^{-1} \left(\sum (y_i - m_i \pi_i^{(k)}) x_i \right)$$

If X is the matrix with rows x_i^T, $W^{(k)}$ is the diagonal matrix with diagonal elements

$$W_{ii}^{(k)} = m_i \pi_i^{(k)}(1-\pi_i^{(k)})$$

and $V^{(k)}$ is the vector with elements

$$V_i^{(k)} = \frac{y_i - m_i \pi_i^{(k)}}{W_{ii}}$$

then this update can be written as

$$\beta^{(k+1)} = \beta^{(k)} + (X^T W^{(k)} X)^{-1} X^T W^{(k)} V^{(k)}$$

Thus the change in β is the result of fitting the linear model $V^{(k)} \sim X$ by weighted least squares with weights $W^{(k)}$.

- This type of algorithm is called an *iteratively reweighted least squares* (IRWLS) algorithm.
- Similar results hold for all generalized linear models.
- In these models Fisher scoring and Newton's method are identical when the *canonical link* function is chosen which makes the natural parameter linear in β.

Gauss-Newton Method

Suppose
$$Y_i \sim N(\eta(x_i,\theta),\sigma^2).$$
For example, $\eta(x,\theta)$ might be of the form
$$\eta(x,\theta) = \theta_0 + \theta_1 e^{-\theta_2 x}.$$

Then the MLE minimizes
$$f(\theta) = \sum_{i=1}^n (y_i - \eta(x_i,\theta))^2.$$
We can approximate η near a current guess $\theta^{(k)}$ by the Taylor series expansion
$$\eta_k(x_i,\theta) \approx \eta(x_i,\theta^{(k)}) + \nabla_\theta \eta(x_i,\theta^{(k)})(\theta - \theta^{(k)})$$
This leads to the approximate objective function
$$\begin{aligned} f_k(\theta) &\approx \sum_{i=1}^n (y_i - \eta_k(x_i,\theta))^2 \\ &= \sum_{i=1}^n (y_i - \eta(x_i,\theta^{(k)}) - \nabla_\theta \eta(x_i,\theta^{(k)})(\theta - \theta^{(k)}))^2 \end{aligned}$$
This is the sum of squared deviations for a linear regression model, so is minimized by
$$\theta^{(k+1)} = \theta^{(k)} + (J_k^T J_k)^{-1} J_k^T (y - \eta(x,\theta^{(k)}))$$
where $J_k = \nabla_\theta \eta(x,\theta_{(k)})$ is the Jacobian matrix of the mean function.

- The Gauss-Newton algorithm works well for problems with small residuals, where it is close to Newton's method.
- Like Newton's method it can suffer from taking too large a step.
- Backtracking or line search can be used to address this.
- An alternative is the Levenberg-Marquardt algorithm that uses
$$\theta^{(k+1)} = \theta^{(k)} + (J_k^T J_k + \lambda I)^{-1} J_k^T (y - \eta(x,\theta^{(k)}))$$
for some *damping parameter* λ. Various strategies for choosing λ are available.

The R function `nls` uses these approaches.

Termination and Scaling

Optimization algorithms generally use three criteria for terminating the search:

- relative change in x
- relative or absolute change in the function values and/or derivatives
- number of iterations

How well these work often depends on problem scaling

- Implementations often allow specification of scaling factors for parameters.
- Some also allow scaling of the objective function.

Dennis and Schnabel (1983) recommend

- a relative gradient criterion

$$\max_{1 \leq i \leq n} \left| \frac{\nabla f(x)_i \max\{|x_i|, \text{typ}x_i\}}{\max\{|f(x)|, \text{typ}f\}} \right| \leq \varepsilon_{\text{grad}}$$

  where typx_i and typ$f$ are typical magnitudes of x_i and f.

- a relative change in x criterion

$$\max_{1 \leq i \leq n} \frac{|\Delta x_i|}{\max\{|x_i|, \text{typ}x_i\}} \leq \varepsilon_{\text{step}}$$

Practical use of optimization algorithms often involves

- trying different starting strategies
- adjusting scaling after preliminary runs
- other reparameterizations to improve conditioning of the problem

Nelder-Mead Simplex Method

This is *not* related to the simplex method for linear programming!

- The method uses only function values and is fairly robust but can we quite slow and need repeated restarts. It can work reasonably even if the objective function is not differentiable.
- The method uses function values at a set of $n+1$ affinely independent points in n dimensions, which form a simplex.
- Affine independence means that the points $x_1, \dots, x_{n+1}$ are such that $x_1 - x_{n+1}, \dots, x_n - x_{n+1}$ are linearly independent.
- The method goes through a series of reflection, expansion, contraction , and reduction steps to transform the simplex until the function values do not change much or the simplex becomes very small.

One version of the algorithm, adapted from Wikipedia:

1. Order the vertices according to their function values,
$$f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})$$
2. Calculate $x_0 = \frac{1}{n}\sum_{i=1}^{n} x_i$, the center of the face opposite the worst point x_{n+1}.
3. *Reflection*: compute the reflected point
$$x_r = x_0 + \alpha(x_0 - x_{n+1})$$
for some $\alpha \geq 1$, e.g. $\alpha = 1$. If $f(x_1) \leq f(x_r) < f(x_n)$, i.e. x_r is better than the second worst point x_n but not better than the best point x_1, then replace the worst point x_{n+1} by x_r and go to Step 1.
4. *Expansion:* If $f(x_r) < f(x_1)$, i.e. x_r is the best point so far, then compute the expanded point
$$x_e = x_0 + \gamma(x_0 - x_{n+1})$$
for some $\gamma > \alpha$, e.g. $\gamma = 2$. If $f(x_e) < f(x_r)$ then replace x_{n+1} by x_e and go to Step 1. Otherwise replace x_{n+1} with x_r and go to Step 1.
5. *Contraction*: If we reach this step then we know that $f(x_r) \geq f(x_n)$. Compute the contracted point
$$x_c = x_0 + \rho(x_0 - x_{n+1})$$
for some $\rho < 0$, e.g. $\rho = -1/2$. If $f(x_c) < f(x_{n+1})$ replace x_{n+1} by x_c and go to Step 1. Otherwise go to Step 6.
6. *Reduction*: For $i = 2, \ldots, n+1$ set
$$x_i = x_1 + \sigma(x_i - x_1)$$
for some $\sigma < 1$, e.g. $\sigma = 1/2$.

The Wikipedia page shows some examples as animations.

Simulated Annealing

- Simulated annealing is motivated by an analogy to slowly cooling metals in order to reach the minimum energy state.
- It is a stochastic global optimization method.
- It can be very slow but can be effective for difficult problems with many local minima.
- For any value $T > 0$ the function

  $$f_T(x) = e^{f(x)/T}$$

  has minima at the same locations as $f(x)$.
- Increasing the *temperature parameter* T flattens f_T; decreasing T sharpens the local minima.
- Suppose we have a current estimate of the minimizer, $x^{(k)}$ and a mechanism for randomly generating a *proposal* y for the next estimate.
- A step in the simulated annealing algorithm given a current estimate $x^{(k)}$:
  - Generate a proposal y for the next estimate.
  - If $f(y) \leq f(x^{(k)})$ then *accept* y and set $x_{(k+1)} = y$.
  - If $f(y) > f(x^{(k)})$ then with probability $f_T(x^{(k)})/f_T(y)$ *accept* y and set $x_{(k+1)} = y$.
  - Otherwise, *reject* y and set $x_{(k+1)} = x_{(k)}$.
  - Adjust the temperature according to a *cooling schedule* and repeat.
- Occasionally accepting steps that increase the objective function allows escape from local minima.

- A common way to generate y is as a multivariate normal vector centered at $x_{(k)}$.
- A common choice of cooling schedule is of the form $T = 1/\log(k)$.
- The standard deviations of the normal proposals may be tied to the current temperature setting.
- Under certain conditions this approach can be shown to converge to a global minimizer with probability one.
- Using other forms of proposals this approach can also be used for discrete optimization problems.

EM and MCEM Algorithms

Suppose we have a problem where *complete data* X,Z has likelihood

$$L^c(\theta|x,z) = f(x,z|\theta)$$

but we only observe *incomplete data* X with likelihood

$$L(\theta|x) = g(x|\theta) = \int f(x,z|\theta)dz$$

- Often the complete data likelihood is much simpler than the incomplete data one.
- The conditional density of the unobserved data given the observed data is
$$k(z|x,\theta) = \frac{f(x,z|\theta)}{g(x|\theta)}$$
- Define, taking expectations with respect to $k(z|x,\phi)$,
$$\begin{aligned} Q(\theta|\phi,x) &= E_\phi[\log f(x,z|\theta)|x] \\ H(\theta|\phi,x) &= E_\phi[\log k(z|x,\theta)|x] \end{aligned}$$
- The EM algorithm consists of starting with an initial estimate $\widehat{\theta}^{(0)}$ and repeating two steps until convergence:
  - *E(xpectation)-Step:* Construct $Q(\theta|\widehat{\theta}^{(i)})$
  - *M(aximization)-Step:* Compute $\widehat{\theta}^{(i+1)} = \text{argmax}_\theta\, Q(\theta|\widehat{\theta}^{(i)})$

- This algorithm was introduced by Dempster, Laird and Rubin (1977); it unified many special case algorithms in the literature.
- The EM algorithm increases the log likelihood at every step; this helps to ensure a very stable algorithm.
- The EM algorithm is typically linearly convergent.
- Acceleration methods may be useful. Givens and Hoeting describe some; others are available in the literature.
- If $Q(\theta|\phi,x)$ cannot be constructed in closed form, it can often be computed by Monte Carlo methods; this is the MCEM algorithm of Wei and Tanner (1990).
- In many cases MCMC methods will have to be used in the MCEM algorithm.

Example: Normal Mixture Models

Suppose $X_1, \dots, X_n$ are independent and identically distributed with density

$$f(x|\theta) = \sum_{j=1}^{M} p_j f(x|\mu_j, \sigma_j^2)$$

with $p_1, \dots, p_M \geq 0$, $\sum p_j = 1$, and $f(x|\mu, \sigma^2)$ is a Normal(μ, σ^2) density. M is assumed known.

We can think of X_i as generated in two stages:

- First a category J is selected from $\{1, \dots, M\}$ with probabilities $p_1, \dots, p_M$.
- Then, given $J = j$, a normal random variable is generated from $f(x|\mu_j, \sigma_j^2)$.

We can represent the unobserved category using indicator variables

$$Z_{ij} = \begin{cases} 1 & \text{if } J_i = j \\ 0 & \text{otherwise.} \end{cases}$$

The complete data log likelihood is

$$\log L(\theta|x,z) = \sum_{i=1}^{n} \sum_{j=1}^{M} z_{ij} \left[\log p_j - \log \sigma_j - \frac{1}{2} \left(\frac{x_i - \mu_j}{\sigma_j} \right)^2 \right]$$

The Expectation step produces

$$Q(\theta|\widehat{\theta}^{(k)}) = \sum_{i=1}^{n}\sum_{j=1}^{M}\widehat{p}_{ij}^{(k)}\left[\log p_j - \log\sigma_j - \frac{1}{2}\left(\frac{x_i-\mu_j}{\sigma_j}\right)^2\right]$$

with

$$\widehat{p}_{ij}^{(k)} = E[Z_{ij}|X=x,\theta^{(k)}] = \frac{\widehat{p}_j^{(k)} f\left(x_i\middle|\widehat{\mu}_j^{(k)},\widehat{\sigma}_j^{(k)2}\right)}{\sum_{\ell=1}^{M}\widehat{p}_\ell^{(k)} f\left(x_i\middle|\widehat{\mu}_\ell^{(k)},\widehat{\sigma}_\ell^{(k)2}\right)}$$

The Maximization step then produces

$$\begin{aligned}
\widehat{\mu}_j^{(k+1)} &= \frac{\sum_{i=1}^{n}\widehat{p}_{ij}^{(k)}x_i}{\sum_{i=1}^{n}\widehat{p}_{ij}^{(k)}}\\
\widehat{\sigma}_j^{(k+1)2} &= \frac{\sum_{i=1}^{n}\widehat{p}_{ij}^{(k)}(x_i-\widehat{\mu}_j^{(k+1)})^2}{\sum_{i=1}^{n}\widehat{p}_{ij}^{(k)}}\\
\widehat{p}_j^{(k+1)} &= \frac{1}{n}\sum_{i=1}^{n}\widehat{p}_{ij}^{(k)}
\end{aligned}$$

A simple R function to implement a single EM step might be written as

```
EMmix1 <- function(x, theta) {
    mu <- theta$mu
    sigma <- theta$sigma
    p <- theta$p
    M <- length(mu)

    ## E step
    Ez <- outer(x, 1:M, function(x, i) p[i] * dnorm(x, mu[i], sigma[i]))
    Ez <- sweep(Ez, 1, rowSums(Ez), "/")
    colSums.Ez <- colSums(Ez)

    ## M step
    xp <- sweep(Ez, 1, x, "*")
    mu.new <- colSums(xp) / colSums.Ez

    sqRes <- outer(x, mu.new, function(x, m) (x - m)^2)
    sigma.new <- sqrt(colSums(Ez * sqRes) / colSums.Ez)

    p.new <- colSums.Ez / sum(colSums.Ez)

    ## pack up result
    list(mu = mu.new, sigma = sigma.new, p = p.new)
}
```

Some notes:

- Reasonable starting values are important.
- We want a local maximum near a good starting value, not a global maximum.
- Code to examine the log likelihood for a simplified example:

  ```
  http://www.stat.uiowa.edu/~luke/classes/STAT7400/
                     examples/mixll.R
  ```

- Some recent papers:
  - Tobias Ryden (2008). EM versus Markov chain Monte Carlo for estimation of hidden Markov models: a computational perspective, *Bayesian Analysis* 3 (4), 659–688.
  - A. Berlinet and C. Roland (2009). Parabolic acceleration of the EM algorithm. *Statistics and Computing* 19 (1), 35–48.
  - Chen, Lin S., Prentice, Ross L., and Wang, Pei (2014). A penalized EM algorithm incorporating missing data mechanism for Gaussian parameter estimation, *Biometrics* 70 (2), 312–322.

Theoretical Properties of the EM Algorithm

- The conditional density of the unobserved data given the observed data is
$$k(z|x,\theta) = \frac{f(x,z|\theta)}{g(x|\theta)}$$
- Define, taking expectations with respect to $k(z|x,\phi)$,
$$\begin{aligned} Q(\theta|\phi,x) &= E_\phi[\log f(x,z|\theta)|x] \\ H(\theta|\phi,x) &= E_\phi[\log k(z|x,\theta)|x] \end{aligned}$$
- For any ϕ
$$\log L(\theta|x) = Q(\theta|\phi,x) - H(\theta|\phi,x)$$
since for any z
$$\begin{aligned} \log L(\theta|x) &= \log g(x|\theta) \\ &= \log f(x,z|\theta) - \log f(x,z|\theta) + \log g(x|\theta) \\ &= \log f(x,z|\theta) - \log \frac{f(x,z|\theta)}{g(x|\theta)} \\ &= \log f(x,z|\theta) - \log k(z|x,\theta) \end{aligned}$$
and therefore, taking taking expectations with respect to $k(z|x,\phi)$,
$$\begin{aligned} \log L(\theta|x) &= E_\phi[\log f(x,z|\theta)] - E_\phi[\log k(z|x,\theta)] \\ &= Q(\theta|\phi,x) - H(\theta|\phi,x) \end{aligned}$$

- Furthermore, for all θ

$$H(\theta|\phi) \leq H(\phi|\phi)$$

since, by Jensen's inequality,

$$\begin{aligned}
H(\theta|\phi) - H(\phi|\phi) &= E_\phi[\log k(z|\theta,x)] - E_\phi[\log k(z|\phi,x)] \\
&= E_\phi\left[\log \frac{k(z|\theta,x)}{k(z|x,\phi)}\right] \\
&\leq \log E_\phi\left[\frac{k(z|\theta,x)}{k(z|x,\phi)}\right] \\
&= \log \int \frac{k(z|\theta,x)}{k(z|x,\phi)} k(z|x,\phi)dz \\
&= \log \int k(z|x\theta)dz = 0.
\end{aligned}$$

- This implies that for any θ and ϕ

$$\log L(\theta|x) \geq Q(\theta|\phi,x) - H(\phi|\phi,x)$$

with equality when $\theta = \phi$, and therefore

  - if $\widehat{\theta}$ maximizes $L(\theta|x)$ then $\widehat{\theta}$ maximizes $Q(\theta|\widehat{\theta},x)$ with respect to θ:

$$\begin{aligned}
Q(\theta|\widehat{\theta},x) - H(\widehat{\theta}|\widehat{\theta},x) &\leq \log L(\theta|x) \\
&\leq \log L(\widehat{\theta}|x) = Q(\widehat{\theta}|\widehat{\theta},x) - H(\widehat{\theta}|\widehat{\theta},x)
\end{aligned}$$

  - if $\widehat{\theta}(\phi)$ maximizes $Q(\theta|\phi,x)$ for a given ϕ then $\log L(\phi|x) \leq \log L(\widehat{\theta}(\phi)|x)$:

$$\begin{aligned}
\log L(\phi|x) &= Q(\phi|\phi,x) - H(\phi|\phi,x) \\
&\leq Q(\widehat{\theta}(\phi)|\phi,x) - H(\phi|\phi,x) \\
&\leq Q(\widehat{\theta}(\phi)|\phi,x) - H(\widehat{\theta}(\phi)|\phi,x) \\
&= \log L(\widehat{\theta}(\phi)|x)
\end{aligned}$$

MM Algorithms

The EM algorithm can be viewed as a special case of an MM algorithm.

- MM stands for
  - *Minimization and Majorization*, or
  - *Maximization and Minorization*.
- Suppose the objective is to *maximize* a function $f(\theta)$.
- The assumption is that a *surrogate function* $g(\theta|\phi)$ is available such that

  $$f(\theta) \geq g(\theta|\phi)$$

  for all θ, ϕ, with equality when $\theta = \phi$.
- The function g is said to *minorize* the function f.
- The MM algorithm starts with an initial guess $\widehat{\theta}^{(0)}$ and then computes

  $$\widehat{\theta}^{(i+1)} = \text{argmax}_{\theta} g(\theta|\widehat{\theta}^{(i)})$$

- As in the EM algorithm,
  - if $\widehat{\theta}$ maximizes $f(\theta)$ then $\widehat{\theta}$ maximizes $g(\theta|\widehat{\theta})$
  - if $\widehat{\theta}(\phi)$ maximizes $g(\theta|\phi)$ then
$$f(\phi) \leq f(\widehat{\theta}(\phi)).$$
- The objective function values will increase, and under reasonable conditions the algorithm will converge to a local maxizer.
- Full maximization of g is not needed as long as sufficient progress is made (single Newton steps are often sufficient).
- The art in designing an MM algorithm is finding a surrogate minorizing function g that
  - produces an efficient algorithm
  - is easy to maximize
- In p-dimensional problems it is often possible to choose minorizing functions of the form
$$g(\theta|\phi) = \sum_{j=1}^{p} g_j(\theta_j|\phi)$$
so that g can be maximized by separate one-dimensional maximizations of the g_j.

Example: Bradley Terry Model

- In the Bradley Terry competition model each team has a strength θ_i and
$$P(i \text{ beats } j) = \frac{\theta_i}{\theta_i + \theta_j}$$
with $\theta_1 = 1$ for identifiability.
- Assuming independent games in which y_{ij} is the number of times i beats j the log likelihood is
$$\log L(\theta) = \sum_{i,j} y_{ij} (\log(\theta_i) - \log(\theta_i + \theta_j))$$
- Since the logarithm is concave, for any x and y
$$-\log y \geq -\log x - \frac{y-x}{x}$$
and therefore for any θ and ϕ
$$\begin{aligned}\log L(\theta) &\geq \sum_{i,j} y_{ij} \left(\log(\theta_i) - \log(\phi_i + \phi_j) - \frac{\theta_i + \theta_j - \phi_i - \phi_j}{\phi_i + \phi_j} \right) \\ &= g(\theta|\phi) \\ &= \sum_i g_i(\theta_i|\phi)\end{aligned}$$
with
$$g_i(\theta_i|\phi) = \sum_j y_{ij} \log \theta_i - \sum_j (y_{ij} + y_{ji}) \frac{\theta_i - \phi_i}{\phi_i + \phi_j}.$$
- The parameters are separated in g, and the one-dimensional maximizers can be easily computed as
$$\widehat{\theta}_i(\phi) = \frac{\sum_j y_{ij}}{\sum_j (y_{ij} + y_{ji})/(\phi_i + \phi_j)}$$

Some References on MM algorithms:

- Hunter DR and Lange K (2004), A Tutorial on MM Algorithms, *The American Statistician*, 58: 30-37.
- Lange, K (2004). *Optimization*, Springer-Verlag, New York.

Constrained Optimization

- Equality constraints arise, for example, in
  - restricted maximum likelihood estimation needed for the null hypothesis in likelihood ratio tests
  - estimating probabilities that sum to one
- Inequality constraints arise in estimating
  - probabilities or rates that are constrained to be non-negative
  - monotone functions or monotone sets of parameters
  - convex functions
- Box constraints are the simplest and the most common.
- Linear inequality constraints also occur frequently.
- Inequality constraints are often handled by converting a constrained problem to an unconstrained one:
  - Minimizing $f(x)$ subject to $g_i(x) \geq 0$ for $i = 1, \ldots, M$ is equivalent to minimizing

$$F(x) = \begin{cases} f(x) & \text{if } g_i(x) \geq 0 \text{ for } i = 1, \ldots, M \\ \infty & \text{otherwise.} \end{cases}$$

  - This objective function is not continuous.
- Sometimes a complicated unconstrained problem can be changed to simpler constrained one.

Example: L_1 Regression

The L_1 regression estimator is defined by the minimization problem

$$\widehat{b} = \underset{b}{\operatorname{argmin}} \sum |y_i - x_i^T b|$$

This unconstrained optimization problem with a non-differentiable objective function can be reformulated as a constrained linear programming problem:

For a vector $z = (z_1, \dots, z_n)^T$ let $[z]_+$ denote the vector of positive parts of z, i.e.

$$([z]_+)_i = \max(z_i, 0)$$

and let

$$\begin{aligned} b_+ &= [b]_+ \\ b_- &= [-b]_+ \\ u &= [y - Xb]_+ \\ v &= [Xb - y]_+ \end{aligned}$$

Then the L_1 estimator satisfies

$$\widehat{b} = \underset{b}{\operatorname{argmin}}\, \mathbf{1}^T u + \mathbf{1}^T v$$

subject to the constraints

$$\begin{aligned} &y = Xb_+ - Xb_- + u - v \\ &u \geq 0, v \geq 0, b_+ \geq 0, b_- \geq 0 \end{aligned}$$

- This linear programming problem can be solved by the standard simplex algorithm.
- A specialized simplex algorithm taking advantage of structure in the constraint matrix allows larger data sets to be handled (Barrodale and Roberts, 1974).
- Interior point methods can also be used and are effective for large n and moderate p.
- An alternative approach called *smoothing* is based on approximating the absolute value function by a twice continuously differentiable function and can be effective for moderate n and large p.
- Chen and Wei (2005) provide an overview in the context of the more general *quantile regression* problem.
  - Quantile regression uses a loss function of the form
  $$\rho_\tau(y) = y\left(\tau - 1_{\{y<0\}}\right) = \tau[y]_+ + (1-\tau)[-y]_+.$$
  for $0 < \tau < 1$.
  - For $\tau = \frac{1}{2}$ this is $\rho_{\frac{1}{2}}(y) = \frac{1}{2}|y|$.
- Reformulation as a constrained optimization problem is also sometimes used for LASSO and SVM computations.

Some Approaches and Algorithms

- If the optimum is in the interior of the feasible region then using standard methods on F may work, especially if these use backtracking if they encounter infinite values.
- For interior optima transormations to an unbounded feasible region may help.
- Standard algorithms can often be modified to handle box constraints.
- Other constraints are often handled using *barrier methods* that approximate F by a smooth function

$$F_\gamma(x) = \begin{cases} f(x) - \gamma \sum_{i=1}^{M} \log g_i(x) & \text{if } g(x) \geq 0 \\ \infty & \text{otherwise} \end{cases}$$

  for some $\gamma > 0$. The algorithm starts with a feasible solution, minimizes F_γ for a given γ, reduces γ, and repeats until convergence.
- Barrier methods are also called path-following algorithms.
- Path-following algorithms are useful in other settings where a harder problem can be approached through a sequence of easier problems.
- Path-following algorithms usually start the search for the next γ at the solution for the previous γ; this is sometimes called a *warm start* approach.
- The intermediate results for positive γ are in the interior of the feasible region, hence this is an *interior point method*
- More sophisticated interior point methods are available in particular for convex optimization problems.

An Adaptive Barrier Algorithm

Consider the problem of minimizing $f(x)$ subject to the linear inequality constraints

$$g_i(x) = b_i^T x - c_i \geq 0$$

for $i = 1, \ldots, M$. For an interior point $x^{(k)}$ of the feasible region define the surrogate function

$$R(x|x^{(k)}) = f(x) - \mu \sum_{i=1}^{M} \left[g_i(x^{(k)}) \log(g_i(x)) - b_i^T x \right]$$

for some $\mu > 0$. As a function of x the barrier function

$$f(x) - R(x|x^{(k)}) = \mu \sum_{i=1}^{M} \left[g_i(x^{(k)}) \log(g_i(x)) - b_i^T x \right]$$

is concave and maximized at $x = x^{(k)}$. Thus if

$$x^{(k+1)} = \operatorname*{argmin}_x R(x|x^{(k)})$$

then

$$\begin{aligned}
f(x^{(k+1)}) &= R(x^{(k+1)}|x^{(k)}) + f(x^{(k+1)}) - R(x^{(k+1)}|x^{(k)})\\
&\leq R(x^{(k)}|x^{(k)}) + f(x^{(k+1)}) - R(x^{(k+1)}|x^{(k)})\\
&\leq R(x^{(k)}|x^{(k)}) + f(x^{(k)}) - R(x^{(k)}|x^{(k)})\\
&= f(x^{(k)})
\end{aligned}$$

- The values of the objective function are non-increasing.
- This has strong similarities to an EM algorithm argument.
- The coefficient of a logarithmic barrier term decreases to zero if $x^{(k)}$ approaches the boundary.
- If f is convex and has a unique minimizer x^* then $x^{(k)}$ converges to x^*.
- This algorithm was introduced in K. Lange (1994).

Optimization in R

Several optimization functions are available in the standard R distribution:

- `optim` implements a range of methods:
  - Nelder-Mead simplex
  - Quasi-Newton with the BFGS update
  - A modified Quasi-Newton BFGS algorithm that allows box constraints
  - A conjugate gradient algorithm
  - A version of simulated annealing.

  Some notes:
  - A variety of iteration control options are available.
  - Analytical derivatives can be supplied to methods that use them.
  - Hessian matrices at the optimum can be requested.
- `nlminb` provides an interface to the FORTRAN PORT library developed at Bell Labs. Box constraints are supported.
- `nlm` implements a modified Newton algorithm as described in Dennis and Schnabel (1983).
- `constrOptim` implements the adaptive barrier method of Lange (1994) using `optim` for the optimizations within iterations.

The Optimization Task View on CRAN describes a number of other methods available in contributed packages.

Simple examples illustrating `optim` and `constrOptim` are available at

```
http://www.stat.uiowa.edu/~luke/classes/STAT7400/
                examples/optimpath.R
```

Density Estimation and Smoothing

Density Estimation

- Suppose we have a random sample $X_1, \ldots, X_n$ from a population with density f.
- Nonparametric density estimation is useful if we
  - want to explore the data without a specific parametric model
  - want to assess the fit of a parametric model
  - want a compromise between a parametric and a fully non-parametric approach
- A simple method for estimating f at a point x:

  $$\widehat{f}_n(x) = \frac{\text{no. of } X_i \text{ in } [x-h, x+h]}{2hn}$$

  for some small value of h
- This estimator has bias

  $$\text{Bias}(\widehat{f}_n(x)) = \frac{1}{2h} p_h(x) - f(x)$$

  and variance

  $$\text{Var}(\widehat{f}_n(x)) = \frac{p_h(x)(1 - p_h(x))}{4h^2 n}$$

  with

  $$p_h(x) = \int_{x-h}^{x+h} f(u) du$$

- If f is continuous at x and $f(x) > 0$, then as $h \to 0$
  - the bias tends to zero;
  - the variance tends to infinity.
- Choosing a good value of h involves a *variance-bias tradeoff*.

Kernel Density Estimation

- The estimator $\widehat{f}_n(x)$ can be written as

$$\widehat{f}_n(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)$$

  with

$$K(u) = \begin{cases} 1/2 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

- Other *kernel functions* K can be used; usually
  - K is a density function
  - K has mean zero
  - K has positive, finite variance σ_K^2

  Often K is symmetric.

- Common choices of K:

$K(u)$	Range	Name
$1/2$	$\|u\| < 1$	Uniform, Boxcar
$\frac{1}{\sqrt{2\pi}}e^{-u^2/2}$		Gaussian
$1-\|u\|$	$\|u\| < 1$	Triangular
$\frac{3}{4}(1-u^2)$	$\|u\| < 1$	Epanechnikov
$\frac{15}{16}(1-u^2)^2$	$\|u\| < 1$	Biweight

Mean Square Error for Kernel Density Estimators

- The bias and variance of a kernel density estimator are of the form

$$\begin{aligned}\text{Bias}(\widehat{f}_n(x)) &= \frac{h^2\sigma_K^2 f''(x)}{2} + O(h^4)\\ \text{Var}(\widehat{f}_n(x)) &= \frac{f(x)R(K)}{nh} + o\left(\frac{1}{nh}\right)\end{aligned}$$

  with

$$R(g) = \int g(x)^2 dx$$

  if $h \to 0$ and $nh \to \infty$ and f is reasonable.

- The pointwise asymptotic mean square error is

$$\text{AMSE}(\widehat{f}_n(x)) = \frac{f(x)R(K)}{nh} + \frac{h^4\sigma_K^4 f''(x)^2}{4}$$

  and the asymptotic mean integrated square error is

$$\text{AMISE}(\widehat{f}_n) = \frac{R(K)}{nh} + \frac{h^4\sigma_K^4 R(f'')}{4}$$

- The resulting asymptotically optimal bandwidths h are

$$\begin{aligned}h_0(x) &= \left(\frac{f(x)R(K)}{\sigma_K^4 f''(x)^2}\right)^{1/5} n^{-1/5}\\ h_0 &= \left(\frac{R(K)}{\sigma_K^4 R(f'')}\right)^{1/5} n^{-1/5}\end{aligned}$$

  with optimal AMSE and AMISE

$$\begin{aligned}\text{AMSE}_0(\widehat{f}_n(x)) &= \frac{5}{4}(\sigma_K f(x)R(K))^{4/5} f''(x)^{2/5} n^{-4/5}\\ \text{AMISE}_0(\widehat{f}_n) &= \frac{5}{4}(\sigma_K R(K))^{4/5} R(f'')^{1/5} n^{-4/5}\end{aligned}$$

Choosing a Bandwidth

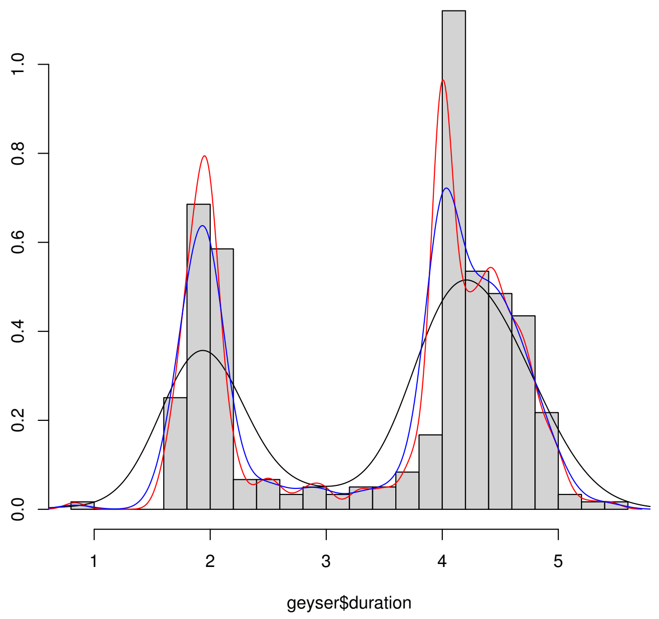
- One way to chose a bandwidth is to target a particular family, such as a Gaussian f:
  - The optimal bandwidth for minimizing AMISE when f is Gaussian and K is Gaussian
    $$h_0 = 1.059\sigma n^{-1/5}$$
  - σ can be estimated using S or the interquartile range
  - The default for `density` in R is
    $$0.9 \times \min(S, \text{IQR}/1.34) n^{-1/5}$$
    based on a suggestion of Silverman (1986, pp 45–47).
- This can often serve as a reasonable starting point.
- It does not adapt to information in the data that suggests departures from normality.
- So-called *plug-in* methods estimate $R(f'')$ to obtain
  $$\widehat{h} = \left(\frac{R(K)}{\sigma_K^4 \widehat{R(f'')}}\right)^{1/5} n^{-1/5}$$
- The Sheather-Jones method uses a different bandwidth (and kernel?) to estimate $\widehat{f}$ and then estimates $R(f'')$ by $R(\widehat{f}'')$.
- Specifying `bw="SJ"` in R's `density` uses the Sheather-Jones method. There are two variants:
  - `SJ-dpi`: direct plug-in
  - `SJ-ste`: solve the equation

  The default for `bw="SJ"` is `ste`.
- Other approaches based on leave-one-out cross-validation are available.
- Many of these are available as options in R's `density` and/or other density estimation functions available in R packages.
- Variable bandwidth approaches can be based on pilot estimates of the density produced with simpler fixed bandwidth rules.

Example: Durations of Eruptions of Old Faithful

- Based on an example in Venables and Ripley (2002).
- Durations, in minutes, of 299 consecutive eruptions of Old Faithful were recorded.
- The data are available as data set `geyser` in package `MASS`.
- Some density estimates are produced by

```
library(MASS)
data(geyser)
truehist(geyser$duration,nbin=25,col="lightgrey")
lines(density(geyser$duration))
lines(density(geyser$duration,bw="SJ"), col="red")
lines(density(geyser$duration,bw="SJ-dpi"), col="blue")
```



- Animation can be a useful way of understanding the effect of smoothing parameter choice. See files `tkdens.R`, `shinydens.R`, and `geyser.R` in

```
http://www.stat.uiowa.edu/~luke/classes/
        STAT7400/examples/
```

Also

```
http://www.stat.uiowa.edu/~luke/classes/
    STAT7400/examples/smoothex.Rmd
```

Issues and Notes

- Kernel methods do not work well at boundaries of bounded regions.
- Transforming to unbounded regions is often a good alternative.
- Variability can be assessed by asymptotic methods or by bootstrapping.
- A crude MCMC bootstrap animation:

```
g <- geyser$duration
for (i in 1:1000) {
    g[sample(299,1)] <- geyser$duration[sample(299,1)]
    plot(density(g,bw="SJ"),ylim=c(0,1.2),xlim=c(0,6))
    Sys.sleep(1/30)
}
```

- Computation is often done with equally spaced bins and fast Fourier transforms.
- Methods that adjust bandwidth locally can be used.
- Some of these methods are based on nearest-neighbor fits and local polynomial fits.
- Spline based methods can be used on the log scale; the `logspline` package implements one approach.

Density Estimation in Higher Dimensions

- Kernel density estimation can in principle be used in any number of dimensions.
- Usually a d-dimensional kernel K_d of the product form

  $$K_d(u) = \prod_{i=1}^{d} K_1(u_i)$$

  is used.
- The kernel density estimate is then

  $$\widehat{f}_n(x) = \frac{1}{n\det(H)} \sum_{i=1}^{n} K(H^{-1}(x - x_i))$$

  for some matrix H.
- Suppose $H = hA$ where $\det(A) = 1$. The asymptotic mean integrated square error is of the form

  $$\text{AMISE} = \frac{R(K)}{nh^d} + \frac{h^4}{4}\int (\text{trace}(AA^T \nabla^2 f(x)))^2 dx$$

  and therefore the optimal bandwidth and AMISE are of the form

  $$\begin{aligned} h_0 &= O(n^{-1/(d+4)}) \\ \text{AMISE}_0 &= O(n^{-4/(d+4)}) \end{aligned}$$

- Convergence is very slow if d is more than 2 or 3 since most of higher dimensional space will be empty—this is known as the *curse of dimensionality*.
- Density estimates in two dimensions can be visualized using perspective plots, surface plots, image plots, and contour plots.
- Higher dimensional estimates can often only be visualized by conditioning, or slicing.
- The `kde2d` function in package `MASS` provides two-dimensional kernel density estimates; an alternative is `bkde2D` in package `KernSmooth`.
- The `kde3d` function in the `misc3d` package provides three-dimensional kernel density estimates.

Example: Eruptions of Old Faithful

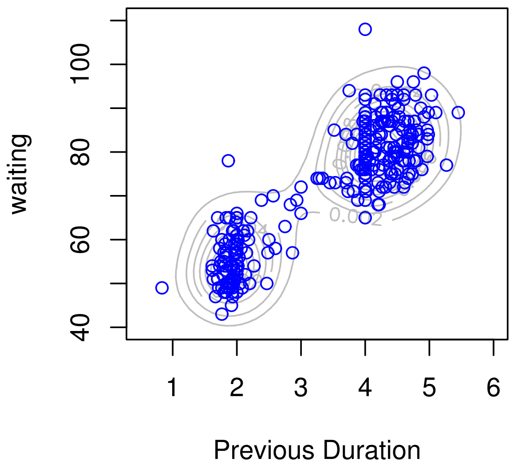
- In addition to duration times, waiting times, in minutes, until the following eruption were recorded.
- The duration of an eruption can be used to predict the waiting time until the next eruption.
- A modified data frame containing the previous duration is constructed by

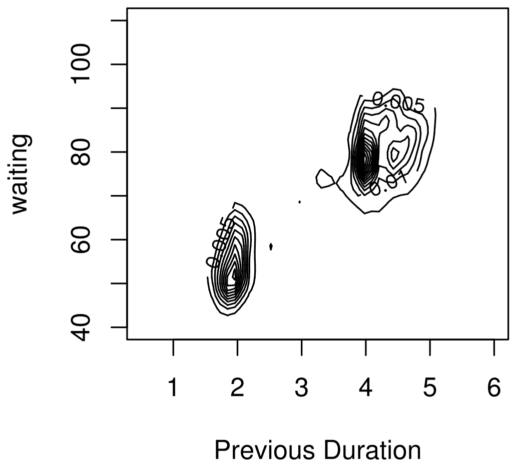
```
geyser2<-data.frame(as.data.frame(geyser[-1,]),
                    pduration=geyser$duration[-299])
```

- Estimates of the joint density of previous eruption duration and waiting time are computed by

```
kd1 <- with(geyser2,
            kde2d(pduration,waiting,n=50,lims=c(0.5,6,40,110)))
contour(kd1,col="grey",xlab="Previous Duration", ylab="waiting")
with(geyser2, points(pduration,waiting,col="blue"))
kd2 <- with(geyser2,
            kde2d(pduration,waiting,n=50,lims=c(0.5,6,40,110),
                  h=c(width.SJ(pduration),width.SJ(waiting))))
contour(kd2,xlab="Previous Duration", ylab="waiting")
```

Rounding of some durations to 2 and 4 minutes can be seen.



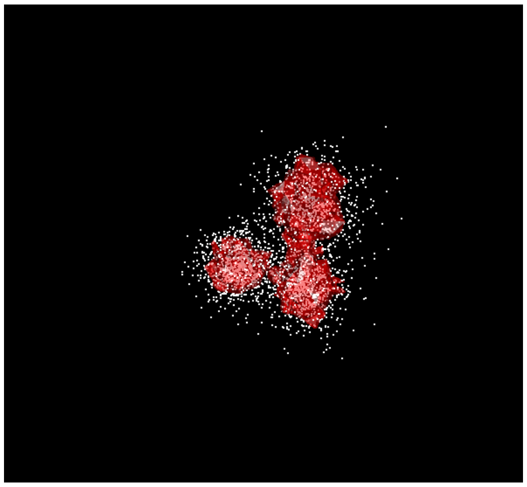


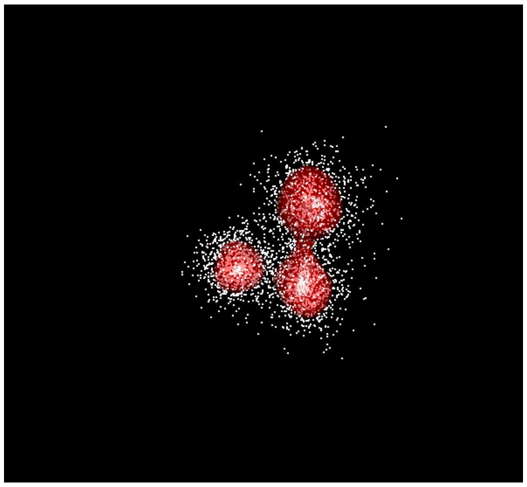
Visualizing Density Estimates

Some examples are given in `geyser.R` and `kd3.R` in

`http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/`

- Animation can be a useful way of understanding the effect of smoothing parameter choice.
- Bootstrap animation can help in visualizing uncertainty.
- For 2D estimates, options include
  - perspective plots
  - contour plots
  - image plots, with or without contours
- For 3D estimates contour plots are the main option
- Example: Data and contours for mixture of three trivariate normals and two bandwidths


BW = 0.2


BW = 0.5

Kernel Smoothing and Local Regression

- A simple non-parametric regression model is

  $$Y_i = m(x_i) + \varepsilon_i$$

  with m a smooth mean function.

- A kernel density estimator of the conditional density $f(y|x)$ is

  $$\widehat{f}_n(y|x) = \frac{\frac{1}{nh^2}\sum K\left(\frac{x-x_i}{h}\right)K\left(\frac{y-y_i}{h}\right)}{\frac{1}{nh}\sum K\left(\frac{x-x_i}{h}\right)} = \frac{1}{h}\frac{\sum K\left(\frac{x-x_i}{h}\right)K\left(\frac{y-y_i}{h}\right)}{\sum K\left(\frac{x-x_i}{h}\right)}$$

- Assuming K has mean zero, an estimate of the conditional mean is

  $$\begin{aligned}\widehat{m}_n(x) = \int y\widehat{f}_n(y|x)dy &= \frac{\sum K\left(\frac{x-x_i}{h}\right)\int y\frac{1}{h}K\left(\frac{y-y_i}{h}\right)dy}{\sum K\left(\frac{x-x_i}{h}\right)} \\ &= \frac{\sum K\left(\frac{x-x_i}{h}\right)y_i}{\sum K\left(\frac{x-x_i}{h}\right)} = \sum w_i(x)y_i\end{aligned}$$

  This is the *Nadaraya-Watson* estimator.

- This estimator can also be viewed as the result of a *locally constant* fit: $\widehat{m}_n(x)$ is the value β_0 that minimizes

  $$\sum w_i(x)(y_i - \beta_0)^2$$

- Higher degree local polynomial estimators estimate $m(x)$ by minimizing

  $$\sum w_i(x)(y_i - \beta_0 - \beta_1(x-x_i) - \cdots - \beta_p(x-x_i)^p)^2$$

- Odd values of p have advantages, and $p = 1$, local linear fitting, generally works well.

- Local cubic fits, $p = 3$, are also used.

- Problems exist near the boundary; these tend to be worse for higher degree fits.

- Bandwidth can be chosen globally or locally.
- A common local choice uses a fraction of nearest neighbors in the x direction.
- Automatic choices can use estimates of σ and function roughness and plug in to asymptotic approximate mean square errors.
- Cross-validation can also be used; it often undersmooths.
- Autocorrelation creates an identifiability problem.
- Software available in R includes
  - `ksmooth` for compatibility with S (but much faster).
  - `locpoly` for fitting and `dpill` for bandwidth selection in package `KernSmooth`.
  - `lowess` and `loess` for nearest neighbor based methods; also try to robustify.
  - `supsmu`, Friedman's *super smoother*, a very fast smoother.
  - package `locfit` on CRAN

  All of these are also available for R; some are available as stand-alone code.

Spline Smoothing

- Given data $(x_1,y_1),\dots,(x_n,y_n)$ with $x_i \in [a,b]$ one way to fit a smooth mean function is to choose m to minimize

  $$S(m,\lambda) = \sum (y_i - m(x_i))^2 + \lambda \int_a^b m''(u)^2 du$$

  The term $\lambda \int_a^b m''(u)^2 du$ is a *roughness penalty*.

- Among all twice continuously differentiable functions on $[a,b]$ this is minimized by a *natural cubic spline* with *knots* at the x_i. This minimizer is called a *smoothing spline*.

- A *cubic spline* is a function g on an interval $[a,b]$ such that for some *knots* t_i with $a = t_0 < t_1 < \cdots < t_{n+1} = b$
  - on (t_{i-1}, t_i) the function g is a cubic polynomial
  - at $t_1,\dots,t_n$ the function values, first and second derivatives are continuous.

- A cubic spline is *natural* if the second and third derivatives are zero at a and b.

- A natural cubic spline is linear on $[a,t_1]$ and $[t_n,b]$.

- For a given λ the smoothing spline is a linear estimator.

- The set of equations to be solved is large but banded.

- The fitted values $\widehat{m}_n(x_i,\lambda)$ can be viewed as

  $$\widehat{m}_n(x,\lambda) = A(\lambda)y$$

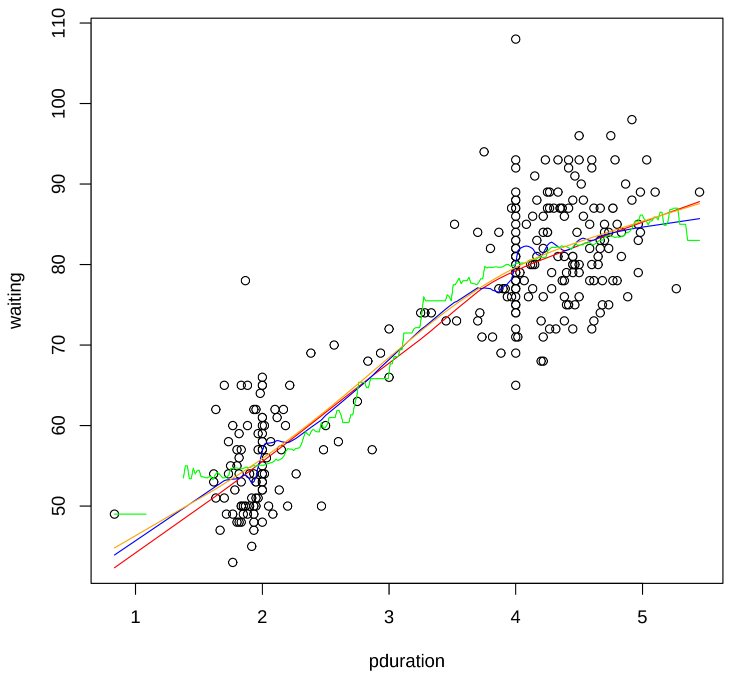
  where $A(\lambda)$ is the *smoothing matrix* or *hat matrix* for the linear fit.

- The function `smooth.spline` implements smoothing splines in R.

Example: Old Faithful Eruptions

- A nonparametric fit of waiting time to previous duration may be useful in predicting the time of the next eruption.
- The different smoothing methods considered produce the following:

```
with(geyser2, {
     plot(pduration,waiting)
     lines(lowess(pduration,waiting), col="red")
     lines(supsmu(pduration,waiting), col="blue")
     lines(ksmooth(pduration,waiting), col="green")
     lines(smooth.spline(pduration,waiting), col="orange")
})
```



- An animated version of the smoothing spline (available on line) shows the effect of varying the smoothing parameter.

Degrees of Freedom of a Linear Smoother

- For a linear regression fit with hat matrix

  $$H = X(X^TX)^{-1}X^T$$

  and full rank regressor matrix X

  $$\text{tr}(H) = \text{number of fitted parameters} = \text{degrees of freedom of fit}$$

- By analogy define the degrees of freedom of a linear smoother as

  $$\text{df}_{\text{fit}} = \text{tr}(A(\lambda))$$

  For the geyser data, the degrees of freedom of a smoothing spline fit with the default bandwidth selection rule are

```
> sum(with(geyser2,smooth.spline(pduration,waiting))$lev)
[1] 4.169843
> with(geyser2,smooth.spline(pduration,waiting))$df
[1] 4.169843
```

- For residual degrees of freedom the definition usually used is

  $$\text{df}_{\text{res}} = n - 2\text{tr}(A(\lambda)) + \text{tr}(A(\lambda)A(\lambda)^T)$$

- Assuming constant error variance, a possible estimate is

  $$\widehat{\sigma}_\varepsilon^2 = \frac{\sum(y_i - \widehat{m}_n(x_i,\lambda))^2}{\text{df}_{\text{res}}(\lambda)} = \frac{\text{RSS}(\lambda)}{\text{df}_{\text{res}}(\lambda)}$$

- The simpler estimator

  $$\widehat{\sigma}_\varepsilon^2 = \frac{\text{RSS}(\lambda)}{\text{tr}(I - A(\lambda))} = \frac{\text{RSS}(\lambda)}{n - \text{df}_{\text{fit}}}$$

  is also used.

- To reduce bias it may make sense to use a rougher smooth for variance estimation than for mean function estimation.

Choosing Smoothing Parameters for Linear Smoothers

- Many smoothing methods are linear for a given value of a smoothing parameter λ.
- Choice of the smoothing parameter λ can be based on leave-one-out cross-validation, i.e. minimizing the *cross-validation score*

$$\mathrm{CV}(\lambda) = \frac{1}{n}\sum (y_i - \widehat{m}_n^{(-i)}(x_i,\lambda))^2$$

- If the smoother satisfies (at least approximately)

$$\widehat{m}_n^{(-i)}(x_i,\lambda) = \frac{\sum_{j\neq i} A(\lambda)_{ij} y_j}{\sum_{j \neq i} A(\lambda)_{ij}}$$

  and

$$\sum_{j=1}^{n} A(\lambda)_{ij} = 1 \quad \text{for all } i$$

  then the cross-validation score can be computed as

$$\mathrm{CV}(\lambda) = \frac{1}{n}\sum \left(\frac{y_i - \widehat{m}_n(x_i,\lambda)}{1 - A_{ii}(\lambda)}\right)^2$$

- The *generalized cross-validation criterion*, or GCV, uses average leverage values:

$$\mathrm{GCV}(\lambda) = \frac{1}{n}\sum \left(\frac{y_i - \widehat{m}_n(x_i,\lambda)}{1 - n^{-1}\mathrm{trace}(A(\lambda))}\right)^2$$

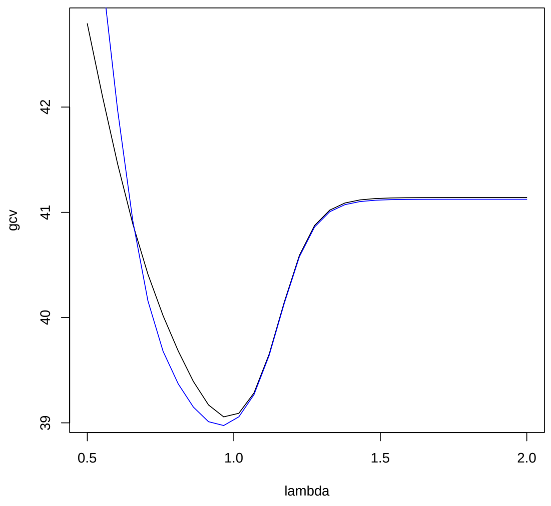
- The original motivation for GCV was computational; with better algorithms this is no longer an issue.

- An alternative motivation for GCV:
  - For an orthogonal transformation Q one can consider fitting $y_Q = QY$ with $A_Q(\lambda) = QA(\lambda)Q^T$.
  - Coefficient estimates and SS_{res} are the same for all Q, but the CV score is not.
  - One can choose an orthogonal transformation such that the diagonal elements of $A_Q(\lambda)$ are constant.
  - For any such Q we have $A_Q(\lambda)_{ii} = n^{-1}\text{trace}(A_Q(\lambda)) = n^{-1}\text{trace}(A(\lambda))$
- Despite the name, GCV does not generalize CV.
- Both CV and GCV have a tendency to undersmooth.

- For the geyser data the code

  ```
  with(geyser2, {
      lambda <- seq(0.5,2,len=30)
      f <- function(s, cv = FALSE)
          smooth.spline(pduration,waiting, spar=s, cv=cv)$cv
      gcv <- sapply(lambda, f)
      cv <- sapply(lambda, f, TRUE)
      plot(lambda, gcv, type="l")
      lines(lambda, cv, col="blue")
  })
  ```

  extracts and plots GCV and CV values:



- Both criteria select a value of λ close to 1.

- Other smoothing parameter selection criteria include
  - Mallows C_p,
$$C_p = \text{RSS}(\lambda) + 2\widehat{\sigma}_\varepsilon^2 \text{df}_{\text{fit}}(\lambda)$$
  - Akaike's information criterion (AIC)
$$\text{AIC}(\lambda) = \log\{\text{RSS}(\lambda)\} + 2\text{df}_{\text{fit}}(\lambda)/n$$
  - Corrected AIC of Hurvich, Simonoff, and Tsai (1998)
$$\text{AIC}_C(\lambda) = \log\{\text{RSS}(\lambda)\} + \frac{2(\text{df}_{\text{fit}}(\lambda)+1)}{n - \text{df}_{\text{fit}}(\lambda) - 2}$$

Spline Representations

- Splines can be written in terms of many different bases,
  - B-splines
  - truncated power basis
  - radial or thin plate basis

  Some are more useful numerically, others have interpretational advantages.

- One useful basis for a cubic spline with knots $\{\kappa_1, \ldots, \kappa_K\}$ is the *radial basis* or *thin plate basis*

$$1, x, |x-\kappa_1|^3, \ldots, |x-\kappa_K|^3$$

- More generally, a basis for splines of order $2m-1$ is

$$1, x, \ldots, x^{m-1}, |x-\kappa_1|^{2m-1}, \ldots, |x-\kappa_K|^{2m-1}$$

  for $m = 1, 2, 3, \ldots$.
  - $m = 2$ produces cubic splines
  - $m = 1$ produces linear splines

- In terms of this basis a spline is a function of the form

$$f(x) = \sum_{j=0}^{m-1} \beta_j x^j + \sum_{k=1}^{K} \delta_k |x-\kappa_k|^{2m-1}$$

- References:
  - P. J. Green and B. W. Silverman (1994). *Nonparametric Regression and Generalied Linear Models*
  - D. Ruppert, M. P. Wand, and R. J. Carroll (2003). *Semiparametric Regression*. `SemiPar` is an R package implementing the methods of this book.
  - G. Wahba (1990). *Spline Models for Observational Data.*
  - S. Wood (2017). *Generalized Additive Models: An Introduction with R, 2nd Ed.*. This is related to the `mgcv` package.

- A generic form for the fitted values is

$$\widehat{y} = X_0\beta + X_1\delta.$$

- *Regression splines* refers to models with a small number of knots K fit by ordinary least squares, i.e. by choosing β, δ to minimize

$$\|y - X_0\beta - X_1\delta\|^2$$

- *Penalized spline smoothing* fits models with a larger number of knots subject to a quadratic constraint

$$\delta^T D\delta \leq C$$

  for a positive definite D and some C.

- Equivalently, by a Lagrange multiplier argument, the solution minimizes the penalized least squares criterion

$$\|y - X_0\beta - X_1\delta\|^2 + \lambda\delta^T D\delta$$

  for some $\lambda > 0$.

- A common form of D is

$$D = \left[|\kappa_i - \kappa_j|^{2m-1}\right]_{1\leq i,j\leq K}$$

- A variant uses

$$D = \Omega^{1/2}(\Omega^{1/2})^T$$

  with

$$\Omega = \left[|\kappa_i - \kappa_j|^{2m-1}\right]_{1\leq i,j\leq K}$$

  where the *principal square root* $M^{1/2}$ of a matrix M with SVD

$$M = U\text{diag}(d)V^T$$

  is defined as

$$M^{1/2} = U\text{diag}(\sqrt{d})V^T$$

  This form ensures that D is at least positive semi-definite.

- Smoothing splines are penalized splines of degree $2m-1=3$ with knots $\kappa_i = x_i$ and
$$D = \left[|\kappa_i - \kappa_j|^3\right]_{1 \le i,j \le n}$$
and the added natural boundary constraint
$$X_0^T \delta = 0$$
- For a natural cubic spline
$$\int g''(t)^2 dt = \delta^T D \delta$$
The quadratic form $\delta^T D \delta$ is strictly positive definite on the subspace defined by $X_0^T \delta = 0$.
- Penalized splines can often approximate smoothing splines well using far fewer knots.
- The detailed placement of knots and their number is usually not critical as long as there are enough.
- Simple default rules that often work well (Ruppert, Wand, and Carroll 2003):
  - knot locations:
$$\kappa_k = \left(\frac{k+1}{K+2}\right)\text{th sample quantile of unique } x_i$$
  - number of knots:
$$K = \min\left(\frac{1}{4} \times \text{ number of unique } x_i,\, 35\right)$$

  The `SemiPar` package actually seems to use the default
$$K = \max\left(\frac{1}{4} \times \text{ number of unique } x_i,\, 20\right)$$
- More sophisticated methods for choosing number and location of knots are possible but not emphasized in the penalized spline literature at this point.

A Useful Computational Device

To minimize

$$\|Y - X_0\beta - X_1\delta\|^2 + \lambda\delta^T D\delta$$

for a given λ, suppose B satisties

$$\lambda D = B^T B$$

and

$$Y^* = \begin{bmatrix} Y \\ 0 \end{bmatrix} \qquad X^* = \begin{bmatrix} X_0 & X_1 \\ 0 & B \end{bmatrix} \qquad \beta^* = \begin{bmatrix} \beta \\ \delta \end{bmatrix}$$

Then

$$\|Y^* - X^*\beta^*\|^2 = \|Y - X_0\beta - X_1\delta\|^2 + \lambda\delta^T D\delta$$

So $\widehat{\beta}$ and $\widehat{\delta}$ can be computed by finding the OLS coefficients for the regression of Y^* on X^*.

Penalized Splines and Mixed Models

- For strictly positive definite D and a given λ minimizing the objective function
$$\|y - X_0\beta - X_1\delta\|^2 + \lambda\delta^T D\delta$$
is equivalent to maximizing the log likelihood for the mixed model
$$Y = X_0\beta + X_1\delta + \varepsilon$$
with fixed effects parameters β and
$$\begin{aligned}&\varepsilon \sim \mathrm{N}(0, \sigma_\varepsilon^2 I)\\ &\delta \sim \mathrm{N}(0, \sigma_\delta^2 D^{-1})\\ &\lambda = \sigma_\varepsilon^2/\sigma_\delta^2\end{aligned}$$
with λ known.
- Some consequences:
  - The penalized spline fit at x is the BLUP for the mixed model with known mixed effects covariance structure.
  - Linear mixed model software can be used to fit penalized spline models (the R package `SemiPar` does this).
  - The smoothing parameter λ can be estimated using ML or REML estimates of σ_ε^2 and σ_δ^2 from the linear mixed model.
  - Interval estimation/testing formulations from mixed models can be used.
- Additional consequences:
  - The criterion has a Bayesian interpretation.
  - Extension to models containing smoothing and mixed effects are immediate.
  - Extension to generalized linear models can use GLMM methodology.

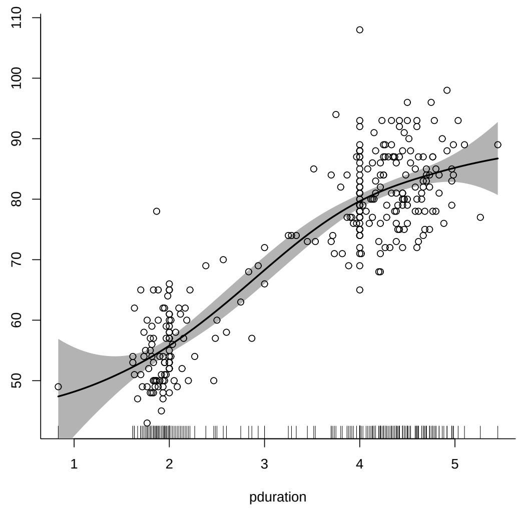
Example: Old Faithful Eruptions

- Using the function `spm` from `SemiPar` a penalized spline model can be fit with

```
> library(SemiPar)
> attach(geyser2) # needed because of flaws in spm implementation
> summary(spm(waiting ~ f(pduration)))
Summary for non-linear components:

                df spar knots
f(pduration) 4.573  2.9    28

Note this includes 1 df for the intercept.
```

- The plot method for the `spm` result produces a plot with pointwise error bars:

```
> plot(spm(waiting ~ f(pduration)), ylim = range(waiting))
> points(pduration, waiting)
```



A fit using `mgcv`:

```
> library(mgcv)
> gam.fit <- gam(waiting ~ s(pduration), data = geyser2)
> summary(gam.fit)

Family: gaussian
Link function: identity

Formula:
waiting ~ s(pduration)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  72.2886     0.3594   201.1   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Approximate significance of smooth terms:
               edf Ref.df     F p-value
s(pduration) 3.149  3.987 299.8  <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

R-sq.(adj) =  0.801   Deviance explained = 80.3%
GCV = 39.046  Scale est. = 38.503    n = 298
```

A plot of the smooth component with the mean-adjusted waiting times is produced by

```
> plot(gam.fit)
> with(geyser2, points(pduration, waiting - mean(waiting)))
```

Smoothing with Multiple Predictors

- Many methods have natural generalizations
- All suffer from the curse of dimensionality.
- Generalizations to two or three variables can work reasonably.
- Local polynomial fits can be generalized to p predictors.
- `loess` is designed to handle multiple predictors, in principle at least.
- Spline methods can be generalized in two ways:
  - *tensor product splines* use all possible products of single variable spline bases.
  - *thin plate splines* generalize the radial basis representation.
- A thin plate spline of order m in d dimensions is of the form

$$f(x) = \sum_{i=1}^{M} \beta_i \phi_i(x) + \sum_{k=1}^{K} \delta_k r(x - \kappa_k)$$

  with

$$r(u) = \begin{cases} \|u\|^{2m-d} & \text{for } d \text{ odd} \\ \|u\|^{2m-d} \log \|u\| & \text{for } d \text{ even} \end{cases}$$

  and where the ϕ_i are a basis for the space of polynomials of total degree $\leq m-1$ in d variables. The dimension of this space is

$$M = \binom{d+m-1}{d}$$

  If $d = 2, m = 2$ then $M = 3$ and a basis is

$$\phi_1(x) = 1, \phi_2(x) = x_1, \phi_3(x) = x_2$$

Penalized Thin Plate Splines

- Penalized thin plate splines usually use a penalty with

  $$D = \Omega^{1/2}(\Omega^{1/2})^T$$

  where

  $$\Omega = [r(\kappa_i - \kappa_j)]_{1 \leq i,j \leq K}$$

  This corresponds at least approximately to using a squared derivative penalty.

- Simple knot selection rules are harder for $d > 1$.

- Some approaches:

  - space-filling designs (Nychka and Saltzman, 1998)
  - clustering algorithms, such as `clara`

Multivariate Smoothing Splines

- The bivariate smoothing spline objective of minimizing

$$\sum (y_i - g(x_i))^2 + \lambda J(g)$$

  with

$$J(g) = \int\int \left(\frac{\partial^2 g}{\partial x_1^2}\right)^2 + 2\left(\frac{\partial^2 g}{\partial x_1 \partial x_2}\right)^2 + \left(\frac{\partial^2 g}{\partial x_2^2}\right)^2 dx_1 dx_2$$

  is minimized by a thin plate spline with knots at the x_i and a constraint on the δ_k analogous to the natural spline constraint.

- Scaling of variables needs to be addressed
- Thin-plate spline smoothing is closely related to *kriging*.
- The general smoothing spline uses

$$D = X_1 = [r(\kappa_i - \kappa_i)]$$

  with the constraint $X_0^T \delta = 0$.

- Challenge: the linear system to be solved for each λ value to fit a smoothing spline is large and not sparse.

Thin Plate Regression Splines

- Wood (2017) advocates an approach called *thin plate regression splines* that is implemented in the `mgcv` package.
- The approach uses the spectral decomposition of X_1

  $$X_1 = UEU^T$$

  with E the diagonal matrix of eigen values, and the columns of U the corresponding eigen vectors.
- The eigen values are ordered so that $|E_{ii}| \geq |E_{jj}|$ for $i \leq j$.
- The approach replaces X_1 with a lower rank approximation

  $$X_{1,k} = U_k E_k U_k^T$$

  using the k largest eigen values in magnitude.
- The implementation uses an iterative algorithm (Lanczos iteration) for computing the largest k eigenvalues/singular values and vectors.
- The k leading eigenvectors form the basis for the fit.
- The matrix X_1 does not need to be formed explicitly; it is enough to be able to compute $X_1 v$ for any v.
- k could be increased until the change in estimates is small or a specified limit is reached.
- As long as k is large enough results are not very sensitive to the particular value of k.
- `mgcv` by default uses $k = 10 \times 3^{d-1}$ for a d-dimensional smooth.

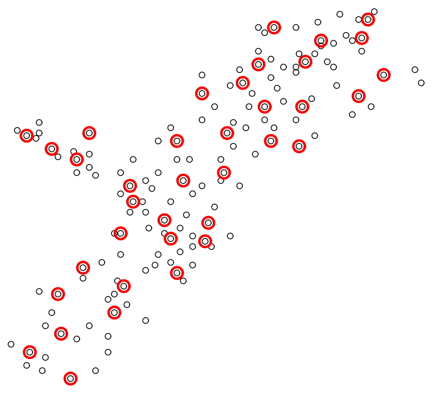
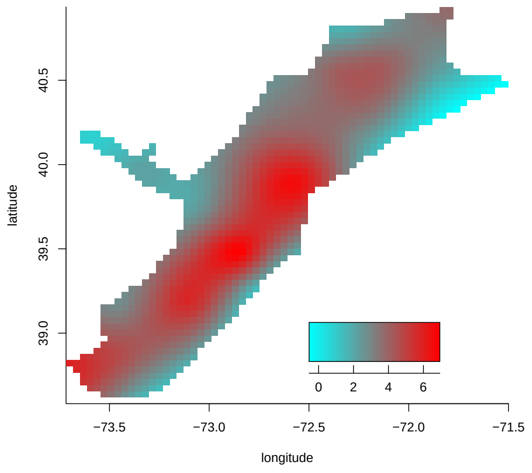
- This approach seems to be very effective in practice and avoids the need to specify a set of knots.
- The main drawback is that the choice of k and its impact on the basis used are less interpretable.
- With this approach the computational cost is reduced from $O(n^3)$ to $O(n^2k)$.
- For large n Wood (2017) recommends using a random sample of n_r rows to reduce the computation cost to $O(n_r^2k)$. (From the help files the approach in `mgcv` looks more like $O(n \times n_r \times k)$ to me).

Example: Scallop Catches

- Data records location and size of scallop catches off Long Island.
- A bivariate penalized spline fit is computed by

```
> data(scallop)
> attach(scallop)
> log.catch <- log(tot.catch + 1)
> fit <- spm(log.catch ~ f(longitude, latitude))
> summary(fit)

Summary for non-linear components:

                              df   spar knots
f(longitude,latitude) 25.12 0.2904    37
```

- Default knot locations are determined using `clara`
- Knot locations and fit:

A fit using `mgcv` would use

```
> scallop.gam <- gam(log.catch ~ s(longitude, latitude), data = scallop)
> summary(scallop.gam)

Family: gaussian
Link function: identity

Formula:
log.catch ~ s(longitude, latitude)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.4826     0.1096   31.77   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Approximate significance of smooth terms:
                        edf Ref.df     F p-value
s(longitude,latitude) 26.23  28.53 8.823  <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

R-sq.(adj) =  0.623   Deviance explained =   69%
GCV = 2.1793  Scale est. = 1.7784    n = 148
> plot(scallop.gam)
```

Computational Issues

- Algorithms that select the smoothing parameter typically need to compute smooths for many parameter values.
- Smoothing splines require solving an $n \times n$ system.
  - For a single variable the fitting system can be made tri-diagonal.
  - For thin plate splines of two or more variables the equations are not sparse.
- Penalized splines reduce the computational burden by choosing fewer knots, but then need to select knot locations.
- Thin plate regression splines (implemented in the `mgcv` package) use a rank k approximation for a user-specified k.
- As long as the number of knots or the number of terms k is large enough results are not very sensitive to the particular value of k.
- Examples are available in

```
http://www.stat.uiowa.edu/~luke/classes/
    STAT7400/examples/smoothex.Rmd
```

Statistical Learning

Some Machine Learning Terminology

- Two forms of learning:
  - *supervised learning*: *features* and responses are available for a *training set*, and a way of predicting response from features of new data is to be *learned*.
  - *unsupervised learning*: no distinguished responses are available; the goal is to discover patterns and associations among features.
- Classification and regression are supervised learning methods.
- Clustering, multi-dimensional scaling, and principal curves are unsupervised learning methods.
- *Data mining* involves extracting information from large (many cases and/or many variables), messy (many missing values, many different kinds of variables and measurement scales) data bases.
- Machine learning often emphasizes methods that are sufficiently fast and automated for use in data mining.
- Machine learning is now often considered a branch of *Artificial Intelligence (AI)*.

- Tree models are popular in machine learning
  - supervised: as predictors in classification and regression settings
  - unsupervised: for describing clustering results.
- Some other methods often associated with machine learning:
  - Bagging
  - Boosting
  - Random Forests
  - Support Vector Machines
  - Neural Networks
- References:
  - T. Hastie, R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning, 2nd Ed.*.
  - G. James, D. Witten, T. Hastie, and R. Tibshirani (2013). *An Introduction to Statistical Learning, with Applications in R*.
  - D. Hand, H, Mannila, and P. Smyth (2001). *Principles of Data Mining*.
  - C. M. Bishop (2006). *Pattern Recognition and Machine Learning*.
  - M. Shu (2008). Kernels and ensembles: perspectives on statistical learning, *The American Statistician* 62(2), 97–109.

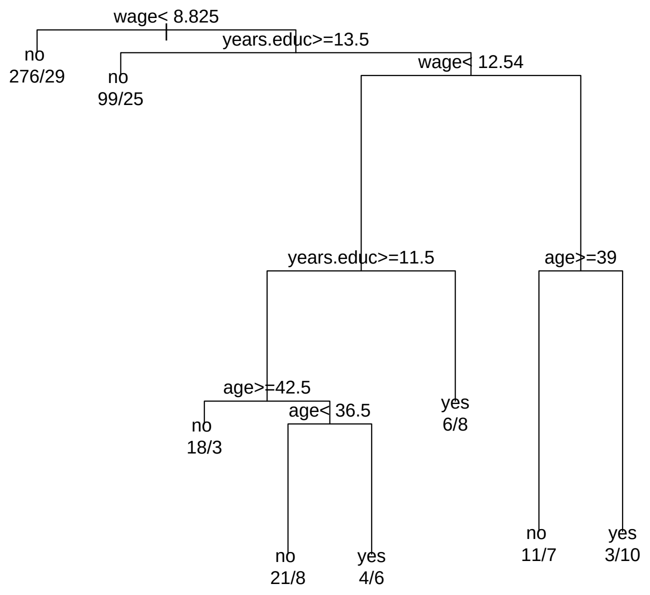
Some examples are available in

`http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/learning.Rmd`

Tree Models

- Tree models were popularized by a book and software named CART, for *Classification and Regression Trees*.
- The name CART was trademarked and could not be used by other implementations.
- Tree models partition the predictor space based on a series of binary splits.
- Leaf nodes predict a response
  - a category for *classification trees*
  - a numerical value for *regression trees*
- Regression trees may also use a simple linear model within leaf nodes of the partition.
- Using `rpart` a tree model for predicting union membership can be constructed by

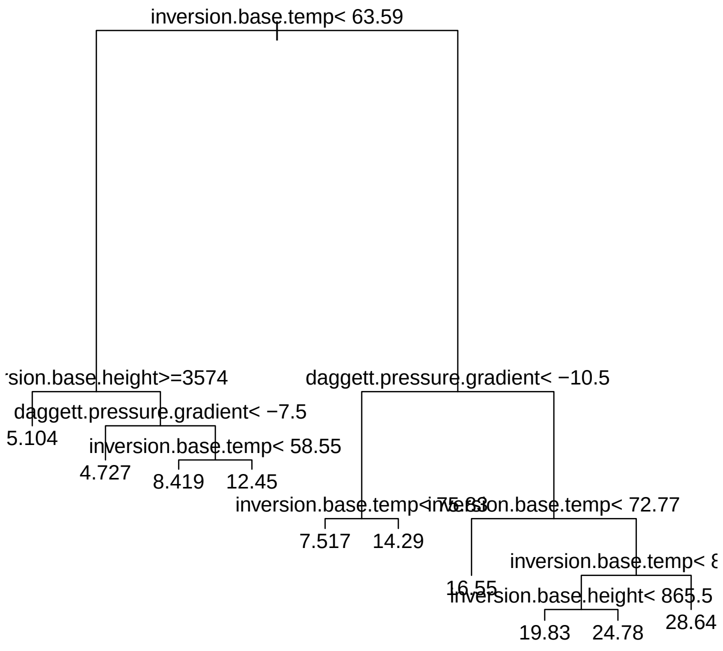
```
library(SemiPar) # for trade union data
library(rpart)
trade.union$member.fac <-
    as.factor(ifelse(trade.union$union.member, "yes", "no"))
fit <- rpart(member.fac ~ wage + age + years.educ,
             data = trade.union)
plot(fit)
text(fit, use.n = TRUE)
```



Left branch is TRUE, right branch is FALSE.

- Regression trees use a constant fit by default.
- A regression tree for the California air pollution data:

```
library(SemiPar) # for air pollution data
library(rpart)
fit2 <- rpart(ozone.level ~ daggett.pressure.gradient +
                             inversion.base.height +
                             inversion.base.temp,
              data = calif.air.poll)
plot(fit2)
text(fit2)
```


inversion.base.temp< 63.59
sion.base.height>=3574
daggett.pressure.gradient< −7.5
5.104
inversion.base.temp< 58.55
4.727
8.419
12.45
daggett.pressure.gradient< −10.5
inversion.base.temp< 75.83
inversion.base.temp< 72.77
7.517
14.29
16.55
inversion.base.temp<
inversion.base.height< 865.5
19.83
24.78
28.64

- Tree models are flexible but simple
  - results are easy to explain to non-specialists
- Small changes in data
  - can change tree structure substantially
  - usually do not change predictive performance much
- Fitting procedures usually consist of two phases:
  - growing a large tree
  - pruning back the tree to reduce over-fitting
- Tree growing usually uses a greedy approach.
- Pruning usually minimizes a penalized goodness of fit measure

  $$R(\mathscr{T}) + \lambda \text{size}(\mathscr{T})$$

  with R a raw measure of goodness of fit.
- The parameter λ can be chosen by some form of cross-validation.
- For regression trees, mean square prediction error is usually used for both growing and pruning.
- For classification trees
  - growing usually uses a loss function that rewards class purity, e.g. a Gini index

    $$G_m = \sum_{k=1}^{K} \widehat{p}_{mk}(1 - \widehat{p}_{mk})$$

    or a cross-entropy

    $$D_m = -\sum_{k=1}^{K} \widehat{p}_{mk} \log \widehat{p}_{mk}$$

    with $\widehat{p}_{mk}$ the proportion of training observations in region m that are in class k.
  - Pruning usually focuses on minimizing classification error rates.
- The `rpart` package provides one implementation; the `tree` and `party` packages are also available, among others.

Bagging, Boosting, and Random Forests

- All three are *ensemble methods*: They combine weaker predictors, or *learners*, to form a stronger one.
- A related idea is *Bayesian Model Averaging (BMA)*

Bagging: Bootstrap AGGregation

- Bootstrapping in prediction models produces a sample of predictors

$$T_1^*(x), \ldots, T_R^*(x).$$

- Usually bootstrapping is viewed as a way of assessing the variability of the predictor $T(x)$ based on the original sample.
- For predictors that are not linear in the data an aggregated estimator such as

$$T_{\text{BAG}}(x) = \frac{1}{R}\sum_{i=1}^{R} T_i^*(x)$$

  may be an improvement.
- Other aggregations are possible; for classification trees two options are
  - averaging probabilities
  - majority vote
- Bagging can be particularly effective for tree models.
  - Less pruning, or even no pruning, is needed since variance is reduced by averaging.
- Each bootstrap sample will use about 2/3 of the observations; about 1/3 will be *out of bag*, or OOB. The OOB observations can be used to construct an error estimate.
- For tree methods:

- The resulting predictors are more accurate than simple trees, but lose the simple interpretability.
- The total reduction in RSS or the Gini index due to splits on a particular variable can be used as a measure of variable importance.

• *Bumping* (Bootstrap umbrella of model parameters) is another approach:

- Given a bootstrap sample of predictors $T_1^*(x),\dots,T_R^*(x)$ choose the one that best fits the original data.
- The original sample is included in the bootstrap sample so the original predictor can be chosen if it is best.

Random Forests

- Introduced by Breiman (2001).
- Also covered by a trademark.
- Similar to bagging for regression or classification trees.
- Draws n_{tree} bootstrap samples.
- For each sample a tree is grown *without* pruning.
  - At each node m_{try} out of p available predictors are sampled at random.
  - A common choice is $m_{\text{try}} \approx \sqrt{p}$.
  - The best split among the sampled predictors is used.
- Form an ensemble predictor by aggregating the trees.
- Error rates are measured by
  - at each bootstrap iteration predicting data not in the sample (out-of-bag, OOB, data).
  - Combine the OOB error measures across samples.
- Bagging without pruning for tree models is equivalent to a random forest with $m_{\text{try}} = p$.
- A motivation is to reduce correlation among the bootstrap trees and so increase the benefit of averaging.
- The R package `randomForest` provides an interface to FORTRAN code of Breiman and Cutler.
- The software provides measures of
  - "importance" of each predictor variable
  - similarity of observations
- Some details are available in A. Liaw and M. Wiener (2002). "Classification and Regression by randomForest," *R News*.

- Other packages implementing random forests are a available as well.
- A recent addition is the `ranger` package.

Boosting

- Boosting is a way of improving on a weak supervised learner.
- The basic learner needs to be able to work with weighted data
- The simplest version applies to binary classification with responses $y_i = \pm 1$.
- A binary classifier produced from a set of weighted training data is a function
$$G(x) : \mathscr{X} \to \{-1, +1\}$$
- The *AdaBoost.M1* (adaptive boosting) algorithm:
  1. Initialize observation weights $w_i = 1/n, i = 1, \ldots, n$.
  2. For $m = 1, \ldots, M$ do
     (a) Fit a classifier $G_m(x)$ to the training data with weights w_i.
     (b) Compute the weighted error rate
     $$\text{err}_m = \frac{\sum_{i=1}^n w_i 1_{\{y_i \neq G_m(x_i)\}}}{\sum_{i=1}^n w_i}$$
     (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$
     (d) Set $w_i \leftarrow w_i \exp(\alpha_m 1_{\{y_i \neq G_m(x_i)\}})$
  3. Output $G(x) = \text{sign}\left(\sum_{i=1}^M \alpha_m G_m(x)\right)$
- The weights are adjusted to put more weight on points that were classified incorrectly.
- These ideas extend to multiple categories and to continuous responses.
- Empirical evidence suggests boosting is successful in a range of problems.
- Theoretical investigations support this.
- The resulting classifiers are closely related to additive models constructed from a set of elementary basis functions.

- The number of steps M plays the role of a model selection parameter
  - too small a value produces a poor fit
  - too large a value fits the training data too well

  Some form of regularization, e.g. based on a validation sample, is needed.
- Other forms of regularization, e.g. variants of shrinkage, are possible as well.

- A variant for boosting regression trees:

  1. Set $\widehat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
  2. For $m = 1, \ldots, M$:
     (a) Fit a tree $\widehat{f^m}(x)$ with d splits to the training data X, r.
     (b) Update $\widehat{f}$ by adding a shrunken version of $\widehat{f^m}(x)$,
     $$\widehat{f}(x) \leftarrow \widehat{f}(x) + \lambda \widehat{f^m}(x).$$
     (c) Update the residuals
     $$r_i \leftarrow r_i - \lambda \widehat{f^m}(x)$$
  3. Return the boosted model
  $$\widehat{f}(x) = \sum_{m=1}^{M} \lambda \widehat{f^m}(x)$$

- Using a fairly small d often works well.
- With $d = 1$ this fits an additive model.
- Small values of λ, e.g. 0.01 or 0.001, often work well.
- M is generally chosen by cross-validation.

References on Boosting

P. Bühlmann and T. Hothorn (2007). “Boosting algorithms: regularization, prediction and model fitting (with discussion),” *Statistical Science*, 22(4),477–522.

Andreas Mayr, Harald Binder, Olaf Gefeller, Matthias Schmid (2014). “The evolution of boosting algorithms - from machine learning to statistical modelling,” *Methods of Information in Medicine* 53(6), arXiv:1403.1452.

California Air Pollution Data

- Load data and split out a training sample:

```
library(SemiPar)
data(calif.air.poll)
library(mgcv)
train <- sample(nrow(calif.air.poll), nrow(calif.air.poll) / 2)
```

- Fit the additive linear model to the training data and compute the mean square prediction error for the test data:

```
fit <- gam(ozone.level ~ s(daggett.pressure.gradient)
                       + s(inversion.base.height)
                       + s(inversion.base.temp),
           data=calif.air.poll[train,])
mean((calif.air.poll$ozone.level[-train] -
      predict(fit, calif.air.poll[-train,]))^2)
```

- Fit a tree to the training data using all pedictors:

```
library(rpart)
tree.ca <- rpart(ozone.level ~ ., data = calif.air.poll[train,])
mean((calif.air.poll$ozone.level[-train] -
      predict(tree.ca, calif.air.poll[-train,]))^2)
```

- Use bagging on the training set:

```
library(randomForest)
bag.ca <- randomForest(ozone.level ~ .,
                       data = calif.air.poll[train,],
                       mtry = ncol(calif.air.poll) - 1)
mean((calif.air.poll$ozone.level[-train] -
      predict(bag.ca, calif.air.poll[-train,]))^2)
```

- Fit a random forest:

```
rf.ca <- randomForest(ozone.level ~ .,
                      data = calif.air.poll[train,])
mean((calif.air.poll$ozone.level[-train] -
      predict(rf.ca, calif.air.poll[-train,]))^2)
```

- Use `gbm` from the `gbm` package to fit booted regression trees:

```
library(gbm)
boost.ca <- gbm(ozone.level ~ ., data = calif.air.poll[train,],
                n.trees = 5000)
mean((calif.air.poll$ozone.level[-train] -
      predict(boost.ca, calif.air.poll[-train,],
              n.trees = 5000))^2)
boost.ca2 <- gbm(ozone.level ~ ., data = calif.air.poll[train,],
                 n.trees = 10000, interaction.depth=2)
mean((calif.air.poll$ozone.level[-train] -
      predict(boost.ca2, calif.air.poll[-train,],
              n.trees = 5000))^2)
```

- Results:

gam	18.34667	
tree	26.94041	
bagged	21.35568	
randomForest	19.13683	
boosted	19.90317	$M = 5000$
	19.04439	$M = 5000, d = 2$

These results were obtained without first re-scaling the predictors.

Support Vector Machines

- Support vector machines are a method of classification.
- The simplest form is for binary classification with training data $(x_1, y_1), \dots, (x_n, y_n)$ with

$$\begin{aligned} x_i &\in \mathbb{R}^p \\ y_i &\in \{-1, +1\} \end{aligned}$$

- Various extensions to multiple classes are available; one uses a form of majority vote among all pairwise classifiers.
- Extensions to continuous resposes are also available.
- An R implementation is `svm` in package `e1071`.

Support Vector Classifiers

- A linear binary classifier is of the form

$$G(x) = \text{sign}(x^T \beta + \beta_0)$$

- One way to choose a classifier is to minimize a penalized measure of misclassification

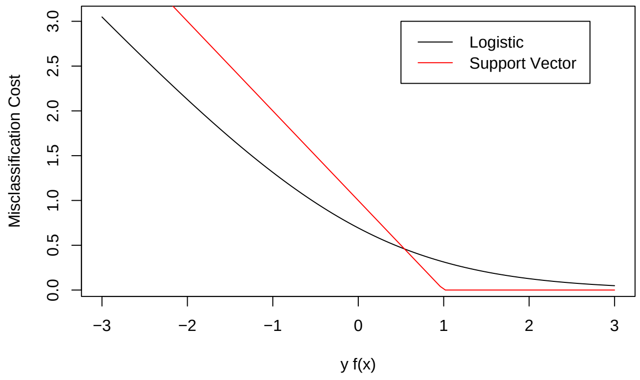
$$\min_{\beta, \beta_0} \sum_{i=1}^{n} (1 - y_i f(x))_+ + \lambda \|\beta\|^2$$

  with $f(x) = x^T \beta + \beta_0$.

  - The misclassification cost is zero for correctly classified points far from the bundary
  - The cost increases for misclassified points farther from the boundary.

- The misclassification cost is qualitatively similar to the negative log-likelihood for a logistic regression model,

$$\rho(y_i, f(x)) = -y_i f(x) + \log\left(1 + e^{y_i f(x)}\right) = \log\left(1 + e^{-y_i f(x)}\right)$$



- The support vector classifier loss function is sometimes called *hinge loss*.
- Via rewriting in terms of equivalent convex optimization problems it can be shown that the minimizer $\widehat{\beta}$ has the form.

$$\widehat{\beta} = \sum_{i=1}^{n} \widehat{\alpha}_i y_i x_i$$

for some values $\widehat{\alpha}_i \in [0, 1/(2\lambda)]$, and therefore

$$\widehat{f}(x) = x^T\widehat{\beta} + \widehat{\beta}_0 = \widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i y_i x^T x_i = \widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i y_i \langle x, x_i \rangle$$

- The values of $\widehat{\alpha}_i$ are only non-zero for x_i close to the plane $f(x) = 0$. These x_i are called *support vectors*.

- To allow for non-linear decision boundaries, we can use an extended feature set
$$h(x_i) = (h_1(x_i), \ldots, h_M(x_i))$$
- A linear boundary in $\mathbb{R}^M$ maps down to a nonlinear boundary in $\mathbb{R}^p$.
- For example, for $p = 2$ and
$$h(x) = (x_1, x_2, x_1 x_2, x_1^2, x_2^2)$$
then $M = 5$ and a linear boundary in $\mathbb{R}^5$ maps down to a quadratic boundary in $\mathbb{R}^2$.
- The estimated classification function will be of the form
$$\widehat{f}(x) = \widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i y_i \langle h(x), h(x_i) \rangle = \widehat{\beta}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i y_i K(x, x_i)$$
where the *kernel function* K is
$$K(x, x') = \langle h(x), h(x') \rangle$$
- The kernel function is symmetric and positive semi-definite.
- We don't need to specify h explicitly, only K is needed.
- Any symmetric, positive semi-definite function can be used.
- Some common choices:
$$\begin{aligned} d\text{th degree polynimial:} K(x,x') &= (1 + \langle x, x' \rangle)^d \\ \text{radial basis:} K(x,x') &= \exp(-\|x - x'\|^2 / c) \\ \text{neural network:} K(x,x') &= \tanh(a \langle x, x' \rangle + b) \end{aligned}$$
- The parameter λ in the optimization criterion is a regularization parameter. It can be chosen by cross-validation.
- Particular kernels and their parameters also need to be chosen.
  - This is analogous/equivalent to choosing sets of basis functions.
- Smoothing splines can be expressed in terms of kernels as well
  - this leads to *reproducing kernel Hilbert spaces*
  - this does not lead to the sparseness of the SVM approach

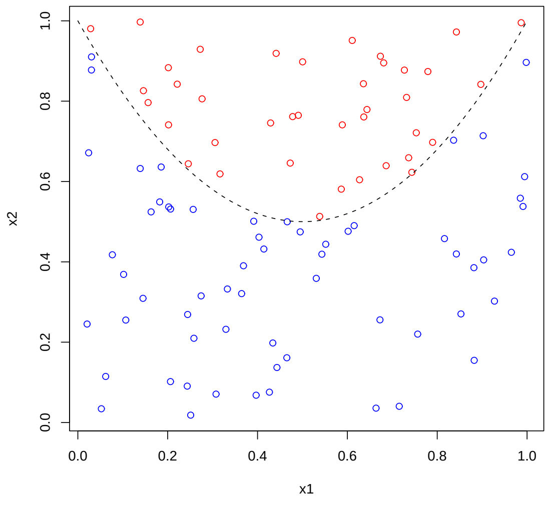
An Artificial Example

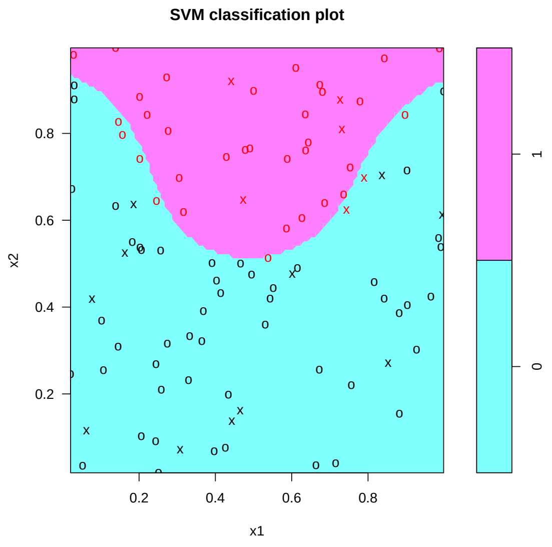
Classify random data as above or below a parabola:

```
x1 <- runif(100)
x2 <- runif(100)
z <- ifelse(x2 > 2 * (x1 - .5)^2 + .5, 1, 0)
plot(x1,x2,col=ifelse(z, "red", "blue"))
x <- seq(0,1,len=101)
lines(x, 2* (x - .5)^2 + .5, lty = 2)
```

Fit a support vector classifier using $\lambda = \frac{1}{2\texttt{cost}}$:

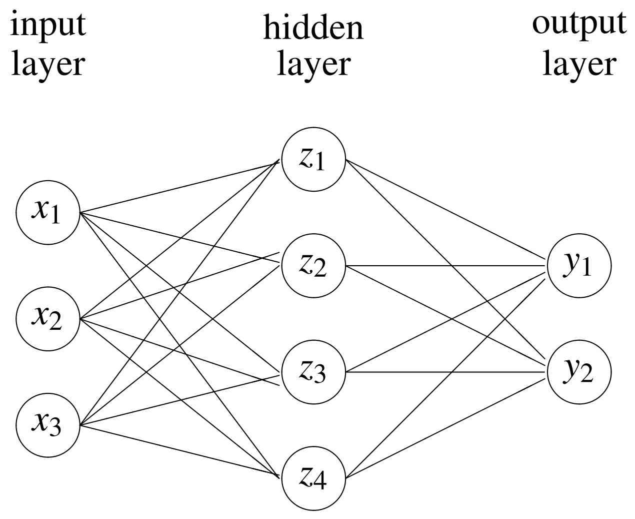
```
> library(e1071)
> fit <- svm(factor(z) ~ x1 + x2, cost = 10)
> fit

Call:
svm(formula = factor(z) ~ x1 + x2, cost = 10)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10
      gamma:  0.5

Number of Support Vectors:  17

plot(fit, data.frame(z=z,x1=x1,x2=x2), formula = x2 ~ x1, grid=100)
```


x1
x2
0.0
0.2
0.4
0.6
0.8
1.0


SVM classification plot
x1
x2
0.2
0.4
0.6
0.8
1
0

Neural Networks

- Neural networks are flexible nonlinear models.
- They are motivated by simple models for the working of neurons.
- They connect *input nodes* to *output nodes* through one or more layers of *hidden nodes*
- The simplest form is the feed-forward network with one hidden layer, inputs $x_1, \dots, x_p$ and outputs $y_1, \dots, y_k$
  - a graphical representation:



  - mathematical form:

$$
\begin{aligned}
z_m &= h(\alpha_{0m} + x^T \alpha_m)\\
t_k &= \beta_{0k} + z^T \beta_k\\
f_k(x) &= g_k(T)
\end{aligned}
$$

    The *activation function* h is usually a *sigmoidal* function, like the logistic CDF

$$h(x) = 1/(1+e^{-x})$$

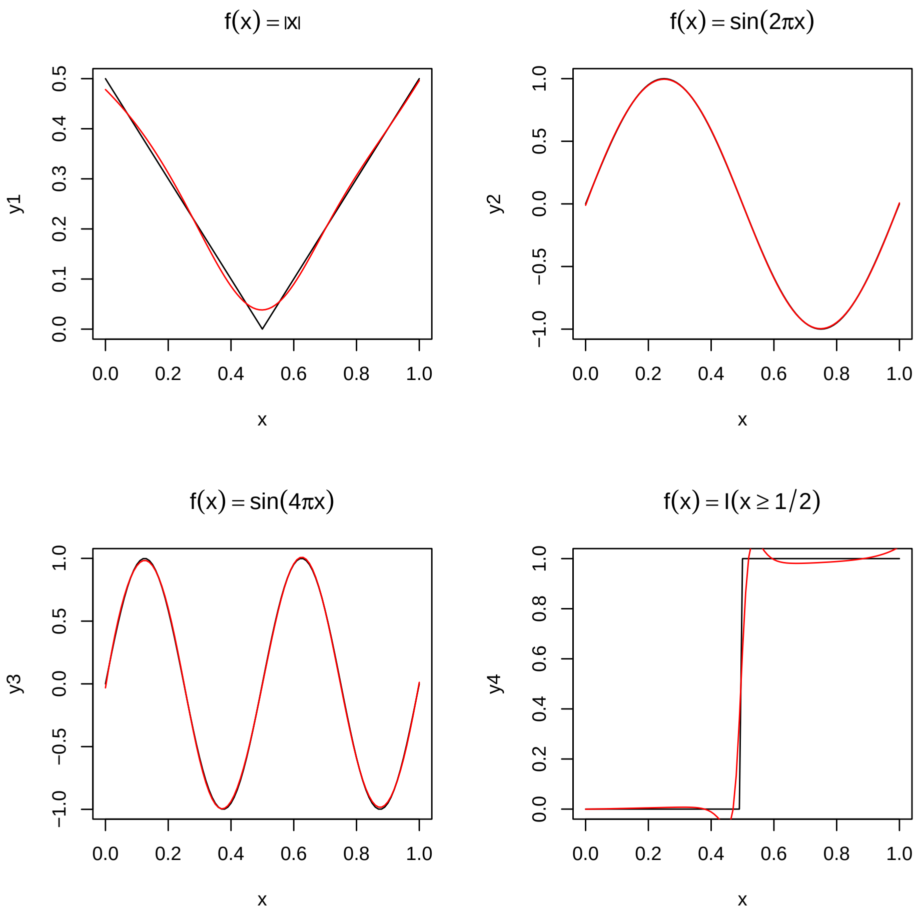
  - For regression there is usually one output with $g_1(t)$ the identity function.

- For binary classification there is usually one output with $g_1(t) = 1/(1+e^{-t})$
- For k-class classification with $k > 2$ usually there are k outputs, corresponding to binary class indicator data, with

$$g_k(t) = \frac{e^{t_k}}{\sum_j e^{t_j}}$$

  This is often called a *softmax* criterion.

- By increasing the size of the hidden layer M a neural network can uniformly approximate any continuous function on a compact set arbitrarily well.
- Some examples, fit to $n = 101$ data points using function `nnet` from package `nnet` with a hidden layer with $M = 5$ nodes:



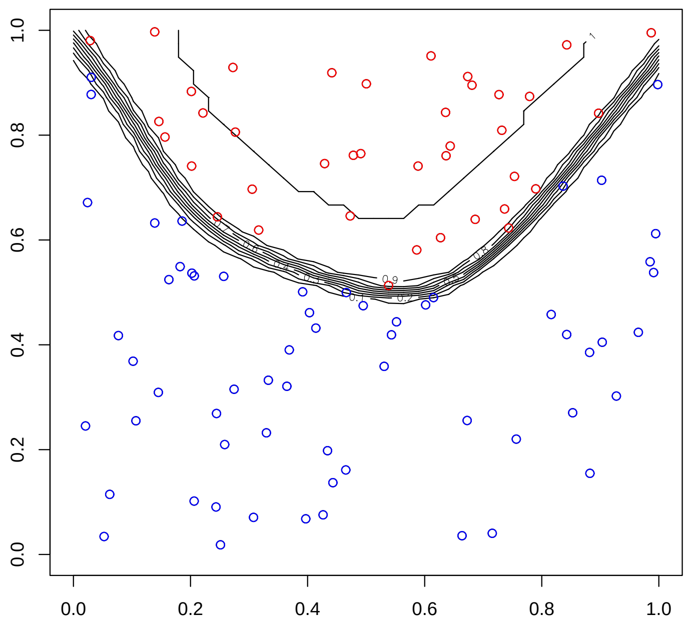
- Fitting is done by maximizing a log likelihood $L(\alpha, \beta)$ assuming
  - normal errors for regression
  - a logistic model for classification
- The likelihood is highly multimodal and the parameters are not identified
  - relabeling hidden nodes does not change the model, for example
  - random starting values are usually used
  - parameters are not interpretable
- If M is large enough to allow flexible fitting then over-fitting is a risk.
- Regularization is used to control overfitting: a penalized log likelihood of the form

$$L(\alpha,\beta) - \lambda\left(\sum_m \|\alpha_m\|^2 + \sum_k \|\beta_k\|^2\right)$$

  is maximized.
  - For this to make sense it is important to center and scale the features to have comparable units.
  - This approach is referred to as *weight decay* and λ is the decay parameter.
- As long as M is large enough and regularization is used, the specific value of M seems to matter little.
- The weight decay parameter is often determined by N-fold cross validation, often with $N = 10$
- Because of the random starting points, results in repeated runs can differ.
  - one option is to make several runs and pick the best fit
  - another is to combine results from several runs by averaging or majority voting.

- Fitting a neural net to the artificial data example:

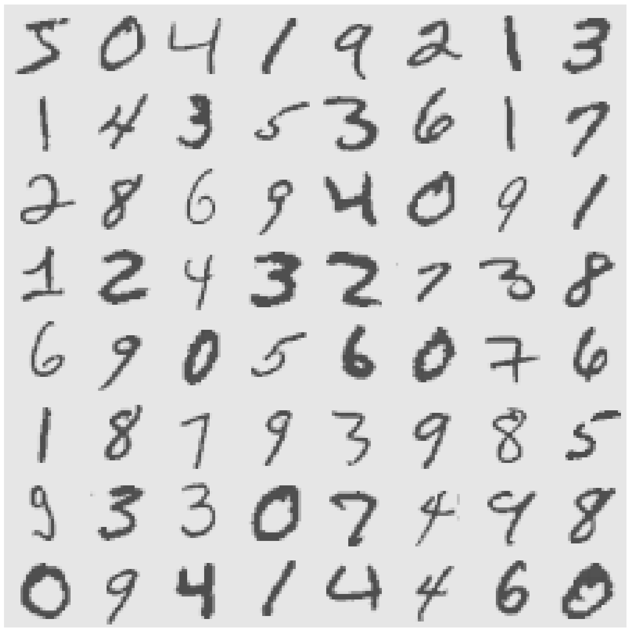
```
nnet(z ~ x1 + x2, size=10, entropy = TRUE, decay = .001,
     maxit = 300)
```



Example: Recognizing Handwritten Digits

- Data consists of scanned ZIP code digist from the U.S. postal service. Available at `http://yann.lecun.com/exdb/mnist/` as a binary file.

  Training data consist of a small number of original images, around 300, and additional images generated by random shifts. Data are 28×28 grayscale images, along with labels.

  This has become a standard machine learning test example.

- Data can be read into R using `readBin`.
- The fit, using 6000 observations and $M = 100$ nodes in the hidden layer took 11.5 hours on `r-lnx400`:

```
fit <- nnet(X, class.ind(lab), size = 100,
            MaxNWts = 100000, softmax = TRUE)
```

  and produced a training misclassification rate of about 8% and a test misclassification rate of about 12%.

- Other implementations are faster and better for large problems.

Deep Learning

- Deep learning models are multi-level non-linear models
- A supervised model with observed responses Y and features X with M layers would be

  $$Y \sim f_1(y|Z_1), Z_1 \sim f_2(z_1|Z_2), \ldots, Z_M \sim f_M(z_M|X)$$

  with $Z_1, \ldots, Z_M$ unobserved latent values.
- An unsupervised model with observed features X would be

  $$X \sim f_1(x|Z_1), Z_1 \sim f_2(z_1|Z_2), \ldots, Z_M \sim f_M(z_M)$$

- These need to be nonlinear so they don't collapse into one big linear model.
- The layers are often viewed as capturing features at different levels of granularity.
- For image classification these might be
  - X: pixel intensities
  - Z_1: edges
  - Z_2: object parts (e.g. eyes, noses)
  - Z_3: whole objects (e.g. faces)
- Multi-layer, or deep, neural networks are one approach, that has become very successful.

- Deep learning methods have become very successful in recent years due to a combination of increased computing power and algorithm improvements.
- Some key algorithm developments include:
  - Use of *stochastic gradient descent* for optimization.
  - Backpropagation for efficient gradient evaluation.
  - Using the piece-wise linear *Rectified Linear Unit (ReLU)* activation function
    $$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$
  - Specialized structures, such as convolutional and recurrent neural networks.
  - Use of dropout, regularization, and early stopping to avoid overfitting.

Stochastic Gradient Descent

- *Gradient descent* for minimizing a function f tries to improve a current guess by taking a step in the direction of the negative gradient:
$$x' = x - \eta \nabla f(x)$$
- The step size η is sometimes called the *learning rate*.
- In one dimension the best step size near the minimum is $1/f''(x)$.
- A step size that is too small converges to slowly; a step size too large may not converge at all.
- Line search is possible but may be expensive.
- Using a fixed step size, with monitoring to avoid divergence, or using a slowly decreasing step size are common choices.
- For a DNN the function to be minimized with respect to parameters A is typically of the form
$$\sum_{i=1}^{n} L_i(y_i, x_i, A)$$
for large n.
- Computing function and gradient values for all n training cases can be very costly.
- *Stochastic gradient descent* at each step chooses a random *minibatch* of B of the training cases and computes a new step based on the loss function for the minibatch.
- The minibatch size can be as small as $B = 1$.
- Stochastic gradient descent optimizations are usually divided into *epochs*, with each epoch expected to use each training case once.

Backpropagation

- Derivatives of the objective function are computed by the chain rule.
- This is done most efficiently by working backwards; this corresponds to the *reverse mode* of automatic differentiation.
- A DNN with two hidden layers can be represented as

$$F(x;A) = G(A_3 H_2(A_2 H_1(A_1 x)))$$

  If G is elementwise the identity, and the H_i are elementwise ReLU, then this is a piece-wise linear function of x.
- The computation of $w = F(x;A)$ can be broken down into intermediate steps as

$$\begin{aligned} t_1 &= A_1 x & z_1 &= H_1(t_1)\\ t_2 &= A_2 z_1 & z_2 &= H_2(t_2)\\ t_3 &= A_3 z_2 & w &= G(t_3) \end{aligned}$$

- The gradient components are then computed as

$$\begin{aligned} B_3 &= \nabla G(t_3) & \frac{\partial w}{\partial A_3} &= \nabla G(t_3) z_2 = B_3 z_2\\ B_2 &= B_3 A_3 \nabla H_2(t_2) & \frac{\partial w}{\partial A_2} &= \nabla G(t_3) A_3 \nabla H_2(t_2) z_1 = B_2 z_1\\ B_1 &= B_2 A_2 \nabla H_1 x & \frac{\partial w}{\partial A_1} &= \nabla G(t_3) A_3 \nabla H_2(t_2) A_2 \nabla H_1(t_1) x = B_1 x \end{aligned}$$

- For ReLU activations the elements of $\nabla H_i(t_i)$ will be 0 or 1.
- For n parameters the computation will typically be of order $O(n)$.
- Many of the computations can be effectively parallelized.

Convolutional and Recurrent Neural Networks

- In image processing features (pixel intensities) have a neighborhood structure.
- A convolutional neural network uses one or more hidden layers that are:
  - only locally connected;
  - use the same parameters at each location.
- A simple convolution layer might use a pixel and each of its 4 neighbors with
  $$t = (a_1 R + a_2 L + a_3 U + a_4 D) z$$
  where, e.g.
  $$R_{ij} = \begin{cases} 1 & \text{if pixel } i \text{ is immediately to the right of pixel } j \\ 0 & \text{otherwise.} \end{cases}$$
- With only a small nunber of parameters per layer it is feasible to add tens of layers.
- Similarly, a recurrent neural network can be designed to handle temporal dependencies for time series or speech recognition.

Avoiding Over-Fitting

- Both L_1 and L_2 regularization are used.
- Another strategy is *dropout*:
  - In each epoch keep a node with probability p and drop with probability $1-p$.
  - In the final fit multiply each node's output by p.

  This simulates an ensemble method fitting many networks, but costs much less.
- Random starts are an important component of fitting networks.
- Stopping early, combined with random starts and randomness from stochastic gradient descent, is also thought to be an effective regularization.
- Cross-validation during training can be used to determine when to stop.

Notes and References

- Deep learning methods have been very successful in a number of areas, such as:
  - Image classification and face recognition. *AlexNet* is a very successful image classifier.
  - *Google Translate* is now based on a deep neural network approach.
  - Speech recognition.
  - Playing Go and chess.
- Being able to effectively handle large data sets is an important consideration in this research.
- Highly parallel GPU based and distributed architectures are often needed.
- Some issues:
  - Very large training data sets are often needed.
  - In high dimensional problems having a high signal to noise ratio seems to be needed.
  - Models can be very brittle – small data perturbations can lead to very wrong results.
  - Biases in data will lead to biases in predictions. A probably harmless example deals with evaluating selfies in social media; there are much more serious examples.
- Some R packages for deep learning include `darch`, `deepnet`, `deepr`, `domino`, `h2o`, `keras`.
- Some references:
  - A nice introduction was provided by Thomas Lumley in a 2019 Ihaka Lecture
  - deeplearning.net web site
  - Li Deng and Dong Yu (2014), *Deep Learning: Methods and Applications*
  - Charu Aggarwal (2018), *Neural Networks and Deep Learning*.

- A Primer on Deep Learning
- A blog post on deep learning software in R.
- A nice simulator.

Some examples are available in

```
http://www.stat.uiowa.edu/~luke/classes/STAT7400/
                examples/keras.Rmd
```

Mixture of Experts

- Mixture models for prediction of y based on fearures x produce predictive distributions of the form
$$f(y|x) = \sum_{i=1}^{M} f_i(y|x)\pi_i$$
with f_i depending on parameters that need to be learned from training data.
- A generalization allows the mixing probabilities to depend on the features:
$$f(y|x) = \sum_{i=1}^{M} f_i(y|x)\pi_i(x)$$
with f_i and π_i depending on parameters that need to be learned.
- The f_i are referred to as *experts*, with different experts being better informed about different ranges of x values, and f this is called a *mixture of experts*.
- Tree models can be viewed as a special case of a mixture of experts with $\pi_i(x) \in \{0,1\}$.
- The mixtures π_i can themselves be modeled as a mixture of experts. This is the *hierarchical mixture of experts* (HME) model.

Symbolic Computation

- Symbolic computations include operations such as symbolic differentiation or integration.
- Symbolic computation is often done using specialized systems, e.g.
  - Mathematica
  - Maple
  - Macsyma
  - Yacas
- R interfaces are available for a number of these.
- R code can be examined and constructed using R code.
- This is sometimes referred to as *computing on the language*.

- Some simple examples:

```
> e <- quote(x+y)
> e
x + y
> e[[1]]
`+`
> e[[2]]
x
> e[[3]]
y
> e[[3]] <- as.name("z")
> e
x + z
> as.call(list(as.name("log"), 2))
log(2)
```

- One useful application is symbolic computation of derivatives.
- R provides functions `D` and `deriv` that do this. These are implemented in `C`.
- The `Deriv` package is another option.

- The start of a simplified symbolic differentiator implemented in R as a function `d` is available in `d.R`.
- Some simple examples:

```
> source("http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/derivs/d.R")

> d(quote(x), "x")
[1] 1
> d(quote(y), "x")
[1] 0
> d(quote(2 + x), "x")
0 + 1
> d(quote(2 * x), "x")
0 * x + 2 * 1
> d(quote(y * x), "x")
0 * x + y * 1
```

- The results are correct but are not ideal.
- There are many things `d` cannot handle yet, such as

```
d(quote(-x), "x")
d(quote(x/y), "x")
d(quote(x+(y+z)), "x")
```

- Simplifying expressions like those produced by `d` is a challenging task, but the results can be made a bit more pleasing to look at by avoiding creating some expressions that have obvious simplifications, like
  - sums where one operand is zero
  - products where one operand is one.
- Symbolic computation is used by all functions that support model formulas.
- Symbolic computation can also be useful for identifying full conditional distributions, e.g. for constructing Gibbs samplers.
- The byte code compiler for R also uses symbolic computation to analyze and compile R expressions, as do the R source code analysis tools in the `codetools` package.

Simulation

Computer Simulation

- Computer simulations are experiments performed on the computer using computer-generated random numbers.
- Simulation is used to
  - study the behavior of complex systems such as
    - biological systems
    - ecosystems
    - engineering systems
    - computer networks
  - compute values of otherwise intractable quantities such as integrals
  - maximize or minimize the value of a complicated function
  - study the behavior of statistical procedures
  - implement novel methods of statistical inference
- Simulations need
  - uniform random numbers
  - non-uniform random numbers
  - random vectors, stochastic processes, etc.
  - techniques to design good simulations
  - methods to analyze simulation results

Uniform Random Numbers

- The most basic distribution is the uniform distribution on $[0,1]$
- Ideally we would like to be able to obtain a sequence of independent draws from the uniform distribution on $[0,1]$.
- Since we can only use finitely many digits, we can also work with
  - A sequence of independent discrete uniform random numbers on $\{0,1,\ldots,M-1\}$ or $\{1,2,\ldots,M\}$ for some large M.
  - A sequence of independent random bits with equal probability for 0 and 1.
- Some methods are based on physical processes such as
  - nuclear decay

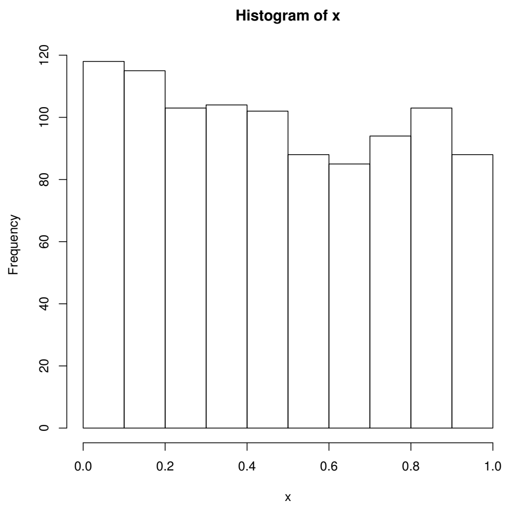
    `http://www.fourmilab.ch/hotbits/`
  - atmospheric noise

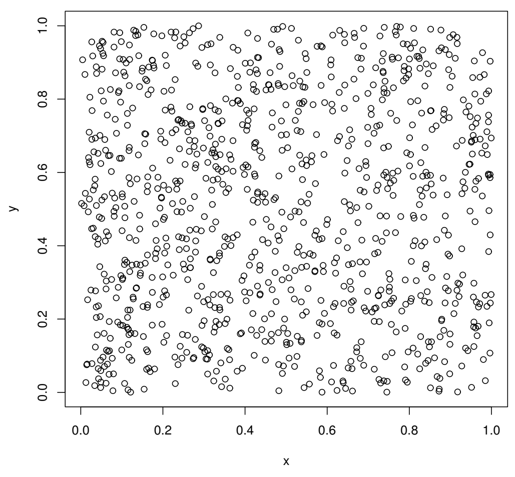
    `http://www.random.org/`

    The R package random provides an interface.
  - air turbulence over disk drives or thermal noise in a semiconductor (Toshiba Random Master PCI device)
  - event timings in a computer (Linux `/dev/random`)

Using `/dev/random` from R

```
devRand <- file("/dev/random", open="rb")
U <- function()
    (as.double(readBin(devRand, "integer"))+2^31) / 2^32
x <-numeric(1000)
for (i in seq(along=x)) x[i] <- U()
hist(x)
y <- numeric(1000)
for (i in seq(along=x)) y[i] <- U()
plot(x,y)
close(devRand)
```





Issues with Physical Generators

- can be very slow
- not reproducible except by storing all values
- distribution is usually not exactly uniform; can be off by enough to matter
- departures from independence may be large enough to matter
- mechanisms, defects, are hard to study
- can be improved by combining with other methods

Pseudo-Random Numbers

Pseudo-random number generators produce a sequence of numbers that is

- not random
- easily reproducible
- "unpredictable;" "looks random"
- behaves in many respects like a sequence of independent draws from a (discretized) uniform $[0,1]$ distribution
- fast to produce

Pseudo-random generators come in various qualities

- Simple generators
  - easy to implement
  - run very fast
  - easy to study theoretically
  - usually have known, well understood flaws
- More complex
  - often based on combining simpler ones
  - somewhat slower but still very fast
  - sometimes possible to study theoretically, often not
  - guaranteed to have flaws; flaws may not be well understood (yet)
- Cryptographic strength

  `https://www.schneier.com/fortuna.html`

  - often much slower, more complex
  - thought to be of higher quality
  - may have legal complications
  - weak generators can enable exploits, a recent issue in iOS 7

We use mostly generators in the first two categories.

General Properties

- Most pseudo-random number generators produce a sequence of integers $x_1, x_2, \ldots$ in the range $\{0, 1, \ldots, M-1\}$ for some M using a recursion of the form
$$x_n = f(x_{n-1}, x_{n-2}, \ldots, x_{n-k})$$
- Values $u_1, u_2, \ldots$ are then produced by
$$u_i = g(x_{di}, x_{di-1}, \ldots, x_{di-d+1})$$
- Common choices of M are
  - $M = 2^{31}$ or $M = 2^{32}$
  - $M = 2^{31} - 1$, a *Mersenne prime*
  - $M = 2$ for bit generators
- The value k is the *order* of the generator
- The set of the most recent k values is the *state* of the generator.
- The initial state $x_1, \ldots, x_k$ is called the *seed*.
- Since there are only finitely many possible states, eventually these generators will repeat.
- The length of a cycle is called the *period* of a generator.
- The maximal possible period is on the order of M^k
- Needs change:
  - As computers get faster, larger, more complex simulations are run.
  - A generator with period 2^{32} used to be good enough.
  - A current computer can run through 2^{32} pseudo-random numbers in under one minute.
  - Most generators in current use have periods 2^{64} or more.
  - Parallel computation also raises new issues.

Linear Congruential Generators

- A linear congruential generator is of the form

$$x_i = (ax_{i-1} + c) \mod M$$

  with $0 \leq x_i < M$.

  - a is the *multiplier*
  - c is the *increment*
  - M is the *modulus*

- A multiplicative generator is of the form

$$x_i = ax_{i-1} \mod M$$

  with $0 < x_i < M$.

- A linear congruential generator has full period M if and only if three conditions hold:

  - $\gcd(c, M) = 1$
  - $a \equiv 1 \mod p$ for each prime factor p of M
  - $a \equiv 1 \mod 4$ if 4 divides M

- A multiplicative generator has period at most $M - 1$. Full period is achieved if and only if M is prime and a is a *primitive root modulo M*, i.e. $a \neq 0$ and $a^{(M-1)/p} \not\equiv 1 \mod M$ for each prime factor p of $M - 1$.

Examples

- Lewis, Goodman, and Miller ("minimal standard" of Park and Miller):

  $$x_i = 16807 x_{i-1} \mod (2^{31}-1) = 7^5 x_{i-1} \mod (2^{31}-1)$$

  Reasonable properties, period $2^{31}-2 \approx 2.15 * 10^9$ is very short for modern computers.

- RANDU:

  $$x_i = 65538 x_{i-1} \mod 2^{31}$$

  Period is only 2^{29} but that is the least of its problems:

  $$u_{i+2} - 6u_{i+1} + 9u_i = \text{an integer}$$

  so (u_i, u_{i+1}, u_{i+2}) fall on 15 parallel planes. Using the `randu` data set and the `rgl` package:

  ```
     library(rgl)
     points3d(randu)
     par3d(FOV=1)  ## removes perspective distortion
  ```

  With a larger number of points:

  ```
  seed <- as.double(1)
  RANDU <- function() {
      seed <<- ((2^16 + 3) * seed) %% (2^31)
      seed/(2^31)
  }

  U <- matrix(replicate(10000 * 3, RANDU()), ncol = 3, byrow = TRUE)
  clear3d()
  points3d(U)
  par3d(FOV=1)
  ```

  This generator used to be the default generator on IBM 360/370 and DEC PDP11 machines.

Some examples are available in

`http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/sim.Rmd`

Lattice Structure

- All linear congruential sequences have a *lattice structure*
- Methods are available for computing characteristics, such as maximal distance between adjacent parallel planes
- Values of M and a can be chosen to achieve good lattice structure for $c = 0$ or $c = 1$; other values of c are not particularly useful.

Shift-Register Generators

- Shift-register generators take the form

  $$x_i = a_1 x_{i-1} + a_2 x_{i-2} + \cdots + a_p x_{i-p} \mod 2$$

  for binary constants $a_1, \ldots, a_p$.

- values in $[0,1]$ are often constructed as

  $$u_i = \sum_{s=1}^{L} 2^{-s} x_{ti+s} = 0.x_{it+1} x_{it+2} \ldots x_{it+L}$$

  for some t and $L \leq t$. t is the *decimation*.

- The maximal possible period is $2^p - 1$ since all zeros must be excluded.

- The maximal period is achieved if and only if the polynomial

  $$z^p + a_1 z^{p-1} + \cdots + a_{p-1} z + a_p$$

  is irreducible over the finite field of size 2.

- Theoretical analysis is based on k-distribution: A sequence of M bit integers with period $2^p - 1$ is *k-distributed* if every k-tuple of integers appears 2^{p-kM} times, except for the zero tuple, which appears one time fewer.

- Generators are available that have high periods and good k-distribution properties.

Lagged Fibonacci Generators

- Lagged Fibonacci generators are of the form

$$x_i = (x_{i-k} \circ x_{i-j}) \mod M$$

  for some binary operator $\circ$.

- Knuth recommends

$$x_i = (x_{i-100} - x_{i-37}) \mod 2^{30}$$

  - There are some regularities if the full sequence is used; one recommendation is to generate in batches of 1009 and use only the first 100 in each batch.
  - Initialization requires some care.

Combined Generators

- Combining several generators may produce a new generator with better properties.
- Combining generators can also fail miserably.
- Theoretical properties are often hard to develop.
- Wichmann-Hill generator:

$$\begin{aligned} x_i &= 171 x_{i-1} \mod 30269 \\ y_i &= 172 y_{i-1} \mod 30307 \\ z_i &= 170 z_{i-1} \mod 30323 \end{aligned}$$

  and
$$u_i = \left(\frac{x_i}{30269} + \frac{y_i}{30307} + \frac{z_i}{30232}\right) \mod 1$$

  The period is around 10^{12}.

  This turns out to be equivalent to a multiplicative generator with modulus

$$M = 27817185604309$$

- Marsaglia's Super-Duper used in S-PLUS and others combines a linear congruential and a feedback-shift generator.

Other Generators

- Mersenne twister
- Marsaglia multicarry
- Parallel generators
  - SPRNG `http://sprng.cs.fsu.edu`.
  - L'Ecuyer, Simard, Chen, and Kelton

    `http://www.iro.umontreal.ca/~lecuyer/myftp/streams00/`

Pseudo-Random Number Generators in R

R provides a number of different basic generators:

Wichmann-Hill: Period around 10^{12}

Marsaglia-Multicarry: Period at least 10^{18}

Super-Duper: Period around 10^{18} for most seeds; similar to S-PLUS

Mersenne-Twister: Period $2^{19937}-1 \approx 10^{6000}$; equidistributed in 623 dimensions; current default in R.

Knuth-TAOCP: Version from second edition of *The Art of Computer Programming, Vol. 2*; period around 10^{38}.

Knuth-TAOCP-2002: From third edition; differs in initialization.

L'Ecuyer-CMRG: A combined multiple-recursive generator from L'Ecuyer (1999). The period is around 2^{191}. This provides the basis for the multiple streams used in package `parallel`.

user-supplied: Provides a mechanism for installing your own generator; used for parallel generation by

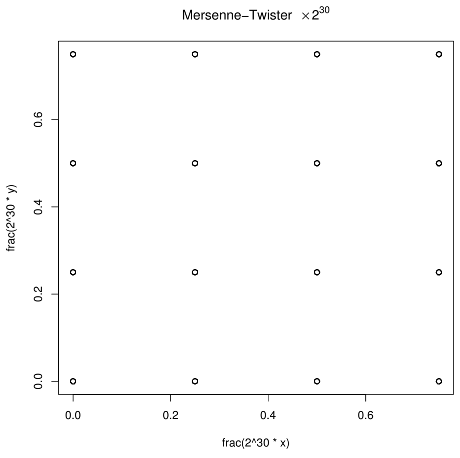
- `rsprng` package interface to SPRNG
- `rlecuyer` package interface to L'Ecuyer, Simard, Chen, and Kelton system
- `rstreams` package, another interface to L'Ecuyer et al.

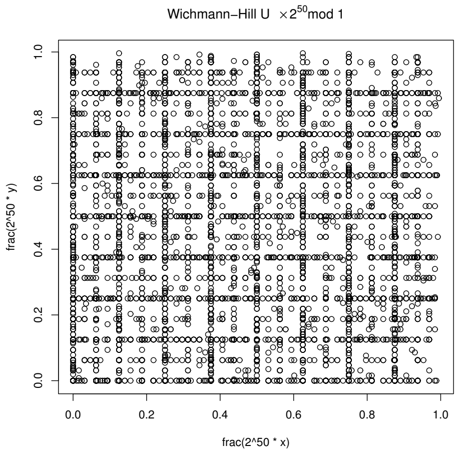
Testing Generators

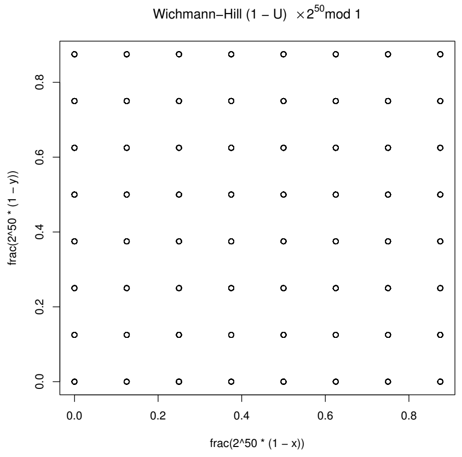
- All generators have flaws; some are known, some are not (yet).
- Tests need to look for flaws that are likely to be important in realistic statistical applications.
- Theoretical tests look for
  - bad lattice structure
  - lack of k-distribution
  - other tractable properties
- Statistical tests look for simple simulations where pseudo-random number streams produce results unreasonably far from known answers.
- Some batteries of tests:
  - DIEHARD `http://stat.fsu.edu/pub/diehard/`
  - DIEHARDER `http://www.phy.duke.edu/~rgb/General/dieharder.php`
  - NIST Test Suite `http://csrc.nist.gov/groups/ST/toolkit/rng/`
  - TestU01 `http://www.iro.umontreal.ca/~lecuyer`

Issues and Recommendations

- Good choices of generators change with time and technology.
  - Faster computers need longer periods.
  - Parallel computers need different methods.
- All generators are flawed
  - Bad simulation results due to poor random number generators are very rare; coding errors in simulations are not.
  - Testing a generator on a “similar” problem with known answers is a good idea (and may be useful to make results more accurate).
  - Using multiple generators is a good idea; R makes this easy to do.
  - Be aware that some generators can produce uniforms equal to 0 or 1 (I believe R’s will not).
  - Avoid methods that are sensitive to low order bits


Mersenne−Twister $\times 2^{30}$
frac(2^30 * y)
frac(2^30 * x)


Wichmann−Hill U $\times 2^{50}$mod 1
frac(2^50 * y)
frac(2^50 * x)


Wichmann−Hill (1 − U) $\times 2^{50}$mod 1
frac(2^50 * (1 − y))
frac(2^50 * (1 − x))

Non-Uniform Random Variate Generation

- Starting point: Assume we can generate a sequence of independent uniform $[0,1]$ random variables.
- Develop methods that generate general random variables from uniform ones.
- Considerations:
  - Simplicity, correctness
  - Accuracy, numerical issues
  - Speed
    - Setup
    - Generation
- General approaches:
  - Univariate transformations
  - Multivariate transformations
  - Mixtures
  - Accept/Reject methods

Inverse CDF Method

Suppose F is a cumulative distribution function (CDF). Define the inverse CDF as

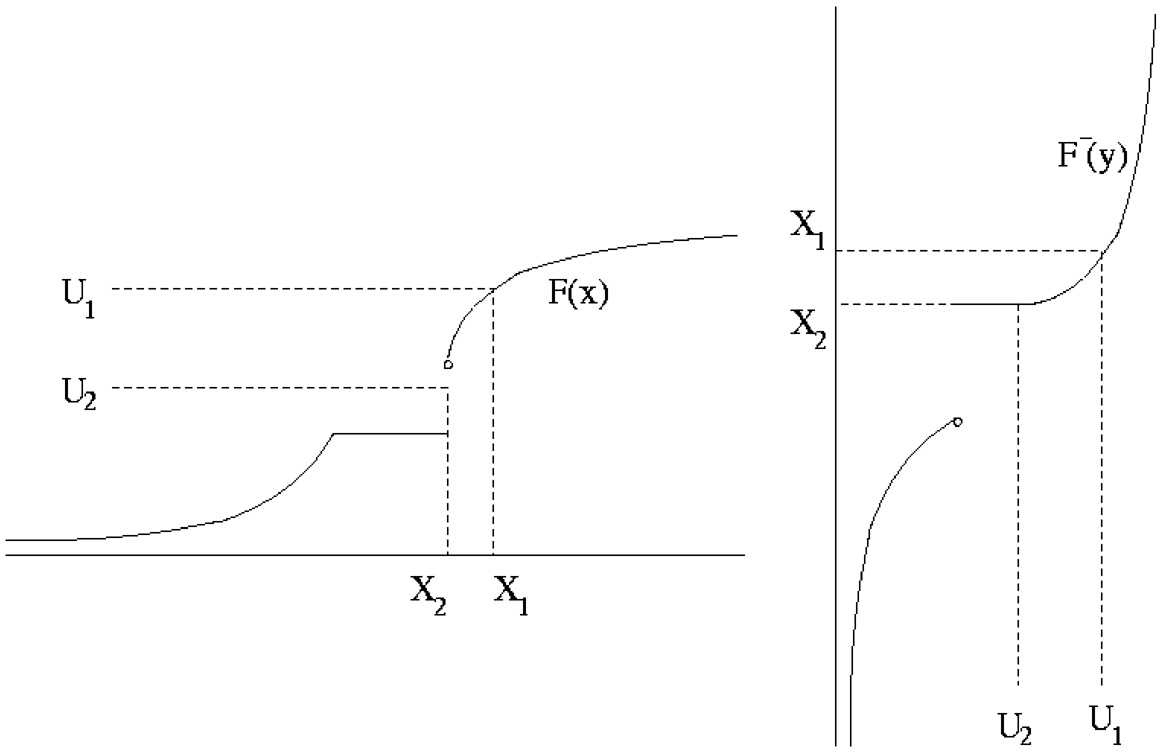
$$F^{-}(u) = \min\{x : F(x) \geq u\}$$

If $U \sim \mathrm{U}[0,1]$ then $X = F^{-}(U)$ has CDF F.

Proof. Since F is right continuous, the minimum is attained. Therefore $F(F^{-}(u)) \geq u$ and $F^{-}(F(x)) = \min\{y : F(y) \geq F(x)\}$. So

$$\{(u,x) : F^{-}(u) \leq x\} = \{(u,x) : u \leq F(x)\}$$

and thus $P(X \leq x) = P(F^{-}(U) \leq x) = P(U \leq F(x)) = F(x)$. □



Example: Unit Exponential Distribution

The unit exponential CDF is

$$F(x) = \begin{cases} 1 - e^{-x} & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

with inverse CDF

$$F^{-}(u) = -\log(1-u)$$

So $X = -\log(1-U)$ has an exponential distribution.

Since $1-U \sim \text{U}[0,1]$, $-\log U$ also has a unit exponential distribution.

If the uniform generator can produce 0, then these should be rejected.

Example: Standard Cauchy Distribution

The CDF of the standard Cauchy distribution is

$$F(x) = \frac{1}{2} + \frac{1}{\pi}\arctan(x)$$

with inverse CDF

$$F^{-}(u) = \tan(\pi(u - 1/2))$$

So $X = \tan(\pi(U - 1/2))$ has a standard Cauchy distribution.

An alternative form is: Let U_1, U_2 be independent $\text{U}[0,1]$ random variables and set

$$X = \begin{cases} \tan(\pi(U_2/2) & \text{if } U_1 \geq 1/2 \\ -\tan(\pi(U_2/2) & \text{if } U_1 < 1/2 \end{cases}$$

- U_1 produces a random sign
- U_2 produces the magnitude
- This will preserve fine structure of U_2 near zero, if there is any.

Example: Standard Normal Distribution

The CDF of the standard normal distribution is

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-z^2/2} dz$$

and the inverse CDF is Φ^{-1}.

- Neither Φ nor Φ^{-1} are available in closed form.
- Excellent numerical routines are available for both.
- Inversion is currently the default method for generating standard normals in R.
- The inversion approach uses two uniforms to generate one higher-precision uniform via the code

```
    case INVERSION:
#define BIG 134217728 /* 2^27 */
        /* unif_rand() alone is not of high enough precision */
        u1 = unif_rand();
        u1 = (int)(BIG*u1) + unif_rand();
        return qnorm5(u1/BIG, 0.0, 1.0, 1, 0);
```

Example: Geometric Distribution

The geometric distribution with PMF $f(x) = p(1-p)^x$ for $x = 0, 1, \ldots$, has CDF

$$F(x) = \begin{cases} 1-(1-p)^{\lfloor x+1 \rfloor} & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases}$$

where $\lfloor y \rfloor$ is the integer part of y. The inverse CDF is

$$\begin{aligned} F^{-}(u) &= \lceil \log(1-u)/\log(1-p) \rceil - 1 \\ &= \lfloor \log(1-u)/\log(1-p) \rfloor \qquad \text{except at the jumps} \end{aligned}$$

for $0 < u < 1$. So $X = \lfloor \log(1-U)/\log(1-p) \rfloor$ has a geometric distribution with success probability p.

Other possibilities:

$$X = \lfloor \log(U)/\log(1-p) \rfloor$$

or

$$X = \lfloor -Y/\log(1-p) \rfloor$$

where Y is a unit exponential random variable.

Example: Truncated Normal Distribution

Suppose $X \sim \mathrm{N}(\mu, 1)$ and

$$y \sim X | X > 0.$$

The CDF of Y is

$$F_Y(y) = \begin{cases} \frac{P(0<X\leq y)}{P(0<X)} & \text{for } y \geq 0 \\ 0 & \text{for } y < 0 \end{cases} = \begin{cases} \frac{F_X(y)-F_X(0)}{1-F_X(0)} & \text{for } y \geq 0 \\ 0 & \text{for } y < 0 \end{cases}.$$

The inverse CDF is

$$F_Y^{-1}(u) = F_X^{-1}(u(1-F_X(0))+F_X(0)) = F_X^{-1}(u+(1-u)F_X(0)).$$

An R function corresponding to this definition is

```
Q1 <- function(p, m) qnorm(p + (1 - p)* pnorm(0, m), m)
```

This seems to work well for positive μ but not for negative values far from zero:

```
> Q1(0.5, c(1, 3, 5, 10, -10))
[1]  1.200174  3.001692  5.000000 10.000000       Inf
```

The reason is that `pnorm(0, -10)` is rounded to one.

A mathematically equivalent formulation of the inverse CDF is

$$F_Y^{-1}(u) = F_X^{-1}(1-(1-u)(1-F_X(0)))$$

which leads to

```
Q2 <- function(p, m)
    qnorm((1 - p)* pnorm(0, m, lower.tail = FALSE),
          m, lower.tail = FALSE)
```

and

```
> Q2(0.5, c(1, 3, 5, 10, -10))
[1]  1.20017369  3.00169185  5.00000036 10.00000000  0.06841184
```

Issues

- In principle, inversion can be used for any distribution.
- Sometimes routines are available for F^- but are quite expensive:

  ```
  > system.time(rbeta(1000000, 2.5, 3.5))
     user  system elapsed
    0.206   0.000   0.211
  > system.time(qbeta(runif(1000000), 2.5, 3.5))
     user  system elapsed
    4.139   0.001   4.212
  ```

  `rbeta` is about 20 times faster than inversion.
- If F^- is not available but F is, then one can solve the equation $u = F(x)$ numerically for x.
- Accuracy of F or F^- may be an issue, especially when writing code for a parametric family that is to work well over a wide parameter range.
- Even when inversion is costly,
  - the cost of random variate generation may be a small fraction of the total cost of a simulation
  - using inversion creates a simple relation between the variables and the underlying uniforms that may be useful

Multivariate Transformations

Many distributions can be expressed as the marginal distribution of a function of several variables.

Box-Muller Method for the Standard Normal Distribution

Suppose X_1 and X_2 are independent standard normals. The polar coordinates θ and R are independent,

- θ is uniform on $[0, 2\pi)$
- R^2 is χ^2_2, which is exponential with mean 2

So if U_1 and U_2 are independent and uniform on $[0, 1]$, then

$$\begin{aligned} X_1 &= \sqrt{-2\log U_1}\cos(2\pi U_2) \\ X_2 &= \sqrt{-2\log U_1}\sin(2\pi U_2) \end{aligned}$$

are independent standard normals. This is the *Box-Muller method.*

Polar Method for the Standard Normal Distribution

The trigonometric functions are somewhat slow to compute. Suppose (V_1, V_2) is uniform on the unit disk

$$\{(v_1, v_2) : v_1^2 + v_2^2 \leq 1\}$$

Let $R^2 = V_1^2 + V_2^2$ and

$$\begin{aligned} X_1 &= V_1 \sqrt{-(2 \log R^2)/R^2} \\ X_2 &= V_2 \sqrt{-(2 \log R^2)/R^2} \end{aligned}$$

Then X_1, X_2 are independent standard normals.

This is the *polar method* of Marsaglia and Bray.

Generating points uniformly on the unit disk can be done using *rejection sampling*, or *accept/reject sampling:*

```
**repeat**
    generate independent $V_1, V_2 \sim \mathrm{U}(-1,1)$
**until** $V_1^2 + V_2^2 \leq 1$
**return** $(V_1, V_2)$
```

- This independently generates pairs (V_1, V_2) uniformly on the square $(-1,1) \times (-1,1)$ until the result is inside the unit disk.
- The resulting pair is uniformly distributed on the unit disk.
- The number of pairs that need to be generated is a geometric variable with success probability

  $$p = \frac{\text{area of disk}}{\text{area of square}} = \frac{\pi}{4}$$

  The expected number of generations needed is $1/p = 4/\pi = 1.2732$.
- The number of generations needed is independent of the final pair.

Polar Method for the Standard Cauchy Distribution

The ratio of two standard normals has a Cauchy distribution.

Suppose two standard normals are generated by the polar method,

$$X_1 = V_1\sqrt{-(2\log R^2)/R^2}$$
$$X_2 = V_2\sqrt{-(2\log R^2)/R^2}$$

with $R^2 = V_1^2 + V_2^2$ and (V_1, V_2) uniform on the unit disk. Then

$$Y = \frac{X_1}{X_2} = \frac{V_1}{V_2}$$

is the ratio of the two coordinates of the pair that is uniformly distributed on the unit disk.

This idea leads to a general method, the *Ratio-of-Uniforms method*.

Student's t Distribution

Suppose

- Z has a standard normal distribution,
- Y has a χ^2_ν distribution,
- Z and Y are independent.

Then

$$X = \frac{Z}{\sqrt{Y/\nu}}$$

has a t distribution with ν degrees of freedom.

To use this representation we will need to be able to generate from a χ^2_ν distribution, which is a Gamma$(\nu/2, 2)$ distribution.

Beta Distribution

Suppose $\alpha > 0$, $\beta > 0$, and

- $U \sim \text{Gamma}(\alpha, 1)$
- $V \sim \text{Gamma}(\beta, 1)$
- U, V are independent

Then

$$X = \frac{U}{U+V}$$

has a Beta(α, β) distribution.

F Distribution

Suppose $a > 0$, $b > 0$, and

- $U \sim \chi^2_a$
- $V \sim \chi^2_b$
- U, V are independent

Then

$$X = \frac{U/a}{V/b}$$

has an F distribution with a and b degrees of freedom.

Alternatively, if $Y \sim \text{Beta}(a/2, b/2)$, then

$$X = \frac{b}{a}\frac{Y}{1-Y}$$

has an F distribution with a and b degrees of freedom.

Non-Central t Distribution

Suppose

- $Z \sim \mathrm{N}(\mu, 1)$,
- $Y \sim \chi^2_\nu$,
- Z and Y are independent.

Then

$$X = \frac{Z}{\sqrt{Y/\nu}}$$

has non-central t distribution with ν degrees of freedom and non-centrality parameter μ.

Non-Central Chi-Square, and F Distributions

Suppose

- $Z_1, \dots, Z_k$ are independent
- $Z_i \sim \mathrm{N}(\mu_i, 1)$

Then

$$Y = Z_1^2 + \cdots + Z_k^2$$

has a non-central chi-square distribution with k degrees of freedom and non-centrality parameter

$$\delta = \mu_1^2 + \cdots + \mu_k^2$$

An alternative characterization: if $\widetilde{Z}_1, \dots, \widetilde{Z}_k$ are independent standard normals then

$$Y = (\widetilde{Z}_1 + \sqrt{\delta})^2 + \widetilde{Z}_2^2 \cdots + \widetilde{Z}_k^2 = (\widetilde{Z}_1 + \sqrt{\delta})^2 + \sum_{i=2}^{k} \widetilde{Z}_i^2$$

has a non-central chi-square distribution with k degrees of freedom and non-centrality parameter δ.

The non-central F is of the form

$$X = \frac{U/a}{V/b}$$

where U, V are independent, U is a non-central χ_a^2 and V is a central χ_b^2 random variable.

Bernoulli and Binomial Distributions

Suppose $p \in [0,1]$, $U \sim \text{U}[0,1]$, and

$$X = \begin{cases} 1 & \text{if } U \leq p \\ 0 & \text{otherwise} \end{cases}$$

Then X bas a Bernoulli(p) distribution.

If $Y_1, \dots, Y_n$ are independent Bernoulli(p) random variables, then

$$X = Y_1 + \cdots + Y_n$$

has a Binomial(n, p) distribution.

For small n this is an effective way to generate binomials.

Mixtures and Conditioning

Many distributions can be expressed using a hierarchical structure:

$$\begin{aligned} X|Y &\sim f_{X|Y}(x|y) \\ Y &\sim f_Y(y) \end{aligned}$$

The marginal distribution of X is called a *mixture distribution*. We can generate X by

Generate Y from $f_Y(y)$.
Generate $X|Y = y$ from $f_{X|Y}(x,y)$.

Student's t Distribution

Another way to think of the t_ν distribution is:

$$\begin{aligned} X|Y &\sim \mathrm{N}(0, \nu/Y) \\ Y &\sim \chi^2_\nu \end{aligned}$$

The t distribution is a *scale mixture of normals*.

Other choices of the distribution of Y lead to other distributions for X.

Negative Binomial Distribution

The negative binomial distribution with PMF

$$f(x) = \binom{x+r-1}{r-1} p^r (1-p)^x$$

for $x = 0, 1, 2, \ldots$, can be written as a gamma mixture of Poissons: if

$$\begin{aligned} X|Y &\sim \text{Poisson}(Y) \\ Y &\sim \text{Gamma}(r, (1-p)/p) \end{aligned}$$

then $X \sim$ Negative Binomial(r, p).

[The notation Gamma(α, β) means that β is the scale parameter.]

This representation makes sense even when r is not an integer.

Non-Central Chi-Square

The density of the non-central χ^2_ν distribution with non-centrality parameter δ is

$$f(x) = e^{-\delta/2} \sum_{i=0}^{\infty} \frac{(\delta/2)^i}{i!} f_{\nu+2i}(x)$$

where $f_k(x)$ central χ^2_k density. This is a Poisson-weighted average of χ^2 densities, so if

$$\begin{aligned} X|Y &\sim \chi^2_{\nu+2Y} \\ Y &\sim \text{Poisson}(\delta/2) \end{aligned}$$

then X has a non-central χ^2_ν distribution with non-centrality parameter δ.

Composition Method

Suppose we want to sample from the density

$$f(x) = \begin{cases} x/2 & 0 \le x < 1 \\ 1/2 & 1 \le x < 2 \\ 3/2 - x/2 & 2 \le x \le 3 \\ 0 & \text{otherwise} \end{cases}$$

We can write f as the mixture

$$f(x) = \frac{1}{4} f_1(x) + \frac{1}{2} f_2(x) + \frac{1}{4} f_3(x)$$

with

$$\begin{aligned} f_1(x) &= 2x && 0 \le x < 1 \\ f_2(x) &= 1 && 1 \le x < 2 \\ f_3(x) &= 6 - 2x && 2 \le x \le 3 \end{aligned}$$

and $f_i(x) = 0$ for other values of x.

Generating from the f_i is straight forward. So we can sample from f using:

```
Generate $I$ from $\{1,2,3\}$ with probabilities $1/4, 1/2, 1/4$.
Generate $X$ from $f_I(x)$ by inversion.
```

This approach can be used in conjunction with other methods.

One example: The polar method requires sampling uniformly from the unit disk. This can be done by

- encloseing the unit disk in a regular hexagon
- using composition to sample uniformly from the hexagon until the result is in the unit disk.

Alias Method

Suppose $f(x)$ is a probability mass function on $\{1,2,\ldots,k\}$. Then $f(x)$ can be written as

$$f(x) = \sum_{i=1}^{k} \frac{1}{k} f_i(x)$$

where

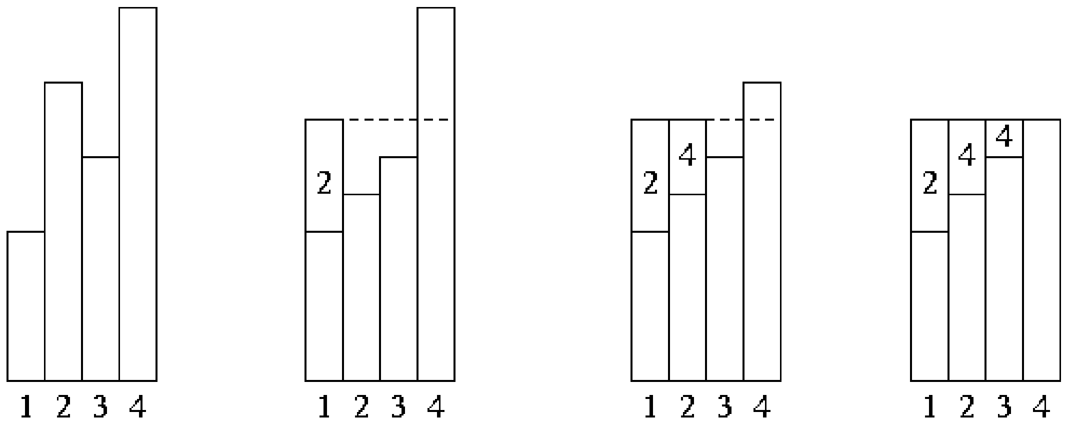
$$f_i(x) = \begin{cases} q_i & x = i \\ 1 - q_i & x = a_i \\ 0 & \text{otherwise} \end{cases}$$

for some $q_i \in [0,1]$ and some $a_i \in \{1,2,\ldots,k\}$.

Once values for q_i and a_i have been found, generation is easy:

```
Generate $I$ uniform on $\{1,\ldots,k\}$
Generate $U$ uniform on $[0,1]$
if $U \leq q_I$
    return $I$
else
    return $a_I$
```

The setup process used to compute the q_i and a_i is called *leveling the histogram*:



This is *Walker's alias method*.

A complete description is in Ripley (1987, Alg 3.13B).

The alias method is an example of trading off a setup cost for fast generation.

The alias method is used by the `sample` function for unequal probability sampling with replacement when there are enough reasonably probable values.

`https://svn.r-project.org/R/trunk/src/main/random.c`

Accept/Reject Methods

Sampling Uniformly from the Area Under a Density

Suppose h is a function such that

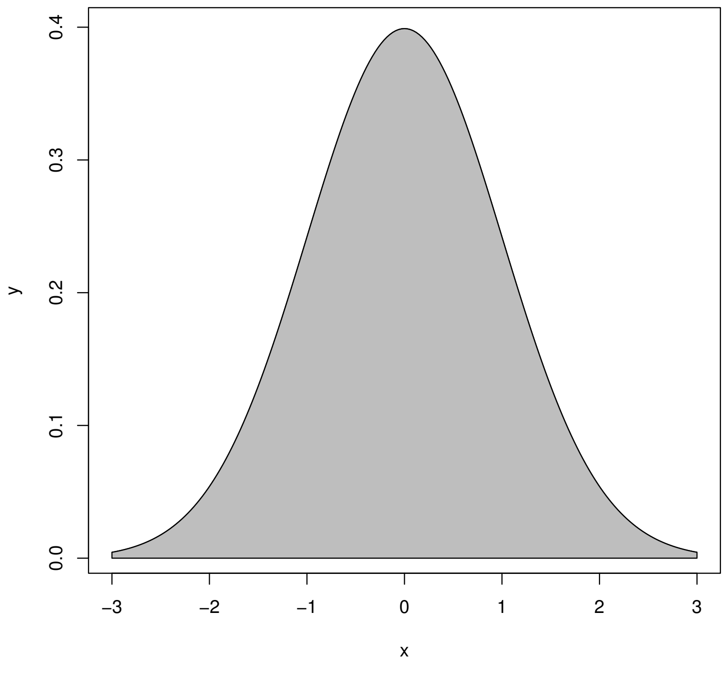
- $h(x) \geq 0$ for all x
- $\int h(x)dx < \infty$.

Let

$$\mathscr{G}_h = \{(x,y) : 0 < y \leq h(x)\}$$

The area of $\mathscr{G}_h$ is

$$|\mathscr{G}_h| = \int h(x)dx < \infty$$



Suppose (X,Y) is uniformly distributed on $\mathscr{G}_h$. Then

- The conditional distribution of $Y|X = x$ is uniform on $(0, h(x))$.

- The marginal distribution of X has density $f_X(x) = h(x)/\int h(y)dy$:

$$f_X(x) = \int_0^{h(x)} \frac{1}{|\mathscr{G}_h|} dy = \frac{h(x)}{\int h(y)dy}$$

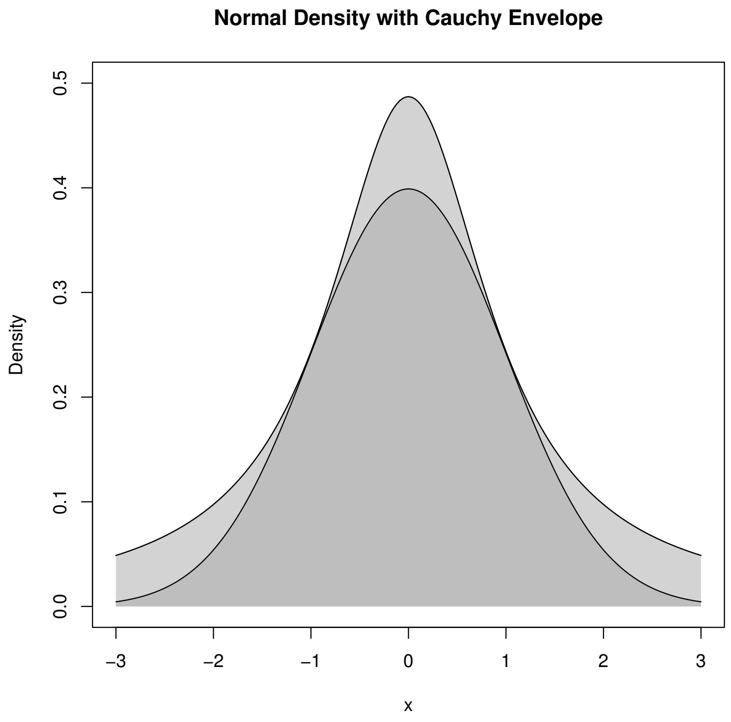
Rejection Sampling Using an Envelope Density

Suppose g is a density and $M > 0$ is a real number such that

$$h(x) \leq Mg(x) \quad \text{for all } x$$

or, equivalently,

$$\sup \frac{h(x)}{g(x)} \leq M \quad \text{for all } x$$



$Mg(x)$ is an *envelope* for $h(x)$.

Suppose

- we want to sample from a density proportional to h
- we can find a density g and a constant M such that
  - $Mg(x)$ is an envelope for $h(x)$
  - it is easy to sample from g

Then

- we can sample X from g and $Y|X = x$ from $\mathrm{U}(0, Mg(x))$ to get a pair (X,Y) uniformly distributed on $\mathscr{G}_{Mg}$
- we can repeat until the pair (X,Y) satisfies $Y \leq h(X)$
- the resulting pair (X,Y) is uniformly distributed on $\mathscr{G}_h$
- so the marginal density of the resulting X is $f_X(x) = h(x)/\int h(y)dy$.
- the number of draws from the uniform distribution on $\mathscr{G}_{Mg}$ needed until we obtain a pair in $\mathscr{G}_h$ is independent of the final pair
- the number of draws has a geometric distribution with success probability
$$p = \frac{|\mathscr{G}_h|}{|\mathscr{G}_{Mg}|} = \frac{\int h(y)dy}{M\int g(y)dy} = \frac{\int h(y)dy}{M}$$
since g is a probability density. p is the *acceptance probability*.
- the expected number of draws needed is
$$E[\text{number of draws}] = \frac{1}{p} = \frac{M\int g(y)dy}{\int h(y)dy} = \frac{M}{\int h(y)dy}$$
- if h is also a proper density, then $p = 1/M$ and
$$E[\text{number of draws}] = \frac{1}{p} = M$$

The Basic Algorithm

The rejection, or accept/reject, sampling algorithm:

```
**repeat**
    generate independent $X \sim g$ and $U \sim \mathrm{U}[0,1]$
**until** $UMg(X) \leq h(X)$
**return** $X$
```

Alternate forms of the test:

$$U \leq \frac{h(X)}{Mg(X)}$$
$$\log(U) \leq \log(h(X)) - \log(M) - \log(g(X))$$

Care may be needed to ensure numerical stability.

Example: Normal Distribution with Cauchy Envelope

Suppose

- $h(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ is the standard normal density
- $g(x) = \frac{1}{\pi(1+x^2)}$ is the standard Cauchy density

Then

$$\frac{h(x)}{g(x)} = \sqrt{\frac{\pi}{2}}(1+x^2)e^{-x^2/2} \leq \sqrt{\frac{\pi}{2}}(1+1^2)e^{-1^2/2} = \sqrt{2\pi e^{-1}} = 1.520347$$

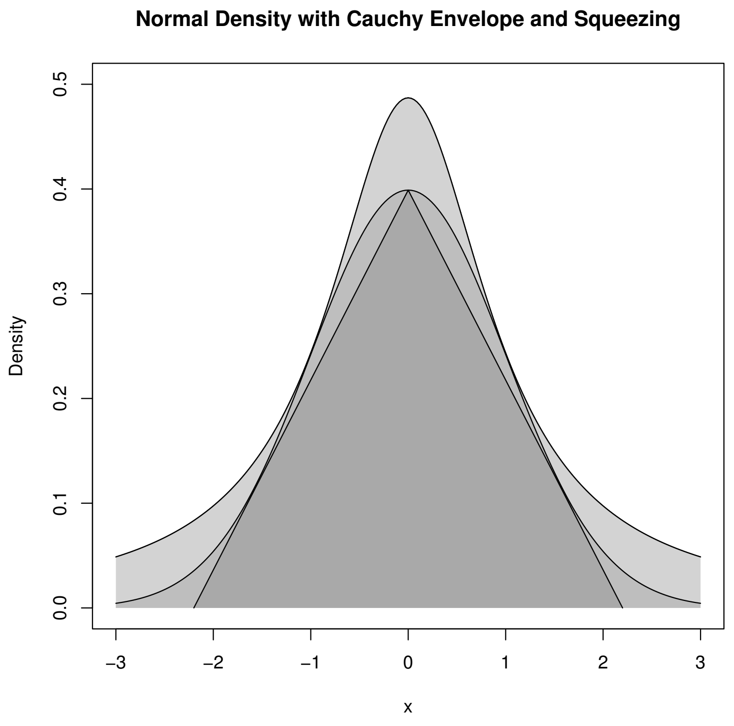
The resulting accept/reject algorithm is

```
**repeat**
    generate independent standard Cauchy $X$ and $U \sim \mathrm{U}[0,1]$
**until** $U \leq \frac{e^{1/2}}{2}(1+X^2)e^{-X^2/2}$
**return** $X$
```

Squeezing

Performance can be improved by *squeezing*:

- Accept if point is inside the triangle:



- Squeezing *can* speed up generation.
- Squeezing *will* complicate the code (making errors more likely).

Rejection Sampling for Discrete Distributions

For simplicity, just consider integer valued random variables.

- If h and g are probability mass functions on the integers and $h(x)/g(x)$ is bounded, then the same algorithm can be used.
- If p is a probability mass function on the integers then

  $$h(x) = p(\lfloor x \rfloor)$$

  is a probability density.

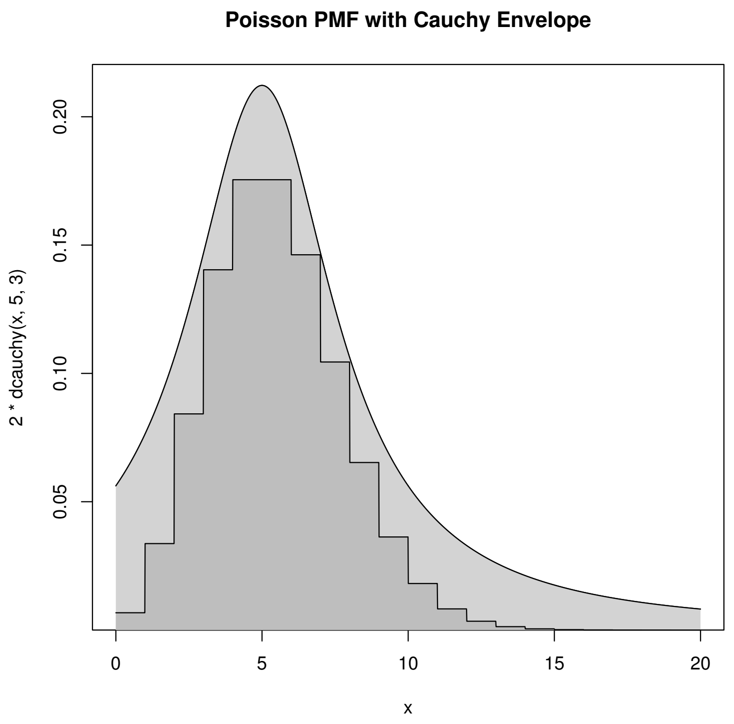
  If X has density h, then $Y = \lfloor X \rfloor$ has PMF p.

Example: Poisson Distribution with Cauchy Envelope

Suppose

- p is the PMF of a Poisson distribution with mean 5
- g is the Cauchy density with location 5 and scale 3.
- $h(x) = p(\lfloor x \rfloor)$

Then, by careful analysis or graphical examination, $h(x) \leq 2g(x)$ for all x.



Comments

- The Cauchy density is often a useful envelope.
- More efficient choices are often possible.
- Location and scale need to be chosen appropriately.
- If the target distribution is non-negative, a truncated Cauchy can be used.
- Careful analysis is needed to produce generators for a parametric family (e.g. all Poisson distributions).
- Graphical examination can be very helpful in guiding the analysis.
- Carefully tuned envelopes combined with squeezing can produce very efficient samplers.
- Errors in tuning and squeezing will produce garbage.

Ratio-of-Uniforms Method

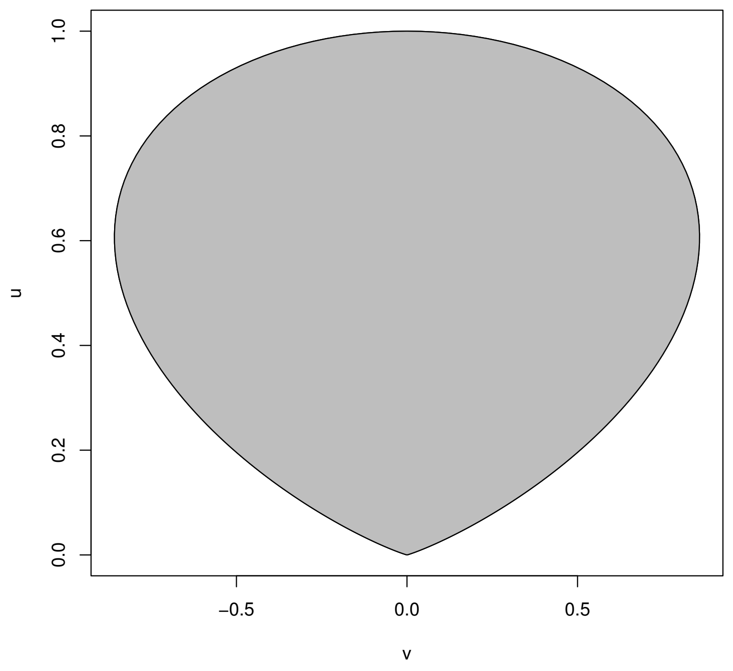
Basic Method

- Introduced by Kinderman and Monahan (1977).
- Suppose
  - $h(x) \geq 0$ for all x
  - $\int h(x)dx < \infty$
- Let (V,U) be uniform on

  $$\mathscr{C}_h = \{(v,u) : 0 < u \leq \sqrt{h(v/u)}\}$$

  Then $X = V/U$ has density $f(x) = h(x)/\int h(y)dy$.

- For $h(x) = e^{-x^2/2}$ the region $\mathscr{C}_h$ looks like



  - The region is bounded.
  - The region is convex.

Properties

- The region $\mathscr{C}_h$ is convex if h is log concave.
- The region $\mathscr{C}_h$ is bounded if $h(x)$ and $x^2h(x)$ are bounded.
- Let

$$\begin{aligned} u^* &= \max_x \sqrt{h(x)} \\ v_-^* &= \min_x x\sqrt{h(x)} \\ v_+^* &= \max_x x\sqrt{h(x)} \end{aligned}$$

  Then $\mathscr{C}_h$ is contained in the rectangle $[v_-^*, v_+^*] \times [0, u^*]$.

- The simple Ratio-of-Uniforms algorithm based on rejection sampling from the enclosing rectangle is

```
repeat
    generate $U \sim \mathrm{U}[0, u^*]$
    generate $V \sim \mathrm{U}[v_-^*, v_+^*]$
until $U^2 \leq h(V/U)$
return $X = V/U$
```

- If $h = e^{-x^2/2}$ then

$$\begin{aligned} u^* &= 1 \\ v_-^* &= -\sqrt{2e^{-1}} \\ v_+^* &= \sqrt{2e^{-1}} \end{aligned}$$

  and the expected number of draws is

$$\frac{\text{area of rectangle}}{\text{area of } \mathscr{C}_h} = \frac{u^*(v_+^* - v_-^*)}{\frac{1}{2}\int h(x)dx} = \frac{2\sqrt{2e^{-1}}}{\sqrt{\pi/2}} = 1.368793$$

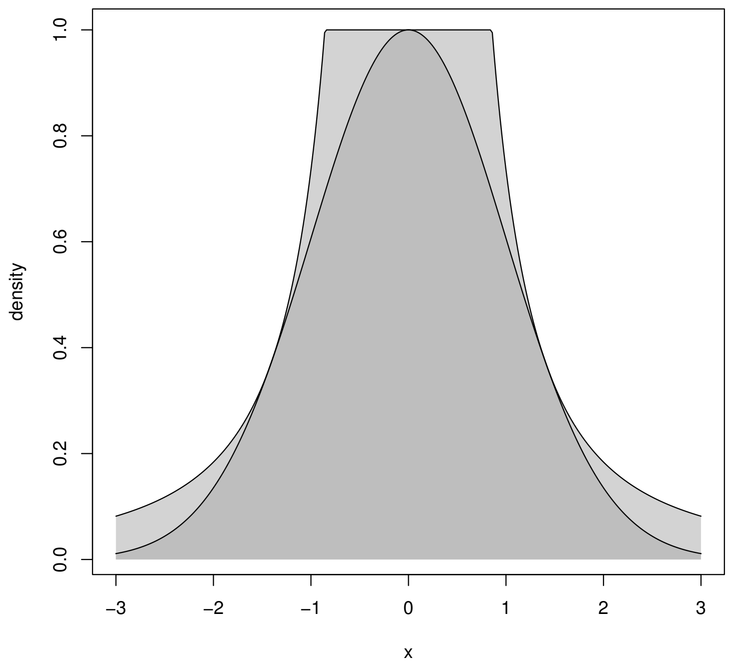
- Various squeezing methods are possible.
- Other approaches to sampling from $\mathscr{C}_h$ are also possible.

Relation to Rejection Sampling

Ratio of Uniforms with rejection sampling from the enclosing rectangle is equivalent to ordinary rejection sampling using an envelope density

$$g(x) \propto \begin{cases} \left(\frac{v_-^*}{x}\right)^2 & \text{if } x < v_-^*/u^* \\ (u^*)^2 & \text{if } v_-^*/u^* \le x \le v_+^*/u^* \\ \left(\frac{v_+^*}{x}\right)^2 & \text{if } x > v_+^*/u^* \end{cases}$$

This is sometimes called a *table mountain density*



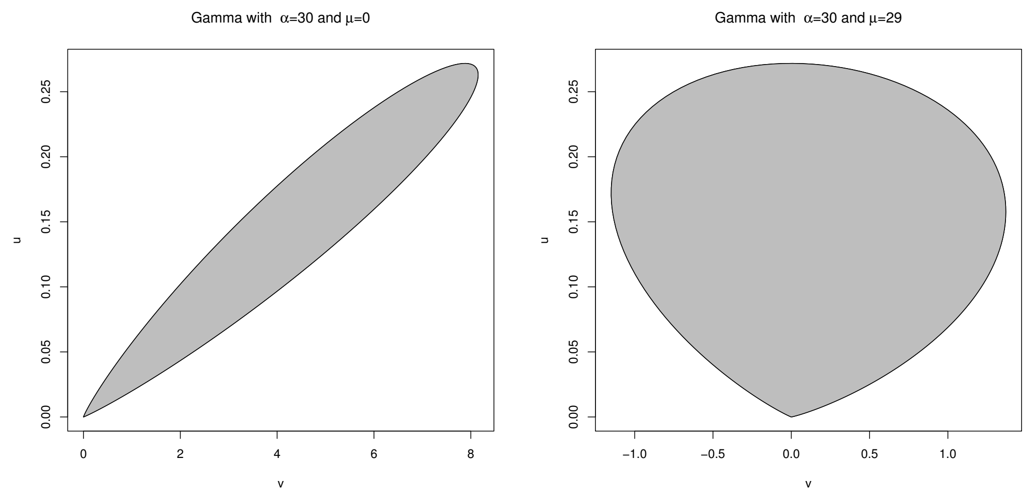
Generalizations

A more general form of the basic result: For any μ and any $r > 0$ let

$$\mathscr{C}_{h,\mu,r} = \{(v,u) : 0 < u \leq h(v/u^r + \mu)^{1/(r+1)}\}$$

If (U,V) is uniform on $\mathscr{C}_{h,\mu,r}$, then $X = V/U^r + \mu$ has density $f(x) = h(x)/\int h(y)dy$.

- μ and r can be chosen to minimize the rejection probability.
- $r = 1$ seems adequate for most purposes.
- Choosing μ equal to the mode of h can help.
- For the Gamma distribution with $\alpha = 30$,



Adaptive Rejection Sampling

First introduced by Gilks and Wild (1992).

Convexity

- A set C is convex if $\lambda x + (1-\lambda)y \in C$ for all $x, y \in C$ and $\lambda \in [0,1]$.
- C can be a subset or $\mathbb{R}$, or $\mathbb{R}^n$, or any other set where the *convex combination*

  $$\lambda x + (1-\lambda)y$$

  makes sense.
- A real-valued function f on a convex set C is convex if

  $$f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda)f(y)$$

  $x, y \in C$ and $\lambda \in [0,1]$.
- $f(x)$ is concave if $-f(x)$ is convex, i.e. if

  $$f(\lambda x + (1-\lambda)y) \ge \lambda f(x) + (1-\lambda)f(y)$$

  $x, y \in C$ and $\lambda \in [0,1]$.
- A concave function is always below its tangent.

Log Concave Densities

- A density f is *log concave* if $\log f$ is a concave function
- Many densities are log concave:
  - normal densities
  - Gamma(α, β) with $\alpha \geq 1$
  - Beta(α, β) with $\alpha \geq 1$ and $\beta \geq 1$.
- Some are not but may be related to ones that are: The t densities are not, but if

$$\begin{aligned} X|Y = y &\sim \mathrm{N}(0, 1/y) \\ Y &\sim \mathrm{Gamma}(\alpha, \beta) \end{aligned}$$

  then

  - the marginal distribution of X is t for suitable choice of β
  - and the joint distribution of X and Y has density

$$f(x,y) \propto \sqrt{y} e^{-\frac{y}{2}x^2} y^{\alpha-1} e^{-y/\beta} = y^{\alpha-1/2} e^{-y(\beta+x^2/2)}$$

    which is log concave for $\alpha \geq 1/2$

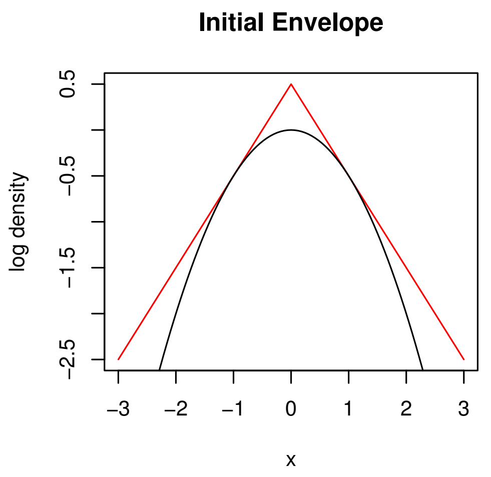
Tangent Approach

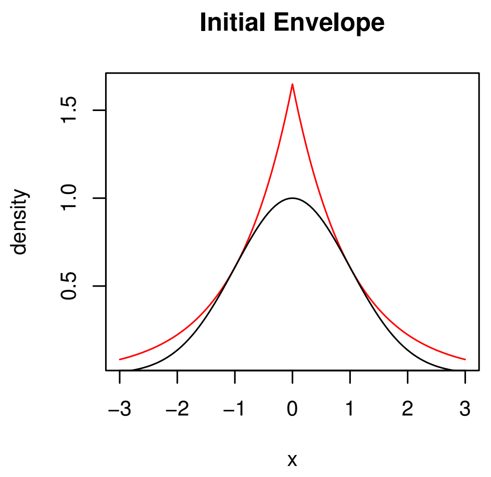
Suppose

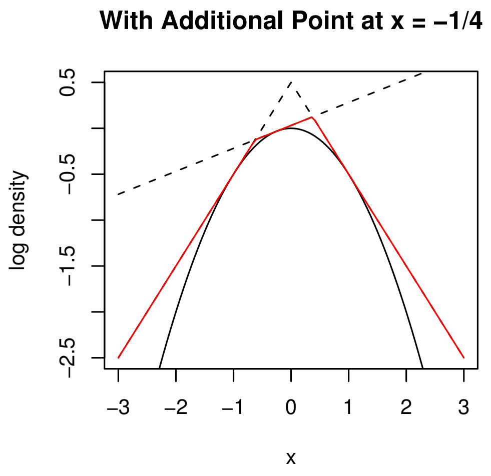
- f is log concave
- f has an interior mode

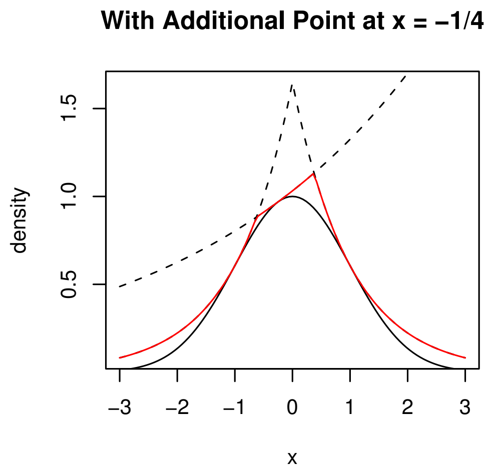
Need log density, derivative at two points, one each side of the mode

- piece-wise linear envelope of log density
- piece-wise exponential envelope of density
- if first point is not accepted, can use to make better envelope

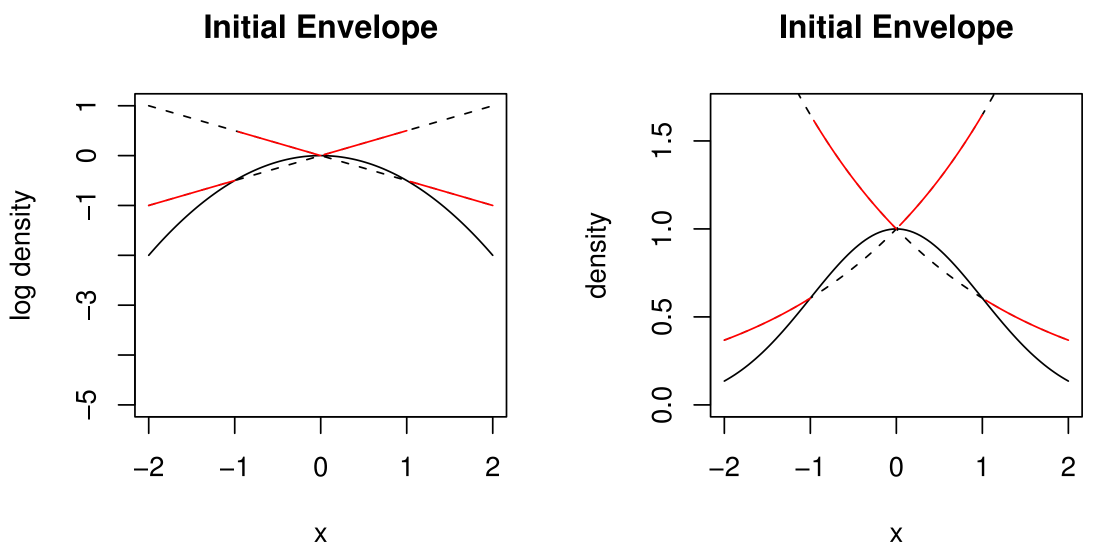








Secant Approach



- Need three points to start
- Do not need derivatives
- Get larger rejection rates
- Both approaches need numerical care

Notes and Comments

- Many methods depend on properties of a particular distribution.
- Inversion is one general method that can often be used.
- Other general-purpose methods are
  - rejection sampling
  - adaptive rejection sampling
  - ratio-of-uniforms
- Some references:
  - Devroye, L. (1986). *Non-Uniform Random Variate Generation*, Springer-Verlag, New York.
  - Gentle, J. E. (2003). *Random Number Generation and Monte Carlo Methods*, Springer-Verlag, New York.
  - Hörmann, W., Leydold, J., and Derflinger, G. (2004). *Automatic Nonuniform Random Variate Generation*, Springer-Verlag, New York.

A Recent Publication

Karney, C.F.F. (2016). “Sampling Exactly from the Normal Distribution.” *ACM Transactions on Mathematical Software* 42 (1).

Random Variate Generators in R

- Generators for most standard distributions are available
  - `rnorm`: normal
  - `rgamma`: gamma
  - `rt`: *t*
  - `rpois`: Poisson
  - etc.
- Most use standard algorithms from the literature.
- Source code is in `src/nmath/` in the source tree,

  `https://svn.r-project.org/R/trunk`

- The normal generator can be configured by `RNGkind`. Options are
  - Kinderman-Ramage
  - Buggy Kinderman-Ramage (available for reproducing results)
  - Ahrens-Dieter
  - Box-Muller
  - Inversion (the current default)
  - user-supplied

Generating Random Vectors and Matrices

- Sometimes generating random vectors can be reduced to a series of univariate generations.
- One approach is conditioning:

  $$f(x,y,z) = f_{Z|X,Y}(z|x,y) f_{Y|X}(y|x) f_X(x)$$

  So we can generate

  - X from $f_X(x)$
  - $Y|X=x$ from $f_{Y|X}(y|x)$
  - $Z|X=x, Y=y$ from $f_{Z|X,Y}(z|x,y)$
- One example: $(X_1, X_2, X_3) \sim \text{Multinomial}(n, p_1, p_2, p_3)$ Then

  $$\begin{aligned} X_1 &\sim \text{Binomial}(n, p_1) \\ X_2|X_1 = x_1 &\sim \text{Binomial}(n - x_1, p_2/(p_2+p_3)) \\ X_3|X_1 = x_1, X_2 = x_2 &= n - x_1 - x_2 \end{aligned}$$

- Another example: X, Y bivariate normal $(\mu_X, \mu_Y, \sigma_X^2, \sigma_Y^2, \rho)$. Then

  $$\begin{aligned} X &\sim \text{N}(\mu_X, \sigma_X^2) \\ Y|X = x &\sim \text{N}\left(\mu_Y + \rho\frac{\sigma_Y}{\sigma_X}(x - \mu_X), \sigma_Y^2(1-\rho^2)\right) \end{aligned}$$

- For some distributions special methods are available.
- Some general methods extend to multiple dimensions.

Multivariate Normal Distribution

- Marginal and conditional distributions are normal; conditioning can be used in general.
- Alternative: use linear transformations.

  Suppose $Z_1,\dots,Z_d$ are independent standard normals, $\mu_1,\dots\mu_d$ are constants, and A is a constant $d \times d$ matrix. Let

  $$Z = \begin{bmatrix} Z_1 \\ \vdots \\ Z_d \end{bmatrix} \qquad \mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_d \end{bmatrix}$$

  and set

  $$X = \mu + AZ$$

  Then X is multivariate normal with mean vector μ and covariance matrix AA^T,

  $$X \sim \text{MVN}_d(\mu, AA^T)$$

- To generate $X \sim \text{MVN}_d(\mu,\Sigma)$, we can
  - find a matrix A such that $AA^T = \Sigma$
  - generate elements of Z as independent standard normals
  - compute $X = \mu + AZ$
- The Cholesky factorization is one way to choose A.
- If we are given Σ^{-1}, then we can
  - decompose $\Sigma^{-1} = LL^T$
  - solve $L^T Y = Z$
  - compute $X = \mu + Y$

Spherically Symmetric Distributions

- A joint distribution with density of the form

$$f(x) = g(x^T x) = g(x_1^2 + \cdots + x_d^2)$$

  is called spherically symmetric (about the origin).

- If the distribution of X is spherically symmetric then

$$\begin{aligned} R &= \sqrt{X^T X} \\ Y &= X/R \end{aligned}$$

  are independent,

  - Y is uniformly distributed on the surface of the unit sphere.
  - R has density proportional to $g(r)r^{d-1}$ for $r > 0$.

- We can generate $X \sim f$ by

  - generating $Z \sim \text{MVN}_d(0, I)$ and setting $Y = Z/\sqrt{Z^T Z}$
  - generating R from the density proportional to $g(r)r^{d-1}$ by univariate methods.

Elliptically Contoured Distributions

- A density f is elliptically contoured if

$$f(x) = \frac{1}{\sqrt{\det \Sigma}} g((x-\mu)^T \Sigma^{-1} (x-\mu))$$

  for some vector μ and symmetric positive definite matrix Σ.

- Suppose Y has spherically symmetric density $g(y^T y)$ and $AA^T = \Sigma$. Then $X = \mu + AY$ has density f.

Wishart Distribution

- Suppose $X_1, \ldots X_n$ are independent and $X_i \sim \text{MVN}_d(\mu_i, \Sigma)$. Let
$$W = \sum_{i=1}^{n} X_i X_i^T$$
Then W has a non-central Wishart distribution $\text{W}(n, \Sigma, \Delta)$ where $\Delta = \sum \mu_i \mu_i^T$.
- If $X_i \sim \text{MVN}_d(\mu, \Sigma)$ and
$$S = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X})(X_i - \overline{X})^T$$
is the sample covariance matrix, then $(n-1)S \sim \text{W}(n-1, \Sigma, 0)$.
- Suppose $\mu_i = 0$, $\Sigma = AA^T$, and $X_i = AZ_i$ with $Z_i \sim \text{MVN}_d(0, I)$. Then $W = AVA^T$ with
$$V = \sum_{i=1}^{n} Z_i Z_i^T$$
- *Bartlett decomposition*: In the Cholesky factorization of V
  - all elements are independent
  - the elements below the diagonal are standard normal
  - the square of the i-th diagonal element is χ^2_{n+1-i}
- If $\Delta \neq 0$ let $\Delta = BB^T$ be its Cholesky factorization, let b_i be the columns of B and let $Z_1, \ldots, Z_n$ be independent $\text{MVN}_d(0, I)$ random vectors. Then for $n \geq d$
$$W = \sum_{i=1}^{d} (b_i + AZ_i)(b_i + AZ_i)^T + \sum_{i=d+1}^{n} AZ_i Z_i^T A^T \sim \text{W}(n, \Sigma, \Delta)$$

Rejection Sampling

- Rejection sampling can in principle be used in any dimensions
- A general envelope that is sometimes useful is based on generating X as

  $$X = b + AZ/Y$$

  where

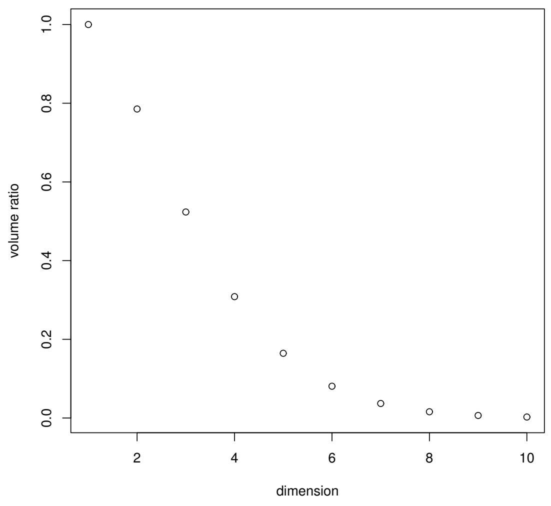
  - Z and Y are independent
  - $Z \sim \text{MVN}_d(0, I)$
  - $Y^2 \sim \text{Gamma}(\alpha, 1/\alpha)$, a scalar
  - b is a vector of constants
  - A is a matrix of constants

  This is a kind of multivariate t random vector.
- This often works in modest dimensions.
- Specially tailored envelopes can sometimes be used in higher dimensions.
- Without special tailoring, rejection rates tend to be too high to be useful.

Ratio of Uniforms

- The ratio-of-uniforms method also works in $\mathbb{R}^d$: Suppose
  - $h(x) \geq 0$ for all x
  - $\int h(x)dx < \infty$

  Let
  $$\mathscr{C}_h = \{(v,u) : v \in \mathbb{R}^d, 0 < u \leq \sqrt[d+1]{h(v/u+\mu)}\}$$
  for some μ. If (V,U) is uniform on $\mathscr{C}_h$, then $X = V/U + \mu$ has density $f(x) = h(x)/\int h(y)dy$.
- If $h(x)$ and $\|x\|^{d+1}h(x)$ are bounded, then $\mathscr{C}_h$ is bounded.
- If $h(x)$ is log concave then $\mathscr{C}_h$ is convex.
- Rejection sampling from a bounding hyper rectangle works in modest dimensions.
- It will not work for dimensions larger than 8 or so:
  - The shape of $\mathscr{C}_h$ is vaguely spherical.
  - The volume of the unit sphere in d dimensions is
  $$V_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$
  - The ratio of this volume to the volume of the enclosing hyper cube, 2^d tends to zero very fast:


volume ratio
dimension
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10

Order Statistics

- The order statistics for a random sample $X_1,\ldots,X_n$ from F are the ordered values
$$X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$$
  - We can simulate them by ordering the sample.
  - Faster $O(n)$ algorithms are available for individual order statistics, such as the median.
- If $U_{(1)} \leq \cdots \leq U_{(n)}$ are the order statistics of a random sample from the $\mathrm{U}[0,1]$ distribution, then
$$\begin{aligned} X_{(1)} &= F^{-}(U_{(1)}) \\ &\vdots \\ X_{(n)} &= F^{-}(U_{(n)}) \end{aligned}$$
are the order statistics of a random sample from F.
- For a sample of size n the marginal distribution of $U_{(k)}$ is
$$U_{(k)} \sim \text{Beta}(k, n-k+1).$$
- Suppose $k < \ell$.
  - Then $U_{(k)}/U_{(\ell)}$ is independent of $U_{(\ell)},\ldots,U_{(n)}$
  - $U_{(k)}/U_{(\ell)}$ has a $\text{Beta}(k,\ell-k)$ distribution.

  We can use this to generate any subset or all order statistics.
- Let $V_1,\ldots,V_{n+1}$ be independent exponential random variables with the same mean and let
$$W_k = \frac{V_1+\cdots+V_k}{V_1+\cdots+V_{n+1}}$$
Then $W_1,\ldots,W_n$ has the same joint distribution as $U_{(1)},\ldots,U_{(n)}$.

Homogeneous Poisson Process

- For a homogeneous Poisson process with rate λ
  - The number of points $N(A)$ in a set A is Poisson with mean $\lambda|A|$.
  - If A and B are disjoint then $N(A)$ and $N(B)$ are independent.
- Conditional on $N(A) = n$, the n points are uniformly distributed on A.
- We can generate a Poisson process on $[0,t]$ by generating exponential variables $T_1, T_2, \ldots$ with rate λ and computing

  $$S_k = T_1 + \cdots + T_k$$

  until $S_k > t$. The values $S_1, \ldots, S_{k-1}$ are the points in the Poisson process realization.

Inhomogeneous Poisson Processes

- For an inhomogeneous Poisson process with rate $\lambda(x)$
  - The number of points $N(A)$ in a set A is Poisson with mean $\int_A \lambda(x)dx$.
  - If A and B are disjoint then $N(A)$ and $N(B)$ are independent.
- Conditional on $N(A) = n$, the n points in A are a random sample from a distribution with density $\lambda(x)/\int_A \lambda(y)dy$.
- To generate an inhomogeneous Poisson process on $[0,t]$ we can
  - let $\Lambda(s) = \int_0^s \lambda(x)dx$
  - generate arrival times $S_1,\dots,S_N$ for a homogeneous Poisson process with rate one on $[0,\Lambda(t)]$
  - Compute arrival times of the inhomogeneous process as

$$\Lambda^{-1}(S_1),\dots,\Lambda^{-1}(S_N).$$

- If $\lambda(x) \le M$ for all x, then we can generate an inhomogeneous Poisson process with rate $\lambda(x)$ by *thinning*:
  - generate a homogeneous Poisson process with rate M to obtain points $X_1,\dots,X_N$.
  - independently delete each point X_i with probability $1-\lambda(X_i)/M$.

  The remaining points form a realization of an inhomogeneous Poisson process with rate $\lambda(x)$.
- If N_1 and N_2 are independent inhomogeneous Poisson processes with rates $\lambda_1(x)$ and $\lambda_2(x)$, then their superposition N_1+N_2 is an inhomogeneous Poisson process with rate $\lambda_1(x)+\lambda_2(x)$.

Other Processes

- Many other processes can be simulated from their definitions
  - Cox processes (doubly stochastic Poisson process)
  - Poisson cluster processes
  - ARMA, ARIMA processes
  - GARCH processes
- Continuous time processes, such as Brownian motion and diffusions, require discretization of time.
- Other processes may require Markov chain methods
  - Ising models
  - Strauss process
  - interacting particle systems

Variance Reduction

Most simulations involve estimating integrals or expectations:

$$
\begin{aligned}
\theta &= \int h(x)f(x)dx && \text{mean}\\
\theta &= \int 1_{\{X \in A\}} f(x)dx && \text{probability}\\
\theta &= \int (h(x)-E[h(X)])^2 f(x)dx && \text{variance}\\
&\vdots
\end{aligned}
$$

- The *crude simulation*, or *crude Monte Carlo*, or *naïve Monte Carlo*, approach:
  - Sample $X_1,\dots,X_N$ independently from f
  - Estimate θ by $\widehat{\theta}_N = \frac{1}{N}\sum h(X_i)$.

  If $\sigma^2 = \text{Var}(h(X))$, then $\text{Var}(\widehat{\theta}_N) = \frac{\sigma^2}{N}$.
- To reduce the error we can
  - increase N: requires CPU time and clock time; diminishing returns.
  - try to reduce σ^2: requires thinking time, programming effort.
- Methods that reduce σ^2 are called
  - tricks
  - swindles
  - Monte Carlo methods

Control Variates

Suppose we have a random variable Y with mean θ and a correlated random variable W with known mean $E[W]$. Then for any constant b

$$\widetilde{Y} = Y - b(W - E[W])$$

has mean θ.

- W is called a *control variate*.
- Choosing $b = 1$ often works well if the correlation is positive and θ and $E[W]$ are close.
- The value of b that minimizes the variance of $\widetilde{Y}$ is $\text{Cov}(Y,W)/\text{Var}(W)$.
- We can use a guess or a pilot study to estimate b.
- We can also estimate b from the same data used to compute Y and W.
- This is related to the regression estimator in sampling.

Example

Suppose we want to estimate the expected value of the sample median T for a sample of size 10 from a Gamma$(3,1)$ population.

- Crude estimate:
$$Y = \frac{1}{N}\sum T_i$$
- Using the sample mean as a control variate with $b = 1$:

$$\widehat{Y} = \frac{1}{N}\sum(T_i - \overline{X}_i) + E[\overline{X}_i] = \frac{1}{N}\sum(T_i - \overline{X}_i) + \alpha$$

```
> x <- matrix(rgamma(10000, 3), ncol = 10)
> md <- apply(x, 1, median)
> mn <- apply(x, 1, mean)
> mean(md)
[1] 2.711137
> mean(md - mn) + 3
[1] 2.694401
> sd(md)
[1] 0.6284996
> sd(md-mn)
[1] 0.3562479
```

The standard deviation is cut roughly in half. The optimal b seems close to 1.

Control Variates and Probability Estimates

- Suppose T is a test statistic and we want to estimate $\theta = P(T \leq t)$.
- Crude Monte Carlo:
$$\widehat{\theta} = \frac{\#\{T_i \leq t\}}{N}$$
- Suppose S is "similar" to T and $P(S \leq t)$ is known. Use
$$\widehat{\theta} = \frac{\#\{T_i \leq t\} - \#\{S_i \leq t\}}{N} + P(S \leq t) = \frac{1}{N}\sum Y_i + P(S \leq t)$$
with $Y_i = 1_{\{T_i \leq t\}} - 1_{\{S_i \leq t\}}$.
- If S mimics T, then Y_i is usually zero.
- Could use this to calibrate
$$T = \frac{\text{median}}{\text{interquartile range}}$$
for normal data using the t statistic.

Importance Sampling

- Suppose we want to estimate

$$\theta = \int h(x) f(x) dx$$

  for some density f and some function h.

- Crude Mote Carlo samples $X_1, \ldots, X_N$ from f and uses

$$\widehat{\theta} = \frac{1}{N} \sum h(X_i)$$

  If the region where h is large has small probability under f then this can be inefficient.

- Alternative: Sample $X_1, \ldots X_n$ from g that puts more probability near the "important" values of x and compute

$$\widetilde{\theta} = \frac{1}{N} \sum h(X_i) \frac{f(X_i)}{g(X_i)}$$

  Then, if $g(x) > 0$ when $h(x) f(x) \neq 0$,

$$E[\widetilde{\theta}] = \int h(x) \frac{f(x)}{g(x)} g(x) dx = \int h(x) f(x) dx = \theta$$

  and

$$\text{Var}(\widetilde{\theta}) = \frac{1}{N} \int \left(h(x) \frac{f(x)}{g(x)} - \theta \right)^2 g(x) dx = \frac{1}{N} \left(\int \left(h(x) \frac{f(x)}{g(x)} \right)^2 g(x) dx - \theta^2 \right)$$

  The variance is minimized by $g(x) \propto |h(x) f(x)|$

Importance Weights

- Importance sampling is related to stratified and weighted sampling in sampling theory.
- The function $w(x) = f(x)/g(x)$ is called the *weight function*.
- Alternative estimator:

  $$\theta^* = \frac{\sum h(X_i)w(X_i)}{\sum w(X_i)}$$

  This is useful if f or g or both are unnormalized densities.
- Importance sampling can be useful for computing expectations with respect to posterior distributions in Bayesian analyses.
- Importance sampling can work very well if the weight function is bounded.
- Importance sampling can work very poorly if the weight function is unbounded—it is easy to end up with infinite variances.

Computing Tail Probabilities

- Suppose $\theta = P(X \in R)$ for some region R.
- Suppose we can find g such that $f(x)/g(x) < 1$ on R. Then

$$\widetilde{\theta} = \frac{1}{N}\sum 1_R(X_i)\frac{f(X_i)}{g(X_i)}$$

and

$$\begin{aligned}
\text{Var}(\widetilde{\theta}) &= \frac{1}{N}\left(\int_R \left(\frac{f(x)}{g(x)}\right)^2 g(x)dx - \theta^2\right)\\
&= \frac{1}{N}\left(\int_R \frac{f(x)}{g(x)} f(x)dx - \theta^2\right)\\
&< \frac{1}{N}\left(\int_R f(x)dx - \theta^2\right)\\
&= \frac{1}{N}(\theta - \theta^2) = \text{Var}(\widehat{\theta})
\end{aligned}$$

- For computing $P(X > 2)$ where X has a standard Cauchy distribution we can use a shifted distribution:

```
> y <- rcauchy(10000,3)
> tt <- ifelse(y > 2, 1, 0) * dcauchy(y) / dcauchy(y,3)
> mean(tt)
[1] 0.1490745
> sd(tt)
[1] 0.1622395
```

- The asymptotic standard deviation for crude Monte Carlo is approximately

```
> sqrt(mean(tt) * (1 - mean(tt)))
[1] 0.3561619
```

- A *tilted* density $g(x) \propto f(x)e^{\beta x}$ can also be useful.

Antithetic Variates

- Suppose S and T are two unbiased estimators of θ with the same variance σ^2 and correlation ρ, and compute

  $$V = \frac{1}{2}(S+T)$$

  Then

  $$\mathrm{Var}(V) = \frac{\sigma^2}{4}(2+2\rho) = \frac{\sigma^2}{2}(1+\rho)$$

- Choosing $\rho < 0$ reduces variance.
- Such negatively correlated pairs are called *antithetic variates*.
- Suppose we can choose between generating independent $T_1, \dots, T_N$

  $$\widehat{\theta} = \frac{1}{N}\sum_{i=1}^{N} T_i$$

  or independent pairs $(S_1, T_1), \dots, (S_{N/2}, T_{N/2})$ and computing

  $$\widetilde{\theta} = \frac{1}{N}\sum_{i=1}^{N/2}(S_i + T_i)$$

  If $\rho < 0$, then $\mathrm{Var}(\widetilde{\theta}) < \mathrm{Var}(\widehat{\theta})$.

- If $T = f(U)$, $U \sim \mathrm{U}[0,1]$, and f is monotone, then $S = f(1-U)$ is negatively correlated with T and has the same marginal distribution.
- If inversion is used to generate variates, computing T from $U_1, \dots$ and S from $1-U_1, \dots$ often works.
- Some uniform generators provide an option in the seed to switch between returning U_i and $1-U_i$.

Example

For estimating the expected value of the median for samples of size 10 from the Gamma$(3,1)$ distribution:

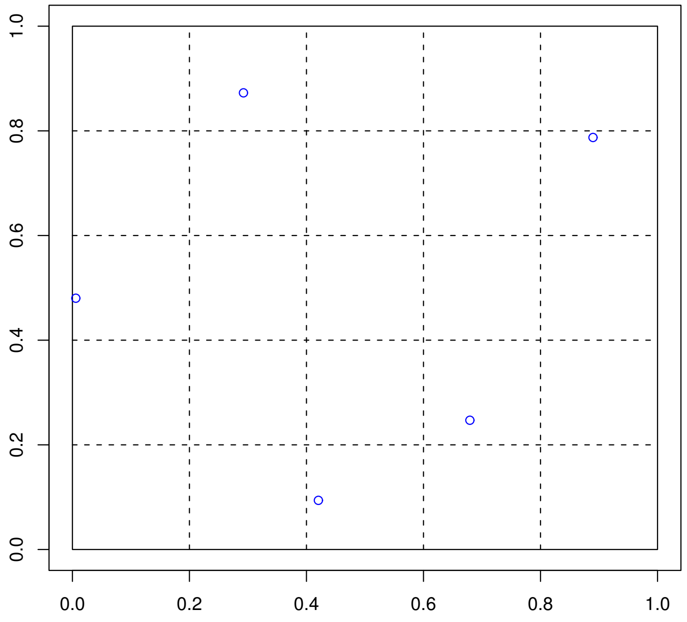
```
> u <- matrix(runif(5000), ncol = 10)
> x1 <- qgamma(u, 3)
> x2 <- qgamma(1 - u, 3)
> md1 <- apply(x1, 1, median)
> md2 <- apply(x2, 1, median)
> sqrt(2) * sd((md1 + md2) / 2)
[1] 0.09809588
```

Control variates helps further a bit but need $b = 0.2$ or so.

```
> mn1 <- apply(x1, 1, mean)
> mn2 <- apply(x2, 1, mean)
> sqrt(2) * sd((md1 + md2 - 0.2 * (mn1 + mn2)) / 2)
[1] 0.09216334
```

Latin Hypercube Sampling

- Suppose we want to compute
$$\theta = E[f(U_1, \ldots, U_d)]$$
with $(U_1, \ldots, U_d)$ uniform on $[0,1]^d$.
- For each i
  - independently choose permutation π_i of $\{1, \ldots, N\}$
  - generate $U_i^{(j)}$ uniformly on $[\pi_i(j)/N, (\pi_i(j)+1)/N]$.
- For $d = 2$ and $N = 5$:



This is a random Latin square design.

- In many cases this reduces variance compared to unrestricted random sampling (Stein, 1987; Avramidis and Wilson, 1995; Owen, 1992, 1998)

Common Variates and Blocking

- Suppose we want to estimate $\theta = E[S] - E[T]$
- One approach is to chose independent samples $T_1, \dots, T_N$ and $S_1, \dots, S_M$ and compute
$$\widehat{\theta} = \frac{1}{M}\sum_{i=1}^{M} S_i - \frac{1}{N}\sum_{i=1}^{N} T_i$$
- Suppose $S = S(X)$ and $T = T(X)$ for some X. Instead of generating independent X values for S and T we may be able to
  - use the common X values to generate pairs $(S_1, T_1), \dots, (S_N, T_N)$
  - compute
$$\widetilde{\theta} = \frac{1}{N}\sum_{i=1}^{N}(S_i - T_i)$$
- This use of *paired comparisons* is a form of *blocking*.
- This idea extends to comparisons among more than two statistics.
- In simulations, we can often do this by using the same random variates to generate S_i and T_i. This is called using *common variates*.
- This is easiest to do if we are using inversion; this, and the ability to use antithetic variates, are two strong arguments in favor of inversion.
- Using common variates may be harder when rejection-based methods are involved.
- In importance sampling, using
$$\theta^* = \frac{\sum h(X_i)w(X_i)}{\sum w(X_i)}$$
can be viewed as a paired comparison; for some forms of h is can have lower variance than the estimator that does not normalize by the sum of the weights.

Conditioning or Rao-Blackwellization

- Suppose we want to estimate $\theta = E[X]$
- If X, W are jointly distributed, then

  $$\theta = E[X] = E[E[X|W]]$$

  and

  $$\text{Var}(X) = E[\text{Var}(X|W)] + \text{Var}(E[X|W]) \geq \text{Var}(E[X|W])$$

- Suppose we can compute $E[X|W]$. Then we can
  - generate $W_1, \dots, W_N$
  - compute

    $$\widetilde{\theta} = \frac{1}{N} \sum E[X|W_i]$$

- This is often useful in Gibbs sampling.
- Variance reduction is not guaranteed if $W_1, \dots, W_N$ are not independent.
- Conditioning is particularly useful for density estimation: If we can compute $f_{X|W}(x|w)$ and generate $W_1, \dots, W_N$, then

  $$\widehat{f}_X(x) = \frac{1}{N} \sum f_{X|W}(x|W_i)$$

  is much more accurate than, say, a kernel density estimate based on a sample $X_1, \dots, X_N$.

Example

Suppose we want to estimate $\theta = P(X > t)$ where $X = Z/W$ with Z, W independent, $Z \sim \text{N}(0,1)$ and $W > 0$. Then

$$P(X > t | W = w) = P(Z > tw) = 1 - \Phi(tw)$$

So we can estimate θ by generating $W_1, \dots, W_N$ and computing

$$\widetilde{\theta} = \frac{1}{N} \sum (1 - \Phi(tW_i))$$

Independence Decomposition

- Suppose $X_1, \dots, X_n$ is a random sample from a $\mathrm{N}(0,1)$ distribution and
$$\widetilde{X} = \mathrm{median}(X_1, \dots, X_n)$$
We want to estimate $\theta = \mathrm{Var}(\widetilde{X}) = E[\widetilde{X}^2]$.
- Crude Monte Carlo estimate: generate independent medians $\widetilde{X}_1, \dots, \widehat{X}_N$ and compute
$$\widehat{\theta} = \frac{1}{N}\sum \widetilde{X}_i^2$$
- Alternative: Write
$$\widetilde{X} = (\widetilde{X} - \overline{X}) + \overline{X}$$
$(\widetilde{X} - \overline{X})$ and $\overline{X}$ are independent, for example by Basu's theorem. So
$$E[\widetilde{X}^2|\overline{X}] = \overline{X}^2 + E[(\widetilde{X} - \overline{X})^2]$$
and
$$\theta = \frac{1}{n} + E[(\widetilde{X} - \overline{X})^2]$$
- So we can estimate θ by generating pairs $(\widetilde{X}_i, \overline{X}_i)$ and computing
$$\widetilde{\theta} = \frac{1}{n} + \frac{1}{N}\sum (\widetilde{X}_i - \overline{X}_i)^2$$
- Generating these pairs may be more costly than generating medians alone.

Example

```
> x <- matrix(rnorm(10000), ncol = 10)
> mn <- apply(x, 1, mean)
> md <- apply(x, 1, median)
> # estimates:
> mean(md^2)
[1] 0.1446236
> 1 / 10 + mean((md - mn)^2)
[1] 0.1363207
> # asymptotic standard errors:
> sd(md^2)
[1] 0.2097043
> sd((md - mn)^2)
[1] 0.0533576
```

Princeton Robustness Study

D. F. Andrews, P. J. Bickel, F. R. Hampel, P. J. Huber, W. H. Rogers, and J. W. Tukey, *Robustness of Location Estimates*, Princeton University Press, 1972.

- Suppose $X_1, \ldots, X_n$ are a random sample from a symmetric density
$$f(x-m).$$
- We want an estimator $T(X_1, \ldots, X_n)$ of m that is
  - accurate
  - robust (works well for a wide range of f's)
- Study considers many estimators, various different distributions.
- All estimators are unbiased and *affine equivariant*, i.e.
$$\begin{aligned} E[T] &= m \\ T(aX_1+b, \ldots, aX_n+b) &= aT(X_1, \ldots, X_n)+b \end{aligned}$$
  for any constants $a, b.$ We can thus take $m = 0$ without loss of generality.

Distributions Used in the Study

- Distributions considered were all of the form

$$X = Z/V$$

  with $Z \sim \mathrm{N}(0,1)$, $V > 0$, and Z,V independent.

- Some examples:
  - $V \equiv 1$ gives $X \sim \mathrm{N}(0,1)$.
  - Contaminated normal:

$$V = \begin{cases} c & \text{with probability } \alpha \\ 1 & \text{with probability } 1-\alpha \end{cases}$$

  - Double exponential: $V \sim f_V(v) = v^{-3}e^{-v^{-2}/2}$
  - Cauchy: $V = |Y|$ with $Y \sim \mathrm{N}(0,1)$.
  - t_ν: $V \sim \sqrt{\chi^2_\nu/\nu}$.
- The conditional distribution $X|V = v$ is $\mathrm{N}(0,1/v^2)$.
- Study generates X_i as Z_i/V_i.
- Write $X_i = \widehat{X} + \widehat{S}C_i$ with

$$\widehat{X} = \frac{\sum X_i V_i^2}{\sum V_i^2} \qquad \widehat{S}^2 = \frac{1}{n-1}\sum (X_i - \widehat{X})^2 V_i^2$$

  Then

$$T(X) = \widehat{X} + \widehat{S}T(C)$$

- Can show that $\widehat{X}, \widehat{S}, C$ are conditionally independent given V.

Estimating Variances

- Suppose we want to estimate $\theta = \text{Var}(T) = E[T^2]$. Then

$$\begin{aligned}\theta &= E[(\widehat{X} + \widehat{S}T(C))^2] \\ &= E[\widehat{X}^2 + 2\widehat{S}\widehat{X}T(C) + \widehat{S}^2 T(C)^2] \\ &= E[E[\widehat{X}^2 + 2\widehat{S}\widehat{X}T(C) + \widehat{S}^2 T(C)^2 | V]]\end{aligned}$$

  and

$$\begin{aligned}E[\widehat{X}^2|V] &= \frac{1}{\sum V_i^2} \\ E[\widehat{X}|V] &= 0 \\ E[\widehat{S}^2|V] &= 1\end{aligned}$$

  So

$$\theta = E\left[\frac{1}{\sum V_i^2}\right] + E[T(C)^2]$$

- Strategy:
  - Compute $E[T(C)^2]$ by crude Monte Carlo
  - Compute $E\left[\frac{1}{\sum V_i^2}\right]$ the same way or analytically.

  Exact calculations:

  - If $V_i \sim \sqrt{\chi^2_\nu/\nu}$, then

$$E\left[\frac{1}{\sum V_i^2}\right] = E\left[\frac{\nu}{\chi^2_{n\nu}}\right] = \frac{\nu}{n\nu - 2}$$

  - Contaminated normal:

$$E\left[\frac{1}{\sum V_i^2}\right] = \sum_{r=0}^{n} \binom{n}{r} \alpha^r (1-\alpha)^{n-r} \frac{1}{n - r + rc}$$

Comparing Variances

If T_1 and T_2 are two estimators, then

$$\text{Var}(T_1) - \text{Var}(T_2) = E[T_1(C)^2] - E[T_2(C)^2]$$

We can reduce variances further by using common variates.

Estimating Tail Probabilities

- Suppose we want to estimate

$$\begin{aligned}
\theta &= P(T(X) > t) \\
&= P(\widehat{X} + \widehat{S}T(C) > t) \\
&= E\left[P\left(\sqrt{\sum V_i^2}\frac{t-\widehat{X}}{\widehat{S}} < \sqrt{\sum V_i^2}T(C) \middle| V, C\right)\right] \\
&= E\left[F_{t,n-1}\left(\sqrt{\sum V_i^2}T(C)\right)\right]
\end{aligned}$$

  where $F_{t,n-1}$ is the CDF of a non-central t distribution (t is not the usual non-centrality parameter).

- This CDF can be evaluated numerically, so we can estimate θ by

$$\widehat{\theta}_N = \frac{1}{N}\sum_{k=1}^{N} F_{t,n-1}\left(T(C^{(k)})\sqrt{\sum {V_i^{(k)}}^2}\right)$$

- An alternative is to condition on $V, C, \widehat{S}$ and use the conditional normal distribution of $\widehat{X}$.

Markov Chain Monte Carlo

Simulation with Dependent Observations

- Suppose we want to compute
$$\theta = E[h(X)] = \int h(x)f(x)dx$$
- Crude Monte Carlo: generate $X_1, \dots, X_N$ that are
  - independent
  - identically distributed from f

  and compute $\widehat{\theta} = \frac{1}{N}\sum h(X_i)$.
- Then
  - $\widehat{\theta} \to \theta$ by the law of large numbers
  - $\widehat{\theta} \sim \text{AN}(\theta, \sigma^2/N)$ by the central limit theorem
  - σ^2 can be estimated using the sample standard deviation.
- Sometimes generating independently from f is not possible or too costly.
- Importance sampling: generate independently from g and reweight.
- Alternative: generate *dependent* samples in a way that
  - preserves the law of large numbers
  - has a central limit theorem if possible
- Variance estimation will be more complicated.

A Simple Example

Suppose X, Y are bivariate normal with mean zero, variance 1, and correlation ρ,

$$(X,Y) \sim \mathrm{BVN}(0,1,0,1,\rho).$$

Then

$$\begin{aligned} Y|X = x &\sim \mathrm{N}(\rho x, 1-\rho^2) \\ X|Y = y &\sim \mathrm{N}(\rho y, 1-\rho^2). \end{aligned}$$

Suppose we start with some initial values X_0, Y_0 and generate

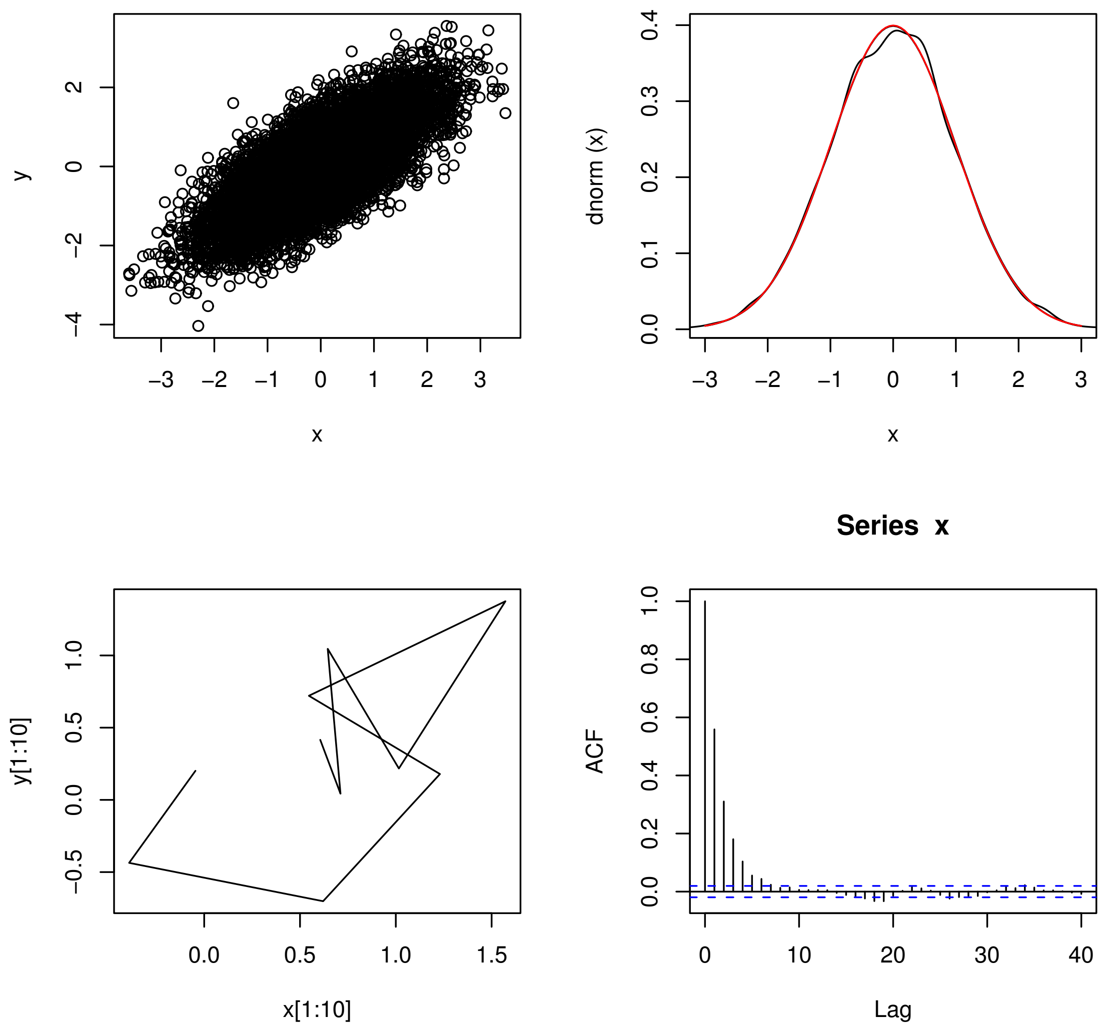
$$\begin{aligned} X_1 &\sim \mathrm{N}(\rho Y_0, 1-\rho^2) \\ Y_1 &\sim \mathrm{N}(\rho X_1, 1-\rho^2) \end{aligned}$$

(X_0 is not used), and continue for $i = 1, \ldots, N-1$

$$\begin{aligned} X_{i+1} &\sim \mathrm{N}(\rho Y_i, 1-\rho^2) \\ Y_{i+1} &\sim \mathrm{N}(\rho X_{i+1}, 1-\rho^2) \end{aligned}$$

For $\rho = 0.75$ and $Y_0 = 0$:

```
r <- 0.75
x <- numeric(10000)
y <- numeric(10000)
x[1] <- rnorm(1, 0, sqrt(1 - r^2))
y[1] <- rnorm(1, r * x[1], sqrt(1 - r^2))
for (i in 1:(length(x) - 1)) {
    x[i+1] <- rnorm(1, r * y[i], sqrt(1 - r^2))
    y[i+1] <- rnorm(1, r * x[i + 1], sqrt(1 - r^2))
}
```



```
par(mfrow = c(2, 2))
plot(x, y)
plot(dnorm, -3, 3)
lines(density(x))
z <- seq(-3, 3, len = 101)
dz <- sapply(z, function(z) mean(dnorm(z, r * x, sqrt(1 - r^2))))
lines(z, dz, col = "red")
plot(x[1:10], y[1:10], type = "l")
acf(x)

> cor(x,y)
[1] 0.7443691
```

- The sequence of pairs (X_i,Y_i) form a *continuous state space Markov chain*.
- Suppose $(X_0,Y_0) \sim \text{BVN}(0,1,0,1,\rho)$. Then
  - $(X_1,Y_0) \sim \text{BVN}(0,1,0,1,\rho)$
  - $(X_1,Y_1) \sim \text{BVN}(0,1,0,1,\rho)$
  - $(X_2,Y_1) \sim \text{BVN}(0,1,0,1,\rho)$
  - $\dots$

  So $\text{BVN}(0,1,0,1,\rho)$ is a *stationary distribution* or *invariant distribution* of the Markov chain.
- $\text{BVN}(0,1,0,1,\rho)$ is also the *equilibrium distribution* of the chain, i.e. for any starting distribution the joint distribution of (X_n,Y_n) converges to the $\text{BVN}(0,1,0,1,\rho)$ distribution.
- For this example, $X_1,X_2,\dots$ is an AR(1) process

  $$X_i = \rho^2 X_{i-1} + \varepsilon_i$$

  with the ε_i independent and $N(0,1-\rho^4)$.
- Standard time series results show that the equilibrium distribution of this chain is $\text{N}(0,1)$.
- If

  $$\overline{X}_N = \frac{1}{N}\sum_{i=1}^{N} X_i$$

  then

  $$\text{Var}(\overline{X}_N) \approx \frac{1}{N}\frac{1-\rho^4}{(1-\rho^2)^2} = \frac{1}{N}\frac{1+\rho^2}{1-\rho^2}$$

Markov Chain Monte Carlo

- Objective is to compute

$$\theta = E[h(X)] = \int h(x)f(x)dx$$

- Basic idea:
  - Construct a Markov chain with invariant distribution f.
  - Make sure the chain has f as its equilibrium distribution.
  - Pick a starting value X_0.
  - Generate $X_1, \ldots, X_N$ and compute

$$\widehat{\theta} = \frac{1}{N}\sum h(X_i)$$

  - Possibly repeat independently several times, maybe with different starting values.
- Some issues:
  - How to construct a Markov chain with a particular invariant distribution.
  - How to estimate the variance of $\widehat{\theta}$.
  - What value of N to use.
  - Should an initial portion be discarded; if so, how much?

Some MCMC Examples

Markov chain Monte Carlo (MCMC) is used for a wide range of problems and applications:

- generating spatial processes
- sampling from equilibrium distributions in physical chemistry
- computing likelihoods in missing data problems
- computing posterior distributions in Bayesian inference
- optimization, e.g. simulated annealing
- ...

Strauss Process

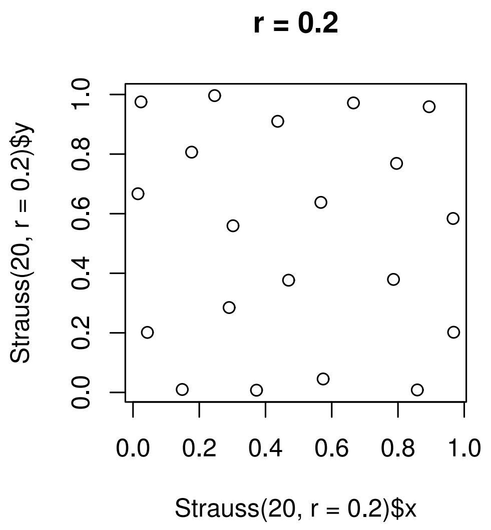
- The Strauss process is a model for random point patterns with some regularity.
- A set of n points is distributed on a region D with finite area or volume.
- The process has two parameters, $c \in [0,1]$ and $r > 0$.
- The joint density of the points is

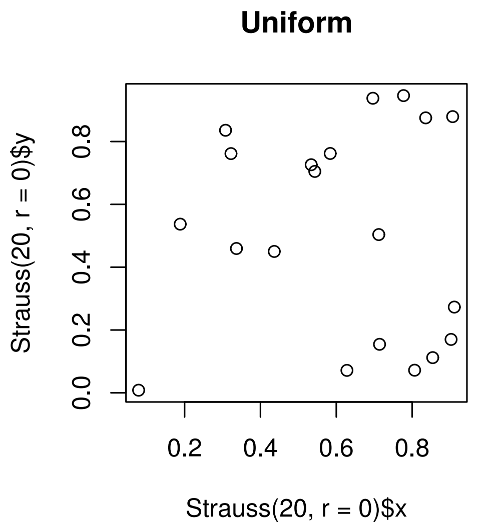
$$f(x_1,\dots,x_n) \propto c^{\text{number of pairs within } r \text{ of each other}}$$

- For $c = 0$ the density is zero if any two points are withing r of each other.
- Simulating independent draws from f is possible in principle but very inefficient.
- A Markov chain algorithm:
  - Start with n points $X_1,\dots,X_n$ with $f(X_1,\dots,X_n) > 0$.
  - choose an index i at random, remove point X_i, and replace it with a draw from

$$f(x|x_1,\dots,x_{i-1},x_{i+1},\dots,x_n) \propto c^{\text{number of remaining } n-1 \text{ points within } r \text{ of } x}$$

  - This can be sampled reasonably efficiently by rejection sampling.
  - Repeat for a reasonable number of steps.
- `Strauss` in package `spatial` implements this algorithm. [**Caution:** it used to be easy to hang R because the C code would go into an infinite loop for some parameters. More recent versions may have modified C code that checks for interrupts.]
- The algorithm is due to Ripley (1979); Ripley's algorithm applies to a general multivariate density.


r = 0.2
Strauss(20, r = 0.2)$x
Strauss(20, r = 0.2)$y
0.0
0.2
0.4
0.6
0.8
1.0


Uniform
Strauss(20, r = 0)$x
Strauss(20, r = 0)$y
0.2
0.4
0.6
0.8
0.0

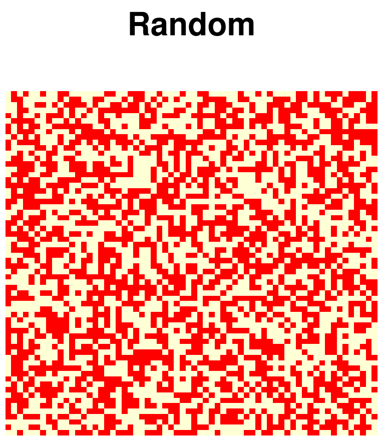
Markov Random Fields

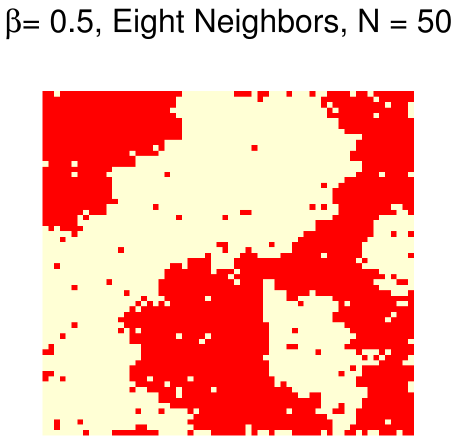
- A Markov random field is a spatial process in which
  - each index i has a set of neighbor indices N_i
  - X_i and $\{X_k : k \neq i \text{ and } k \notin N_i\}$ are conditionally independent given $\{X_j : j \in N_i\}$.
- In a binary random field the values of X_i are binary.
- Random fields are used as models in
  - spatial statistics
  - image processing
  - ...
- A simple model for an $n \times n$ image with c colors:

  $$f(x) \propto \exp\{\beta \times (\text{number of adjacent pixel pairs with the same color})\}$$

  - This is called a Potts model
  - For a pixel i, the conditional PMF of the pixel color X_i given the other pixel colors, is

    $$f(x_i|\text{rest}) \propto \exp\{\beta \times (\text{number of neighbors with color } x_i\}$$





Simple Image Reconstruction

- Suppose a binary image is contaminated with noise.
- The noise process is assumed to act independently on the pixels.
- Each pixel is left unchanged with probability p and flipped to the opposite color with probability $1-p$.
- The likelihood for the observed image Y_i is

  $$f(y|x) \propto p^m (1-p)^{n^2-m}$$

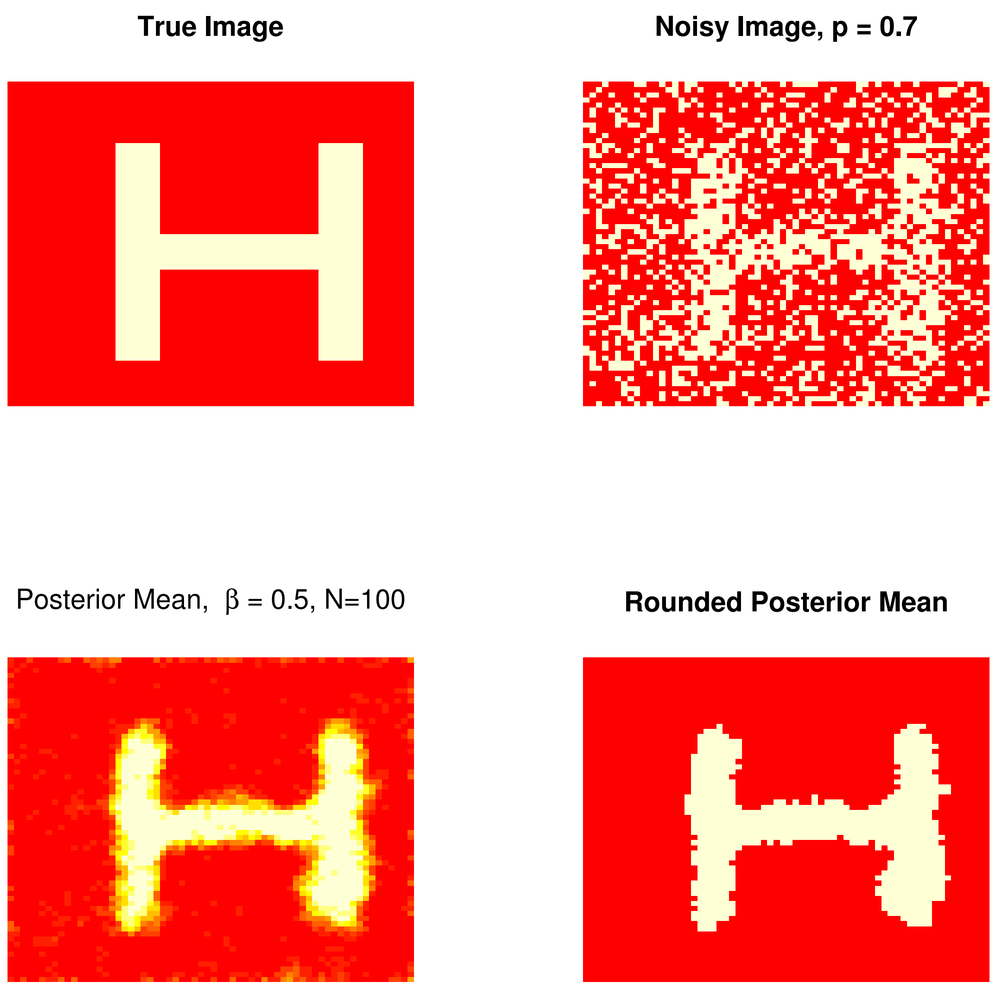
  with m the number of pixels with $y_i = x_i$.
- A Markov random field is used as the prior distribution of the true image.
- The posterior distribution of the true image is

  $$f(x|y) \propto p^m (1-p)^{n^2-m} e^{\beta w}$$

  with w the number of adjacent pixel pairs in the true image x with the same color.
- The posterior distribution is also a Markov random field, and

  $$f(x_i|y, x_j \text{ for } j \neq i) \propto p^{m_i}(1-p)^{1-m_i} e^{\beta w_i}$$

  with $m_i = 1$ if $x_i = y_i$ and $m_i = 0$ otherwise, and w_i the number of neighbors with color x_i.
- Images drawn from the posterior distribution can be averaged to form a posterior mean.
- The posterior mean can be rounded to a binary image.
- Many other variations are possible.



Another example using $\beta = 0.5$ and $\beta = 0.35$ and $N = 100$ will be shown in class.

- This is a simple version of the ideas in Geman, S. and Geman D, (1984) Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**, 721–741.
- The posterior distributions are related to Gibbs distributions in physics.
- Geman and Geman call the Markov chain algorithm *Gibbs sampling*.

Code for the Image Sampler

```
simImg<-function (m, img, N, beta, p)
{
    colors <- 1:2
    inboff <- c(-1, -1, -1,  0, 0,  1, 1, 1)
    jnboff <- c(-1,  0,  1, -1, 1, -1, 0, 1)
    for (k in 1:N) {
        for (i in 1:nrow(m)) {
            for (j in 1:ncol(m)) {
                w <- double(length(colors))
                inb <- i + inboff
                jnb <- j + jnboff
                omit <- inb == 0 | inb == nrow(m) + 1 |
                        jnb == 0 | jnb == ncol(m) + 1
                inb <- inb[!omit]
                jnb <- jnb[!omit]
                for (ii in 1:length(inb)) {
                    kk <- m[inb[ii], jnb[ii]]
                    w[kk] <- w[kk] + 1
                }
                if (is.null(img)) lik <- 1
                else lik <- ifelse(img[i,j]==colors, p, 1-p)
                prob <- lik * exp(beta * w)
                m[i, j] <- sample(colors, 1, TRUE, prob = prob)
            }
        }
    }
    m
}
```

Profiling to Improve Performance

- `Rprof` can be used to turn on profiling.
- During profiling a *stack trace* is written to a file, `Rprof.out` by default, 50 times per second.
- `summaryRprof` produces a summary of the results.

```
> Rprof();system.time(simImg(m,img,10,.35,.7));Rprof(NULL)
   user  system elapsed
  1.547   0.004   1.552
> summaryRprof()
$by.self
              self.time self.pct total.time total.pct
"simImg"           0.68    43.59       1.56    100.00
"ifelse"           0.30    19.23       0.36     23.08
"sample.int"       0.20    12.82       0.22     14.10
"double"           0.08     5.13       0.08      5.13
"sample"           0.06     3.85       0.28     17.95
...

$by.total
              total.time total.pct self.time self.pct
"simImg"            1.56    100.00      0.68    43.59
"system.time"       1.56    100.00      0.00     0.00
"ifelse"            0.36     23.08      0.30    19.23
"sample"            0.28     17.95      0.06     3.85
"sample.int"        0.22     14.10      0.20    12.82
"double"            0.08      5.13      0.08     5.13
...

$sampling.time
[1] 1.56
```

Replacing `ifelse` in

```
else lik <- ifelse(img[i,j]==colors, p, 1-p)
```

with

```
else if (img[i, j] == 1) lik <- c(p, 1-p)
else lik <- c(1 - p, p)
```

speeds things up considerably:

```
> Rprof();system.time(simImg1(m,img,10,.35,.7));Rprof(NULL)
   user  system elapsed
  1.261   0.002   1.263
> summaryRprof()
$by.self
              self.time self.pct total.time total.pct
"simImg1"          0.86    67.19       1.28    100.00
"sample"           0.08     6.25       0.12      9.38
"c"                0.08     6.25       0.08      6.25
"double"           0.06     4.69       0.06      4.69
"+"                0.04     3.12       0.04      3.12
"|"                0.04     3.12       0.04      3.12
"sample.int"       0.04     3.12       0.04      3.12
"!"                0.02     1.56       0.02      1.56
"*"                0.02     1.56       0.02      1.56
"=="               0.02     1.56       0.02      1.56
"ncol"             0.02     1.56       0.02      1.56

$by.total
              total.time total.pct self.time self.pct
"system.time"       3.00    100.00      0.00     0.00
"simImg1"           1.28    100.00      0.86    67.19
"system.time"       1.28    100.00      0.00     0.00
"sample"            0.12      9.38      0.08     6.25
"c"                 0.08      6.25      0.08     6.25
"double"            0.06      4.69      0.06     4.69
"+"                 0.04      3.12      0.04     3.12
"|"                 0.04      3.12      0.04     3.12
"sample.int"        0.04      3.12      0.04     3.12
"!"                 0.02      1.56      0.02     1.56
"*"                 0.02      1.56      0.02     1.56
```

```
"=="                  0.02      1.56      0.02     1.56
"ncol"                0.02      1.56      0.02     1.56
...
```

Starting profiling with

```
Rprof(line.profiling = TRUE)
```

will enable source profiling if our funciton is defined in a file and sourced.

Using the new version of package `proftools` from GitHub with

```
> pd <- readProfileData()
> annotateSource(pd)
```

produces

```
          :  simImg1 <- function (m, img, N, beta, p)
          :  {
          :      colors <- 1:2
          :      inboff <- c(-1, -1, -1, 0, 0, 1, 1, 1)
          :      jnboff <- c(-1, 0, 1, -1, 1, -1, 0, 1)
          :      for (k in 1:N) {
          :          for (i in 1:nrow(m)) {
          :              for (j in 1:ncol(m)) {
   1.56%  :                  w <- double(length(colors))
   1.56%  :                  inb <- i + inboff
   1.56%  :                  jnb <- j + jnboff
  14.06%  :                  omit <- inb == 0 | inb == nrow(m) + 1 | jnb ==
          :                    0 | jnb == ncol(m) + 1
   1.56%  :                  inb <- inb[!omit]
          :                  jnb <- jnb[!omit]
   4.69%  :                  for (ii in 1:length(inb)) {
   7.81%  :                    kk <- m[inb[ii], jnb[ii]]
  23.44%  :                    w[kk] <- w[kk] + 1
          :                  }
   3.12%  :                  if (is.null(img))
          :                    lik <- 1
          :                  else if (img[i, j] == 1) lik <- c(p, 1-p)
          :                  else lik <- c(1 - p, p)
   1.56%  :                  prob <- lik * exp(beta * w)
  35.94%  :                  m[i, j] <- sample(colors, 1, TRUE, prob = prob)
          :              }
          :          }
          :      }
          :      m
          :  }
```

This suggests one more useful change: move the loop-invariant computations `nrow(m)+1` and `ncol(m)+1` out of the loop:

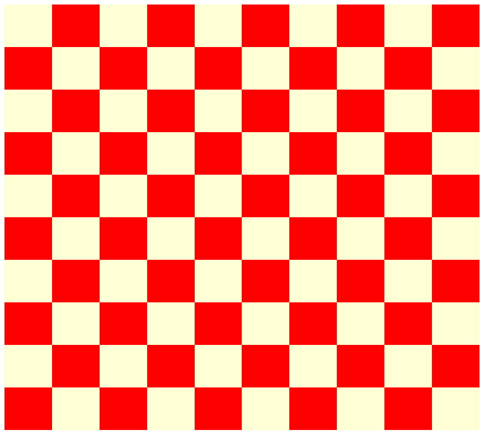
```
simImg2 <- function (m, img, N, beta, p) {
    ...
    nrp1 <- nrow(m) + 1
    ncp1 <- ncol(m) + 1
    for (k in 1:N) {
        for (i in 1:nrow(m)) {
            for (j in 1:ncol(m)) {
                ...
                omit <- inb == 0 | inb == nrp1 |
                        jnb == 0 | jnb == ncp1
                ...
            }
        }
    }
    m
}
```

This helps as well:

```
> system.time(simImg2(m,img,10,.35,.7))
   user  system elapsed
  0.692   0.000   0.690
```

Exploiting Conditional Independence

- We are trying to sample a joint distribution of a collection of random variables
$$X_i, i \in \mathscr{C}$$
- Sometimes it is possible to divide the index set $\mathscr{C}$ into k groups
$$\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$$
such that for each j the indices $X_i, i \in \mathscr{C}_j$ are conditionally independent given the other values $\{X_i, i \notin \mathscr{C}_j\}$.
- For a 4-neighbor lattice we can use two groups,



with $\mathscr{C}_1 =$ red and $\mathscr{C}_2 =$ white
- For an 8-neighbor lattice four groups are needed.
- Each group can be updated as a group, either
  - using vectorized arithmetic, or
  - using parallel computation

A vectorized implementation based on this approach:

```
nn <- function(m, c) {
    nr <- nrow(m)
    nc <- ncol(m)
    nn <- matrix(0, nr, nc)
    nn[1:(nr)-1,] <- nn[1:(nr)-1,] + (m[2:nr,] == c)
    nn[2:nr,] <- nn[2:nr,] + (m[1:(nr-1),] == c)
    nn[,1:(nc)-1] <- nn[,1:(nc)-1] + (m[,2:nc] == c)
    nn[,2:nc] <- nn[,2:nc] + (m[,1:(nc-1)] == c)
    nn
}

simGroup <- function(m, l2, l1, beta, which) {
    pp2 <- l2 * exp(beta * nn(m, 2))
    pp1 <- l1 * exp(beta * nn(m, 1))
    pp <- pp2 / (pp2 + pp1)
    ifelse(runif(sum(which)) < pp[which], 2, 1)
}

simImgV <- function(m, img, N, beta, p) {
    white <- outer(1:nrow(m), 1:ncol(m), FUN=`+`) %% 2 == 1
    black <- ! white
    if (is.null(img)) {
        l2 <- 1
        l1 <- 1
    }
    else {
        l2 <- ifelse(img == 2, p, 1 - p)
        l1 <- ifelse(img == 1, p, 1 - p)
    }
    for (i in 1:N) {
        m[white] <- simGroup(m, l2, l1, beta, white)
        m[black] <- simGroup(m, l2, l1, beta, black)
    }
    m
}
```

The results:

```
> system.time(simImgV(m,img,10,.35,.7))
   user  system elapsed
  0.037   0.000   0.037
```

More MCMC Examples

Monte Carlo Maximum Likelihood

- Suppose we have an exponential family likelihood

$$h(x|\theta) = c(\theta)e^{\theta x - \nu(x)} = c(\theta)\tilde{h}(x|\theta)$$

- In many problems $c(\theta)$ is not available in closed form:
  - Strauss process with $\theta = \log c$.
  - MRF image model with $\theta = \beta$.
- Geyer and Thompson (1992) write

$$\log \frac{h(x|\theta)}{h(x|\eta)} = \log \frac{\tilde{h}(x|\theta)}{\tilde{h}(x|\eta)} - \log E_\eta \left[\frac{\tilde{h}(X|\theta)}{\tilde{h}(X|\eta)}\right]$$

  This follows from the fact that

$$\frac{c(\theta)}{c(\eta)} = \frac{1}{c(\eta)\int \tilde{h}(x|\theta)dx} = \left(\int \frac{\tilde{h}(x|\theta)}{\tilde{h}(x|\eta)} c(\eta)\tilde{h}(x|\eta)dx\right)^{-1} = \left(E_\eta \left[\frac{\tilde{h}(X|\theta)}{\tilde{h}(X|\eta)}\right]\right)^{-1}$$

- Using a sample $x_1, \dots, x_N$ from $h(x|\eta)$ this can be approximated by

$$\log \frac{h(x|\theta)}{h(x|\eta)} \approx \log \frac{\tilde{h}(x|\theta)}{\tilde{h}(x|\eta)} - \log \frac{1}{N} \sum_{i=1}^{N} \frac{\tilde{h}(x_i|\theta)}{\tilde{h}(x_i|\eta)}$$

- The sample $x_1, \dots, x_N$ from $h(x|\eta)$ usually needs to be generated using MCMC methods.

Data Augmentation

- Suppose we have a problem where data Y, Z have joint density $f(y,z|\theta)$ but we only observe z.
- Suppose we have a prior density $f(\theta)$.
- The joint density of Y, Z, θ is then

  $$f(y,z,\theta) = f(y,z|\theta)f(\theta)$$

  and the joint posterior density of θ, y given z is

  $$f(\theta,y|z) = \frac{f(y,z|\theta)f(\theta)}{f(z)} \propto f(y,z|\theta)f(\theta)$$

- Suppose it is easy to sample from the conditional distribution of
  - the missing data y, given θ and the observed data z
  - the parameter θ given the complete data y, z.

  Then we can start with $\theta^{(0)}$ and for each $i = 1, 2, \ldots$
  - draw $y^{(i)}$ from $f(y|\theta^{(i-1)}, z)$
  - draw $\theta^{(i)}$ from $f(\theta|y^{(i)}, z)$.

  This is the *data augmentation algorithm* of Tanner and Wong (1987)
- The result is a Markov chain with stationary distribution $f(\theta,y|z)$
- If we discard the y values then we have a (dependent) sample from the marginal posterior density $f(\theta|z)$.
- In this alternating setting, the marginal sequence $\theta^{(i)}$ is a realization of a Markov chain with invariant distribution $f(\theta|z)$.

Probit Model for Pesticide Effectiveness

- Batches of 20 tobacco budworms were subjected to different doses of a pesticide and the number killed was recorded.

Dose	1	2	4	8	16	32
Died	1	4	9	13	18	20

- A probit model assumes that binary responses Z_i depend on covariate values x_i though the relationship

$$Z_i \sim \text{Bernoulli}(\Phi(\alpha + \beta(x_i - \overline{x})))$$

- A direct likelihood or Bayesian analysis is possible.
- An alternative is to assume that there are latent variables Y_i with

$$Y_i \sim \text{N}(\alpha + \beta(x_i - \overline{x}), 1)$$
$$Z_i = \begin{cases} 1 & \text{if } Y_i \geq 0 \\ 0 & \text{if } Y_i < 0 \end{cases}$$

- For this example assume a flat, improper, prior density.

- The full data posterior distribution is

$$f(\alpha,\beta|y) \propto \exp\left\{-\frac{n}{2}(\alpha-\overline{y})^2 - \frac{\sum(x_i-\overline{x})^2}{2}(\beta-\widehat{\beta})^2\right\}$$

  with

$$\widehat{\beta} = \frac{\sum(x_i-\overline{x})y_i}{\sum(x_i-\overline{x})^2}$$

  So α, β are independent given y and x, and

$$\begin{aligned}\alpha|y,z &\sim \mathrm{N}(\overline{y},1/n)\\ \beta|y,z &\sim \mathrm{N}(\widehat{\beta},1/\textstyle\sum(x_i-\overline{x})^2)\end{aligned}$$

- Given z, α, and β the Y_i are conditionally independent, and

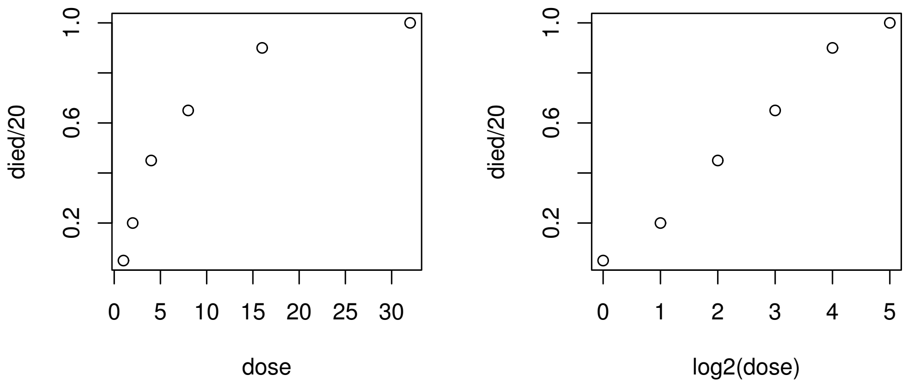
$$Y_i|z,\alpha,\beta \sim \begin{cases}\mathrm{N}(\alpha+\beta(x_i-\overline{x})^2,1) \text{ conditioned to be positive} & \text{if } z_i=1\\ \mathrm{N}(\alpha+\beta(x_i-\overline{x})^2,1) \text{ conditioned to be negative} & \text{if } z_i=0\end{cases}$$

  The inverse CDF's are

$$F^-(u|z_i,\mu_i) = \begin{cases}\mu_i+\Phi^{-1}(\Phi(-\mu_i)+u(1-\Phi(-\mu_i))) & \text{if } z_i=1\\ \mu_i+\Phi^{-1}(u\Phi(-\mu_i)) & \text{if } z_i=0\end{cases}$$

  with $\mu_i = \alpha+\beta(x_i-\overline{x})$.

- A plot of the proportion killed against dose is curved, but a plot against the logarithm is straight in the middle. So use $x = \log_2(\text{dose})$.



```
dose <- c(1, 2, 4, 8, 16, 32)
died <- c(1, 4, 9, 13, 18, 20)
x <- log2(dose)
```

- We need to generate data for individual cases:

```
xx <- rep(x - mean(x), each = 20)
z <- unlist(lapply(died,
                   function(x) c(rep(1, x), rep(0, 20 - x))
```

- We need functions to generate from the conditional distributions:

```
genAlpha <- function(y)
    rnorm(1, mean(y), 1 / sqrt(length(y)))
genBeta <- function(y, xx, sxx2)
    rnorm(1, sum(xx * y) / sxx2, 1 / sqrt(sxx2))
genY <- function(z, mu) {
    p <- pnorm(-mu)
    u <- runif(length(z))
    mu + qnorm(ifelse(z == 1, p + u * (1 - p), u * p))
}
```

- A function to produce a sample of parameter values by data augmentation is then defined by

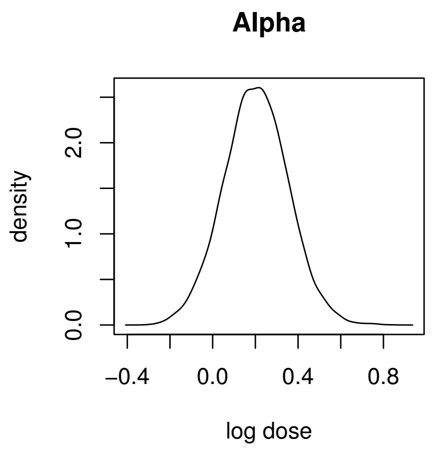
```
da <- function(z, alpha, beta, xx, N) {
    val <- matrix(0, nrow = N, ncol = 2)
    colnames(val) <- c("alpha", "beta")
    sxx2 <- sum(xx^2)
    for (i in 1 : N) {
        y <- genY(z, alpha + beta * xx)
        alpha <- genAlpha(y)
        beta <- genBeta(y, xx, sxx2)
        val[i,1] <- alpha
        val[i,2] <- beta
    }
    val
}
```

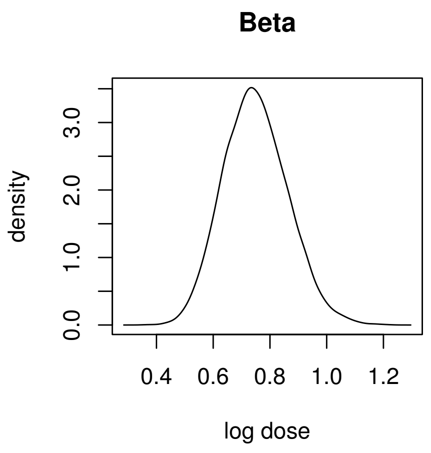
- Initial values are

```
alpha0 <- qnorm(mean(z))
beta0 <- 0
```

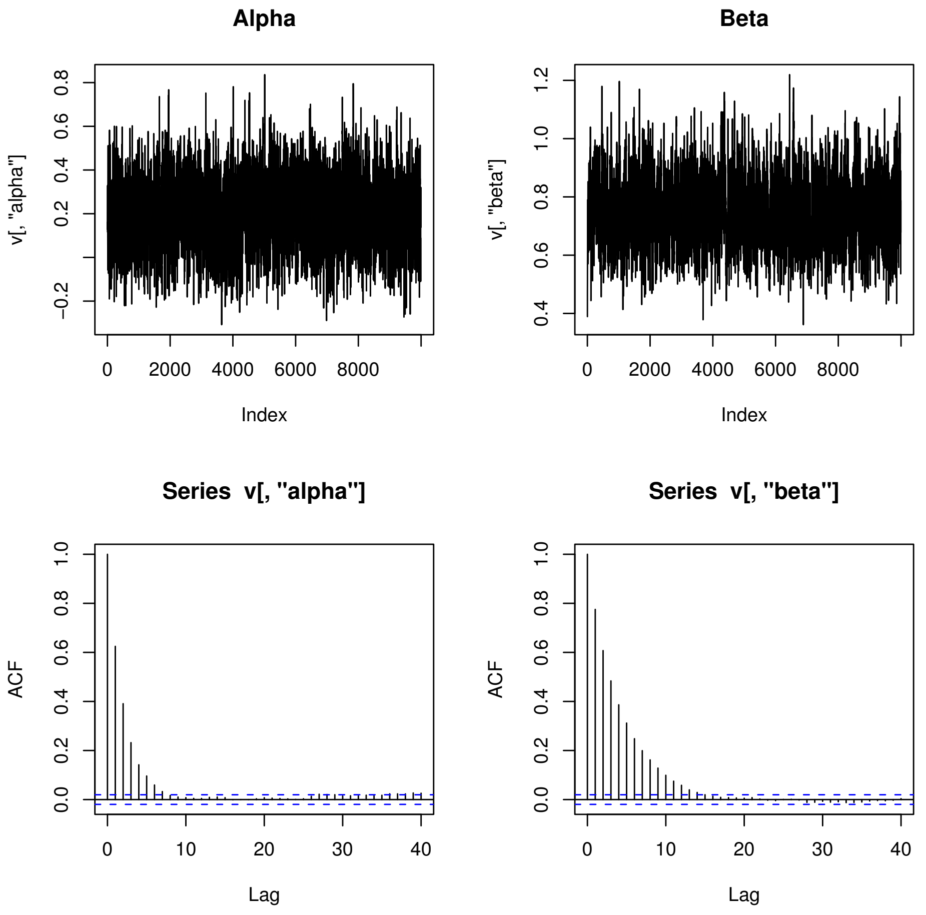
- A run of 10000:

```
v <- da(z, alpha0, beta0, xx, 10000)
> apply(v,2,mean)
    alpha      beta
0.2030763 0.7494578
> apply(v,2,sd)
    alpha      beta
0.1503186 0.1132519
```


Alpha
density
0.0
1.0
2.0
−0.4
0.0
0.4
0.8
log dose


Beta
density
0.0
1.0
2.0
3.0
0.4
0.6
0.8
1.0
1.2
log dose

- Some diagnostics:



Using a simple AR(1) model,

$$\mathrm{SD}(\overline{\alpha}) \approx \frac{\mathrm{SD}(\alpha|z)}{\sqrt{N}}\sqrt{\frac{1+\rho_\alpha}{1-\rho_\alpha}} = \frac{0.1503186}{100}\sqrt{\frac{1+0.65}{1-0.65}} = 0.003263778$$

$$\mathrm{SD}(\overline{\beta}) \approx \frac{\mathrm{SD}(\beta|z)}{\sqrt{N}}\sqrt{\frac{1+\rho_\beta}{1-\rho_\beta}} = \frac{0.1132519}{100}\sqrt{\frac{1+0.8}{1-0.8}} = 0.003397557$$

Approxiamte effective sample sizes:

$$\mathrm{ESS}(\overline{\alpha}) \approx N\left(\frac{1-\rho_\alpha}{1+\rho_\alpha}\right) = 10000\left(\frac{1-0.65}{1+0.65}\right) = 2121.212$$

$$\mathrm{ESS}(\overline{\beta}) \approx N\left(\frac{1-\rho_\beta}{1+\rho_\beta}\right) = 10000\left(\frac{1-0.8}{1+0.8}\right) = 1111.111$$

Practical Bayesian Inference

- In Bayesian inference we have
  - a prior density or PMF $f(\theta)$
  - a data density or PMF, or likelihood, $f(x|\theta)$
- We compute the posterior density or PMF as

$$f(\theta|x) = \frac{f(x|\theta)f(\theta)}{f(x)} \propto f(x|\theta)f(\theta)$$

- At this point, in principle, we are done.
- In practice, if $\theta = (\theta_1, \dots, \theta_p)$ then we want to compute things like
  - the posterior means $E[\theta_i|x]$ and variances $\text{Var}(\theta_i|x)$
  - the marginal posterior densities $f(\theta_i|x)$
  - posterior probabilities, such as $P(\theta_1 > \theta_2|x)$

  These are all integration problems.
- For a few limited likelihood/prior combinations we can compute these integrals analytically.
- For most reasonable likelihood/prior combinations analytic computation is impossible.

Numerical Integration

- For $p = 1$
  - we can plot the posterior density
  - we can compute means and probabilities by numerical integration
- General one dimensional numerical integration methods like
  - the trapezoidal rule
  - Simpson's rule
  - adaptive methods (as in `integrate` in R)

  often use $N \approx 100$ function evaluations.
- If $p = 2$ we can
  - plot the joint posterior density
  - compute marginal posterior densities by one dimensional numerical integrations and plot them
  - compute means and probabilities by iterated one dimensional numerical integration
- In general, iterated numerical integration requires N^p function evaluations.
- If a one dimensional f looks like

  $$f(x) \approx (\text{ low degree polynomial}) \times (\text{normal density})$$

  then *Gaussian quadrature* (Monahan, p. 268–271; Givens and Hoeting, Section 5.3) may work with $N = 3$ or $N = 4$.
  - This approach is used in Naylor and Smith (1984).
  - Getting the location and scale of the Gaussian right is critical.
  - Even 3^p gets very large very fast.

Large Sample Approximations

- If the sample size n is large,

$$\begin{aligned}
\widehat{\theta} &= \text{mode of joint posterior density } f(\theta|x)\\
H &= -\nabla^2_\theta \log f(\widehat{\theta}|x)\\
&= \text{Hessian (matrix of second partial derivatives) of } -\log f \text{ at } \widehat{\theta}
\end{aligned}$$

  then under often reasonable conditions

$$f(\theta|x) \approx \mathrm{MVN}_p(\widehat{\theta}, H^{-1})$$

  The relative error in the density approximation is generally of order $O(n^{-1/2})$ near the mode.

- More accurate second order approximations based on Laplace's method are also sometimes available.

- To approximate the marginal posterior density of θ_1, compute

$$\begin{aligned}
\widehat{\theta}_2(\theta_1) &= \mathrm{argmax}_{\theta_2} f(\theta_1, \theta_2|x)\\
H(\theta_1) &= -\nabla^2_{\theta_2} \log f(\theta_1, \widehat{\theta}_2(\theta_1)|x)
\end{aligned}$$

  Then

$$\hat{f}(\theta_1|x) \propto \sqrt{\det H(\theta_1)} f(\theta_1, \widehat{\theta}_2(\theta_1)|x)$$

  approximates $f(\theta_1|x)$ with a relative error near the mode of order $O(n^{-3/2})$.

- The component $f(\theta_1, \widehat{\theta}_2(\theta_1)|x)$ is analogous to the profile likelihood.

- The term $\sqrt{\det H(\theta_1)}$ adjusts for differences in spread in the parameter being maximized out.

Monte Carlo Methods

- Early Monte Carlo approaches used importance sampling.
  - Usually some form of multivariate t is used to get heavy tails and bounded weights.
  - Guaranteeing bounded weights in high dimensions is very hard.
  - Even bounded weights may have too much variation to behave well.
- Gelfand and Smith (1989) showed that many joint posterior distributions have simple *full conditional distributions*

$$f(\theta_i|x,\theta_1,\ldots,\theta_{i-1},\theta_{i+1},\ldots,\theta_p)$$

- These can be sampled using a Markov chain, called a Gibbs sampler.

- The *systematic scan* Gibbs sampler starts with some initial values $\theta_1^{(0)},\ldots,\theta_p^{(0)}$ and then for each k generates

$$\begin{aligned}
\theta_1^{(k+1)} &\sim f(\theta_1|x,\theta_2=\theta_2^{(k)},\ldots,\theta_p=\theta_p^{(k)})\\
&\vdots\\
\theta_i^{(k+1)} &\sim f(\theta_i|x,\theta_1=\theta_1^{(k+1)},\ldots,\theta_{i-1}=\theta_{i-1}^{(k+1)},\theta_{i+1}=\theta_{i+1}^{(k)},\ldots,\theta_p=\theta_p^{(k)})\\
&\vdots\\
\theta_p^{(k+1)} &\sim f(\theta_p|x,\theta_1=\theta_1^{(k+1)},\ldots,\theta_{p-1}=\theta_{p-1}^{(k+1)})
\end{aligned}$$

- The *random scan* Gibbs sampler picks an index $i=1,\ldots,p$ at random and updates that component from its full conditional distribution.
- Many other variations are possible.
- All generate a Markov chain $\theta^{(0)},\theta^{(1)},\theta^{(2)},\ldots$, with invariant distribution $f(\theta|x)$.

Example: Pump Failure Data

Numbers of failures and times of observation for 10 pumps in a nuclear power plant:

Pump	1	2	3	4	5	6	7	8	9	10
Failures	5	1	5	14	3	19	1	1	4	22
Time	94.32	15.72	62.88	125.76	5.24	31.44	1.05	1.05	2.10	10.48

Times are in 1000's of hours.

- Suppose the failures follow Poisson processes with rates λ_i for pump i; so the number of failures on pump i is $X_i \sim \text{Poisson}(\lambda_i t_i)$.
- The rates λ_i are drawn from a $\text{Gamma}(\alpha, 1/\beta)$ distribution.
- Assume $\alpha = 1.8$
- Assume $\beta \sim \text{Gamma}(\gamma, 1/\delta)$ with $\gamma = 0.01$ and $\delta = 1$.
- The joint posterior distribution of $\lambda_1, \dots, \lambda_{10}, \beta$ is

$$\begin{aligned}
f(\lambda_1,\dots,\lambda_{10},\beta|t_1,\dots,t_{10},x_1,\dots,x_{10}) &\propto \left(\prod_{i=1}^{10}(\lambda_i t_i)^{x_i} e^{-\lambda_i t_i}\lambda_i^{\alpha-1}\beta^{\alpha}e^{-\beta\lambda_i}\right)\beta^{\gamma-1}e^{-\delta\beta}\\
&\propto \left(\prod_{i=1}^{10}\lambda_i^{x_i+\alpha-1}e^{-\lambda_i(t_i+\beta)}\right)\beta^{10\alpha+\gamma-1}e^{-\delta\beta}\\
&\propto \left(\prod_{i=1}^{10}\lambda_i^{x_i+\alpha-1}e^{-\lambda_i t_i}\right)\beta^{10\alpha+\gamma-1}e^{-(\delta+\sum_{i=1}^{10}\lambda_i)\beta}
\end{aligned}$$

- Full conditionals:

$$\begin{aligned}
\lambda_i|\beta,t_i,x_i &\sim \text{Gamma}(x_i+\alpha,(t_i+\beta)^{-1})\\
\beta|\lambda_i,t_i,x_i &\sim \text{Gamma}(10\alpha+\gamma,(\delta+\sum\lambda_i)^{-1})
\end{aligned}$$

- It is also possible to integrate out the λ_i analytically to get

$$f(\beta|t_i,x_i) \propto \left(\prod_{i=1}^{10}(t_i+\beta)^{x_i+\alpha}\right)^{-1} \beta^{10\alpha+\gamma-1}e^{-\delta\beta}$$

  This can be simulated by rejection or RU sampling; the λ_i can then be sampled conditionally given β.

- Suppose α is also unknown and given an exponential prior distribution with mean one.

- The joint posterior distribution is then

$$f(\lambda_1,\ldots,\lambda_{10},\beta,\alpha|t_i,x_i) \propto \left(\prod_{i=1}^{10}\lambda_i^{x_i+\alpha-1}e^{-\lambda_i(t_i+\beta)}\right)\beta^{10\alpha+\gamma-1}e^{-\delta\beta}\frac{e^{-\alpha}}{\Gamma(\alpha)^{10}}$$

- The full conditional density for α is

$$f(\alpha|\beta,\lambda_i,t_i,x_i) \propto \frac{\left(\beta^{10}e^{-1}\prod_{i=1}^{10}\lambda_i\right)^{\alpha}}{\Gamma(\alpha)^{10}}$$

  This is not a standard density

- This density is log-concave and can be sampled by adaptive rejection sampling.

- Another option is to use the Metropolis-Hastings algorithm.

Metropolis-Hasting Algorithm

- Introduced in N, Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (1953), "Equations for state space calculations by fast computing machines," *Journal of Chemical Physics*.
- Extended in Hastings (1970), "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*.
- Suppose we want to sample from a density f.
- We need a family of *proposal distributions* $Q(x,dy)$ with densities $q(x,y)$.
- Suppose a Markov chain with stationary distribution f is currently at $X^{(i)} = x$. Then
  - Generate a *proposal* Y for the new location by drawing from the density $q(x,y)$.
  - Accept the proposal with probability

    $$\alpha(x,y) = \min\left\{\frac{f(y)q(y,x)}{f(x)q(x,y)}, 1\right\}$$

    and set $X^{(i+1)} = Y$.
  - Otherwise, reject the proposal and remain at x, i.e. set $X^{(i+1)} = x$.
- The resulting transition densities satisfy the *detailed balance equations* for $x \neq y$ and initial distribution f:

  $$f(x)q(x,y)\alpha(x,y) = f(y)q(y,x)\alpha(y,x)$$

  The chain is therefore reversible and has invariant distribution f.

Symmetric Proposals

- Suppose $q(x,y) = q(y,x)$. Then

$$\alpha(x,y) = \min\left\{\frac{f(y)}{f(x)}, 1\right\}$$

  So:

  - If $f(y) \geq f(x)$ then the proposal is accepted.
  - If $f(y) < f(x)$ then the proposal is accepted with probability

$$\alpha(x,y) = \frac{f(y)}{f(x)} < 1$$

- Symmetric proposals are often used in the *simulated annealing* optimization method.
- *Symmetric random walk proposals* with $q(x,y) = g(y-x)$ where g is a symmetric density are often used.
- Metropolis et al. (1953) considered only the symmetric proposal case.
- Hastings (1970) introduced the more general approach allowing for non-symmetric proposals.

Independence Proposals

- Suppose $q(x,y) = g(y)$, independent of x. Then

$$\alpha(x,y) = \min\left\{\frac{f(y)g(x)}{f(x)g(y)}, 1\right\} = \min\left\{\frac{w(y)}{w(x)}, 1\right\}$$

  with $w(x) = f(x)/g(x)$.

- This is related to importance sampling:
  - If a proposal y satisfies $w(y) \geq w(x)$ then the proposal is accepted.
  - If $w(y) < w(x)$ then the proposal may be rejected.
  - If the weight $w(x)$ at the current location is very large, meaning $g(x)$ is very small compared to $f(x)$, then the chain will remain at x for many steps to compensate.

Metropolized Rejection Sampling

- Suppose $h(x)$ is a possible envelope for f with $\int h(x)dx < \infty$.
- Suppose we sample
  - Y from h
  - U uniformly on $[0, h(Y)]$

  until $U < f(Y)$.
- Then the resulting Y has density

$$g(y) \propto \min(h(x), f(x))$$

- Using g as a proposal distribution, the Metropolis acceptance probability is

$$\begin{aligned}
\alpha(x,y) &= \min\left\{\frac{f(y)\min(h(x),f(x))}{f(x)\min(h(y),f(y))}, 1\right\} \\
&= \min\left\{\frac{\min(h(x)/f(x),1)}{\min(h(y)/f(y),1)}, 1\right\} \\
&= \begin{cases} 1 & \text{if } f(x) \leq h(x) \\ h(x)/f(x) & \text{if } f(x) > h(x) \text{ and } f(y) \leq h(y) \\ \min\left\{\frac{f(y)h(x)}{f(x)h(y)}, 1\right\} & \text{otherwise} \end{cases} \\
&= \min\left\{\frac{h(x)}{f(x)}, 1\right\} \min\left\{\max\left\{\frac{f(y)}{h(y)}, 1\right\}, \max\left\{\frac{f(x)}{h(x)}, 1\right\}\right\} \\
&\geq \min\left\{\frac{h(x)}{f(x)}, 1\right\}
\end{aligned}$$

- If h is in fact an envelope for f, then $\alpha(x,y) \equiv 1$ and the algorithm produces independent draws from f.
- If h is not an envelope, then the algorithm occasionally rejects proposals when the chain is at points x where the envelope fails to hold.
- The dependence can be very mild if the failure is mild; it can be very strong if the failure is significant.

Metropolis-Within-Gibbs

- Suppose $f(x) = f(x_1, \ldots, x_p)$
- The Metropolis-Hastings algorithm can be used on the entire vector x.
- The Metropolis-Hastings algorithm can also be applied to one component of a vector using the full conditional density

  $$f(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_p)$$

  as the target density.
- This approach is sometimes called *Metropolis-within-Gibbs*.
- This is a misnomer: this is what Metropolis et al. (1953) did to sample from the equilibrium distribution of n gas molecules.

Example: Pump Failure Data

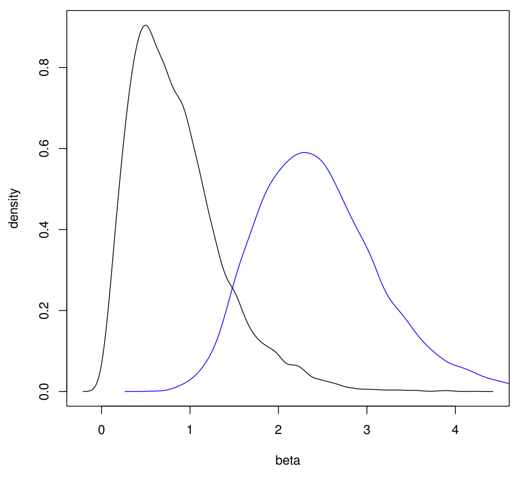
- Suppose α has an exponential prior distribution with mean 1.
- The full conditional density for α is

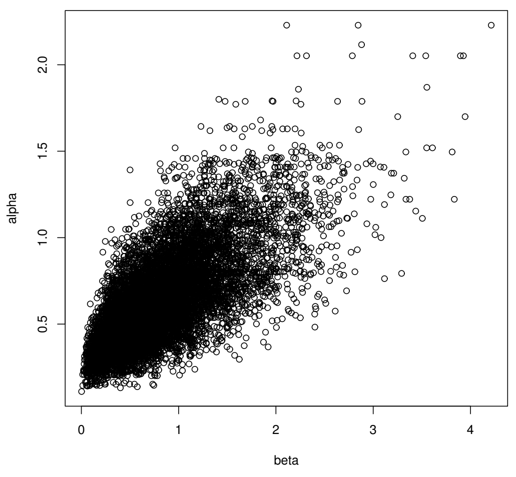
$$f(\alpha|\beta,\lambda_i,t_i,x_i) \propto \frac{\left(\beta^{10}e^{-1}\prod_{i=1}^{10}\lambda_i\right)^{\alpha}}{\Gamma(\alpha)^{10}}$$

- To use a random walk proposal it is useful to make the support of the distribution be the whole real line.
- The full conditional density of $\log\alpha$ is

$$f(\log\alpha|\beta,\lambda_i,t_i,x_i) \propto \frac{\left(\beta^{10}e^{-1}\prod_{i=1}^{10}\lambda_i\right)^{\alpha}\alpha}{\Gamma(\alpha)^{10}}$$

- Using a normal random walk proposal requires choosing a standard deviation; 0.7 seems to work reasonably well.
- We can use a single Metropolis step or several.





Gibbs Sampler in R for Pump Data

```
fail <- c(5, 1, 5, 14, 3, 19, 1, 1, 4, 22)
time <- c(94.32, 15.72, 62.88, 125.76, 5.24,
          31.44, 1.05, 1.05, 2.10, 10.48)
alpha <- 1.8
gamma <- 0.01
delta <- 1
pump <- function(alpha, beta, N, d = 1, K = 1) {
    v <- matrix(0, ncol=12, nrow=N)
    for (i in 1:N) {
        lambda <- rgamma(10, fail + alpha, rate = time + beta)
        beta <- rgamma(1, 10 * alpha + gamma, rate = delta + sum(lambda))
        b <- (10 * log(beta) + sum(log(lambda)) - 1)
        for (j in 1:K) {
            newAlpha <- exp(rnorm(1, log(alpha), d))
            logDensRatio <-
                ((newAlpha - alpha) * b + log(newAlpha) - log(alpha) +
                 10 * (lgamma(alpha) - lgamma(newAlpha)))
            if (is.finite(logDensRatio) &&
                log(runif(1)) < logDensRatio)
                alpha <- newAlpha
        }
        v[i,1:10] <- lambda
        v[i,11] <- beta
        v[i,12] <- alpha
    }
    v
}
```

Markov Chain Theory: Discrete State Space

- A sequence of random variables $X_0, X_1, X_2, \dots$ with values in a finite or countable set E is a Markov chain if for all n

  $$P(X_{n+1} = j | X_n = i, X_{n-1}, X_{n-2}, \dots, X_0) = P(X_{n+1} = j | X_n = i)$$

  i.e. given the present, the future and the past are independent.

- A Markov chain is *time homogeneous* if

  $$P(i, j) = P(X_{n+1} = j | X_n = i)$$

  does not depend on n. $P(i, j)$ is the *transition matrix* of the Markov chain.

- A transition matrix satisfies

  $$\sum_{j \in E} P(i, j) = 1$$

  for all $i \in E$.

- The distribution of X_0 is called the *initial distribution* of a Markov chain.

- The n step transition matrix is

  $$P^n(i,j) = P(X_n = j | X_0 = i)$$

  The $n+m$ step transition matrix satisfies

  $$P^{n+m} = P^n P^m$$

  That is,

  $$P^{n+m}(i,j) = \sum_{k \in E} P^n(i,k) P^m(k,j)$$

  for all $i, j \in E$.

- A distribution π is an *invariant distribution* or a *stationary distribution* for a Markov transition matrix P if

  $$\pi(j) = \sum_{i \in E} \pi(i) P(i,j)$$

  for all i. These equations are sometimes called the *flow balance equations*

Reversibility

- A transition matrix is *reversible* with respect to a distribution π if
$$\pi(i)P(i,j) = \pi(j)P(j,i)$$
for every pair i, j. These equations are called the *detailed balance* equations.

- If P is reversible with respect to π, then π is a stationary distribution for P:
$$\sum_{i \in E} \pi(i)P(i,j) = \sum_{i \in E} \pi(j)P(j,i) = \pi(j) \sum_{i \in E} P(j,i) = \pi(j)$$

- If P is reversible with respect to π and X_0 has initial distribution π, then the vectors
$$(X_0, X_1, X_2, \dots, X_{n-2}, X_{n-1}, X_n)$$
and
$$(X_n, X_{n-1}, X_{n-2}, \dots, X_2, X_1, X_0)$$
have the same joint distributions.

- Reversible transition matrices have many nice properties, including real eigenvalues.

Convergence

- A Markov chain is irreducible if for each pair of states i, j there is an integer n such that
$$P^n(i, j) > 0$$
i.e. if each state can be reached with positive probability from any other state.

- The time to reach state i is
$$\tau_i = \min\{n \geq 1 : X_n = i\}$$
with $\tau_i = \infty$ if $X_n \neq i$ for all n.

- An irreducible Markov chain falls into one of three categories:
  - *Transient:* $P(\tau_i = \infty | X_0 = j) > 0$ for all $i, j \in E$.
  - *Null recurrent:* $P(\tau_i = \infty | X_0 = j) = 0$ and $E[\tau_i | X_0 = j] = \infty$ for all $i, j \in E$.
  - *Positive recurrent:* $P(\tau_i = \infty | X_0 = j) = 0$ and $E[\tau_i | X_0 = j] < \infty$ for all $i, j \in E$.

- If an irreducible Markov chain is transient or null recurrent then it does not have a proper invariant distribution.

- If an irreducible Markov chain is positive recurrent, then
  - it has a unique invariant distribution π
  - for any $i \in E$
  $$\frac{1}{n}\sum_{k=1}^{n} P^k(i, j) \to \pi(j)$$
  - if h is a real-valued function on E such that
  $$\sum_{i \in E} |h(i)|\pi(i) < \infty$$
  then almost surely
  $$\frac{1}{n}\sum_{k=1}^{n} h(X_k) \to \pi h = \sum_{i \in E} h(i)\pi(i)$$

- if the chain is also aperiodic, then

$$P^n(i,j) \Longrightarrow \pi(j)$$

  for all $i, j \in E$. In this case π is an *equilibrium distribution* of the chain.

Different Points of View

- In applied probability problems we usually
  - verify that a chain is irreducible
  - verify that a chain is positive recurrent
  - conclude that a unique stationary distribution exists
  - compute the unique stationary distribution π
  - verify that the chain is aperiodic
  - use π to approximate the distribution of X_n
- In Markov chain Monte Carlo
  - we know by construction that a proper stationary distribution π exists
  - we verify that the chain is irreducible
  - we conclude that the chain must be positive recurrent (since it cannot be transient or null recurrent) and therefore π is the unique stationary distribution
  - we approximate expectations under π by sample path averages
  - aperiodicity is usually not important

Markov Chain Theory: General State Spaces

- Let E be an arbitrary set and $\mathscr{E}$ a countably generated sigma-algebra on E.
- A sequence of $(E,\mathscr{E})$-valued random variables is a time homogeneous Markov chain with transition kernel $P(x,dy)$ if

  $$P(X_{n+1} \in A | X_n, X_{n-1}, \ldots, X_0) = P(X_n, A)$$

  for all $A \in \mathscr{E}$.
- A Markov transition kernel is a function $P(\cdot,\cdot)$ such that
  - $P(x,\cdot)$ is a probability on $(E,\mathscr{E})$ for each $x \in E$.
  - $P(\cdot,A)$ is a $\mathscr{E}$-measurable function for each $A \in \mathscr{E}$.
- The n-step transition kernel of a Markov chain is

  $$P^n(x,A) = P(X_n \in A | X_0 = x)$$

  and satisfies

  $$P^{n+m}(x,A) = \int P^n(x,dy) P^m(y,A)$$

  for all $x \in E$ and all $A \in \mathscr{A}$.
- A distribution π is invariant for a Markov transition kernel P if

  $$\pi(A) = \int \pi(dy) P(y,A)$$

  for all $A \in \mathscr{E}$.

Reversibility

- A transition kernel P is reversible with respect to a distribution π if

$$\pi(dx)P(x,dy) = \pi(dy)P(y,dx)$$

  i.e. these two bivariate distributions must be identical.

- If P is reversible with respect to π then P is invariant with respect to π:

$$\begin{aligned}
\int_{x\in E} \pi(dx)P(x,A) &= \int_{x\in E}\int_{y\in A} \pi(dx)P(x,dy)\\
&= \int_{x\in E}\int_{y\in A} \pi(dy)P(y,dx)\\
&= \int_{y\in A} \pi(dy)\int_{x\in E} P(y,dx)\\
&= \int_{y\in A} \pi(dy)\\
&= \pi(A)
\end{aligned}$$

Convergence

- A Markov chain with transition kernel P is irreducible with respect to a sigma-finite measure ν if for every $x \in E$ and every $A \in \mathscr{E}$ with $\nu(A) > 0$ there exists an integer n such that $P^n(x,A) > 0$
- A Markov chain is irreducible if it is irreducible with respect to ν for some sigma-finite ν.
- The standard definition of irreducibility for discrete state spaces corresponds to irreducibility with respect to counting measure for general state spaces.
- An irreducible Markov chain is either transient, null recurrent, or positive recurrent.
- An irreducible Markov chain is positive recurrent if and only if it has a proper stationary distribution π.
- Essentially all Markov chains used in MCMC that are recurrent are also Harris recurrent.
- If an irreducible Markov chain is positive recurrent, then
  - it has a unique stationary distribution π
  - if the chain is Harris recurrent, then

    $$\sup_{A \in \mathscr{E}} \left| \frac{1}{n} \sum_{k=1}^{n} P^k(x,A) - \pi(A) \right| \to 0$$

    for all x
  - if the chain is Harris recurrent, h is real-valued, $\mathscr{E}$-measurable, and $\pi|h| = \int |h(x)|\pi(dx) < \infty$, then for any initial distribution

    $$\frac{1}{n} \sum_{k=1}^{n} h(X_k) \to \pi h = \int h(x)\pi(dx)$$
  - if the chain is Harris recurrent and aperiodic, then

    $$\sup_{A \in \mathscr{E}} |P^n(x,A) - \pi(A)| \to 0$$

    for all x.

Rates of Convergence

- A Markov chain is *geometrically ergodic* if there exists a nonnegative function $M(x)$ with $\pi M < \infty$ and a constant $\lambda < 1$ such that

$$\sup_{A \in \mathscr{E}} |P^n(x,A) - \pi(A)| \leq M(x)\lambda^n$$

  for all $x \in E$ and all integers $n \geq 1$.

- A Markov chain is *uniformly ergodic* if there exists a finite constant M and a constant $\lambda < 1$ such that

$$\sup_{A \in \mathscr{E}} |P^n(x,A) - \pi(A)| \leq M\lambda^n$$

  for all $x \in E$ and all integers $n \geq 1$.

- Many MCMC samplers are geometrically ergodic; relatively few are uniformly ergodic.

- Restricting parameters to a compact set can often make a chain uniformly ergodic.

Central Limit Theorems

- Suppose $\int h(x)^2 \pi(dx) < \infty$ and let
$$\overline{h}_n = \frac{1}{n} \sum_{k=1}^{n} h(X_k)$$
Let
$$\tau_n = n \text{Var}_\pi(\overline{h}_n)$$
and let
$$\tau = \lim_{n \to \infty} \tau_n = \text{Var}_\pi(h(X_0)) + 2 \sum_{k=1}^{\infty} \text{Cov}_\pi(h(X_k), h(X_0))$$
if the limit exists.
- Suppose the Markov chain is uniformly ergodic. Then the limit τ exists, is finite, and $\sqrt{n}(\overline{h} - \pi h)$ converges in distribution to a $\mathrm{N}(0, \tau)$ random variable.
- Suppose the Markov chain is Harris recurrent and geometrically ergodic and that $\int |h(x)|^{2+\varepsilon} \pi(dx) < \infty$ for some $\varepsilon > 0$. Then the limit τ exists, is finite, and $\sqrt{n}(\overline{h} - \pi h)$ converges in distribution to a $\mathrm{N}(0, \tau)$ random variable.
- If the Markov chain is reversible, Harris recurrent, and geometrically ergodic then $\pi(h^2) < \infty$ is sufficient for a CLT.
- Suppose the Markov chain is reversible. Then the limit τ exists but may be infinite. If the limit is finite, then $\sqrt{n}(\overline{h} - \pi h)$ converges in distribution to a $\mathrm{N}(0, \tau)$ random variable.
- The asymptotic variance τ can be written as
$$\tau = \text{Var}_\pi(h(X_0)) \left[1 + 2 \sum_{k=1}^{\infty} \rho_k(h) \right]$$
with
$$\rho_k(h) = \text{Corr}_\pi(h(X_k), h(X_0))$$
To use a central limit theorem we need to
  - be confident that it is valid
  - be able to estimate τ

Summary of Markov Chain Theory

Suppose $X_1, X_2, \ldots$ is a Markov chain on a state space E with invariant distribution π and h is a function such that

$$\int h(x)^2 \pi(dx) < \infty$$

- *Law of large numbers:* If the chain is irreducible, i.e. can get from any initial point to, or close to, any other point in E, then π is an equilibrium distribution and

  $$\overline{h}_n = \frac{1}{n} \sum_{i=1}^{n} h(X_i) \to \pi h = \int h(x) \pi(dx)$$

  almost surely.

- *Central limit theorem:* Under reasonable conditions, $\sqrt{n}(\overline{h}_n - \pi h)$ converges in distribution to a $\mathrm{N}(0, \tau)$ random variable with

  $$\tau = \mathrm{Var}_\pi(h(X_0)) \left[1 + 2 \sum_{k=1}^{\infty} \rho_k(h) \right]$$

  with

  $$\rho_k(h) = \mathrm{Corr}_\pi(h(X_k), h(X_0))$$

  and $X_0 \sim \pi$.

To use the central limit theorem we need to be able to estimate τ.

Output Analysis

- Simulation output analysis deals mainly with
  - estimation using simulation output
  - estimating variances of simulation estimators
  - assessing convergence, initialization bias, initial transients

  based on data produced by simulations
- General characteristics of such data:
  - Simulation run lengths are often very long.
  - There is usually dependence within runs.
  - Usually runs settle down to some form of stationarity
- Software:
  - CODA (Best, Cowles, and Vines)
    - Developed for S-PLUS, mainly for BUGS output
    - R port by Martyn Plummer; available on our workstations
  - BOA (B. Smith)
    - Major revision of CODA
    - Provides more methods
    - Available as an R package from CRAN and on our workstations
    - `http://www.public-health.uiowa.edu/boa/`

Simulation Estimators

Suppose a simulation produces values $X_1,\ldots,X_N$ and

$$\overline{X}_N = \frac{1}{N}\sum_{i=1}^{N} X_i \to \theta$$

- Usually we will estimate θ by $\overline{X}$
- If the process $X_1, X_2, \ldots,$ is stationary then usually

  $$\theta = E[X_i]$$

  Otherwise we usually have

  $$E[\overline{X}_N] \to \theta$$

  and often also $E[X_i] \to \theta$.
- In some cases we may be able to find a function g such that

  $$\frac{1}{N}\sum_{i=1}^{N} g(X_i) \to \theta$$

  Rao-Blackwellization is one approach that may produce such a function g.
- Often a Rao-Blackwellized estimator will have reduced variance, but this in *not* guaranteed with depended X_i,
- For the rest of this discussion, assume that X_i incorporates any such g.

Variance Estimation

- We often have, or hope to have,
$$\overline{X}_N \sim \text{AN}(\theta, \tau_N/N)$$
where
$$\tau_N = N\text{Var}(\overline{X}_N) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\text{Cov}(X_i, X_j)$$
- If the process $X_1, \ldots, X_N$ is stationary, which is usually the case in the limit, then
$$\begin{aligned}\sigma^2 &= \text{Var}(X_i)\\ \rho_k &= \text{Corr}(X_i, X_{i+k})\end{aligned}$$
do not depend on i. The value ρ_k is the *lag k autocorrelation* of $X_1, X_2, \ldots$.
- For a stationary process
$$\tau_N = \sigma^2\left(1 + 2\sum_{k=1}^{N-1}\left(1 - \frac{k}{N}\right)\rho_k\right)$$
- Typically,
$$\tau_N \to \tau = \sigma^2\left(1 + 2\sum_{k=1}^{\infty}\rho_k\right) = \sigma^2\sum_{k=-\infty}^{\infty}\rho_k$$
- Several methods are available for estimating τ, including
  - modeling the time series and using the estimated autocorrelations
  - estimating the spectral density at zero
  - batching
  - combinations
  - regenerative simulation
  - replication

Time Series Models

- We can fit an ARMA(p,q) model of the form

$$(X_i - \theta) = \sum_{j=1}^{p} \alpha_j (X_{i-j} - \theta) + \sum_{j=1}^{q} \beta_j \varepsilon_{i-j} + \varepsilon_i$$

  with the ε_i independent $\mathrm{N}(0, \sigma_\varepsilon^2)$.

- Then

$$\tau = \sigma_\varepsilon^2 \frac{\left(1 + \sum_{j=1}^{q} \beta_j\right)^2}{\left(1 - \sum_{j=1}^{p} \alpha_j\right)^2}$$

  τ can be estimated by plugging in estimates of α_j, β_j, and σ_ε^2.

- For the AR(1) model $\sigma^2 = \sigma_\varepsilon^2 / (1 - \alpha_1^2)$ and $\rho_1 = \alpha_1$; so

$$\tau = \sigma_\varepsilon^2 \frac{1}{(1-\alpha_1)^2} = \sigma^2 \frac{1 - \alpha_1^2}{(1-\alpha_1)^2} = \sigma^2 \frac{1+\alpha_1}{1-\alpha_1} = \sigma^2 \frac{1+\rho_1}{1-\rho_1}$$

  An estimate is

$$\widehat{\tau} = S^2 \frac{1 + r_1}{1 - r_1}$$

  with S^2 the sample variance and r_1 the lag one sample autocorrelation.

Spectral Density at the Origin

- The autocorrelation function satisfies

$$\sigma^2 \rho_k = 2 \int_0^{\pi} \cos(k\omega) f(\omega) d\omega$$

  where

$$f(\omega) = \frac{\sigma^2}{2\pi} \sum_{k=-\infty}^{\infty} \rho_k \cos(k\omega)$$

  is the *spectral density*.

- The spectral density is sometimes defined as a function on $[0, 1/2)$.
- The spectral density at zero is related to τ as

$$\tau = 2\pi f(0)$$

- Spectral densities are usually estimated by smoothing the *periodogram*, the Fourier transform of the sample autocovariance function.
- Smoothing flattens peaks, and there is typically a peak at zero.
- A number of methods are available for dealing with this.
- The CODA function `spectrum0` computes τ by some of these methods.
- It is usually a good idea to make `spectrum0` use batching by specifying a value for `max.length`

Batching and Replication

- If we replicate sampler runs independently R times then we have R independent sample averages and can use their sample standard deviation in computing a standard error for the overall average.
- Because of concerns about convergence we usually run only relatively few long chains; this does not provide enough degrees of freedom by itself.
- Batching is a form of within chain replication:
  - Suppose $N = KM$ and for $i = 1,\dots,K$ let

    $$\overline{X}_{M,i} = \frac{1}{M} \sum_{j=(i-1)M+1}^{iM} X_i$$

    Then $\overline{X}_{M,1},\dots,\overline{X}_{M,K}$ are means of successive batches of size M.
  - The overall mean is the mean of the batch means,

    $$\overline{X}_N = \frac{1}{N}\sum_{i=1}^{N} X_i = \frac{1}{K}\sum_{i=1}^{K}\overline{X}_{M,i}$$

  - If the batches are large enough, then the batch means will be approximately independent and normal, so t confidence intervals can be used.
- An estimate of τ based on assuming independent batch means is

  $$\widehat{\tau} = \frac{M}{K-1}\sum_{i=1}^{K}(\overline{X}_{M,i} - \overline{X})^2$$

- An alternative:
  - Choose a batch size so that an AR(1) model fits.
  - Estimate τ assuming the batch means follow an AR(1) model.
- Batching and replication can be combined.

Effective Sample Size and Sampler Efficiency

- If the sequence $X_1, \ldots, X_N$ were a random sample from a distribution π, then we would have
$$\text{Var}(\overline{X}_N) = \frac{\sigma^2}{N}$$
- With dependent sampling the variance is
$$\text{Var}(\overline{X}_N) \approx \frac{\tau}{N} = \frac{\sigma^2}{N} \sum_{k=-\infty}^{\infty} \rho_k$$
- So a sample of size N from a sampler with dependence is equivalent to a sample of
$$N_E = N\frac{\sigma^2}{\tau} = N\left(\sum_{k=-\infty}^{\infty} \rho_k\right)^{-1}$$
independent observations. N_E is sometimes called the *effective sample size*.
- By analogy to estimation theory the value
$$\frac{N_E}{N} = \left(\sum_{k=-\infty}^{\infty} \rho_k\right)^{-1}$$
is sometimes called the *asymptotic relative efficiency*, or just the efficiency, of the sampler.
- Thinking about the equivalent number of independent observations is often useful.
- Efficiencies need to be treated with caution: If sampler A is half as efficient but ten times as fast as sampler B, then sampler A is clearly better.
- In the physics literature the quantity
$$\mathscr{T}_{\text{int}} = \sum_{k=-\infty}^{\infty} \rho_k$$
is called the *integrated autocorrelation time*.

Example: Pump Data

Generate a run of 20000 with

```
v <- pump(1.8,1,20000)
colnames(v) <- c(paste("lambda", 1:10, sep=""), "beta", "alpha")
```

Using CODA we can get a summary as

```
> summary(mcmc(v))

Iterations = 1:20000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 20000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

            Mean      SD  Naive SE Time-series SE
lambda1  0.05968 0.02535 0.0001792      0.0001997
lambda2  0.09960 0.07879 0.0005572      0.0007174
lambda3  0.08841 0.03701 0.0002617      0.0003067
lambda4  0.11537 0.03028 0.0002141      0.0002605
lambda5  0.59891 0.31644 0.0022376      0.0026431
lambda6  0.60894 0.13760 0.0009730      0.0007948
lambda7  0.91144 0.75378 0.0053300      0.0061026
lambda8  0.89653 0.74452 0.0052646      0.0065248
lambda9  1.61012 0.78779 0.0055705      0.0061624
lambda10 2.00624 0.42777 0.0030248      0.0037128
beta     0.85510 0.53960 0.0038156      0.0102915
alpha    0.65329 0.28070 0.0019848      0.0066979

2. Quantiles for each variable:
...
```

The function `bmse` computes a standard error for the sample path mean using batching and, optionally, a time series adjustment based on an AR(1) model:

```
bmse <- function(x, M = 1, ts = FALSE) {
    bm <- apply(matrix(x, nrow = M), 2, mean)
    se <- sd(bm) / sqrt(length(bm))
    if (ts) {
        r <- acf(bm, plot = FALSE, lag = 1)$acf[2]
        se <- se * sqrt((1 + r) / (1 - r))
    }
    se
}
```

Results for β:

```
> bmse(v[, 11])
[1] 0.003815577
> bmse(v[, 11], ts = TRUE)
[1] 0.007686438
> sqrt(spectrum0(v[, 11], max.length = NULL)$spec / nrow(v))
[1] 0.006172704
> sqrt(spectrum0(v[, 11])$spec / nrow(v)) # max.length = 200
[1] 0.01029148
> bmse(v[, 11], M = 100) # 200 batches of size 100
[1] 0.01092254
> bmse(v[, 11], M = 100, ts = TRUE)
[1] 0.01041768
```

Results for α:

```
> bmse(v[,12])
[1] 0.001984824
> bmse(v[, 12], ts = TRUE)
[1] 0.007143383
> sqrt(spectrum0(v[, 12], max.length = NULL)$spec / nrow(v))
[1] 0.003597932
> sqrt(spectrum0(v[, 12])$spec / nrow(v))
[1] 0.006697929
> bmse(v[, 12], M=100)
[1] 0.007033827
> bmse(v[, 12], M = 100, ts = TRUE)
[1] 0.006781224
```

Convergence and Mixing

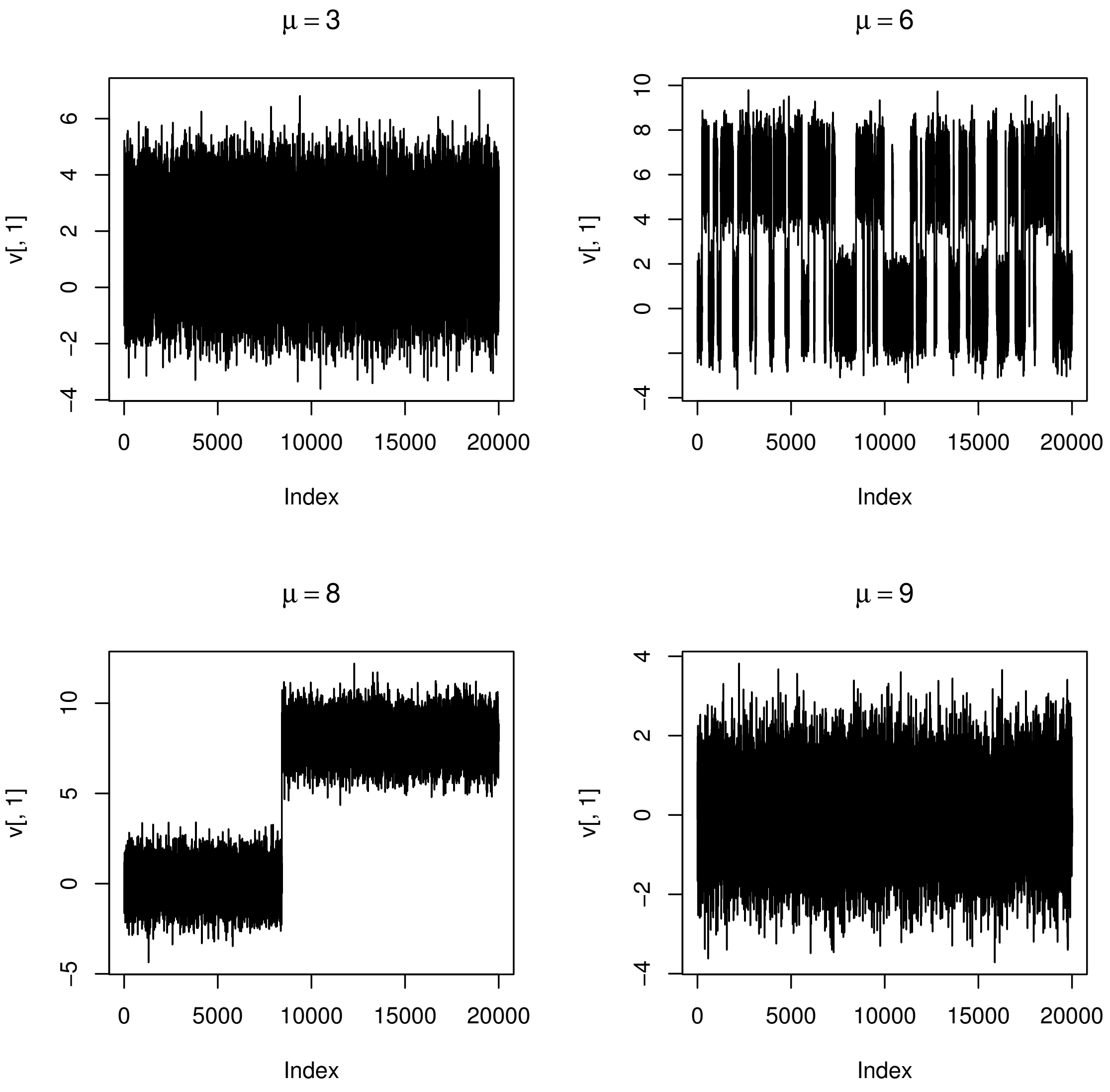
- There are many, many "convergence diagnostics" available in the literature. More are being developed.
- CODA and BOA provide a rich set of choices along with references. Monahan also has some references.
- The simulation literature is less preoccupied with this; a good example is C. Alexopoulos and D. Goldsman (2004) "To Batch or Not To Batch," *ACM Trans. on Modeling and Simulation* **14**, 76–114.
- Convergence is not the issue, *mixing* is:
  - Suppose you could, possibly at great cost, obtain one draw from the target distribution and use it to start a chain.
  - The chain would then be stationary.
  - On the other hand, the conditional chain, given the value of the draw, is not stationary.
  - Mixing conditions deal with how rapidly

$$\sup_{A,B} |P(X_n \in A, X_0 \in B) - P(X_n \in A)P(X_0 \in B)| \to 0$$

- If we knew the value of $\sigma^2 = \text{Var}_\pi(X_i)$ and of $\tau = \lim N \text{Var}(\overline{X}_N)$ then we would know how well the chain mixes and how to choose N.
- For independent sampling, $N = 10000$ is typically sufficient (computational uncertainty about the posterior mean is 1% of the uncertainty in the posterior distribution).

- Unfortunately we do not know σ^2 or τ.
- We can estimate them from one or more sample paths.
- We cannot be certain, by looking at sample paths alone, that the estimates are in the right ballpark.
- An example:

$$f(x,y) = \frac{1}{2}\varphi(x)\varphi(y) + \frac{1}{2}\varphi(x-\mu)\varphi(y-\mu)$$



Outline of Diagnostic Approaches

- Plot the data
- Single chain approaches
  - ANOVA on batches
  - Comparing beginning batch to end batch
  - Reverse chain and look for “out of control”
  - Detect how much to discard as “burn in”
  - Start far from center to see how quickly effect dissipates
- Multiple chain approaches
  - Look for consistency within and between chains
  - Use “over-dispersed” starting points
  - Can be run in parallel
- Dropping the “burn in:” bias/variance trade-off.

Convergence and Mixing Again

Mixing and convergence are two sides of the same issue:

$$E[h(X_0)g(X_n)] = E[h(X_0)E[g(X_n)|X_0]]$$

So mixing behavior such as

$$E[h(X_0)g(X_n)] \to E[h(X_0)]E[g(X_0)]$$

is essentially equivalent to

$$E[g(X_n)|X_0] \to E[g(X_0)]$$

Combining MCMC Samplers

- Some samplers may mix well but be very costly.
- Other samplers may be good at one kind of jump and not so good at others
- Suppose P_1 and P_2 are two transition kernels with common invariant distribution π. Then
  - P_1P_2 is a transition kernel with invariant distribution π.
  - $\alpha P_1 + (1-\alpha)P_2$ is a transition kernel with invariant distribution π for any $\alpha \in [0,1]$.
- For a *mixture kernel* $P = \alpha P_1 + (1-\alpha)P_2$ with $0 < \alpha < 1$
  - if either P_1 or P_2 is irreducible, then P is irreducible
  - if either P_1 or P_2 is uniformly ergodic then P is uniformly ergodic.
- Metropolis-Hasting kernels such as ones that
  - make an independence proposal from an approximation to the posterior distribution
  - propose a value reflected around an axis or through a point
  - propose a rotated version of the current state

  can often be useful.
- Combinations can be used to improve theoretical properties, such as make a chain reversible.

Improving Mixing and Convergence

Some possible strategies:

- transforming parameters
- blocking
- auxiliary variables
- heating, alternating, and reweighting

Many basic ideas from optimization also apply:

- make sure problem is reasonably scaled
- make sure problem is well conditioned
- eliminate range constraints where possible

Transformations

- For random walk MH samplers eliminating range constraints is useful.
- For variable at a time samplers, transforming to make variables nearly uncorrelated helps mixing.
  - For a hierarchical model

    $$\begin{aligned} \mu_i \mid \mu &\sim \text{independent N}(\mu, \sigma^2) \\ \mu &\sim \text{N}(0,1) \end{aligned}$$

    with $i = 1, \ldots, K$, we have

    $$\begin{aligned} \text{Corr}(\mu_i, \mu_j) &= 1/(1+\sigma^2) \\ \text{Corr}(\mu_i, \mu) &= 1/\sqrt{1+\sigma^2} \end{aligned}$$

    These will be close to one if σ^2 is small. But for

    $$\alpha_i = \mu_i - \mu$$

    the parameters $\alpha_1, \ldots, \alpha_K, \mu$ are independent.
- Linear transformations that cause many variables to be updated at once often make the cost of a single update much higher.

Blocking

- Sometimes several parameters can be updated together as a *block*.
- Exact sampling of a block is usually only possible if the joint distribution is related to the multivariate normal distribution.
  - Exact block sampling usually improves mixing.
  - The cost of sampling a block often increases with the square or cube of the block size.
- Block sampling with the Metropolis-Hastings algorithm is also possible.
  - The rejection probability usually increases with block size.
  - The cost of proposals often increases with the square or cube of the block size
- At times choosing overlapping blocks may be useful
- In some cases some level of blocking may be essential to ensure irreducibility.

Auxiliary Variables

- Suppose we want to sample from $\pi(x)$. We can expand this to a joint distribution
$$\pi(x,y) = \pi(x)\pi(y|x)$$
on x,y. This may be useful
  - if the joint distribution $\pi(x,y)$ is simple in some way
  - if the conditional distribution $\pi(x|y)$ is simple in some way
- Data augmentation is one example of this.
- If $\pi(x) = h(x)g(x)$ then it may be useful to take
$$Y|X = x \sim \mathrm{U}[0,g(x)]$$
Then
$$\pi(x,y) = h(x)1_{[0,g(x)]}(y)$$
In particular, if $h(x)$ is constant, then $\pi(x,y)$ is uniform on
$$\{(x,y) : 0 \leq y \leq g(x)\}$$
This is the idea used in rejection sampling.
- The conditional distribution of $X|Y = y$ is uniform on $\{x : \pi(x) \geq y\}$.
- Alternately sampling $X|Y$ and $Y|X$ from these uniform conditionals is called *slice sampling*.
- Other methods of sampling from this uniform distribution are possible:
  - Random direction (hit-and-run) sampling
  - Metropolis-Hastings sampling
- Ratio of uniforms sampling can also be viewed as an auxiliary variable method.

Example: Tobacco Budworms

- We previously used a latent variable approach to a probit regression model

$$
\begin{aligned}
Z_i &= \begin{cases} 1 & \text{if } Y_i \geq 0 \\ 0 & \text{if } Y_i < 0 \end{cases} \\
Y_i | \alpha, \beta, x &\sim \mathrm{N}(\alpha + \beta(x_i - \overline{x}), 1) \\
\alpha, \beta &\sim \text{flat non-informative prior distribution}
\end{aligned}
$$

- It is sometimes useful to introduce an additional non-identified parameter to improve mixing of the sampler.
- One possibility is to add a variance parameter:

$$
\begin{aligned}
Z_i &= \begin{cases} 1 & \text{if } Y_i \geq 0 \\ 0 & \text{if } Y_i < 0 \end{cases} \\
Y_i | \widetilde{\alpha}, \widetilde{\beta}, x &\sim \mathrm{N}(\widetilde{\alpha} + \widetilde{\beta}(x_i - \overline{x}), \sigma^2) \\
\widetilde{\alpha}, \widetilde{\beta} &\sim \text{flat non-informative prior distribution} \\
\sigma^2 &\sim \text{TruncatedInverseGamma}(\nu_0, a_0, T) \\
\alpha &= \widetilde{\alpha}/\sigma \\
\beta &= \widetilde{\beta}/\sigma
\end{aligned}
$$

- Several schemes are possible:
  - Generate Y, σ, $\widetilde{\alpha}$, and $\widetilde{\beta}$ from their full conditional distributions.
  - Generate σ from its conditional distribution given Y, i.e. integrating out $\widetilde{\alpha}$ and $\widetilde{\beta}$, and the others from their full conditional distributions.
  - Generate Y from it's conditional distribution given the identifiable α and β by generating a σ^* value from the prior distribution; generate the others from their full conditional distributions.
- Sample code is available in

`http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/worms.Rmd`

Swendsen-Wang Algorithm

- For the Potts model with C colors

$$\pi(x) \propto \exp\{\beta \sum_{(i,j)\in\mathscr{N}} 1_{x_i=x_j}\} = \prod_{(i,j)\in\mathscr{N}} \exp\{\beta 1_{x_i=x_j}\}$$

  with $\beta > 0$. $\mathscr{N}$ is the set of neighboring pairs.

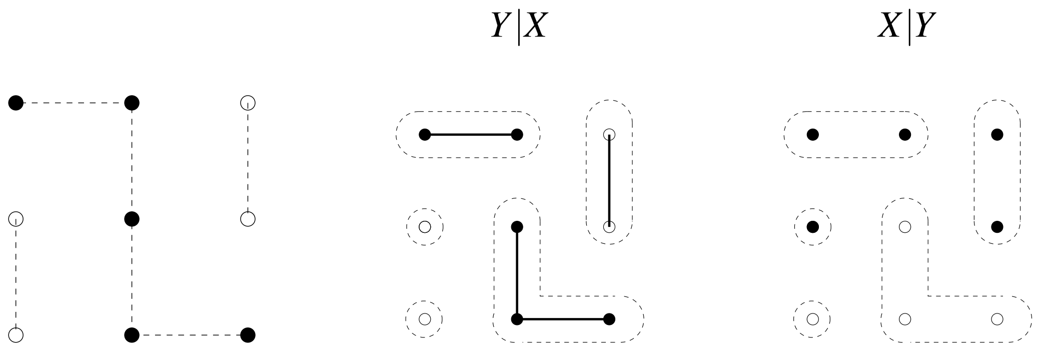
- Single-site updating is easy but may mix slowly if β is large.

- Taking $Y_{ij}|X = x$ for $(i,j) \in \mathscr{N}$ to be independent and uniform on $[0, \exp\{\beta 1_{x_i=x_j}\}]$ makes the joint density

$$\pi(x,y) \propto \prod_{(i,j)\in\mathscr{N}} 1_{[0,\exp\{\beta 1_{x_i=x_j}\}]}(y_{ij})$$

  - The conditional distribution of X given $Y = y$ is uniform on the possible configurations.
  - If $y_{ij} > 1$ then $x_i = x_j$; otherwise there are no further constraints.
  - The nodes can be divided into patches that are constrained to be the same color.
  - The colors of the patches are independent and uniform on the available colors.

  The algorithm that alternates generating $Y|X$ and $X|Y$ was introduced by Swendsen and Wang (1987).



- For models without an external field this approach mixes much more rapidly than single-site updating.

- Nott and Green (2004) propose a Bayesian variable selection algorithm based on the Swendsen-Wang approach.

Hamiltonian Monte Carlo

- Hamiltonian Monte Carlo (HMC) is an auxiliary variable method for producing a Markov chain with invariant distribution $f(\theta)$.
- HMC is also known ad Hybrid Monte Carlo.
- HMC requires that $f(\theta)$ be differentiable and that the gradient of $\log f(\theta)$ be computable.
- A motivation for the method:
  - View θ as the position of an object on a surface, with potential energy $-\log f(\theta)$.
  - Add a random momentum vector r, with kinetic energy $\frac{1}{2} r \cdot r$.
  - Compute where the object will be after time T by solving the differential equation of Hamiltonian dynamics.
  - The numerical solution uses a discrete approximation with L steps of size ε, with $T = \varepsilon L$.
- The random momentum values are sampled as independent standard normals.
- The algorithm produces a Markov chain with invariant density

$$h(\theta, r) \propto f(\theta) \exp\left\{-\frac{1}{2} r \cdot r\right\}.$$

- If the differential equation is solved exactly then the θ, r pair moves along contours of the energy surface $-\log h(\theta, r)$.
- With discretization this is not exactly true, and Metropolis Hasting step is used to correct for discretization errors.
- With a good choice of $T = \varepsilon L$ the algorithm can take very large steps and mix much better than a random walk Metropolis algorithm or simple Gibbs sampler.
- ε has to be chosen to be large enough to move a reasonable distance but small enough to keep the acceptance probability from becoming too small.

- The basic algorithm:

```
Given $\theta^0, \varepsilon, L, \mathscr{L}, M$
**for** $m = 1$ to $M$ **do**
    Sample $r \sim \mathrm{N}(0, I)$
    Set $\tilde{\theta}, \tilde{r} \leftarrow \mathrm{Leapfrog}(\theta^{m-1}, r, \varepsilon, L)$
    With probability $\alpha = \min\left\{1, \frac{\exp\{\mathscr{L}(\tilde{\theta}) - \frac{1}{2}\tilde{r}\cdot\tilde{r}\}}{\exp\{\mathscr{L}(\theta^{m-1}) - \frac{1}{2}r^0\cdot r^0\}}\right\}$ set $\theta^m \leftarrow \tilde{\theta}$
    Otherwise, set $\theta^m \leftarrow \theta^{m-1}$
**end for**

**function** Leapfrog($\theta, r, \varepsilon, L$)
    **for** $i = 1$ to $L$ **do**
        Set $r \leftarrow (\varepsilon/2)\nabla_\theta \mathscr{L}(\theta)$          ▷ half step for $r$
        Set $\theta \leftarrow \theta + \varepsilon r$                               ▷ full step for $\theta$
        Set $r \leftarrow (\varepsilon/2)\nabla_\theta \mathscr{L}(\theta)$          ▷ another half step for $r$
    **end for**
    **return** $\theta, -r$
**end function**
```

- The Leapfrog step produces a deterministic proposal $(\tilde{\theta},\tilde{r}) = \text{Leapfrog}(\theta,r)$.
- It is reversible: $(\theta,r) = \text{Leapfrog}(\tilde{\theta},\tilde{r})$
- It is also satisfies $|\det \nabla_{\theta,r}\text{Leapfrog}(\theta,r)| = 1$.
- Without this property a Jacobian correction would be needed in the acceptance probability.
- Scaling of the distribution of θ will affect the sampler's performance; it is useful to scale so the variation in the r_i is comparable to the variation in the θ_i.
- Since L gradients are needed for each step the algorithm can be very expensive.
- Pilot runs are usually used to tune ε and L.
- It is also possible to choose values of ε and L random, independently of θ and r, before each Leapfrog step
- The No-U-Turn Sampler (NUTS) provides an approach to automatically tuning ε and L.
- NUTS is the basis of the Stan framework for automate posterior sampling.
- Duane S, Kennedy AD, Pendleton BJ, Roweth D (1987). "Hybrid Monte Carlo." Physics Letters, B(195), 216-222.
- Neal R (2011). "MCMC for Using Hamiltonian Dynamics." In S Brooks, A Gelman, G Jones, M Xiao-Li (eds.), Handbook of Markov Chain Monte Carlo, p. 113-162. Chapman & Hall, Boca Raton, FL.
- Hoffman M, Gelman A (2012). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." Journal of Machine Learning Research, 1-30.
- Stan project home page.
- A simple R implementation is available on line.

Pseudo-Marginal Metropolis-Hastings MCMC

The Metropolis-Hastings method using a proposal density $q(x,x')$ for sampling from a target proportional to f uses the acceptance ratio

$$A(x,x') = \frac{f(x')q(x',x)}{f(x)q(x,x')}.$$

- Sometimes the target f is expensive or impossible to compute, but a non-negative unbiased estimate is available.
- Suppose, after generating a proposal x', such an estimate y' of $f(x')$ is produced and used in the acceptance ratio

  $$\hat{A}(x,x') = \frac{y'q(x',x)}{yq(x,x')}.$$

  The previous estimate y for $f(x)$ has to be retained and used.
- This produces a joint chain in in x,y.
- The marginal invariant distribution of the x component has density proportional to $f(x)$.
- To see this, denote the density of the estimate y given x as $h(y|x)$ and write

  $$\hat{A}(x,x') = \frac{y'h(y'|x')}{yh(y|x)} \frac{q(x',x)h(y|x)}{q(x,x')h(y'|x')}.$$
- This is the acceptance ratio for a Metropolis-Hastings chain with target density $yh(y|x)$. Since y is unbiased, the marginal density of x is

  $$\int yh(y|x)dx = f(x).$$

This is known as the *pseudo-marginal* method introduced by Andrieu and Roberts (2009) extending earlier work of Beaumont (2003).

A number of extensions and generalizations are also available.

Doubly-Intractable Posterior Distributions

For some problems a likelihood for data y is of the form

$$p(y|\theta) = \frac{g(y,\theta)}{Z(\theta)}$$

where $g(y,\theta)$ is available but $Z(\theta)$ is expensive or impossible to evaluate.

The posterior distribution is then

$$p(\theta|y) \propto \frac{g(y,\theta)p(\theta)}{Z(\theta)},$$

but is again not computable because of the likelihood normalizing constant $Z(\theta)$.

- For a fixed value $\hat{\theta}$ of θ is is useful to write the posterior density as

$$p(\theta|y) \propto g(y,\theta)p(\theta)\frac{Z(\hat{\theta})}{Z(\theta)},$$

- Suppose it is possible for a given θ to simulate a draw y^* from $p(y|\theta)$.
- Then an unbiased importance-sampling estimate of $p(\theta|y)$ is

$$\hat{p}(\theta|y) = g(y,\theta)p(\theta)\frac{g(y^*,\hat{\theta})}{g(y^*,\theta)}$$

  since

$$E\left[\frac{g(y^*,\hat{\theta})}{g(y^*,\theta)}\right] = \int \frac{g(y^*,\hat{\theta})}{g(y^*,\theta)}p(y^*|\theta)dy^* = \frac{1}{Z(\theta)}\int g(y^*,\hat{\theta})dy^* = \frac{Z(\hat{\theta})}{Z(\theta)}.$$

- Generating multiple y^* samples is also possible.
- Reducing the variance of the estimate generally reduces rejection rates and improves mixing of the sampler.

Heating and Reweighting

- Let $f(x)$ have finite integral and let $f_T(x) = f(x)^{1/T}$.
- If f_T has finite integral then we can run a Markov Chain with invariant distribution f_T
- Increasing T flattens the target density and may lead to a faster mixing chain—this is called heating.
- Decreasing T leads to a more peaked f_T concentrated more around the global maximum of f.
- Careful choice of a *cooling schedule* $T_n \to 0$ can produce an inhomogeneous chain that converges to the global maximum. This is called *simulated annealing*.
- Using a fixed $T > 1$ can produce a faster mixing chain than $T = 1$.
- More generally, using a similar but more dispersed, or more easily sampled, density g may produce a faster mixing chain.
- If $X_1, X_2, \ldots$ is a Markov chain with invariant density g, then, under reasonable conditions,

$$\frac{\sum W_i h(X_i)}{\sum W_i} \to \frac{\int h(x) f(x) dx}{\int f(x) dx}$$

  where $W_i = f(X_i)/g(X_i)$.
- This approach can also be used for sensitivity analysis:
  - Sample from the primary distribution of interest g.
  - Examine how results change for various perturbations f using the original sample from g and reweighting to f.
  - Reusing the sample is a form of common variate use.
- Instead of keeping weights one can resample with probabilities proportional to the weights.

Switching and Parallel Chains

- Suppose $f_1,\dots,f_k$ are unnormalized densities, $a_1,\dots,a_k$ are positive numbers, and p_{ij} are transition probabilities on $\{1,\dots,k\}$.
- A sampler on (X,I) can be run as:
  - when $I=i$, run a sampler with invariant distribution f_i for K steps.
  - Then choose an index $J \in \{1,\dots,k\}$ with probabilities $p_{i1},\dots,p_{ik}$.
  - With probability
  $$\min\left\{\frac{p_{Ji}a_J f_J(X)}{p_{iJ}a_i f_i(X)}, 1\right\}$$
  accept the proposal and set $I=J$; otherwise keep $I=i$
- The resulting chain has an invariant distribution with $f(x|i) \propto f_i$.
- Usually one distribution, say f_1, is the primary target distribution and the others are successively "hotter" alternatives.
- The hottest distribution may allow independent sampling.
- This approach is called *simulated tempering*.
- Care is needed in choosing a_i and p_{ij} to ensure the chain does not get stuck
- A variant runs k chains in parallel and periodically proposes a permutation of states, which is accepted with an appropriate probability. This is called *parallel tempering*.
- Parallel tempering does not require constants a_i; the joint distribution of the chains has density proportional to $f_1(x_1)\cdots f_k(x_k)$.
- Some references:

  Geyer, C. (1991) "Markov chain Monte Carlo maximum likelihood," *Computing Science and Statsitics: The 23sr Symposium on the Interface*, Interface Foundation, 156–153.

  Geyer, C. and Thompson, E (1995) "Annealing Markov chain Monte Carlo with applications to ancestral inference," *JASA*, 909–920.

  Marinari, E. and Parisi, G. (1992) "Simulated tempering: a new Monte Carlo scheme," *Europhysics letters*, 451–458.

Regeneration and MCM

- A process $X_1, X_2, \dots$ is *regenerative* if there exists a sequence of random variables $T_1 \leq T_2 \leq T_3 \dots$ such that
  - The T_i form a (possibly delayed) renewal process
  - The *tour lengths* and *tours*

    $$(T_{i+1} - T_i, X_{T_i+1}, X_{T_i+2}, \dots, X_{T_{i+1}})$$

    are independent and identically distributed.
- Suppose X_n is regenerative with stationary distribution π. Let $T_0 = 0$,

  $$\begin{aligned} N_i &= T_i - T_{i-1} \\ Y_i &= \sum_{j=T_{i-1}+1}^{T_i} h(X_j) \end{aligned}$$

  If $E[|Y_i|] < \infty$ and $E[N_i] < \infty$ then

  $$\widehat{\theta}_n = \frac{\sum_{i=1}^n Y_i}{\sum_{i=1}^n N_i} = \frac{\overline{Y}}{\overline{N}} \to \theta = E_\pi[h(X)]$$

  If $E[Y_i^2] < \infty$ and $E[N_i^2] < \infty$ then

  $$\sqrt{n}(\widehat{\theta}_n - \theta) \to \mathrm{N}(0, \tau)$$

  and τ can be estimated by the variance estimation formula for a ratio estimator:

  $$\widehat{\tau} = \frac{\frac{1}{n}\sum(Y_i - \widehat{\theta}_n N_i)^2}{\overline{N}^2}$$
- For a regenerative process we can simulate tours independently and in any order.

- An irreducible discrete state space Markov chain is regenerative with the T_i corresponding to the hitting times of any chosen state.
- Irreducible general state space chains are also regenerative.
- Finding regeneration times can be hard and may involve using auxiliary variables.
- If we have an approximate envelope h available then
  - we and can use Metropolized rejection sampling for a target distribution f
  - every time we get $f(X_i) \leq h(X_i)$ then the next proposal will be accepted
  - so every step with $f(X_i) \leq h(X_i)$ is a regeneration time.
- Periodically using a Metropolized rejection step is the simplest way to introduce regeneration in MCMC.
- How well it works depends on the quality of the envelope and the other sampler it is used in conjunction with.
- Other methods are available for identifying regeneration points.
- Regenerative analysis does *not* make a sampler better: poorly mixing samplers have tour length distributions with long tails.

Transdimensional MCMC

- A number of problems have parameter spaces that are unions of spaces of different dimensions:
  - model selection problems
  - finite mixture models with unknown number of components
  - model-based clustering
  - partitioned regression models
  - spline models with unknown number of knots
- For each of these the parameter space can be viewed as taking the form

$$\Theta = \bigcup_{k \in \mathscr{K}} (\Theta_k \times \{k\})$$

- A Bayesian formulation usually involves specifying
  - a prior on k
  - a conditional prior, given k, on the parameters in Θ_k

  An MCMC approach needs a way of moving between models.
- Several approaches are available
  - integrating out θ_k (sometimes viable)
  - reversible jump sampler
  - birth and death sampler
  - other special purpose samplers
- A useful review paper by Sisson appeared in JASA, September 2005.

Reversible Jump MCMC

- In Bayesian model selection problems we have
  - a set of M models with parameter spaces $\Theta_1,\dots,\Theta_M$
  - a set of likelihoods $f_i(x|\theta_i)$ with $\theta_i \in \Theta_i$
  - conditional prior distributions given the model $\pi(\theta_i|i)$
  - prior probabilities $\pi(i)$ on the models
- The posterior probabilities of the models are proportional to

$$\pi(i)\int_{\Theta_i} f_i(x|\theta_i)\pi(\theta_i|i)d\theta_i$$

  The odds of model i to model j can be written as

$$\frac{\int_{\Theta_i} f_i(x|\theta_i)\pi(\theta_i|i)d\theta_i}{\int_{\Theta_j} f_j(x|\theta_j)\pi(\theta_j|j)d\theta_j}\frac{\pi(i)}{\pi(j)} = B_{ij}(x)\frac{\pi(i)}{\pi(j)}$$

  $B_{ij}(x)$ is called the *Bayes factor* for model i against model j.

- One computational option is to run separate samplers for each model and estimate the normalizing constants.
- Another option is to run a single sampler that moves both within and between models.
- To move between models we need a proposal distribution $Q_{ij}(u,dv)$ for proposing a value v in model j when currently at u in model i. The proposal is accepted with probability

$$\alpha_{ij}(u,v) = \min\left\{\frac{\pi_j(dv|x)Q_{ji}(v,du)}{\pi_i(du|x)Q_{ij}(u,dv)},1\right\} = \min\left\{r_{ij}(u,v),1\right\}$$

  where $\pi_k(d\psi|x) = \pi(k)f_k(x|\psi)\pi(\psi|k)d\psi$.

- With care the proposal for going from a larger model to a smaller one can be chosen to be deterministic.
- This is the *reversible jump* sampler of Green (1995).

A Simple Example: Normal Means

Suppose X_1, X_2 are independent.

> Model 1: $X_1, X_2 \sim \mathrm{N}(\mu, 1)$, $\mu \sim N(0, b^2)$.
>
> Model 2: $X_1 \sim \mathrm{N}(\mu_1, 1), X_2 \sim \mathrm{N}(\mu_2, 1)$, $\mu_i \sim \mathrm{N}(0, b^2)$ and independent.

The two models are assumed equally likely a priori.

The jump proposals:

- To move from 1 to 2: Generate $\mu_1 \sim \mathrm{N}(\mu, 1)$ and set $\mu_2 = 2\mu - \mu_1$.
- To move from 2 to 1 set $\mu = (\mu_1 + \mu_2)/2$.

Let $\varphi(z)$ be the standard normal density, and let

$$\begin{aligned}
r_{12}(\mu, \mu_1, \mu_2) &= \frac{\frac{1}{2}\varphi(x_1 - \mu_1)\varphi(x_2 - \mu_2)b^{-1}\varphi(\mu_1/b)d\mu_1 b^{-1}\varphi(\mu_2/b)d\mu_2}{\frac{1}{2}\varphi(x_1 - \mu)\varphi(x_2 - \mu)b^{-1}\varphi(\mu/b)d\mu\varphi(\mu_1 - \mu)d\mu_1}\\
&= \frac{\varphi(x_1 - \mu_1)\varphi(\mu_1/b)\varphi(x_2 - \mu_2)\varphi(\mu_2/b)}{b\varphi(x_1 - \mu)\varphi(x_2 - \mu)\varphi(\mu/b)\varphi(\mu_1 - \mu)}\frac{d\mu_2}{d\mu}\\
&= \frac{2}{b}\frac{\varphi(x_1 - \mu_1)\varphi(\mu_1/b)\varphi(x_2 - \mu_2)\varphi(\mu_2.b)}{\varphi(x_1 - \mu)\varphi(x_2 - \mu)\varphi(\mu/b)\varphi(\mu_1 - \mu)}
\end{aligned}$$

Then

$$\begin{aligned}
\alpha_{12}(\mu, \mu_1, \mu_2) &= \min(r_{12}(\mu, \mu_1, \mu_2), 1)\\
\alpha_{21}(\mu_1, \mu_2, \mu) &= \min(1/r_{12}(\mu, \mu_1, \mu_2), 1)
\end{aligned}$$

R code to implement a within-model Gibbs step followed by a jump proposal:

```
rj <- function(m, N, x1=1, x2=-1, b=1) {
    lr12 <- function(m, m1, m2)
        log(2/b) - 0.5 * ((x1-m1)^2 + (m1/b)^2 + (x2-m2)^2 + (m2/b)^2) +
                   0.5 * ((x1-m)^2 + (x2-m)^2 + (m/b)^2 + (m1-m)^2)
    xbar <- (x1 + x2) / 2
    v <- matrix(nrow=N, ncol=3)
    I <- 1
    m <- m1 <- m2 <- 0
    for (i in 1:N) {
        if (I == 1) {
            m <- rnorm(1, xbar * b^2 / (1/2 + b^2) , b / sqrt(1 + 2 * b^2))
            m1 <- rnorm(1, m)
            m2 <- 2 * m - m1
            if (log(runif(1)) < lr12(m, m1, m2)) I <- 2
        }
        else {
            m1 <- rnorm(1, x1 * b^2 / (1 + b^2), b / sqrt(1 + b^2))
            m2 <- rnorm(1, x2 * b^2 / (1 + b^2), b / sqrt(1 + b^2))
            m <- (m1 + m2)/2
            if (log(runif(1)) < -lr12(m, m1, m2)) I <- 1
        }
        if (I == 1) v[i,] <- c(1, m, m)
        else v[i,] <- c(2, m1, m2)
    }
    v
}

> v <- rj(0, 10000, x1 = 2, x2 = -2, b = 1)
> mean(ifelse(v[, 1] == 1, 1, 0))
[1] 0.1248
> v <- rj(0, 10000, x1 = 2, x2 = -2, b = 2)
> mean(ifelse(v[, 1] == 1, 1, 0))
[1] 0.0668
> v <- rj(0, 10000, x1 = 2, x2 = -2, b = 20)
> mean(ifelse(v[, 1] == 1, 1, 0))
[1] 0.2112
> v <- rj(0, 10000, x1 = 2, x2 = -2, b = 100)
> mean(ifelse(v[, 1] == 1, 1, 0))
[1] 0.5733
> v<-rj(0, 10000, x1 = 2,x2 = -2, b = 200)
> mean(ifelse(v[, 1] == 1, 1, 0))
[1] 0.7169
```

The code is available on line.

Alternate Approach: Mixed Distributions

- We can view this as a single model with means μ_1, μ_2 and a prior distribution that says
  - with probability 1/2 the means are equal and the common value has a $\mathrm{N}(0, b^2)$ distribution
  - with probability 1/2 the means are unequal and drawn independently from a $\mathrm{N}(0, b^2)$ distribution.
- The distribution of $\mu_2 | X_1, X_2, \mu_1$ is a mixed discrete-continuous distribution such that

$$\begin{aligned} P(\mu_2 = \mu_1 | x_1, x_2, \mu_1) &= \frac{\frac{1}{2}\varphi(x_2 - \mu_1)}{\frac{1}{2}\varphi(x_2 - \mu_1) + \frac{1}{2}\frac{1}{\sqrt{1+b^2}}\varphi(x_2/\sqrt{1+b^2})} \\ &= \frac{\sqrt{1+b^2}\varphi(x_2 - \mu_1)}{\sqrt{1+b^2}\varphi(x_2 - \mu_1) + \varphi(x_2/\sqrt{1+b^2})} \end{aligned}$$

  and

$$\mu_2 | x_1, x_2, \mu_1, \mu_2 \neq \mu_1 \sim \mathrm{N}(x_2 b^2/(1+b^2), b^2/(1+b^2))$$

- The conditional distribution of $\mu_1 | Y_1, Y_2, \mu_2$ is analogous.
- The Gibbs sampler can therefore be used directly
- Metropolis-Hastings methods can also be used if care is taken in defining densities.
- With more parameters a similar approach can be used to sample pairs of parameters where the distribution can consist of
  - a discrete component
  - a one dimensional component
  - a two dimensional component
- Transformations can again help: if we use

$$\begin{aligned} \theta_1 &= \mu_1 + \mu_2 \\ \theta_2 &= \mu_1 - \mu_2 \end{aligned}$$

  then

  - θ_1, θ_2 are independent under both prior and posterior distributions
  - θ_1 has a continuous posterior distribution
  - θ_2 has a mixed posterior distribution with $P(\theta_2 = 0|X) > 0$.

- Code is available on line.

Birth and Death MCMC

- A number of models have parameters that are point processes:
  - support set for finite mixture models
  - knot set for spline models
- Point process models can be sampled by a continuous time Markov process, called a *spatial birth and death process*.
- A set of points $y = \{y_1, \dots, y_n\}$ changes by
  - *births* that add a point: $y \to y \cup \{\xi\}$
  - *deaths* that remove a point: $y \to y \backslash y_i$
- Births occur at a rate

  $$b(y, \xi) = \beta(y)\tilde{b}(y, \xi)$$

  with $\beta(y) = \int b(y, \xi) d\xi$
- The points in a set $y = \{y_1, \dots, y_n\}$ die independently with rates $d(y \backslash \{y_i\}, y_i)$.
- The total death rate is $\delta(y) = \sum_{i=1}^n d(y \backslash \{y_i\}, y_i)$

- Suppose we wish to simulate a point process with density $h(y)$ with respect to an inhomogeneous Poisson process with rate $\lambda(x)$.
- A spatial birth and death process will have this point process as invariant distribution if it satisfies the detailed balance equations

$$h(y)b(y,\xi) = h(y \cup \{\xi\})\lambda(\xi)d(y,\xi)$$

- Usual approach:
  - pick a reasonable birth rate function
  - solve for the required death rate
- The algorithm: starting with a set of points y
  1. wait for an amount of time exponentially distributed with rate $\beta(y) + \delta(y)$.
  2. at that time, a birth occurs with probability $\beta(y)/(\beta(y)+\delta(y))$.
  3. If a birth occurs, generate the location of the new point ξ from $\tilde{b}(y,\xi)$.
  4. If a death occurs, chose the point to die with probabilities proportional to $d(y\backslash\{y_i\}, y_i)$
- The idea is due to Ripley (1977) and Preston (1977).
- Stephens (2001) introduced a variation for Bayesian inference for finite mixture models.
- Continuous time data for pure jump processes can be represented as the sequence of states and their waiting times.
- Sample path averages are time-weighted averages.
- An alternative is to sample at a discrete grid of time points.
- Some notes:
  1. There are no rejections. Instead, some points die very quickly.
  2. It may be useful to add move steps that pick one point to possibly move based on, say, a Metropolis-Hastings proposal.

Example: Normal Mixture Models

- A normal mixture model assumes that $X_1, \dots, X_n$ are independent draws from the density

$$f(x|K,\mu,\sigma,p) = \sum_{i=1}^{K} p_i \frac{1}{\sigma_i} \varphi\left(\frac{x-\mu_i}{\sigma_i}\right)$$

  with φ the standard normal density and $p_1 + \cdots + p_K = 1$.

- A possible prior distribution for K, p, μ, σ can be specified as

$$\begin{aligned} K &\sim \text{Poisson}(\lambda), \text{conditioned on } 1 \le K \le K_{\max} \\ p|K &\sim \text{Dirichlet}(\alpha, \dots, \alpha) \\ \sigma_i^2|K,p &\overset{\text{ind}}{\sim} \text{IG}(a,b) \\ \mu_i|K,p,\sigma &\overset{\text{ind}}{\sim} \text{N}(m,c) \end{aligned}$$

  A more elaborate formulation might put priors on some of the hyperparameters.

- If we add an auxilliary variable v, independent of K, p, σ, μ, with

$$v|K,p,\sigma,\mu \sim \text{Gamma}(K\alpha, 1)$$

  and set $w_i = vp_i$, then

$$w_i|K,\sigma,\mu \overset{\text{ind}}{\sim} \text{Gamma}(\alpha, 1)$$

- The prior distribution of K, w, σ^2, μ is an inhomogeneous Poisson process on $\mathbb{R}^3$ with rate function

$$\begin{aligned} \lambda(\mu,\sigma,w) = \lambda &\times \text{Gamma density for } w \\ &\times \text{Inverse Gamma density for } \sigma^2 \\ &\times \text{Normal density for } \mu|\sigma \end{aligned}$$

  and $p_i = w_i / \sum_{j=1}^{K} w_j$, conditioned on $1 \le K \le K_{\max}$.

- The posterior distribution has a density

$$h(K,\mu,\sigma,w) \propto 1_{[1,K_{\max}]}(K)\prod_{j=1}^{n} f(x_i|K,\mu,\sigma,p)$$

  with respect to the Poisson process.

- Code is available on line.

Approximate Bayesian Computation (ABC)

- All approaches to posterior sampling so far have required computing the likelihood function $f(x|\theta)$.
- For some problems this is not possible, but it is possible to simulate from $f(\cdot|\theta)$.
- A simple approach:
  1. draw θ^* from the prior distribution
  2. run the model to simulate x^* from $f(x|\theta^*)$
  3. if x^* is close to the observed x then keep θ^*; otherwise, go back to step (1).
- An MCMC variant of this is also used and can lead to higher acceptance rates.
- Closeness might be measured as $d(x,x^*) \leq \varepsilon$ for some distance d and tolerance ε.
- It the tolerance is small enough the distribution of an accepted θ^* should be close to the posterior distribution $f(\theta|x)$.
- If the tolerance is too small the acceptance probability will be too low.
- This problem increases very quickly with the dimension of x.
- If a low dimensional sufficient statistic is available then the distance can be based on the sufficient statistic.
- Generally sufficient statistics are not available in problems where ABC is needed.
- If a modest number of statistics can be chosen that are nearly sufficient then the conditional distribution given these statistics may not be too far from the full posterior distribution.
- Much recent literature has explored ways of selecting a suitable set of conditioning statistics.

- Another direction of work explores the use of sequential Monte Carlo, and adaptive sequential Monte Carlo methods, in the ABC context (Sisson, Fan, and Tanaka, 2007
- The Wikipedia entry provides a good introduction and references.

Other MCMC and Related Approaches

- There are many other approaches and ideas.
  - Particle filters
  - Umbrella sampling
  - Dynamic reweighting
  - Adaptive MCMC (Christophe Andrieu, “Annotated Bibliography: Adaptive Monte Carlo Methods,” *The ISBA Bulletin* 15(1), March 2008; `http://www.bayesian.org/bulletin/0803.pdf`)
  - Special issue on adaptive Monte Carlo, *Statistics and Computing*, December 2008.
  - Sequential Importance Sampling
  - ...
- Several book length treatments are available:
  - Gamerman and Lopes (2006)
  - Robert and Casella (2004)
  - Chen, Shao, and Ibrahim (2000)
  - Liu (2001)
  - Brooks, Gelman, Jones, and Meng (2011)

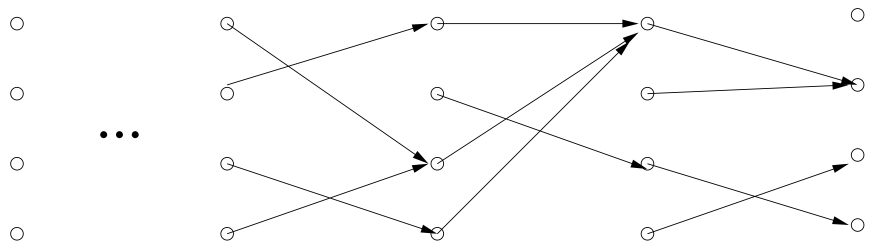
  among a number of others.

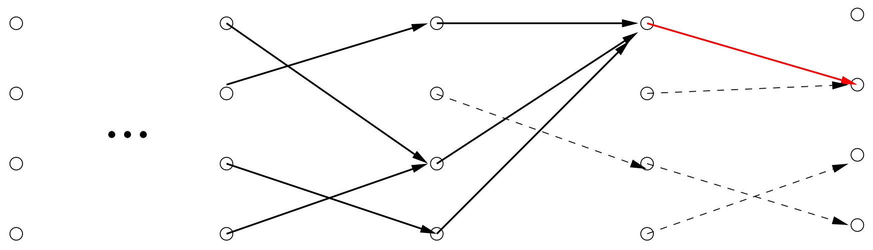
Perfect Sampling and Coupling From The Past

- Suppose π is a distribution on $E = \{1,\dots,M\}$ and P is an irreducible, aperiodic transition matrix with invariant distribution π.
- Let $\phi(u,i)$ be the inverse CDF of $P(i,\cdot)$, so if $U \sim \mathrm{U}[0,1]$ then $\phi(U,i)$ has distribution $P(i,\cdot)$.
- Suppose $U_1, U_2, \dots$ are independent $\mathrm{U}[0,1]$ and suppose

$$X_{i+1} = \phi(U_i, X_i)$$

  for $i = -1, -2, \dots$.
- For this chain started in the infinite past $X_0 \sim \pi$.
- Can we figure out what X_0 is without going infinitely far into the past?





- For $T < 0$ and $k \in E$ define

$$\begin{aligned} X_T^{(T,k)} &= k \\ X_{i+1}^{(T,k)} &= \phi(U_i, X_i^{(T,k)}) \end{aligned}$$

  for $i = T, T+1, \dots, -2, -1$.

  - If $X_0^{(T,k)}$ is the same state for all initial states k, say $X_0^{(T,k)} = j \in E$, then $X_0 = j$. The chains are said to have coupled.
  - With probability one there exists a finite $T < 0$ such that all chains starting at T will have coupled by time zero.

  The *coupling from the past (CFTP)* algorithm:

  - Start with an initial T and determine whether all chains have coupled by time zero. If so, return the common value at time zero.
  - If not, double T and repeat.

  The CFTP algorithm was introduced by Propp and Wilson (1996).

- If $\phi(u,i) \leq \phi(u,j)$ for every u and every $i \leq j$ then it is sufficient to consider the minimal and maximal chains $X_i^{(T,1)}$ and $X_i^{(T,M)}$ since
$$X_i^{(T,1)} \leq X_i^{(T,k)} \leq X_i^{(T,M)}$$
for all $k \in E = \{1,\dots,M\}$. If the minimal and maximal chains have coupled then all chains have coupled.
- This idea can be extended to partially ordered state spaces with a minimal and maximal value.
- Extensions to some continuous state space problems have been developed.
- CFTP samplers for a number of interesting distributions in physics applications have been found.
- Progress in statistics is still limited to somewhat artificial examples.
- One issue is bias: Truncating the backward search for T will change the distribution of X_0. Variations are available to address this.

Example: Image reconstruction with Ising Prior

- States are partially ordered by pixel with “black” > “white”.
- All “white” is minimal, all “black” is maximal.

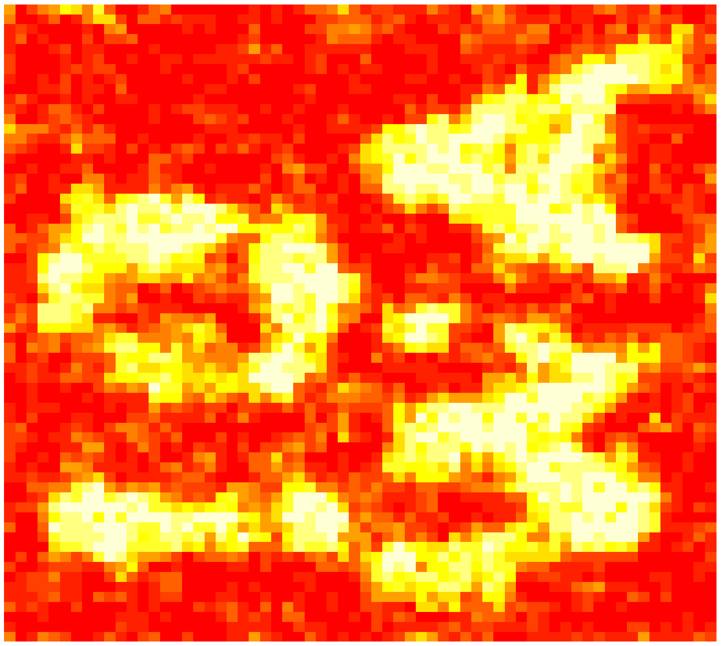
A CFTP version of the vectorized Ising model sampler:

```
simGroupU <- function(m, l2, l1, beta, which, u) {
    pp2 <- l2 * exp(beta * nn(m, 2))
    pp1 <- l1 * exp(beta * nn(m, 1))
    pp <- pp2 / (pp2 + pp1)
    ifelse(u[which] < pp[which], 2, 1)
}


simImgVU <- function(m, img, beta, p, u) {
    white <- outer(1:nrow(m), 1:ncol(m), FUN=`+`) %% 2 == 1
    black <- ! white
    if (is.null(img)) {
        l2 <- 1
        l1 <- 1
    }
    else {
        l2 <- ifelse(img == 2, p, 1 - p)
        l1 <- ifelse(img == 1, p, 1 - p)
    }
    m[white] <- simGroupU(m, l2, l1, beta, white, u)
    m[black] <- simGroupU(m, l2, l1, beta, black, u)
    m
}

isingCFTP <- function(img, N, d, beta, p) {
    u <- array(runif(d * d * N), c(d, d, N))
    repeat {
        m1 <- matrix(1, d, d)
        m2 <- matrix(2, d, d)
        for (i in 1:dim(u)[3]) {
            m1 <- simImgU(m1, img, beta, p, u[,,i])
            m2 <- simImgU(m2, img, beta, p, u[,,i])
        }
        if (identical(m1, m2)) return (m1)
        u <- array(c(array(runif(d * d * N), c(d, d, N)), u),
                   c(d, d, 2 * N))
        N <- 2 * N
    }
}
```

It takes about one minute for 10 images:

```
> img4<-array(0,c(64,64,10))
> system.time(for (i in 1:10)
                  img4[,,i] <- isingCFTP(img, 10, 64, 0.9, 0.7)
   user  system elapsed
 72.019   0.134  72.410
> image(apply(img4,c(1,2),mean), axes=FALSE)
```

and results seem reasonable:



Performance deteriorates as

- dimension increases
- β increases

For $\beta = 1.2$ it took about 10 minutes to generate 10 images (64×64)

Graphical Methods and Visualization

- There are two kind of graphics used in data analysis:
  - static graphics
  - dynamic, or interactive, graphics

  There is overlap:
  - interactive tools for building static graphs
- Graphics is used for several purposes
  - exploration and understanding
    - of raw data
    - of residuals
    - of other aspects of model fit, misfit
  - displaying and communicating results
- Historically, display and communication usually used static graphics
- Dynamic graphs were used mostly for exploration
- With digital publishing, dynamic graphics are also used for communication:
  - 2014 as hottest year on record on Bloomberg
  - Subway crime on New York Daily News
  - Who was helped by Obamacare on New York Times' Upshot

  - Paths to the White House on Upshot
  - LA Times years in graphics: 2014 and 2015

Historical Graphics

- Easy construction of graphics is highly computational, but a computer isn't necessary.
- Many graphical ideas and elaborate statistical graphs were creates in the 1800s.
- Some classical examples:
  - Playfair's *The Commercial and Political Atlas* and *Statistical Breviary* introduced a number of new graphs including
    * a bar graph
    * a pie chart
  - Minard developed many elaborate graphs, some available as thumbnail images, including an illustration of Napoleon's Russia campaign
  - Florence Nightingale uses a polar area diagram to illustrate causes of death among British troops in the Crimean war.
  - John Snow used a map (higher resolution) to identify the source of the 1854 London cholera epidemic. An enhanced version is available on `http://www.datavis.ca/`. A short movie has recently been produced.
  - Statistical Atlas of the US from the late 1800s shows a number of nice examples. The complete atlases are also available.
  - Project to show modern data in a similar style.
- Some references:
  - Edward Tufte (1983), *The Visual Display of Quantitative Information.*
  - Michael Friendly (2008), "The Golden Age of Statistical Graphics," *Statistical Science* 8(4), 502-535
  - Michael Friendly's Historical Milestones on `http://www.datavis.ca/`
  - A Wikipedia entry

Graphics Software

- Most statistical systems provide software for producing static graphics
- Statistical static graphics software typically provides
  - a variety of standard plots with reasonable default configurations for
    - bin widths
    - axis scaling
    - aspect ratio
  - ability to customize plot attributes
  - ability to add information to plots
    - legends
    - additional points, lines
    - superimposed plots
  - ability to produce new kinds of plots

  Some software is more flexible than others.
- Dynamic graphical software should provide similar flexibility but often does not.
- Non-statistical graph or chart software often emphasizes "chart junk" over content
  - results may look pretty
  - but content is hard to extract
  - graphics in newspapers and magazines and advertising
  - Some newspapers and magazines usually have very good information graphics
    - New York Times
    - Economist
    - Guardian
    - LA Times

- Chart drawing packages can be used to produce good statistical graphs but they may not make it easy.
- They may be useful for editing graphics produced by statistical software. NY Times graphics creators often
  - create initial graphs in R
  - enhance in Adobe Illustrator

Graphics in R and S-PLUS

- Graphics in R almost exclusively static.
- S-PLUS has some minimal dynamic graphics
- R can work with `ggobi`
- Dynamic graphics packages available for R include
  - `rgl` for 3D rendering and viewing
  - `iplots` Java-based dynamic graphics
  - a number of others in various stages of development
- Three mostly static graphics systems are widely used in R:
  - standard graphics (`graphics` base package)
  - `lattice` graphics (trellis in S-PLUS) (a standard recommended package)
  - `ggplot` graphics (available as `ggplot2` from CRAN)

  Minimal interaction is possible via the `locator` command
- Lattice is more structured, designed for managing multiple related graphs
- `ggplot` represents a different approach based on Wilkinson's *Grammar of Graphics*.

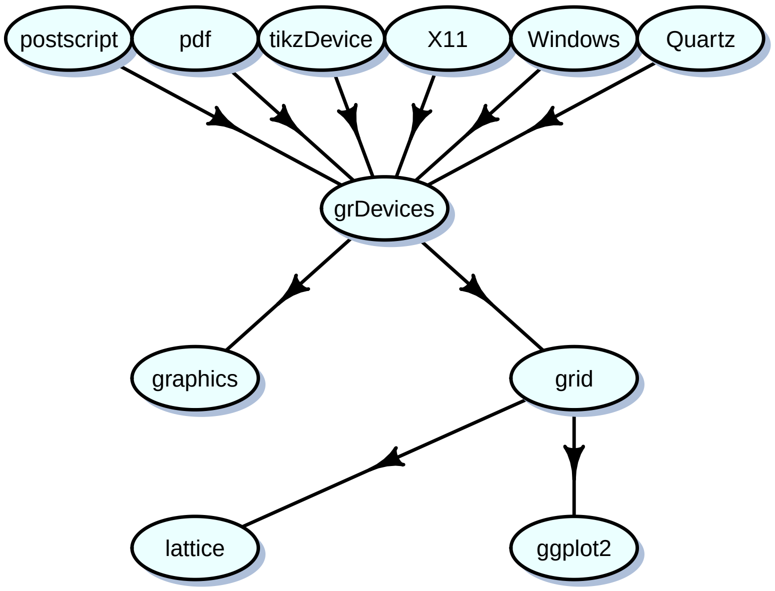
Some References

- Deepayan Sarkar (2008), Lattice: *Multivariate Data Visualization with R*, Springer; has a supporting web page.
- Hadley Wickham (2009), `ggplot`*: Elegant Graphics for Data Analysis*, Springer; has a supporting wep page.
- Paul Murrell (2011), *R Graphics*, 2nd ed., CRC Press; has a supporting web page.
- Josef Fruehwald's introduction to `ggplot`.
- Vincent Zoonekynd's Statistics with R web book; Chapter 3 and Chapter 4 are on graphics.
- Winston Chang (2013), *R Graphics Cookbook*, O'Reilly Media.
- The Graphics task view lists R packages related to graphics.

Some Courses

- Graphics lecture in Thomas Lumley's introductory computing for biostatistics course.
- Ross Ihaka's graduate course on computational data analysis and graphics.
- Ross Ihaka's undergraduate course on information visualization.
- Deborah Nolan's undergraduate course *Concepts in Computing with Data*.
- Hadley Wickham's Data Visualization course

A View of R Graphics



Graphics Examples

- Code for Examples in the remainder of this section is available on line
- Many examples will be from W. S. Cleveland (1993), *Visualizing Data* and N. S. Robbins (2004), *Creating More Effective Graphs*.

Plots for Single Numeric Variables

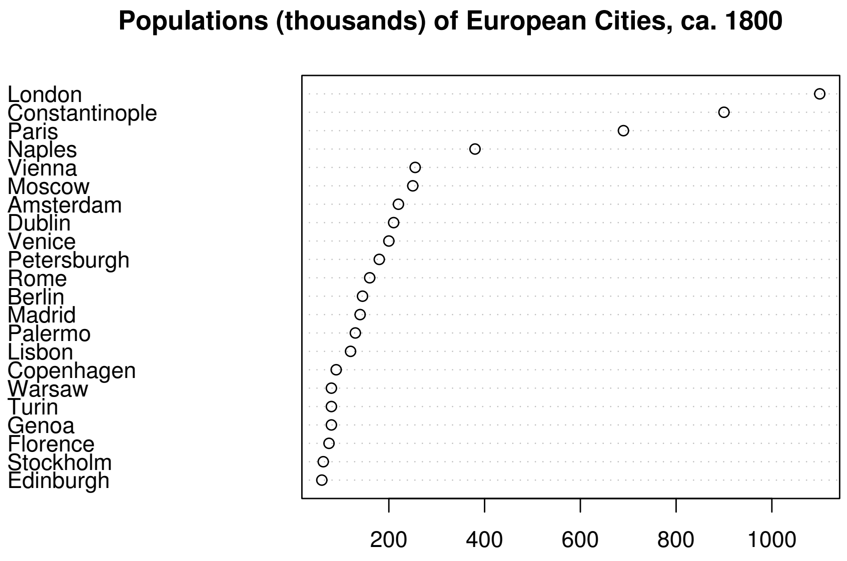
Dot Plots

This uses Playfair's city population data available in the data from Cleveland's *Visualizing Data* book:

```
Playfair <-
  read.table("http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/Playfair")
```

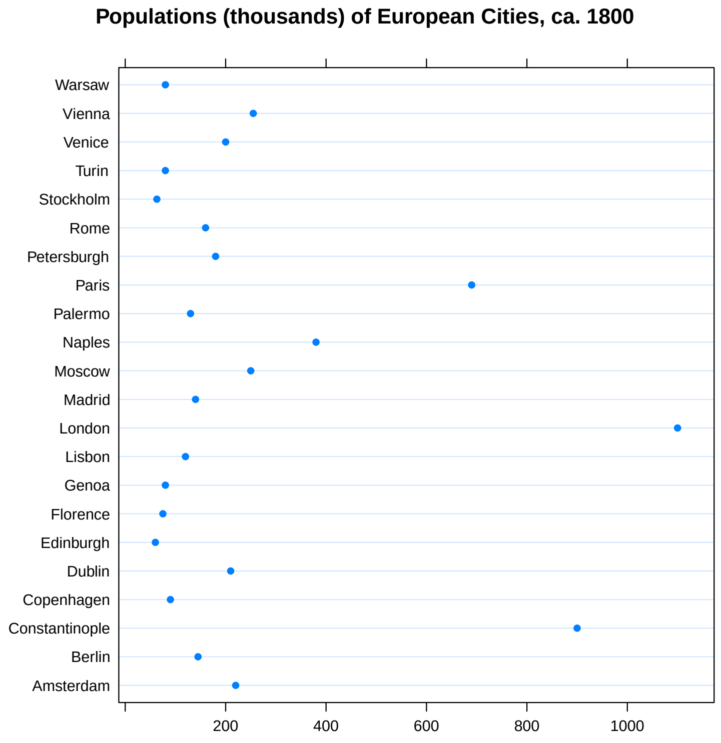
- Useful for modest amounts of data
- Particularly useful for named values.
- Different sorting orders can be useful.
- Standard graphics:

  ```
  dotchart(structure(Playfair[,1],names=rownames(Playfair)))
  title("Populations (thousands) of European Cities, ca. 1800")
  ```



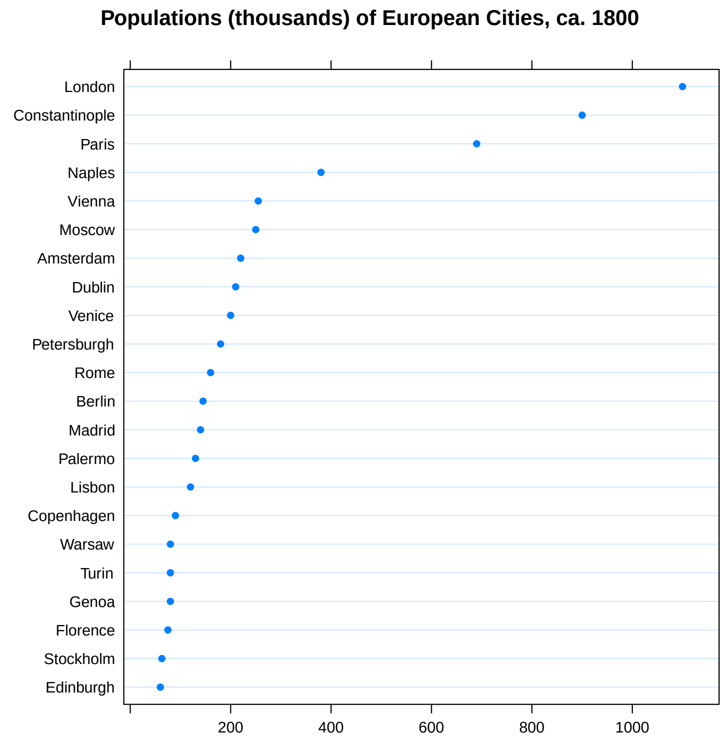
- Lattice uses `dotplot`.

```
library(lattice)
dotplot(rownames(Playfair) ~ Playfair[,1],
        main = "Populations (thousands) of European Cities, ca. 1800",
        xlab = "")
```



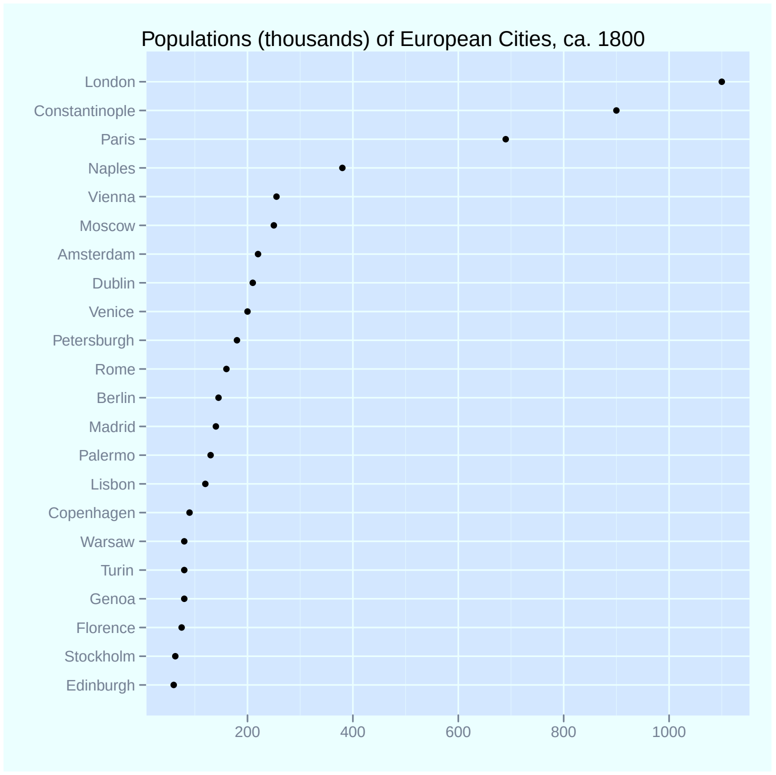
To prevent sorting on names need to convert names to an ordered factor.

```
dotplot(reorder(rownames(Playfair), Playfair[,1]) ~ Playfair[,1],
        main = "Populations (thousands) of European Cities, ca. 1800",
        xlab = "")
```



- `ggplot` graphics

```
library(ggplot2)
qplot(Playfair[,1], reorder(rownames(Playfair), Playfair[,1]),
      main = "Populations (thousands) of European Cities, ca. 1800",
      xlab = "", ylab = "")
```



More Plots for Single Numeric Variables

Bar Charts

An alternative to a dot chart is a bar chart.

- These are more commonly used for categorical data
- They use more "ink" for the same amount of data
- Standard graphics provide `barplot`:

  ```
  barplot(Playfair[,1],names = rownames(Playfair),horiz=TRUE)
  ```

  This doesn't seem to handle the names very well.
- Lattice graphics use `barchart`:

  ```
  barchart(reorder(rownames(Playfair), Playfair[,1]) ~ Playfair[,1],
           main = "Populations (thousands) of European Cities, ca. 1800",
           xlab = "")
  ```
- `ggplot` graphics:

  ```
  p <- qplot(weight = Playfair[,1],
             x = reorder(rownames(Playfair), Playfair[,1]),
             geom="bar")
  p + coord_flip()
  ```

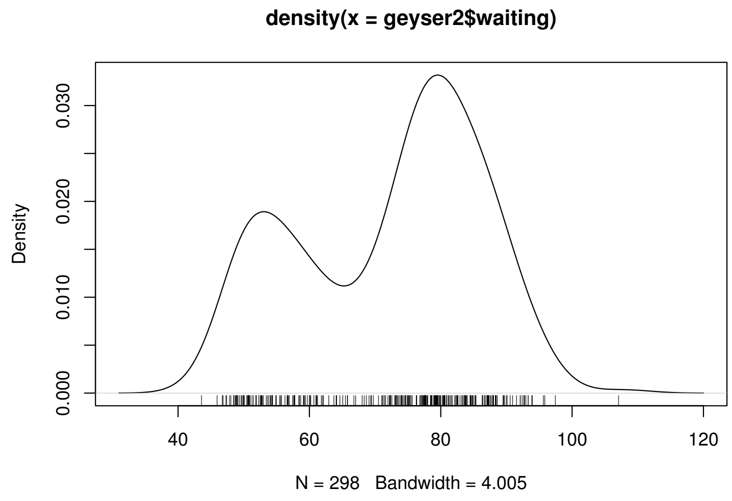
Density Plots

A data set on eruptions of the Old Faithful geyser in Yellowstone:

```
library(MASS)
geyser2 <- data.frame(as.data.frame(geyser[-1,]),
                      pduration=geyser$duration[-299])
```

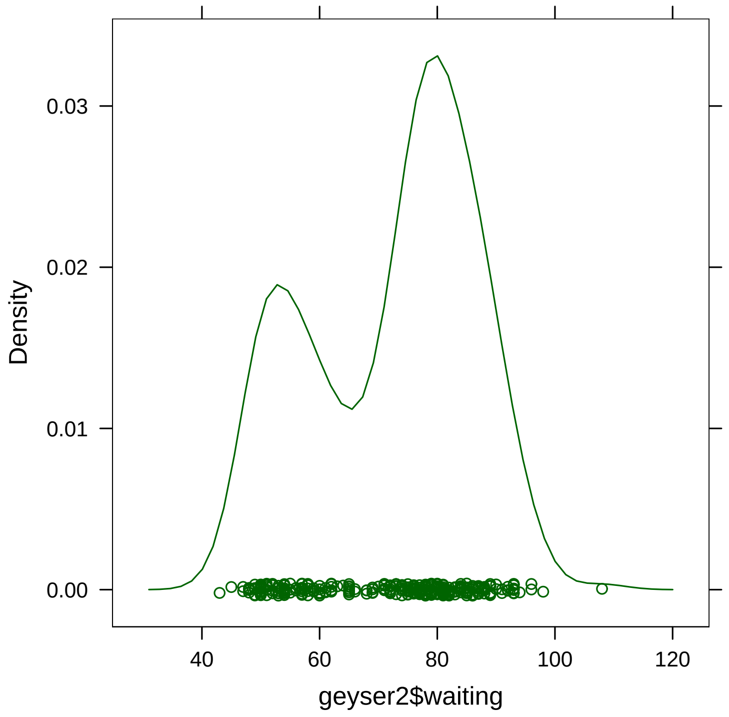
- Standard graphics:

  ```
  plot(density(geyser2$waiting))
  rug(jitter(geyser2$waiting, amount = 1))
  ```



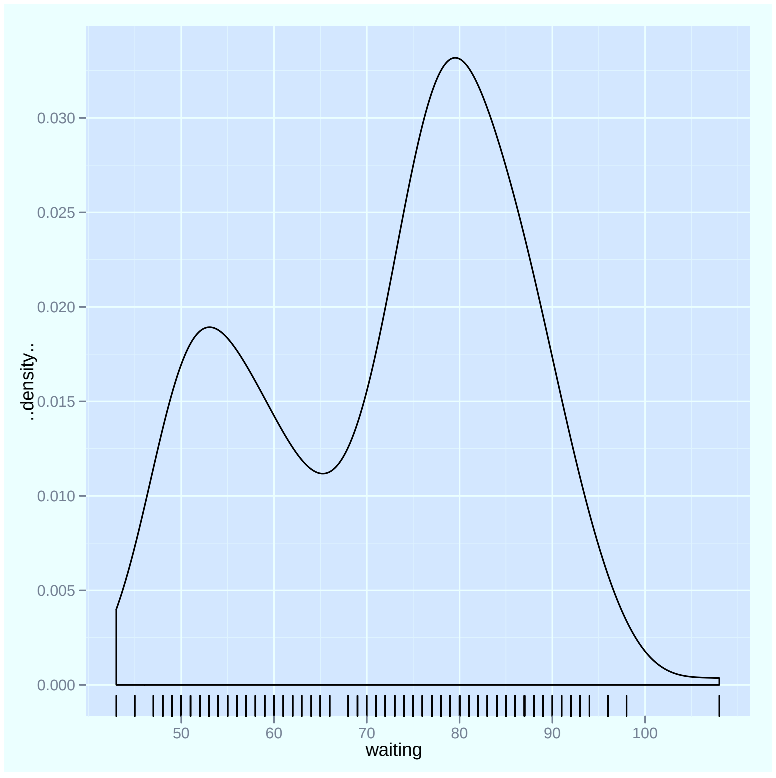
- Lattice graphics:

```
densityplot(geyser2$waiting)
```



- `ggplot2` graphics:

```
qplot(waiting,data=geyser2,geom="density") + geom_rug()
```



Quantile Plots

- Standard graphics

```
data(precip)
qqnorm(precip, ylab = "Precipitation [in/yr] for 70 US cities")
```

- Lattice graphics

```
qqmath(~precip, ylab = "Precipitation [in/yr] for 70 US cities")
```

- `ggplot` graphics

```
qplot(sample = precip, stat="qq")
```

Other Plots

Other options include

- Histograms
- Box plots
- Strip plots; use jittering for larger data sets

Plots for Single Categorical Variables

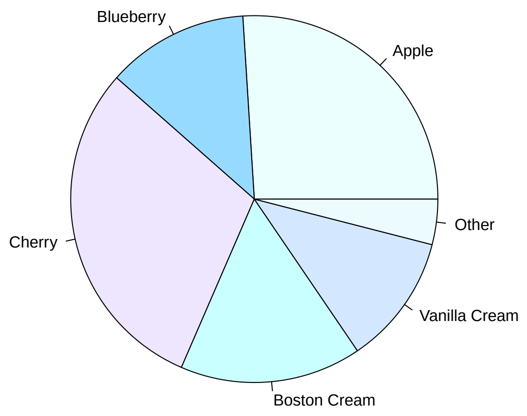
- Categorical data are usually summarized as a contingency table, e.g. using the `table` function.
- A little artificial data set:

```
pie.sales <- c(0.26, 0.125, 0.3, 0.16, 0.115, 0.04)
names(pie.sales) <- c("Apple", "Blueberry", "Cherry",
                      "Boston Cream", "Vanilla Cream",
                      "Other")
```

Pie Charts

- Standard graphics provides the `pie` function:

```
pie(pie.sales)
```



- Lattice does not provide a pie chart, but the Lattice book shows how to define one.
- `ggplot` can create pie charts as stacked bar charts in polar coordinates:

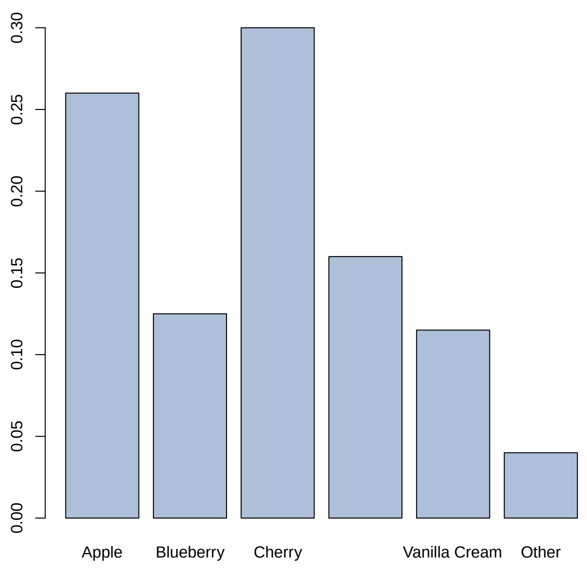
```
qplot(x = "", y = pie.sales, fill = names(pie.sales)) +
    geom_bar(width = 1, stat = "identity") + coord_polar(th

df <- data.frame(sales = as.numeric(pie.sales), pies = name
ggplot(df, aes(x = "", y = sales, fill = pies)) +
    geom_bar(width = 1, stat = "identity") +
    coord_polar(theta = "y")
```

This could use some cleaning up of labels.

Bar Charts

- Standard graphics:

```
barplot(pie.sales)
```



  - One label is skipped to avoid over-printing
  - vertical or rotated text might help.

- Lattice:

```
barchart(pie.sales)
```

- `ggplot`:

```
qplot(x = names(pie.sales), y = pie.sales,
      geom = "bar", stat = "identity")
```

This orders the categories alphabetically.

Plotting Two Numeric Variables

Scatter Plots

- The most important form of plot.
- Not as easy to use as one might think.
- Ability to extract information can depend on aspect ratio.
- Research suggests aspect ratio should be chosen to center absolute slopes of important line segments around 45 degrees.
- A simple example: river flow measurements.

```
river <-
  scan("http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples
plot(river)
xyplot(river~seq_along(river),panel=function(x,y,...) {
    panel.xyplot(x,y,...)
    panel.loess(x,y,...)})
plot(river,asp=4)
plot(river)
lines(seq_along(river),river)
plot(river, type = "b")
```

- Some more Lattice variations

```
xyplot(river~seq_along(river), type=c("p","r"))
xyplot(river~seq_along(river), type=c("p","smooth"))
```

- Some `ggplot` variations

```
qplot(seq_along(river), river)
qplot(seq_along(river), river) + geom_line()
qplot(seq_along(river), river) + geom_line() + stat_smooth(
```

- There is not always a single best aspect ratio.

```
data(co2)
plot(co2)
title("Monthly average CO2 concentrations (ppm) at Mauna Loa Observator
```

Handling Larger Data Sets

An artificial data set:

```
x <- rnorm(10000)
y <- rnorm(10000) + x * (x + 1) / 4
plot(x,y)
```

- Overplotting makes the plot less useful.
- Reducing the size of the plotting symbol can help:

  ```
  plot(x,y, pch=".")
  ```

- Another option is to use translucent colors with *alpha blending*:

  ```
  plot(x,y, col = rgb(0, 0, 1, 0.1, max=1))
  ```

- Hexagonal binning can also be useful:

  ```
  plot(hexbin(x,y))          # standard graphics
  hexbinplot(y ~ x)          # lattice
  qplot(x, y, geom = "hex") # ggplot
  ```

Plotting a Numeric and a Categorical Variable

Strip Charts

- Strip charts can be useful for modest size data sets.

  ```
  stripchart(yield ~ site, data = barley, met)  # standard
  stripplot(yield ~ site, data = barley)        # Lattice
  qplot(site, yield, data = barley)             # ggplot
  ```

- *Jittering* can help reduce overplotting.

  ```
  stripchart(yield ~ site, data = barley, method="jitter")
  stripplot(yield ~ site, data = barley, jitter.data = TRUE)
  qplot(site, yield, data = barley, position = position_jitter(w = 0.1))
  ```

Box Plots

Box plots are useful for larger data sets:

```
boxplot(yield ~ site, data = barley)                 # standard
bwplot(yield ~ site, data = barley)                  # Lattice
qplot(site, yield, data = barley, geom = "boxplot") # ggplot
```

Density Plots

- One approach is to show multiple densities in a single plot.
- We would want
  - a separate density for each site
  - different colors for the sites
  - a legend linking site names to colors
  - all densities to fit in the plot
- This can be done with standard graphics but is tedious:

  ```
  with(barley, plot(density(yield[site == "Waseca"])))
  with(barley, lines(density(yield[site == "Crookston"]), col = "re
  # ...
  ```

- Lattice makes this easy using the `group` argument:

  ```
  densityplot(~yield, group = site, data = barley)
  ```

  A legend can be added with `auto.key=TRUE`:

  ```
  densityplot(~yield, group = site, data = barley, auto.key=T
  ```

- `ggplot` also makes this easy by mapping the site to the `col` aesthetic.

  ```
  qplot(yield, data = barley, geom="density", col = site)
  ```

- Another approach is to plot each density in a separate plot.
- To allow comparisons these plots should use common axes.
- This is a key feature of Lattice/Trellis graphics:

  ```
  densityplot(~yield | site, data = barley)
  ```

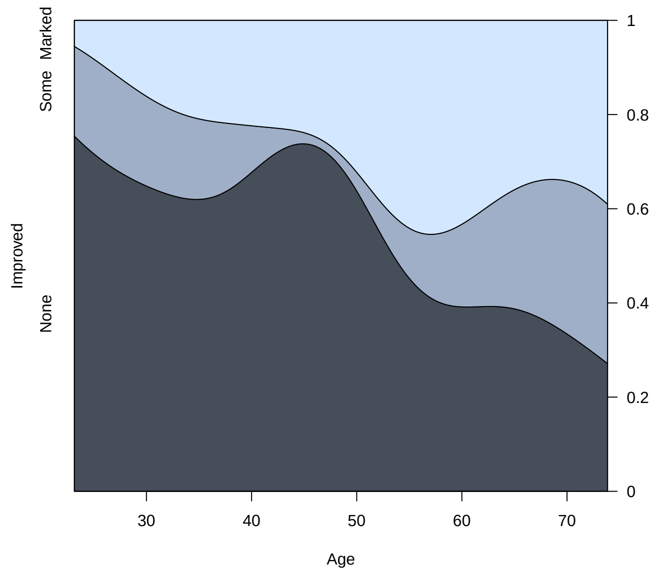
- `ggplot` supports this as *faceting*:

  ```
  qplot(yield, data = barley, geom="density") + facet_wrap(~ site)
  ```

Categorical Response Variable

Conditional density plots estimate the conditional probabilities of the response categories given the continuous predictor:

```
library(vcd)
data("Arthritis")
cd_plot(Improved ~ Age, data = Arthritis)
```



Plotting Two Categorical Variables

Bar Charts

- Standard graphics:

  ```
  tab <- prop.table(xtabs(~Treatment + Improved, data = Arthr
  barplot(t(tab))
  barplot(t(tab),beside=TRUE)
  ```

- Lattice:

  ```
  barchart(tab, auto.key = TRUE)
  barchart(tab, stack = FALSE, auto.key = TRUE)
  ```

  Lattice seems to also require using a frequency table.

- `ggplot`:

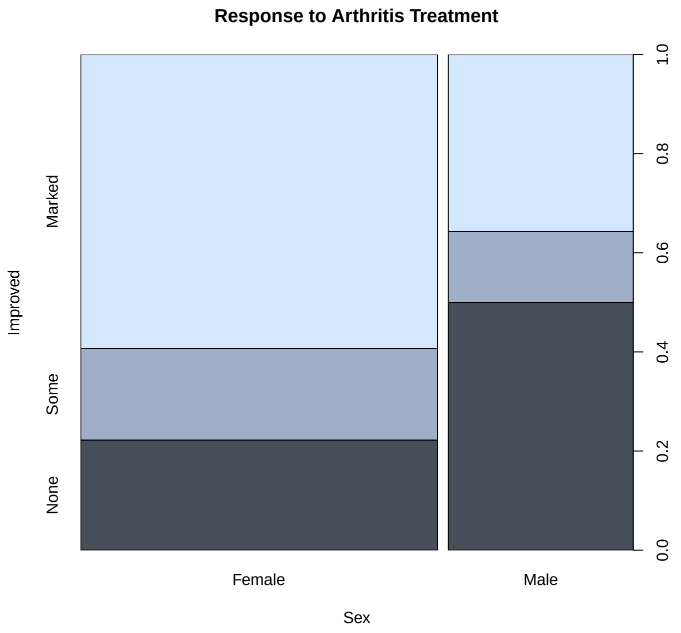
  ```
  qplot(Treatment, geom = "bar", fill = Improved, data = Arth
  qplot(Treatment, geom = "bar", fill = Improved,
        position="dodge", data = Arthritis)
  qplot(Treatment, geom = "bar", fill = Improved,
        position="dodge", weight = 1/nrow(Arthritis),
        ylab="", data = Arthritis)
  ```

Plotting Two Categorical Variables

Spine Plots

Spine plots are a variant of stacked bar charts where the relative widths of the bars correspond to the relative frequencies of the categories.

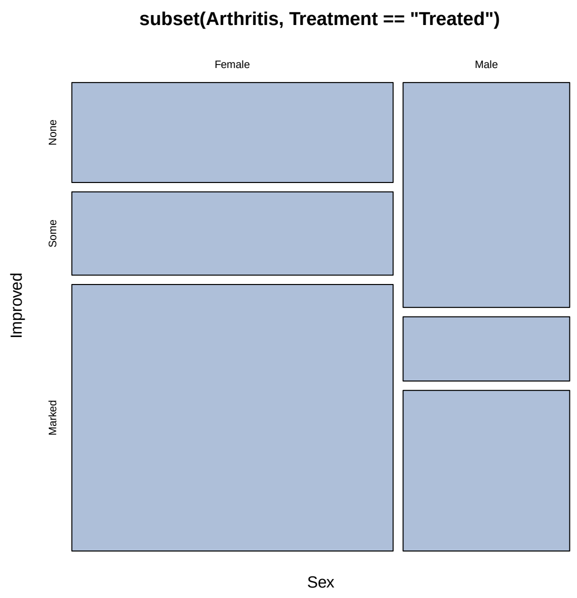
```
spineplot(Improved ~ Sex,
          data = subset(Arthritis, Treatment == "Treated"),
          main = "Response to Arthritis Treatment")
spine(Improved ~ Sex,
      data = subset(Arthritis, Treatment == "Treated"),
      main = "Response to Arthritis Treatment")
```



Mosaic Plots

Mosaic plots for two variables are similar to spine plots:

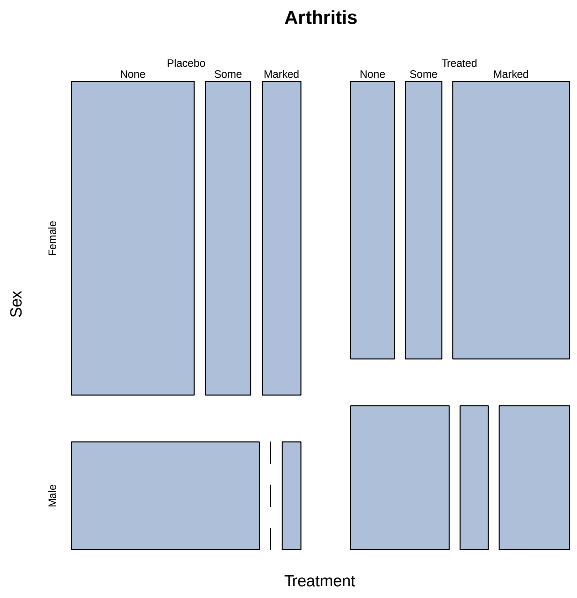
```
mosaicplot(~ Sex + Improved,
           data = subset(Arthritis, Treatment == "Treated"))

mosaic(~ Sex + Improved,
       data = subset(Arthritis, Treatment == "Treated"))
```



Mosaic plots extend to three or more variables:

```
mosaicplot(~ Treatment + Sex + Improved, data = Arthritis)
```

```
mosaic(~ Treatment + Sex + Improved, data = Arthritis)
```



Three or More Variables

- Paper and screens are two-dimensional; viewing more than two dimensions requires some trickery
- For three continuous variables we can use intuition about space together with
  - motion
  - perspective
  - shading and lighting
  - stereo
- For categorical variables we can use forms of conditioning
- Some of these ideas carry over to higher dimensions
- For most viewers intuition does not go beyond three dimensions

Some Examples

Soil Resistivity

- Soil resistivity measurements taken on a tract of land.

```
library(lattice)
soilfile <-
    "http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/soil"
soil <- read.table(soilfile)

p <- cloud(resistivity ~ easting * northing, pch = ".", data = soil)
s <- xyplot(northing ~ easting, pch = ".", aspect = 2.44, data = soil)
print(s, split = c(1, 1, 2, 1), more = TRUE)
print(p, split = c(2, 1, 2, 1))
```

- A loess surface fitted to soil resistivity measurements.

```
eastseq <- seq(.15, 1.410, by = .015)
northseq <- seq(.150, 3.645, by = .015)
soi.grid <- expand.grid(easting = eastseq, northing = northseq)
m <- loess(resistivity ~ easting * northing, span = 0.25,
           degree = 2, data = soil)
soi.fit <- predict(m, soi.grid)
```

- A level/image plot is made with

```
levelplot(soi.fit ~ soi.grid$easting * soi.grid$northing,
          cuts = 9,
          aspect = diff(range(soi.grid$n)) / diff(range(soi.grid$e)),
          xlab = "Easting (km)",
          ylab = "Northing (km)")
```

- An interactive 3D rendered version of the surface:

```
library(rgl)
bg3d(color = "white")
clear3d()
par3d(mouseMode="trackball")
surface3d(eastseq, northseq,
          soi.fit / 100, color = rep("red", length(soi.fit)))
```

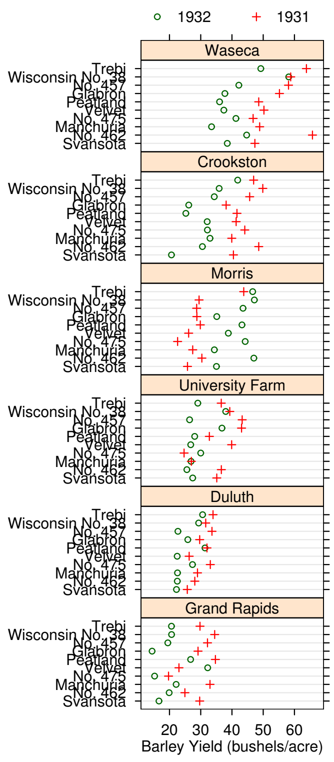
- Partially transparent rendered surface with raw data:

```
clear3d()
points3d(soil$easting, soil$northing, soil$resistivity / 100,
         col = rep("black", nrow(soil)))
surface3d(eastseq, northseq,
          soi.fit / 100, col = rep("red", length(soi.fit)),
          alpha=0.9, front="fill", back="fill")
```

Barley Yields

- Yields of different barley varieties were recorded at several experimental stations in Minnesota in 1931 and 1932
- A dotplot can group on one factor and condition on others:

```
data(barley)
n <- length(levels(barley$year))
dotplot(variety ~ yield | site,
        data = barley,
        groups = year,
        layout = c(1, 6),
        aspect = .5,
        xlab = "Barley Yield (bushels/acre)",
        key = list(points = Rows(trellis.par.get("superpose.symbol"), 1
                   text = list(levels(barley$year)),
                   columns = n))
```



- Cleveland suggests that years for Morris may have been switched.
- A recent article offers another view.

NOx Emissions from Ethanol-Burning Engine

- An experiment examined the relation between nitrous oxide concentration in emissions `NOx` and
  - compression ratio `C`
  - equivalence ratio `E` (richness of air/fuel mixture)
- A scatterplot matrix shows the results

```
data(ethanol)
pairs(ethanol)
splom(ethanol)
```

- Conditioning plots (coplots) can help:

```
with(ethanol, xyplot(NOx ~ E | C))
with(ethanol, {
     Equivalence.Ratio <- equal.count(E, number = 9, overlap = 0.25)
     xyplot(NOx ~ C | Equivalence.Ratio,
            panel = function(x, y) {
                panel.xyplot(x, y)
                panel.loess(x, y, span = 1)
            },
            aspect = 2.5,
            layout = c(5, 2),
            xlab = "Compression Ratio",
            ylab = "NOx (micrograms/J)")
})
```

Three or More Variables

Earth Quakes

- Some measurements on earthquakes recorded near Fiji since 1964
- A scatterplot matrix shows all pairwise distributions:

  ```
  data(quakes)
  splom(quakes)
  ```

- The locations can be related to geographic map data:

  ```
  library(maps)
  map("world2",c("Fiji","Tonga","New Zealand"))
  with(quakes,points(long,lat,col="red"))
  ```

- Color can be used to encode depth or magnitude

  ```
  with(quakes,
       points(long,lat,col=heat.colors(nrow(quakes))[rank(depth)]))
  ```

- Color scale choice has many issues; see `www.colorbrewer.org`
- Conditioning plots can also be used to explore depth:

  ```
  with(quakes,xyplot(lat~long|equal.count(depth)))
  ```

- Perspective plots are useful in principle but getting the right view can be hard

  ```
  with(quakes,cloud(-depth~long*lat))
  library(scatterplot3d)
  with(quakes,scatterplot3d(long,lat,-depth))
  ```

- Interaction with rgl can make this easier:

  ```
  library(rgl)
  clear3d()
  par3d(mouseMode="trackball")
  with(quakes, points3d(long, lat, -depth/50,size=2))
  clear3d()
  par3d(mouseMode="trackball")
  with(quakes, points3d(long, lat, -depth/50,size=2,
                        col=heat.colors(nrow(quakes))[rank(mag)]))
  ```

Other 3D Options

- Stereograms, stereoscopy.
- Anaglyph 3D using red/cyan glasses.
- Polarized 3D.

Design Notes

- Standard graphics
  - provides a number of basic plots
  - modify plots by drawing explicit elements
- Lattice graphics
  - create an expression that describes the plot
  - basic arguments specify layout vie `group` and conditioning arguments
  - drawing is done by a panel function
  - modify plots by defining new panel functions (usually)
- `ggplot` and Grammar of Graphics
  - create an expression that describes the plot
  - aesthetic elements are associated with specific variables
  - modify plots by adding layers to the specification

Dynamic Graphs

- Some interaction modes:
  - identification/querying of points
  - conditioning by selection and highlighting
  - manual rotation
  - programmatic rotation
- Some systems with dynamic graphics support:
  - S-PLUS, JMP, SAS Insight, ...
  - `ggobi`, `http://www.ggobi.org`
  - `Xmdv`, `http://davis.wpi.edu/~xmdv/`
  - Various, `http://stats.math.uni-augsburg.de/software/`
  - `xlispstat`

Color Issues

Some Issues

- different types of scales, palettes:
  - qualitative
  - sequential
  - diverging
- colors should ideally work in a range of situations
  - CRT display
  - LCD display
  - projection
  - color print
  - gray scale print
  - for color blind viewers
- obvious choices like simple interpolation in RGB space do not work well

Some References

- Harrower, M. A. and Brewer, C. M. (2003). ColorBrewer.org: An online tool for selecting color schemes for maps. *The Cartographic Journal*, 40, 27–37. Available on line. The `RColopBrewer` package provides an R interface.
- Ihaka, R. (2003). Colour for presentation graphics," in K. Hornik, F. Leisch, and A. Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, Vienna, Austria. Available on line. See also the R package `colorspace`.
- Lumley, T. (2006). Color coding and color blindness in statistical graphics. *ASA Statistical Computing & Graphics Newsletter*, 17(2), 4–7. Avaivable on line.

- Zeileis, A., Meyer, D. and Hornik, K. (2007). Residual-based shadings for visualizing (conditional) independence. *Journal of Computational and Graphical Statistics*, 16(3), 507–525. See also the R package *vcd*.

- Zeileis, A., Murrell, P. and Hornik, K. (2009). Escaping RGBland: Selecting colors for statistical graphics, *Computational Statistics & Data Analysis*, 53(9), 3259-3270 Available on line.

Perception Issues

- A classic paper:

  William S. Cleveland and Robert McGill (1984), "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods," *Journal of the American Statistical Association* 79, 531–554.

- The paper shows that accuracy of judgements decreases down this scale:
  - position along a common scale
  - position along non-aligned scales
  - length, direction, angle,
  - area
  - shading, color saturation

- A simple example:

```
x <- seq(0, 2*pi, len = 100)
y <- sin(x)
d <- 0.2 - sin(x+pi/2) * 0.1
plot(x,y,type="l", ylim = c(-1,1.2))
lines(x, y + d, col = "red")
lines(x, d, col = "blue", lty = 2)
```

- Bubble plots
  - An example from Bloomberg.
  - An improved version of the lower row:

```
library(ggplot2)
bankName <- c("Credit Suisse", "Goldman Sachs", "Santander",
              "Citygroup", "JP Morgan", "HSBC")
before <- c(75, 100, 116, 255, 165, 215)
after <- c(27, 35, 64, 19, 85, 92)

d <- data.frame(cap = c(before, after),
                year = factor(rep(c(2007,2009), each=6)),
                bank = rep(reorder(bankName, 1:6), 2))

ggplot(d, aes(x = year, y = bank, size = cap, col = year)) +
    geom_point() +
    scale_size_area(max_size = 20) +
    scale_color_discrete(guide="none")
```

  - A bar chart:

```
ggplot(d, aes(x = bank, y = cap, fill = year)) +
    geom_bar(stat = "identity", position = "dodge") + coord_flip()
```

  - Some dot plots:

```
qplot(cap, bank, col = year, data = d)
qplot(cap, bank, col = year, data = d) + geom_point(size = 4)
do <- transform(d, bank = reorder(bank,rep(cap[1:6],2)))
qplot(cap, bank, col = year, data = do) +
    geom_point(size = 4)
qplot(cap, bank, col = year, data = do) +
    geom_point(size = 4) + theme_bw()
library(ggthemes)
qplot(cap, bank, col = year, data = do) +
    geom_point(size = 4) + theme_economist()
qplot(cap, bank, col = year, data = do) +
    geom_point(size = 4) + theme_wsj()
```

- Our perception can also play tricks, leading to optical illusions.
  - Some examples, some created in R.
  - Some implications for circle and bubble charts.
  - The sine illusion.

Some References

- Cleveland, W. S. (1994), *The Elements of Graphing Data*, Hobart Press.
- Cleveland, W. S. (1993), *Visualizing Data*, Hobart Press.
- Robbins, Naomi S. (2004), *Creating More Effective Graphs*, Wiley; Effective Graphs blog.
- Tufte, Edward (2001), *The Visual Display of Quantitative Information*, 2nd Edition, Graphics Press.
- Wilkinson, Leland (2005), *The Grammar of Graphics*, 2nd Edition, Springer.
- Bertin, Jaques (2010), *Semiology of Graphics: Diagrams, Networks, Maps*, ESRI Press.
- Cairo, Alberto (2012), *The Functional Art: An introduction to information graphics and visualization*, New Riders; The Functional Art blog.
- Few, Stephen (2012), *Show Me the Numbers: Designing Tables and Graphs to Enlighten*, 2nd Edition, Analytics Press; Perceptual Edge blog.

Some Web and Related Technologies

- Google Maps and Earth
  - Mapping earthquakes.
  - Baltimore homicides.
  - Mapping twitter trends.
- SVG/JavaSctipt examples
  - SVG device driver.
  - JavaScript D3 and some R experiments:
    - Contour plots
    - rCharts
- Grammar of Graphics for interactive plots
  - animint package
  - ggvis package; source on github
- Flash, Gapminder, and Google Charts
  - Gapminder: `http://www.gapminder.org/`
  - An example showing wealth and health of nations over time.‘
  - Popularized in a video by Hans Rosling.
  - Google Chart Tools: `https://developers.google.com/chart/`
  - googleVis package.
- Plotly
  - A blog post about an R interface.
- Gif animations
  - Bird migration patterns
- Embedding animations and interactive views in PDF files

- Supplemental material to JCGS editorial. (This seems not to be complete; another example is available from my web site.)

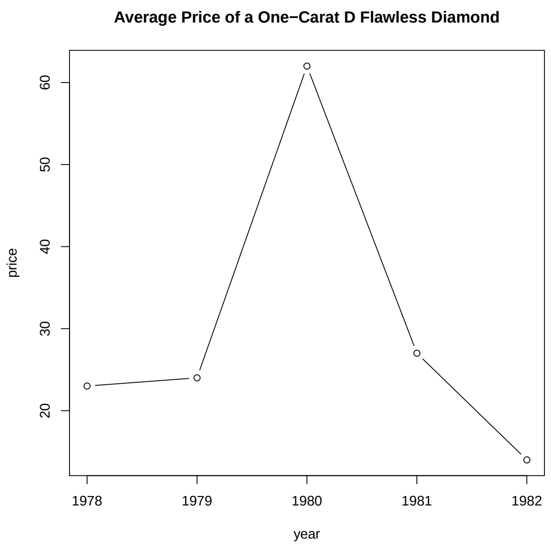
- Animations in R
  - `animation` package; has a supporting web site.
  - A simple example is available at the class web site.
  - Rstudio's shiny package.

- Tableau software
  - Tableau Public.

Further References

- Colin Ware (2004), *Information Visualization, Second Edition: Perception for Design*, Morgan Kaufmann.
- Steele, Julie and Iliinsky, Noah (Editors) (2010), *Beautiful Visualization: Looking at Data through the Eyes of Experts*.
- Tufte, Edward (2001), *The Visual Display of Quantitative Information*, 2nd Edition, Graphics Press.
- Tufte, Edward (1990), *Envisioning Information*, Graphics Press.
- Cairo, Alberto (2012), *The Functional Art: An introduction to information graphics and visualization*, New Riders.
- Gelman, Andrew and Unwin, Antony (2013), "Infovis and Statistical Graphics: Different Goals, Different Looks," *JCGS*; links to discussions and rejoinder; slides for a related talk.
- Stephen Few (2011), The Chartjunk Debate A Close Examination of Recent Findings.
- An article in The Guardian.
- Robert Kosara's Eagereyes blog.
- Data Journalism Awards for 2012.
- The Information is Beautiful Awards.

A classic example:



An alternate representation.

Some More References and Links

- Kaiser Fung's Numbers Rule Your World and Junk Charts blogs.
- Nathan Yao's FlowingData blog.
- JSS Special Volume on Spatial Statistics, February 2015.
- An unemployment visualization from the Wall Street Journal.
- A WebGL example from `rgl`

Some Data Technologies

- Data is key to all statistical analyses.
- Data comes in various forms:
  - text files
  - data bases
  - spreadsheets
  - special binary formats
  - embedded in web pages
  - special web formats (XML, JSON, ...)
- Data often need to be cleaned.
- Data sets often need to be reformatted or merged or partitioned.
- Some useful R tools:
  - `read.table`, `read.csv`, and `read.delim` functions.
  - `merge` function for merging columns of two tables based on common keys (data base join operation).
  - The `reshape` function and the `melt` and `cast` functions from the `reshape` or `reshape2` packages for conversion between long and wide formats.
  - `tapply` and the `plyr` and `dplyr` packages for
    - partitioning data into groups
    - applying statistical operations to the groups
    - assembling the results
  - The `XML` package for reading XML and HTML files.
  - The `scrapeR` and `rvest` packages.
  - Web Technologies Task View.
  - Regular expressions for extracting data from text.
- Some references:

- Paul Murrell (2009), *Introduction to Data Technologies*, CRC Press; available online at the supporting website,
- Phil Spector (2008), *Data Manipulation with R*, Springer; available through Springer Link.
- Deborah Nolan and Duncan Temple Lang (2014), *XML and Web Technologies for Data Sciences with R*, Springer.

Example: Finding the Current Temperature

- A number of web sites provide weather information.
- Some provide web pages intended to be read by humans:
  - Weather Underground.
  - Weather Channel
  - National Weather Service.
- Others provide a web service intended to be accessed by programs:
  - Open Weather Map API.
  - A similar service from Google was shut down in 2012.
  - National Weather Service SOAP API.
  - National Weather Service REST API.
- Historical data is also available, for example from Weather Underground.
- You computer of smart phone uses services like these to display current weather.
- The R package `RWeather` provides access to a number of weather APIs.

- Open Weather Map provides an API for returning weather information in XML format using a URL of the form

  ```
  http://api.openweathermap.org/data/2.5/weather?q=Iowa+
  City,IA&mode=xml&appid=44db6a862fba0b067b1930da0d769e98
  ```

  or

  ```
  http:
  //api.openweathermap.org/data/2.5/weather?lat=41.66&lon=
  -91.53&mode=xml&appid=44db6a862fba0b067b1930da0d769e98
  ```

- Here is a simple function to obtain the current temperature for from Open Weather Map based on latitude and longitude:

  ```
  library(xml2)
  findTempOWM <- function(lat, lon) {
      base <- "http://api.openweathermap.org/data/2.5/weather"
      key <- "44db6a862fba0b067b1930da0d769e98"
      url <- sprintf("%s?lat=%f&lon=%f&mode=xml&units=Imperial&appid=%s",
          base, lat, lon, key)
      page <- read_xml(url)
      as.numeric(xml_text(xml_find_one(page, "//temperature/@value")))
  }
  ```

- For Iowa City you would use

  ```
  findTempOWM(41.7, -91.5)
  ```

- This function should be robust since the format of the response is documented and should not change.
- Using commercial web services should be done with care as there are typically limitations and license terms to be considered.
- They may also come and go: Google's API was shut down in 2012.

Example: Creating a Temperature Map

- The National Weather Service provides a site that produces forecasts in a web page for a URL like this:

```
http://forecast.weather.gov/zipcity.php?inputstring=
                        IowaCity,IA
```

- This function uses the National Weather Service site to find the current temperature:

```
library(xml2)
findTempGov <- function(citystate) {
    url <- paste("http://forecast.weather.gov/zipcity.php?inputstring",
                 url_escape(citystate),
                 sep = "=")
    page <- read_html(url)
    xpath <- "//p[@class=\"myforecast-current-lrg\"]"
    tempNode <- xml_find_one(page, xpath)
    as.numeric(sub("([-+]?[[:digit:]]+).*", "\\1", xml_text(tempNode)))
}
```

- This will need to be revised whenever the format of the page changes, as happened sometime in 2012.
- Murrell's *Data Technologies* book discusses XML, XPATH queries, regular expressions, and how to work with these in R.
- Some other resources for regular expressions:
  - Wikipedia
  - Regular-Expressions.info

- A small selection of Iowa cities

```
places <- c("Ames", "Burlington", "Cedar Rapids", "Clinton",
            "Council Bluffs", "Des Moines", "Dubuque", "Fort Dodge",
            "Iowa City", "Keokuk", "Marshalltown", "Mason City",
            "Newton", "Ottumwa", "Sioux City", "Waterloo")
```

- We can find their current temperatures with

```
temp <- sapply(paste(places, "IA", sep = ", "),
               findTempGov, USE.NAMES = FALSE)
temp
```

- To show these on a map we need their locations. We can optain a file of geocoded cities and read it into R:

```
## download.file("http://www.sujee.net/tech/articles/geocoded/cities.csv.zip",
##               "cities.csv.zip")
download.file("http://www.stat.uiowa.edu/~luke/classes/STAT7400/data/cities.csv.zip",
              "cities.csv.zip")
unzip("cities.csv.zip")
cities <- read.csv("cities.csv", stringsAsFactors=FALSE, header=FALSE)
names(cities) <- c("City", "State", "Lat", "Lon")
head(cities)
```

- Form the temperature data into a data frame and use `merge` to merge in the locations from the cities data frame (a JOIN operation in data base terminology):

```
tframe <- data.frame(City = toupper(places), State = "IA", Temp = temp)
tframe

temploc <- merge(tframe, cities,
                 by.x = c("City", "State"), by.y = c("City", "State"))
temploc
```

- Now use the `map` function from the `maps` package along with the `text` function to show the results:

```
library(maps)
map("state", "iowa")
with(temploc, text(Lon, Lat, Temp, col = "blue"))
```

- To add contours we can use `interp` from the `akima` package and the `contour` function:

```
library(akima)
map("state", "iowa")
surface <- with(temploc, interp(Lon, Lat, Temp, linear = FALSE))
contour(surface, add = TRUE)
with(temploc, text(Lon, Lat, Temp, col = "blue"))
```

- A version using `ggmap`:

```
library(ggmap)
p <- qmplot(Lon, Lat, label = Temp, data = temploc,
            zoom = 7, source = "google") +
    geom_text(color="blue", vjust = -0.5, hjust = -0.3, size = 7)
p
```

- Add contour lines:

```
s <- expand.grid(Lon = surface$x, Lat = surface$y)
s$Temp <- as.vector(surface$z)
s <- s[! is.na(s$Temp),]
p + geom_contour(aes(x = Lon, y = Lat, z = Temp), data = s)
```

Example: 2008 Presidential Election Results

- The New York Times website provides extensive material on the 2008 elections. County by county vote totels and percentages are available, including results for Iowa
- This example shows how to recreate the *choropleth map* shown on the Iowa retults web page.
- The table of results can be extracted using the `XML` package with

  ```
  library(XML)
  url <- "http://elections.nytimes.com/2008/results/states/president/iowa.html"
  tab <- readHTMLTable(url, stringsAsFactors = FALSE)[[1]]
  ```

  Alternatively, using packages `xml2` and `rvest`,

  ```
  library(xml2)
  library(rvest)
  tab <- html_table(read_html(url))[[1]]
  ```

  These results can be formed into a usable data frame with

  ```
  iowa <- data.frame(county = tab[[1]],
                     ObamaPCT = as.numeric(sub("%.*", "", tab[[2]])),
                     ObamaTOT = as.numeric(gsub("votes|,", "", tab[[3]])),
                     McCainPCT = as.numeric(sub("%.*", "", tab[[4]])),
                     McCainTOT = as.numeric(gsub("votes|,", "", tab[[5]])),
                     stringsAsFactors = FALSE)
  head(iowa)
  ```

- We need to match the county data to the county regions. The region names are

  ```
  library(maps)
  cnames <- map("county", "iowa", namesonly = TRUE, plot = FALSE)
  head(cnames)
  ```

- Compare them to the names in the table:

  ```
  which( ! paste("iowa", tolower(iowa$county), sep = ",") == cnames)
  cnames[71]
  iowa$county[71]
  ```

- There is one polygon for each county and they are in alphabetical order, so no elaborate matching is needed.
- An example on the `maps` help page shows how matching on FIPS codes can be done if needed.
- Next, choose cutoffs for the percentage differences and assign codes:

```
cuts <- c(-100, -15, -10, -5, 0, 5, 10, 15, 100)
buckets <- with(iowa, as.numeric(cut(ObamaPCT - McCainPCT, cuts)))
```

- Create a diverging color palette and assign the colors:

```
palette <- colorRampPalette(c("red", "white", "blue"),
                            space = "Lab")(8)
colors <- palette[buckets]
```

- Create the map:

```
map("county", "iowa", col = colors, fill = TRUE)
```

- Versions with no county lines and with the county lines in white:

```
map("county", "iowa", col = colors, fill = TRUE, lty = 0, resolution=0)
map("county", "iowa", col = "white", add = TRUE)
```

- A better pallette:

```
myred <- rgb(0.8, 0.4, 0.4)
myblue <- rgb(0.4, 0.4, 0.8)
palette <- colorRampPalette(c(myred, "white", myblue),
                            space = "Lab")(8)
colors <- palette[buckets]
map("county", "iowa", col = colors, fill = TRUE, lty = 0, resolution=0)
map("county", "iowa", col = "white", add = TRUE)
```

- Some counties have many more total votes than others.
- *Cartograms* are one way to attempt to adjust for this; these have been used to show 2008 and 2012 presidential election results.
- *Tile Grid Maps* are another variation currently in use.
- The New York Times also provides data for 2012 but it seems more difficult to scrape.
- Politoco.com provides results for 2012 that are easier to scrape; the Iowa results are available at

  `http://www.politico.com/2012-election/results/president/iowa/`

ITBS Results for Iowa City Elementary Schools

- The Iowa City Press-Citizen provides data from ITBS results for Iowa City shools.
- Code to read these data is available.
- This code arranges the Standard and Percentile results into a single data frame with additional columns for `Test` and `School`.
- CSV files for the Percentile and Standard results for the elementary schools (except Regina) are also available.
- Read in the Standard results:

```
url <- paste("http://www.stat.uiowa.edu/~luke/classes/STAT7400",
             "examples/ITBS/ICPC-ITBS-Standard.csv", sep = "/")
Standard <- read.csv(url, stringsAsFactors = FALSE, row.names = 1)
names(Standard) <- sub("X", "", names(Standard))
head(Standard)
```

- These data are in *wide* format. To use Lattice or `ggplot` to examine these data we need to convert to *long* format.
- This can be done with the `reshape` function or the function `melt` in the `reshape2` package:

```
library(reshape2)
mS <- melt(Standard, id=c("Grade", "Test", "School"),
           value.name = "Score", variable.name = "Year")
head(mS)
```

- Some Lattice plots:

```
library(lattice)
xyplot(Score ~ Grade | Year, group = Test, type = "l", data = mS,
       auto.key = TRUE)
xyplot(Score ~ Grade | Year, group = Test, type = "l", data = mS,
       subset = School == "Lincoln", auto.key = TRUE)
xyplot(Score ~ Grade | Year, group = Test, type = "l", data = mS,
       subset = Test %in% c("SocialScience", "Composite"),
       auto.key = TRUE)
```

Studying the Web

- Many popular web sites provide information about their use.
- This kind of information is now being actively mined for all sorts of purposes.
- Twitter provides an API for collecting information about "tweets."
  - The R package `twitteR` provides an interface to this API.
  - A simple introduction (deprecated but may still be useful).
  - One example of its use involves mining twitter for airline consumer sentiment.
  - Another example is using twitter activity to detect earthquakes.
- Facebook is another popular framework that provides some programmatic access to its information.
  - The R package `Rfacebook` is available.
  - One blog post shows how to access the data.
  - Another provides a simple illustration.
- Google provides access to a number of services, including
  - Google Maps
  - Google Earth
  - Google Visualization
  - Google Correlate
  - Google Trends

  R packages to connect to some of these and others are available.

- Some other data sites:
  - Iowa Government Data
  - New York Times Data
  - Guardian Data
- Nice summary of a paper on deceptive visualizations.

Bootstrap and Resampling Methods

- Often we have an estimator T of a parameter θ and want to know its sampling properties
  - to informally assess the quality of the estimate
  - for formal confidence intervals
  - for formal hypothesis tests
- In some cases we can obtain exact results; usually this requires
  - particularly convenient statistics (e.g. linear)
  - particularly convenient data models (e.g. normal)
- For more complex problems we can use approximations
  - based on large samples
  - use central limit theorem
  - use Taylor series approximations
- Alternative: use simulation in place of central limit theorem and approximations
- References:
  - Givens and Hoeting, Chapter 9.
  - Davison, A. C. and Hinkley, D. V. (1997) "Bootstrap Methods and their Application," Cambridge University Press.
  - Package `boot`; R port of S-Plus code written to support Davison and Hinkley.

Example: A Parametric Bootstrap

- Ordered times between failures of air conditioning unit on one aircraft:

3	5	7	18	43	85	91	98	100	130	230	487

- Suppose the data are exponentially distributed with mean μ and we want to estimate the failure rate $\theta = 1/\mu$.
- The MLE of μ is $\widehat{\mu} = \overline{X} = 108.0833$.
- The MLE of θ is $T = 1/\overline{X}$.
- The exact sampling distribution is inverse gamma with mean μ, which we do not know.
- We could compute the exact bias and variance of T,

$$\begin{aligned} b(\mu) &= E_\mu[T] - \theta = E_\mu[T] - 1/\mu = \frac{1}{n-1}\frac{1}{\mu} = \frac{1}{n-1}\theta \\ v(\mu) &= \mathrm{Var}_\mu(T) = \frac{1}{\mu^2}\frac{n^2}{(n-1)^2(n-2)} = \theta^2\frac{n^2}{(n-1)^2(n-2)} \end{aligned}$$

  and obtain plug-in estimates

$$\begin{aligned} B &= b(\widehat{\mu}) \\ V &= v(\widehat{\mu}) \end{aligned}$$

- One alternative to exact computation of the sampling distribution is the delta method.
- The first order delta method approximates T by

$$T = 1/\overline{X} \approx 1/\mu - (\overline{X} - \mu)/\mu^2$$

So

$$\begin{aligned} b(\mu) &\approx 0 \\ v(\mu) &\approx \text{Var}(\overline{X})/\mu^4 = \frac{1}{n}\frac{\mu^2}{\mu^4} = \frac{\theta^2}{n} \end{aligned}$$

- A second order delta method for the bias uses

$$T = 1/\overline{X} \approx 1/\mu - (\overline{X} - \mu)/\mu^2 + (\overline{X} - \mu)^2/\mu^3$$

So

$$b(\mu) \approx \text{Var}(\overline{X})/\mu^3 = \frac{\theta}{n}$$

- Instead of working out $b(\cdot)$ and $v(\cdot)$ analytically, we can estimate B and V by simulation:
  - Draw a sample $X_1^*, \dots, X_{12}^*$ from an exponential distribution with mean $\mu = \widehat{\mu} = 108.0833$.
  - Compute T^* from this sample.
  - Repeat R times to obtain $T_1^*, \dots, T_R^*$
  - Estimate the bias and variance of T by

$$\begin{aligned} B^* &= \overline{T}^* - T \\ V^* &= \frac{1}{R-1}\sum_{i=1}^{R}(T_i^* - \overline{T}^*)^2 \end{aligned}$$

- The full sampling distribution can be examined using a histogram or other density estimate.

Bootstrap and Resampling Methods

Example: A Nonparametric Bootstrap

- Instead of assuming an exponential population in assessing the performance of T we can make the weaker assumption that $X_1,\ldots,X_n$ is a random sample from an arbitrary distribution F.
- A delta method approximation to the variance is

$$\mathrm{Var}(T) \approx \frac{S_X^2}{n\overline{X}^4}$$

- A non-parametric bootstrap approach takes the empirical distribution $\widehat{F}$ with

$$\widehat{F}(t) = \frac{\#\{X_i \le t\}}{n}$$

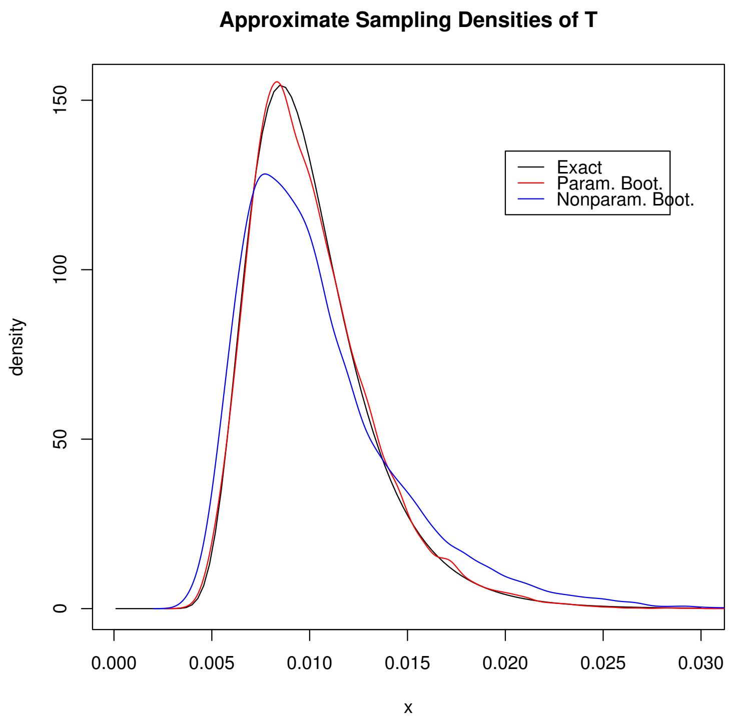
  as an approximation to F, and
  - draws a sample $X_1^*,\ldots,X_{12}^*$ from $\widehat{F}$,
  - computes T^* from this sample,
  - repeats R times to obtain $T_1^*,\ldots,T_R^*$,
  - estimate the bias and variance of T by

$$\begin{aligned} B^* &= \overline{T}^* - T \\ V^* &= \frac{1}{R-1}\sum_{i=1}^{R}(T_i^* - \overline{T}^*)^2 \end{aligned}$$

- Drawing a random sample from $\widehat{F}$ is the same as sampling with replacement from $\{X_1,\ldots,X_n\}$

Some Results

Method	Bias	Stand. Dev.
Exact	0.000841	0.003192
Param. Delta (1st order)	0.0	0.002671
Param. Delta (2nd order)	0.0007710	
Nonparam. Delta (1st order)	0.0	0.003366
Nonparam. Delta (2nd order)	0.001225	
Parametric Bootstrap ($R = 10000$)	0.000857	0.003185
Nonparametric Bootstrap ($R = 10000$)	0.001244	0.004170



Notes

- There are two kinds of errors in the bootstrap approaches (Davison and Hinkley's terminology):
  - The *statistical error* due to using $\widehat{\mu}$ or $\widehat{F}$ instead of μ or F.
  - The *simulation error*.
- The simulation error can be reduced by
  - increasing R
  - using variance reduction methods if applicable
- The statistical error is reduced as the sample size n gets large.
- In some settings the statistical error can be reduced by working with a transformed parameter.
- For the parametric bootstrap, as $R \to \infty$

$$\begin{aligned} B^* &\to b(\widehat{\mu}) \\ V^* &\to v(\widehat{\mu}) \end{aligned}$$

  and

$$\begin{aligned} b(\widehat{\mu}) &= \frac{1}{n-1}\frac{1}{\widehat{\mu}} = b(\mu) + O_P(n^{-3/2}) \\ v(\widehat{\mu}) &= \frac{1}{\widehat{\mu}}\frac{n^2}{(n-1)^2(n-2)} = v(\mu) + O_P(n^{-3/2}) \end{aligned}$$

- Similar results hold, under suitable regularity conditions, for the nonparametric bootstrap.
- Suppose $X_1,\ldots,X_n$ are a random sample from F with mean μ_F and variance σ_F^2. Let

$$\begin{aligned}\theta_F &= \mu_F^2\\ T &= \overline{X}^2\end{aligned}$$

- The bias of T is

$$b(F) = E_F[T] - \theta(F) = E_F[\overline{X}^2] - \mu_F^2 = \text{Var}_F(\overline{X}) = \frac{1}{n}\sigma_F^2$$

- The bootstrap bias ($R = \infty$) is

$$b(\widehat{F}) = \frac{1}{n}\sigma_{\widehat{F}}^2 = \frac{1}{n}\frac{n-1}{n}S^2$$

- If the population has finite fourth moments, then

$$b(\widehat{F}) = b(F) + O_P(n^{-3/2})$$

- Except in some special cases the theoretical justification for the bootstrap is asymptotic in sample size.
- The motivation for the *nonparametric* bootstrap is often similar to the motivation for using the sandwich estimator of variance:
  - use a model to suggest an estimator T
  - do not assume the model in assessing the estimator's performance

A Simple Implementation

- The basic operation of bootstrapping is to generate samples and collect statistics:

  ```
  b <- function(stat, gen, R)
      sapply(1:R, function(i) stat(gen()))
  ```

- A generator for an exponential model can be constructed by

  ```
  makeExpGen <- function(data) {
      rate <- 1 / mean(data)
      n <- length(data)
      function() rexp(n, rate)
  }
  ```

  This uses *lexical scope* to capture the variables `rate` and `n`.

- A parametric bootstrap sample of T is produced by

  ```
  v <- b(function(data) 1 / mean(data),
         makeExpGen(aircondit$hours), 10000)
  ```

- A generator for a nonparametric bootstrap sample drawn from the empirical distribution is constructed by

  ```
  makeEmpGen <- function(data) {
      if (is.vector(data)) {
          n <- length(data)
          function() data[sample(n, n, replace = TRUE)]
      }
      else {
          n <- nrow(data)
          function() data[sample(n, n, replace = TRUE), , drop = FALSE]
      }
  }
  ```

  - The variables `data` and `n` are captured by lexical scope.
  - The `data` can be a vector, a matrix, or a data frame.

- A nonparametric bootstrap sample of T is produced by

  ```
  vv <- b(function(data) 1 / mean(data),
          makeEmpGen(aircondit$hours), 10000)
  ```

A Cautionary Example

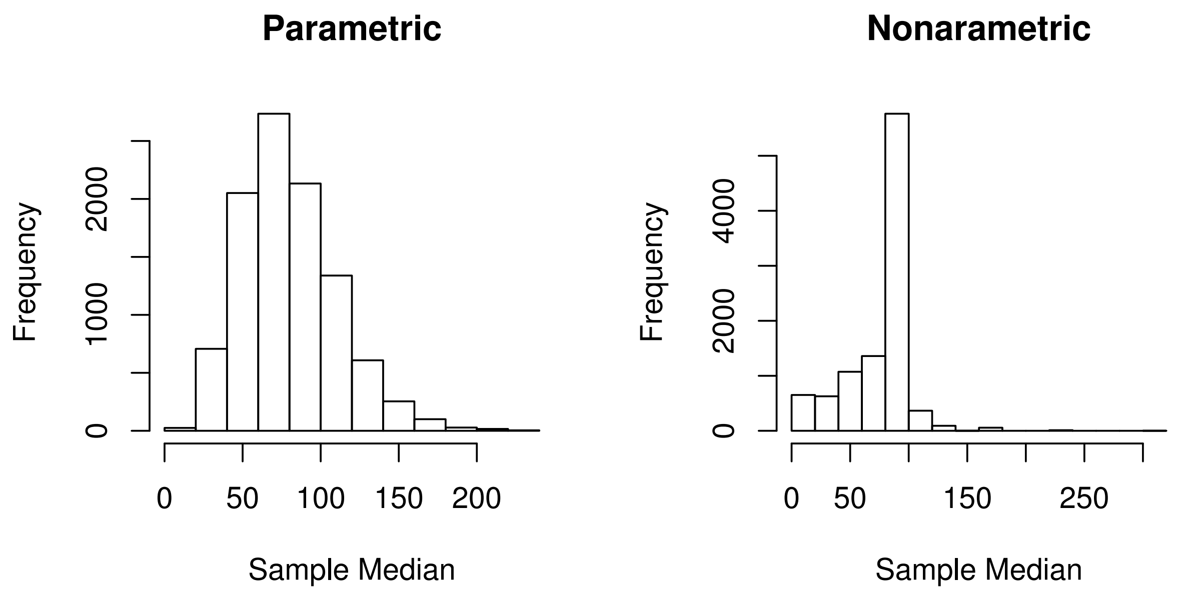
- In many elementary bootstrap applications we have, at least approximately,

$$\begin{aligned} T &= T(\widehat{F}) \\ \theta &= T(F) \end{aligned}$$

- For the basic nonparametric bootstrap to work, some level of smoothness of θ as a function of F is needed.
- Suppose we are interested in examining the sampling distribution of the median for the air conditioner data.
- Parametric and nonparametric bootstrap samples are produced by

```
medpb <- b(median, makeExpGen(aircondit$hours),10000)
mednpb <- b(median, makeEmpGen(aircondit$hours),10000)
```

- The resulting histograms are



- A more complex smoothed bootstrap can be used to address this issue.

- Suppose
  - $v_n(T) = \mathrm{Var}(T)$ for a sample of size n
  - V_n^* is the boostrap estimate for $R = \infty$ for a sample of size n.

  Then, under suitable conditions,

$$\begin{aligned} nv_n(T) &\to \frac{1}{4f(\theta)^2} \\ nV_n^*(T) &\to \frac{1}{4f(\theta)^2} \end{aligned}$$

  In this sense the boostrap is *valid*.

- The mean square error $E[(nV_n^*(T) - nv_n(T))^2]$ tends to zero slower than for smooth functionals ($O(n^{-1/2})$ instead of $O(n^{-1})$).
- The mean square error for a suitable smooth bootstrap converges faster than for an unsmoothed one, but not as fast as for smooth functionals ($O(n^{-4/5})$ can be achieved).

Bootstrapping Regression Models

- Suppose we have independent observations $(x_1, y_1), \dots, (x_n, y_n)$.
- Regression models model the conditional distribution $y|x$.
- Linear regression models assume $E[Y_i|x_i] = x_i\beta$.
- Often we assume $\text{Var}(Y_i|x_i)$ is constant.
- Parametric bootstrapping can be used for a parametric model.
- Two forms of non-parametric bootstrap are used:
  - Case-based bootstrapping: pairs (x^*, y^*) are selected with replacement from $(x_1, y_1), \dots, (x_n, y_n)$. This is also called a *pairs bootstrap*.
  - Model-based bootstrapping: Residuals are formed using the model, and bootstrap samples are constructed by sampling the residuals.

Case-Based Bootstrap

- Assumes the (X_i, Y_i) are sampled from a multivariate population.
- Advantages:
  - Simplicity.
  - Does not depend on an assumed mean model.
  - Does not depend on a constant variance assumption.
  - Does not depend on the notion of a residual.
  - Similar in spirit to the sandwich estimator.
- Disadvantages:
  - Not appropriate for designed experiments.
  - Does not reproduce standard results conditional on x_i
  - For bivariate normal data the bootstrap distribution of $\widehat{\beta}_1$ is estimating a mixture over the x_i of

    $$N(\beta, \sigma^2 / \sum (x_i - \overline{x})^2)$$

    distributions, i.e. a non-central t_{n-1} distribution.
  - Does reproduce standard results for studentized quantities where the conditional distribution does not depend on the x_i.
  - Can lead with high probability to design matrices that are not of full rank.

Model-Based Bootstrap

- This is a semi-parametric approach.
- It assumes
  - $Y_i = \mu(x_i) + \varepsilon_i$
  - the ε_i have a common distribution G with mean zero
  - $\mu(x_i)$ has a parametric form (usually).
- The *raw residuals* are of the form $e_i = y_i - \widehat{\mu}(x_i)$.
- For generalized linear models one can use various definitions of raw residuals (original scale, linear predictor scale, deviance residuals, e.g.).
- The raw residuals usually do not have constant variance; for linear models
  $$\mathrm{Var}(e_i) = \sigma^2(1-h_i)$$
  where h_i is the *leverage* of the i-th case, the i-th diagonal element of the hat matrix $H = X(X^TX)^{-1}X^T$.
- The *modified residuals* are
  $$r_i = \frac{y_i - \widehat{\mu}(x_i)}{(1-h_i)^{1/2}} = \frac{e_i}{(1-h_i)^{1/2}}$$
  Approximate leverages are used for generalized linear and nonlinear models.
- The modified residuals usually do not have mean zero, so need to be adjusted.
- The model-based resampling algorithm:
  - randomly resample ε_i^* with replacement from $r_1 - \overline{r}, \ldots, r_n - \overline{r}$.
  - set $y_i^* = \widehat{\mu}(x_i) + \varepsilon_i^*$
  - fit a model to the (x_i, y_i^*).

Example: Leukemia Survival

- For the leukemia survival data of Feigl and Zelen we can fit a model

$$\log \text{Time}_i = \beta_0 + \beta_1 \log(\text{WBC}_i/10000) + \beta_2 \text{AG}_i + \varepsilon_i$$

  where the ε_i are assumed to have zero mean and constant variance.

- We can compute the fit and some diagnostics with

```
library(boot)
fz <-
    read.table(
        "http://www.stat.uiowa.edu/~luke/classes/STAT7400/feigzel2.dat"
        head = TRUE)
fz.lm <- glm(log(fz$Time) ~ log(fz$WBC / 10000) + fz$AG)
fz.diag <- glm.diag(fz.lm)
```

- A case-based bootstrap is computed by

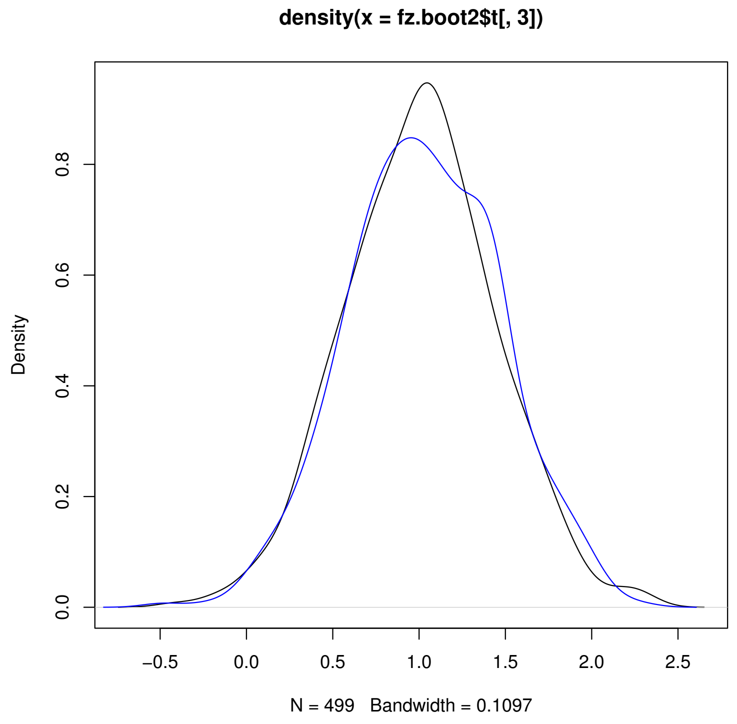
```
fz.fit <- function(d)
    coef(glm(log(d$Time) ~ log(d$WBC / 10000) + d$AG))
fz.case <- function(data, i)
    fz.fit(data[i,])
fz.boot1 <- boot(fz, fz.case, R=499)
```

- A model-based bootstrap is computed by

```
fz.res <- residuals(fz.lm) / sqrt(1 - fz.diag$h)
fz.res <- fz.res - mean(fz.res)
fz.df <- data.frame(fz, res = fz.res, fit = fitted(fz.lm))
fz.model <- function(data, i) {
    d <- data
    d$Time <- exp(d$fit + d$res[i])
    fz.fit(d)
}
fz.boot2 <- boot(fz.df, fz.model, R=499)
```

A comparison of the bootstrap distributions for $\widehat{\beta}_2$:

```
plot(density(fz.boot2$t[,3]))
lines(density(fz.boot1$t[,3]),col="blue")
```



Method	$\widehat{\beta}_2$ or $\overline{\widehat{\beta}}_2^*$	Stand. Error
Least Squares	0.9883	0.4361
Case-based Bootstrap	1.0187	0.4389
Model-based Bootstrap	0.9686	0.4353

Bootstrap Confidence Intervals

Simple Intervals

- A simple interval can be based on $T - \theta$ being approximately normal with mean B^* and variance V^*:

$$[\theta_L, \theta_U] = T - B^* \pm \sqrt{V^*} z_{1-\alpha/2}$$

- With the variance stabilizing logarithm transformation this becomes

$$[\theta_L, \theta_U] = \exp\{\log T - B_L^* \pm \sqrt{V_L^*} z_{1-\alpha/2}\}$$

  with B_L^* and V_L^* the bootstrap estimates of bias and variance for $\log T$.

- Results for the air conditioner data and $\alpha = 0.05$:

Parametric	Log Scale	θ_L	θ_U
yes	no	0.002152	0.01464
no	no	-0.000165	0.01618
yes	yes	0.004954	0.01585
no	yes	0.004238	0.01783

Basic Bootstrap Intervals

- If $P(T-\theta \leq a) = \alpha/2$ and $P(T-\theta \geq b) = \alpha/2$ then

$$P(a \leq T-\theta \leq b) = P(T-b \leq \theta \leq T-a) = 1-\alpha$$

- We can estimate a, b by $T^*_{((R+1)\alpha/2)} - T$ and $T^*_{((R+1)(1-\alpha/2))} - T$ to get the interval

$$\begin{aligned}[\theta_L, \theta_U] &= [T - (T^*_{((R+1)(1-\alpha/2))} - T), T - (T^*_{((R+1)\alpha/2)} - T)] \\ &= [2T - T^*_{((R+1)(1-\alpha/2))}, 2T - T^*_{((R+1)\alpha/2)}]\end{aligned}$$

- This can be done on a variance stabilizing scale as well; for the logarithm:

$$[\theta_L, \theta_U] = [T^2/T^*_{((R+1)(1-\alpha/2))}, T^2/T^*_{((R+1)\alpha/2)}]$$

- Results for the air conditioner data and $\alpha = 0.05$:

Parametric	Log Scale	θ_L	θ_U
yes	no	0.0006166	0.01286
no	no	-0.0026972	0.01325
yes	yes	0.0047855	0.01516
no	yes	0.0040375	0.01628

Studentized Bootstrap Intervals

- Suppose we have
  - an estimator T of θ
  - an estimator U of the variance of T, e.g. from the delta method
- From the bootstrap samples compute
  - estimates $T_1^*, \dots, T_R^*$
  - variance estimates $U_1^*, \dots, U_R^*$
- Compute and rank studentized values

$$Z_i^* = \frac{T_i^* - T}{\sqrt{U_i^*}}$$

- The studentized interval is then

$$[\theta_L, \theta_U] = [T - \sqrt{U} Z^*_{((R+1)(1-\alpha/2))}, T - \sqrt{U} Z^*_{((R+1)\alpha/2)}]$$

- For the air conditioner data the delta method variance estimate for T is

$$U = \frac{1}{n} \frac{S^2}{\overline{X}^4}$$

  For $\log T$ the variance estimate is

$$U_L = \frac{1}{n} \frac{S^2}{\overline{X}^2}$$

Other Bootstrap Confidence Intervals

- Percentile Methods
  - Basic: $[T^*_{((R+1)\alpha/2)}, T^*_{((R+1)(1-\alpha/2))}]$
  - Does not work very well with a nonparametric bootstrap
  - Modified: BC$_a$ method
- ABC method.
- Double bootstrap.
- ...

Bootstrap and Monte Carlo Tests

- Simulation can be used to compute null distributions or p-values.
- If the distribution of the test statistic under H_0 has no unknown parameters, then the p-value
$$P(T \geq t | H_0)$$
can be computed by simulation with no statistical error.
- If the distribution of the test statistic under H_0 does have unknown parameters but a sufficient statistic S for the model under H_0 is available, then it may be possible to compute the conditional p-value
$$P(T \geq t | S = s, H_0)$$
by simulation with no statistical error.
- In other cases, an approximate p-value
$$P(T \geq t | \widehat{F_0})$$
is needed.
- Transformations and reparameterizations can help reduce the statistical error in these cases.

Parametric Tests: Logistic Regression

- Suppose $Y_1, \dots, Y_n$ are independent binary responses with scalar covariate values $x_1, \dots, x_n$.
- A logistic regression model specifies

$$\log \frac{P(Y_i = 1|x_i)}{P(Y_i = 0|x_i)} = \beta_0 + \beta_1 x_i$$

- Under $H_0 : \beta_1 = 0$, $S = \sum Y_i$ is sufficient for β_0.
- A natural test statistic for $H_1 : \beta_1 \neq 0$ is

$$T = \sum x_i Y_i$$

- Conditional on $S = s$ and H_0 the distribution of the Y_i is uniform on the $\binom{n}{s}$ possible arrangements of s ones and $n - s$ zeros.
- We can simulate from this distribution by generating random permutations.
- Another (in this case silly) option is to use MCMC in which each step switches a random y_i, y_j pair.

Parametric Bootstrap Tests: Separate Families

- For the air conditioner data we might consider testing

$$\begin{aligned} H_0 &: \quad \text{data have a Gamma}(\alpha, \beta) \text{ distribution} \\ H_1 &: \quad \text{data have a Log Normal}(\mu, \sigma^2) \text{ distribution} \end{aligned}$$

- A test can be based on

$$T = \frac{1}{n} \sum \log \frac{f_1(x_i | \widehat{\mu}, \widehat{\sigma}^2)}{f_0(x_i | \widehat{\alpha}, \widehat{\beta})}$$

- A normal approximation for the null distribution of T exists but may be inaccurate.
- The bootstrap null distribution of T is computed by:
  - Generate R random samples from the fitted null model, Gamma$(\widehat{\alpha}, \widehat{\beta})$.
  - For each sample calculate the MLE's $\widehat{\alpha}^*$, $\widehat{\beta}^*$, $\widehat{\mu}^*$, and $\widehat{\sigma}^{*2}$.
  - Using these MLE's compute T^* for each sample
- For small samples this sort of test may not be useful:
  - For the air conditioner data there is no significant evidence for rejecting a Gamma model in favor of a log normal model.
  - There is also no significant evidence for rejecting a log normal model in favor of a Gamma model.

  Both p values are quite large.

Nonparametric Permutation Tests: Correlation

- A test for independence of bivariate data $(X_1,Y_1),\ldots,(X_n,Y_n)$ can be based on the sample correlation
$$T = \rho(\widehat{F})$$
- Under the null hypothesis of independence the two sets of order statistics are sufficient.
- The conditional null distribution of T, given the order statistics, is the distribution of
$$T^* = \rho((X_{(1)},Y_1^*),\ldots(X_{(n)},Y_n^*))$$
where $Y_1^*,\ldots,Y_n^*$ is drawn uniformly from all permutations of $Y_1,\ldots,Y_n$.
- The null distribution of T can be simulated by randomly selecting permutations and computing T^* values.
- Any other measure of dependence can be used as well, for example a rank correlation.

Semi-Parametric Bootstrap Test: Equality of Means

- Suppose we have data

$$Y_{ij} = \mu_i + \sigma_i \varepsilon_{ij}, \quad j = 1, \ldots, n_i; i = 1, \ldots, k$$

  and we assume the ε_{ij} are independent from a common distribution G.

- To test $H_0 : \mu_i = \cdots = \mu_k$ against a general alternative we can use

$$T = \sum_{i=1}^{k} w_i (\overline{Y}_i - \widehat{\mu}_0)^2$$

  with

$$\begin{aligned} \widehat{\mu}_0 &= \sum w_i \overline{Y}_i / \sum w_i \\ w_i &= n_i / S_i^2 \end{aligned}$$

- The null distribution would be approximately χ^2_{k-1} for large sample sizes.
- The null model studentized residuals are

$$e_{ij} = \frac{Y_{ij} - \widehat{\mu}_0}{\sqrt{\widehat{\sigma}_{i0}^2 - (\sum w_i)^{-1}}}$$

  with

$$\widehat{\sigma}_{i0}^2 = (n_i - 1) S_i^2 / n_i + (\overline{Y}_i - \widehat{\mu}_0)^2$$

- The bootstrap simulates data sets

$$Y_{ij}^* = \widehat{\mu}_0 + \widehat{\sigma}_{i0} \varepsilon_{ij}^*$$

  with the ε_{ij}^* sampled with replacement from the pooled null studentized residuals.

Fully Nonparametric Bootstrap Tests: Comparing Two Means

- Suppose Y_{ij}, $i = 1,2$, $j = 1,\ldots,n_i$, are independent with $Y_{ij} \sim F_i$ and means μ_i.
- We want to test $H_0 : \mu_1 = \mu_2$.
- Nonparametric maximum likelihood estimates F_1 and F_2 by discrete distributions concentrated on the observed data values.
- A certain nonparametric maximum likelihood argument suggests that the maximum likelihood probabilities, under the constraint of equal means, are of the form

  $$\widehat{p}_{1j,0} = \frac{\exp(\lambda y_{1j})}{\sum_{k=1}^{n_1} \exp(\lambda y_{1k})}$$
  $$\widehat{p}_{2j,0} = \frac{\exp(-\lambda y_{2j})}{\sum_{k=1}^{n_2} \exp(-\lambda y_{2k})}$$

  with λ chosen numerically to satisfy

  $$\frac{\sum y_{1j} \exp(\lambda y_{1j})}{\sum \exp(\lambda y_{1j})} = \frac{\sum y_{2j} \exp(-\lambda y_{2j})}{\sum \exp(-\lambda y_{2j})}$$

  These are called *exponential tilts* of the empirical distribution.
- The tilted bootstrap two-sample comparison:
  - Generate Y_{1j}^*, $j = 1,\ldots,n_1$ by sampling from y_{1j} with weights $\widehat{p}_{1j,0}$
  - Generate Y_{2j}^*, $j = 2,\ldots,n_2$ by sampling from y_{2j} with weights $\widehat{p}_{2j,0}$
  - Compute $T^* = \overline{Y}_2^* - \overline{Y}_1^*$
  - Repeat R times

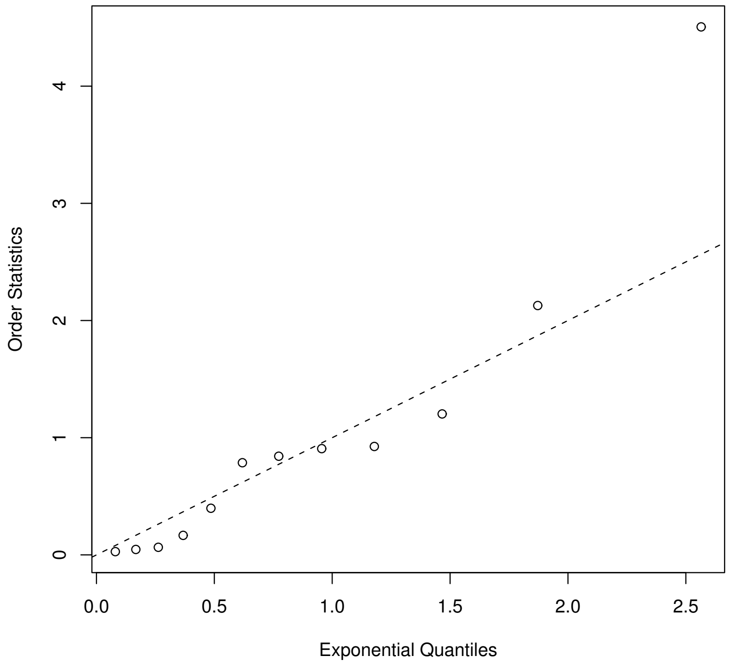
  Then compute a bootstrap p-value, say

  $$p = \frac{1 + \#\{T_i^* \geq T\}}{R+1}$$

- It is a good idea to look at the two tilted distributions to make sure they are not too silly.

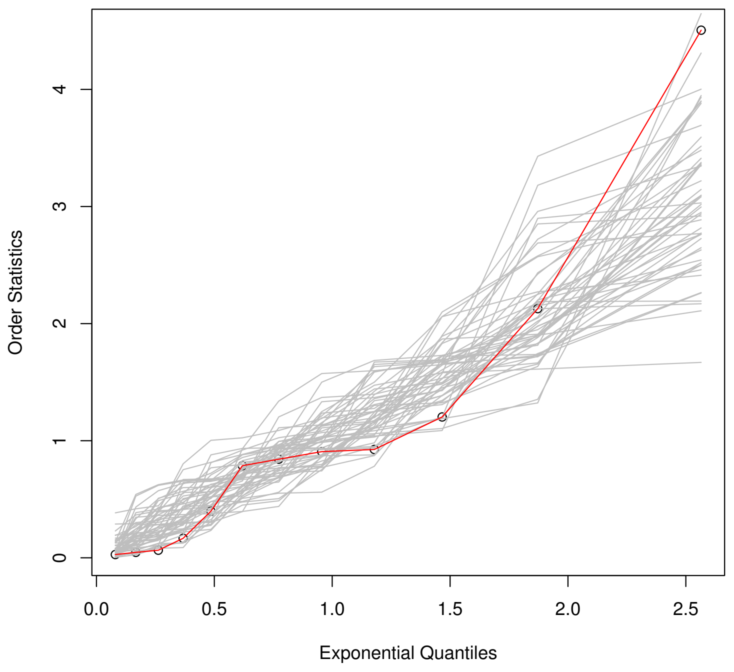
Graphical Tests

- Bootstrapping and resampling ideas can be used to help calibrate graphical procedures.
- As an example, a plot of $X_{(i)}/\overline{X}$ against the quantiles of a unit exponential distribution can be used to asses the appropriateness of the exponential model for the airconditioner data:



- The dashed line is the theoretical line for large samples.
- Is the departure more than might be expected by chance for a sample of this size from an exponential distribution?

- A bootstrap approach:
  - Generate a sample $Y_1^*, \ldots, Y_n^*$ from an exponential distribution (the mean does not matter)
  - plot $Y_{(i)}^* / \overline{Y}^*$ against the unit exponential quantiles $-\log(1 - i/(n+1))$
  - Repeat and add to the plot
- For the air conditioner data with $R = 50$ replications this produces



- Some references on similar ideas:
  - Andreas Buja (1999) Talk given at the Joint Statistics Meetings 1999, on the possibility of valid inference in exploratory data analysis, with Di Cook: Inference for Data Visualization
  - Andrew Gelman (2004) "Exploratory data analysis for complex models." *Journal of Computational and Graphical Statistics*.
  - Hadley Wickham, Dianne Cook, Heike Hofmann, and Andreas Buja (2010) "Graphical inference for infovis." *IEEE Transactions on Visualization and Computer Graphics*

Parallel Computing

- Many computations are virtually instantaneous.
- Some would take hours, days, or months.
- Often multiple processors are available
  - multiple workstations
  - dedicated cluster
- Ideally, p processors should be p times faster than one processor.
- Usually not quite achievable because of uneven granularity, communication overhead.
- Sophisticated parallel algorithms are hard.
  - can be programmed using PVM, MPI
  - R interfaces are packages `rpvm`, `Rmpi`
- Some problems are *embarrassingly parallel*.
- Running embarrassingly parallel programs in parallel should be simple.

Package `snow` (Simple Network of Workstations)

- uses PVM, MPI or sockets for communication.
- manages a cluster of R worker processes from a master process.
- some simple examples:
  - Create a cluster of 10 R worker processes:
    ```
    library(snow)
    cl <- makeCluster(10)
    ```
  - Call function on all nodes:
    ```
    clusterCall(cl, exp, 1)
    ```
  - Evaluate an expression on all nodes:
    ```
    clusterEvalQ(cl, library(boot))
    ```
  - Apply function to list, one element per node:
    ```
    clusterApply(cl, 1:5, get("+"), 2)
    ```
  - Parallel `lapply`
    ```
    > unlist(parLapply(cl, 1:15, get("+"), 2))
     [1]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
    ```
  - Parallel `sapply`
    ```
    > parSapply(cl, 1:15, get("+"), 2)
     [1]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
    ```
  - Stop the cluster:
    ```
    stopCluster(cl)
    ```

Parallel Random Numbers

- Random number generation needs help:

```
> clusterCall(cl, runif, 3)
[[1]]
[1] 0.4351672 0.7394578 0.2008757
[[2]]
[1] 0.4351672 0.7394578 0.2008757
...
[[10]]
[1] 0.4351672 0.7394578 0.2008757
```

- Identical streams are likely, not guaranteed.
- R interfaces to several libraries are available: `rsprng`, `rlecuyer`, `rstream`.
- `snow` provides a convenient interface:

```
> clusterSetupRNG(cl)
> clusterCall(cl, runif, 3)
[[1]]
[1] 0.014266542 0.749391854 0.007316102
[[2]]
[1] 0.8390032 0.8424790 0.8896625
...
[[10]]
[1] 0.591217470 0.121211511 0.002844222
```

  `rlecuyer` is currently the default package used.

Example: Parallel Bootstrap

- Bootstrapping is embarrassingly parallel.
- Replications can be split onto a cluster.
- Random number streams on nodes need to be independent.
- `boot` package allows bootstrapping of any R function.
- Help page shows example of bootstrapping `glm` fit for data on the cost of constructing nuclear power plants.
- 1000 replicates on a single processor:

```
> wallTime(nuke.boot <-
+    boot(nuke.data, nuke.fun, R=1000, m=1,
+         fit.pred=new.fit, x.pred=new.data))
[1] 27.44
```

- Parallel version: 100 replicates on each of 10 cluster nodes:

```
> clusterSetupRNG(cl)
> clusterEvalQ(cl,library(boot))
> wallTime(cl.nuke.boot <-
+    clusterCall(cl,boot,nuke.data, nuke.fun, R=100, m=1,
+                fit.pred=new.fit, x.pred=new.data))
[1] 3.03
```

- More details are available at

  `http://www.stat.uiowa.edu/~luke/R/cluster/cluster.html`

Final Notes

Topics We Covered

- computer basics
- numerical linear algebra
- optimization
- smoothing
- a little machine learning
- simulation
- MCMC
- a (very) little parallel computing

Some Topics We Did Not Cover

- bootstrap
- graphics and visualization
- data technologies
- numerical integration
- symbolic computation
- data mining
- pattern recognition
- functional data analysis
- Parallel computing for big data
  - Distributed arrays, `pbd` packages.
  - MapReduce, distributed file systems, and Hadoop.
  - Distributed data frames and Spark. Some examples on line.
- Cloud computing

Homework and Projects

Some of the things you got from these:

- practice writing non-trivial programs that work
- appreciation of basic numerical issues
- experience using the computer to explore ideas
- a better understanding of R
- experience in understanding and extending a statistical analysis framework
- practice in conducting simulation experiments
- a deeper understanding of some topic of your choice