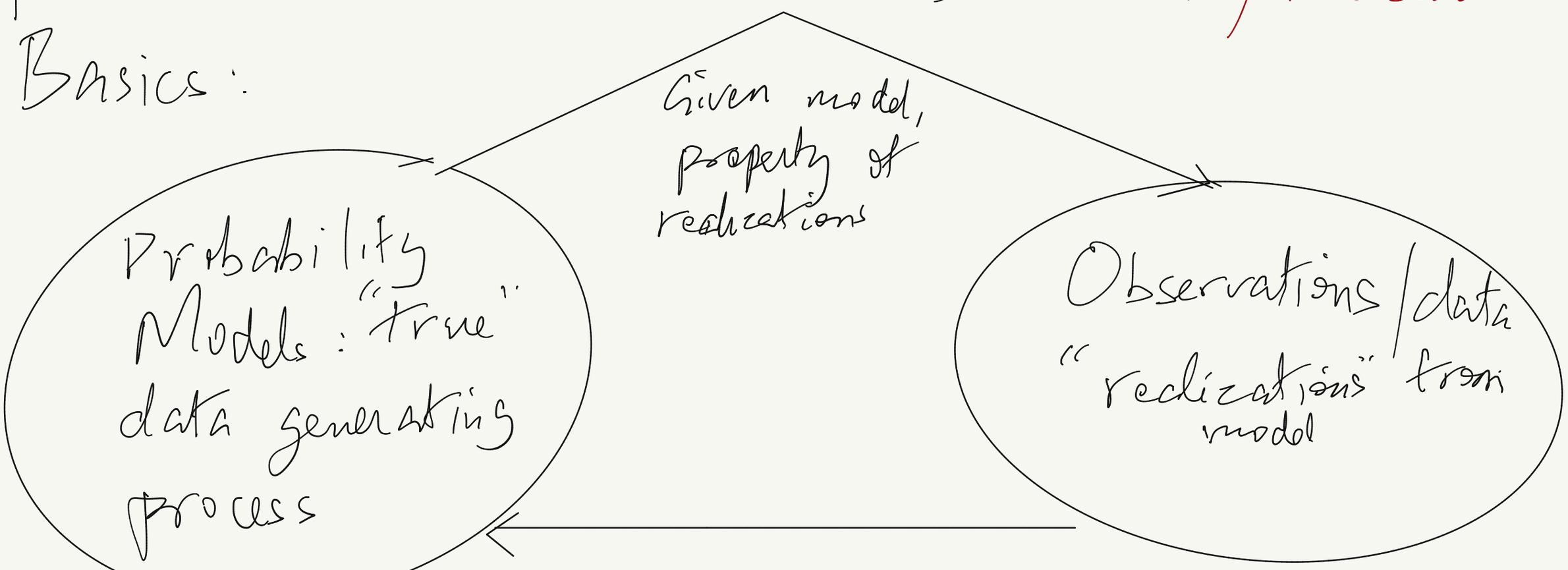
Stat 540

Aug. 2018

Typical intro. to prob. theory + stat. inference: (Stat 513+ 514)
most problems are solved analytically e.g. using tools from multivariate calculus.
In practice: only a small number of problems admit analytical solutions.
Most require computer algorithms.
This is the subject of this course: learn computer algorithms in order to solve probability and stat. inf. problems.

[Above topics: required background for this class!]

Basics:



Statistical Inference:
learn about model based on data computing: optimization; also simulation/Monte Carlo
e.g. Bayesian inf., hyp. testing, goodness of fit

Of course, prob. is central to stat. inf.:
without understanding model, cannot carry out
inference. e.g. need joint distr. to obtain likelihood fn.
e.g. properties of fitted prob. model: used to
assess goodness of fit of model to data.

Probability calculations can be analytically intractable
but easy w/ computer algorithms.

Simple e.g. $\underset{\sim}{X} \sim N_d(\underset{\sim}{\mu}, \Sigma)$. X_{max}, X_{min}: max and min of vector $\underset{\sim}{X}$
For some $\underset{\sim}{\mu}, \Sigma$, what is $\mu = P(X_{max} | X_{min} > c)$?

Analytical form of $X_{max} | X_{min}$ complicated.
E.g. X_{max}, X_{min} distr. for bivariate Normal
(Nadarajah & Kotz, 2008)

But Monte Carlo approximation is easy:
Simulate $\underset{\sim}{X}^{(1)}, \ldots, \underset{\sim}{X}^{(n)} \overset{i.i.d}{\sim} N(\underset{\sim}{\mu}, \Sigma)$

Approximation to μ, $\hat{\mu}_n = \sum_{i=1}^{n} I\left(X_{max}^{(i)} / X_{min}^{(i)} > c\right) / n$

Generalizes to more complicated joint distributions
beyond $N_d(\mu, \Sigma)$, provided we can simulate them.
And to more complicated functions (not just
X_{max} / X_{min}).

Statistical Inference

Simple e.g. $Y_i = \beta X_i + \delta_i + \varepsilon_i$ $\quad \delta_1, \ldots, \delta_n \overset{iid}{\sim} \text{Gamma}(\alpha, \theta X_i)$, $\varepsilon_1, \ldots, \varepsilon_n \overset{iid}{\sim} N(0, \sigma^2)$

Observe: (X_i, Y_i), $i = 1, \ldots, n$

MLE for $\beta, \alpha, \theta, \sigma^2 = \underset{\beta, \alpha, \theta, \sigma^2}{\arg\max}\ l(\beta, \alpha, \theta, \sigma^2; \underset{\sim}{X}, \underset{\sim}{Y})$

Maximize via computer algorithm: e.g. Newton-Raphson alg.

Bayesian inf.: $\pi(\alpha, \beta, \theta, \sigma^2 | \ldots) \propto l(\quad ; (X_1, Y_1), \ldots, (X_n, Y_n)) \times p(\alpha, \beta, \theta, \sigma^2)$ ⇐ prior distr.

MCMC to approximate $\pi(\ | \ldots)$

OR optimization to maximize $\pi(\ | \ldots)$ w.r.t. $(\alpha, \beta, \theta, \sigma^2)$

Modern stats: lots of computing challenges.

Many topics I don't cover will be covered by you in projects!

Projects: central to this course

Talk to me <u>now</u> about potential topics

Substantial prelim. work: complete in 6 weeks

May have to modify topic so good to start early. In past: poor projects typically from students who do not discuss project w/ me.

Project must be primarily focused on algorithms — nice applications are okay, but they can only be motivation, not focus.

Topics and organization of course

[I.] Preliminaries: { Computing basics e.g. precision, complexity
Computational matrix algebra

[II.] Probability: { numerical integration / approximation
Monte Carlo
Markov chain Monte Carlo (MCMC)

[III.] Inference: { Overview of inference
" of optimization for stats
Newton-Raphson & variants (2nd order)
Gradient / stochastic gradient alg (1st order)
Bootstrap
EM & MM + MCEM
MCMC for inference
Variational methods
Likelihood-free methods (e.g. ABC)
Composite likelihood
⋮

Several methods for high-dim data, complex models

+ Student projects (last 2 weeks)

Updates to schedule on course website

Computing preliminaries

(1) Computational complexity / cost of algorithms
- Mathematical elegance is not same as computational efficiency

(2) Computer precision, stability

(3) Pseudo-randomness

(4) Computer memory

(5) Parallel computing

End of Lecture 1
August 21, 2018

Clarity of thinking about computing issues vs statistical issues

"Separation of concerns": keeping statistical ideas separate from computing ideas (original quote from E.W. Dijkstra "A discipline of programming")

E.g. "My bootstrap CIs were narrower than my MCMC intervals"

Better: My percentile bootstrap CIs based on ML estimates were narrower than my 95% credible intervals from a Bayesian approach using MCMC.

E.g. I used EM to estimate parameters, then I used MCMC.

Better: I estimated params by MLE using the EM algorithm, then I used a Bayesian approach, implemented via MCMC.

Why: Same algorithm can be used for different purposes:
MCMC can be used for MLE too!

Also: clearer thinking separates computing issues e.g. Monte Carlo error from inference issue e.g. standard error.

① Computational complexity

Useful to use order notation for algorithm run times.

Let n be the unit of the # of operations

- "Big Oh" (O-mega!) $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ $f: \mathbb{Z}^+ \to \mathbb{R}^+$, $g: \mathbb{Z}^+ \to \mathbb{R}^+$

$O(g(n))$ is a class of functions $f(n) \in O(g(n))$ for which $\exists k > 0$, $\exists n_0$ s.t. $\forall n > n_0$,

$$f(n) \leq k\, g(n)$$

That is, f is bounded asymptotically "from above by g." "grows at the same rate or slower than"

E.g. $n^3/3 \in O(n^3)$

$n^2 \in O(n^2+n)$

$n^2 \in O(10\, n^2)$

Big-Oh: "is of the same order as". Hence, popular for alg. cost

- "Little oh" (o-micron!) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$o(g(n))$ is a class of functions $f(n) \in o(g(n))$ for which $\exists n_0$ s.t. $\forall n > n_0$, $\forall k > 0$,

$$f(n) \leq k\, g(n)$$

That is, f is dominated asymptotically by g. "grows strictly slower than"

E.g. $n^2 \in o(n^3)$

$n^2 \in o(n!)$

$\ln(n) \in o(n)$

Nok: $n^2 \notin O(n^3)$, $n^2 \notin O(n!)$, $\ln(n) \notin O(n)$

Of course, $f \in o(g) \Rightarrow f \in O(g)$ but not vice-versa

- Theta notation

$\Theta(g(n))$ is a class of functions $f(n) \in \Theta(g(n))$ for which $\exists k_1 > k_2 > 0$, $\exists n_0$ s.t. $\forall n > n_0$,

$$k_2\, g(n) \leq f(n) \leq k_1\, g(n)$$

That is, f is bounded asymptotically from above and below by g.

Why is this useful? Allows us to describe, in fairly simple terms, the computational cost of an algorithm. Eg. take two $n \times n$ matrices A, B.

$A+B$ or $A-B$ involves n^2 calculations

Each calculation is a "floating point operation" (flop)

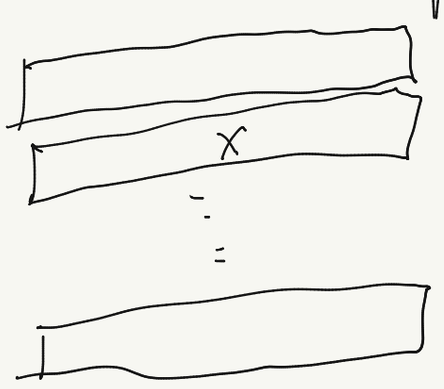
1 flop each: +, −, ×, / for $\mathbb{R}$

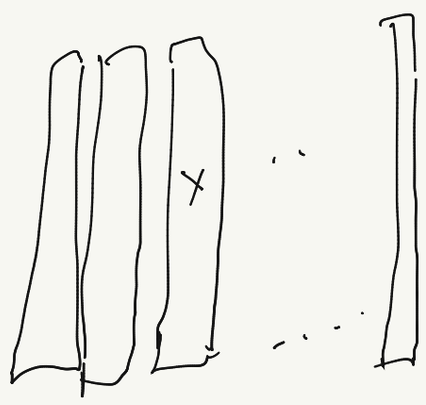
R matrix multiplication example

Matrix multiplication: $C = AB$ Naive:

$C_{23} =$ 

$= n$ multiplications + $(n-1)$ additions





Total # of calculations $= n^2 \times (2n-1) \in O(n^3)$ flops

So, costs increase at the rate n^3 w/ dimension of the matrix. Helps w/ understanding how alg. scales w/ size of matrix.

Common to use Strassen's alg: $O(n^{2.8})$ but less stable and uses more memory — not good for very large matrices

What to do in practice about computing costs:

(1) Begin by "profiling" your code: this helps identify bottlenecks - parts of code that take the most time. Helpful since full-fledged complexity analysis can be difficult.
In R: Rprof

(2) Be aware of computational costs of any potentially expensive parts of the code. E.g. Golub and Van Loan book; Numerical recipes in C; Cormen, Leicersen, ... book

(3) Counting flops not enough! (1) numerical solvers (e.g. BLAS, LAPACK) are optimized to hardware
(2) memory plays a role in costs (more on this later)
(3) file I/O can be expensive. e.g. pay attention to how often you write to file

(4) algorithms can be iterative w/ some stopping criteria
e.g. eigen decomposition

(4) If expensive operations are repeated, pre-calculate and re-use. E.g. some parts of likelihood may stay the same multiple iterations of an algorithm - $l(\theta, \phi; x) = l_1(\theta, \phi; x) + l_2(\theta; x) + l_3(\phi; x)$

no need to re-evaluate if ϕ same

(5) Compiled language may often be much faster
E.g. C/C++/FORTRAN. Easy to do this in R.
Roger Peng's or C.J. Geyer's notes online to connect C code to R.

General rule: your time is more valuable than computer time - code in compiled language if enough payoff!

End of lecture 2, August 23, 2018

(6) Worth being aware of how languages handle matrices
E.g. In R [ ] $1000 \times 1{,}000{,}000$
is v. hard to read in
but [ ] is manageable
$1{,}000.000 \times 1{,}000$

② Computer precision / stability

Computers are not infinitely precise. This can affect accuracy of calculations.

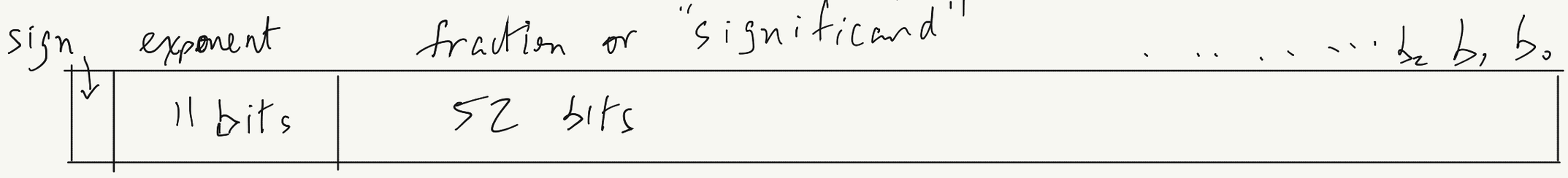
Basics: represent #s as a sequence of 0's and 1's

"bits": term coined by statistician John Tukey
↑
binary digits

[Group of 8 bits is a "byte" (typically) — historically used to represent a single text character]

Representation of #s either 32 bit or 64 bit

IEEE-754 double precision representation:



$$(-1)^{sign}\left[1+\sum_{i=1}^{52} b_{52-i}\,2^{-i}\right]\times 2^{expo-1023}$$

E.g. 1 | 1000...011 | 00.... 10001

$$(-1)^1\times\left[1+\left(\frac{1}{2}\right)^{48}+\left(\frac{1}{2}\right)^{52}\right]\times 2^{(1024+481)-1023}$$

$$\left[=-1.\ldots 037746\times 2^{6}\right]$$

E.g. Exponent 0...1 : 2^{-1022}

11...10 : $2^{2046-1023}=2^{1023}$

E.g. 0 | 011...11 | 0.....010

$$2^{1023-1023}\times\left[1+0+\ldots+\left(\frac{1}{2}\right)^{51}+0\right] = 1.0\ldots 0.4\ldots \approx 1+4\times 10^{-16}$$

In practice: need to pay attention to

(1) how precisely numbers are stored - this can affect results.

(2) how numbers are displayed. In R:
good to learn how to use: paste, format, sprintf
— default presentation of numbers can obscure what is really going on.

Add R examples

(3) common to encounter numbers too large or small to be represented on a computer.
Good to work in log scale in many problems
e.g. frequently in likelihood calculations for complicated models.

$$\frac{\mathcal{L}(\Phi; \underline{x})}{\mathcal{L}(\phi; \underline{x})},$$

use $l(\Phi; \underline{x}) - l(\phi; \underline{x})$

Stability

Calculation carried out on a computer will, due to rounding, often produce results that are different from results based on theory.

Of interest to us: how close is the computer result to the theoretical result?

[Reference: Stewart & Voss, Numerical Linear Algebra]

Backward error analysis: view computed solution to a problem as exact solution to a perturbed problem.

Small perturbation $\Rightarrow$ algorithm is "backward stable"

Not small perturbation $\Rightarrow$ algorithm is "unstable"

We will discuss this more in context of numerical linear algebra.

(3) Pseudo-randomness

Deterministic algorithms are used to mimic random sequences on a computer

Algorithms look roughly like this:

(1) Start w/ initial value u_0, called the "seed". Like an initial state. Vector of integers

(2) Construct a deterministic sequence $u_1, u_2, \ldots$ where successive values use previous values

Simple e.g. $u_{k+1} = f(u_k)$, $k = 1, 2, \ldots$

(3) Resulting sequence of integers $\in [0, u_{max}]$ where u_{max} is set by the algorithm, typically related to how numbers are represented

sequence: $u_0/u_{max}, u_1/u_{max}, \ldots \in [0,1]$

Good random number generator: sequence is virtually "indistinguishable" from Unif(0,1) iid sequence.

$\Leftrightarrow$ Pass several tests e.g. "Diehard battery" (Marsaglia, 1960s, 1970s)

Very Popular: Mersenne-Twister (1997) default in R, Python, Matlab...

All random number generation algorithms are based on iid sequence of Unif(0,1) random variables

Practical consequences:

(1) Deterministic: if use same seed, get same sequence

But this is an advantage more than a disadvantage: helps w/ reproducible work.

Also, useful for debugging code.

In R: set.seed(x)

- Random.seed is global variable that stores state of random number generator

Very useful for long, potential interrupted runs of MCMC, or stochastic gradient alg.

Care w/ parallel computing: make sure different seeds on different processors!

(2) Random number generator has a "period": eventually returns to initial state, then reproduces same sequence: generally a non-issue

E.g. Mersenne-Twister's period is 2^{19337}

End of Lecture 3,
Tuesday August 28,
2018

(4) Computer memory basics

Hierarchy : cache : small, v. fast access } O.S. keeps most frequently accessed items here
RAM : bigger, next fastest } next. O.S. uses RAM

Hard drive / swap space: when oper. sys. runs out of RAM, internal hard drive is used to store items temporarily not being used. Writing / retrieving items from various kinds of memory is called swapping.

Practical consequences:

Useful to know if you are using a lot of RAM, e.g. by constantly accessing huge matrices. This can result in a lot of swapping, which can really slow things down.

This is an e.g. of why calculating real computational costs of an algorithm can be very complicated.

- Careful of algorithms that require lots of memory
  E.g. Strassen's alg vs simple matrix multiplication.

Often tradeoff between memory use and flops.

- Avoid too much dynamic allocation of memory.

E.g. in R

```
myvec = c(1)
for (i in 2:M)        <- dynamic memory allocation
    myvec = c(myvec, myvec[i-1] + i)
```

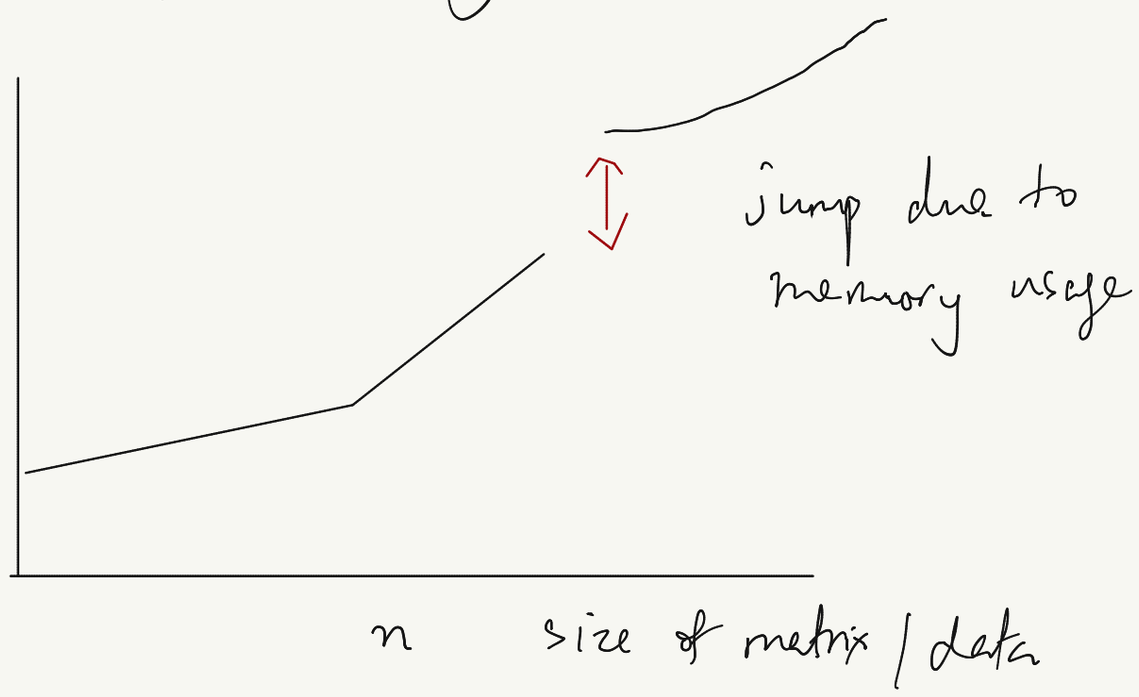
Better :

```
myvec = rep(NA, M)   # fixed allocation of memory
myvec[1] = 1
for (i in 2:M)
    myvec[i] = myvec[i-1] + i
```

In practice, common to not know length/size of data structures in advance. Still good to avoid many memory allocations: allocate estimated size in advance, then increase if necessary, reduce how often this is done.

Point process problem??

Due to memory issues, scaling can look like this:



Parallel computing

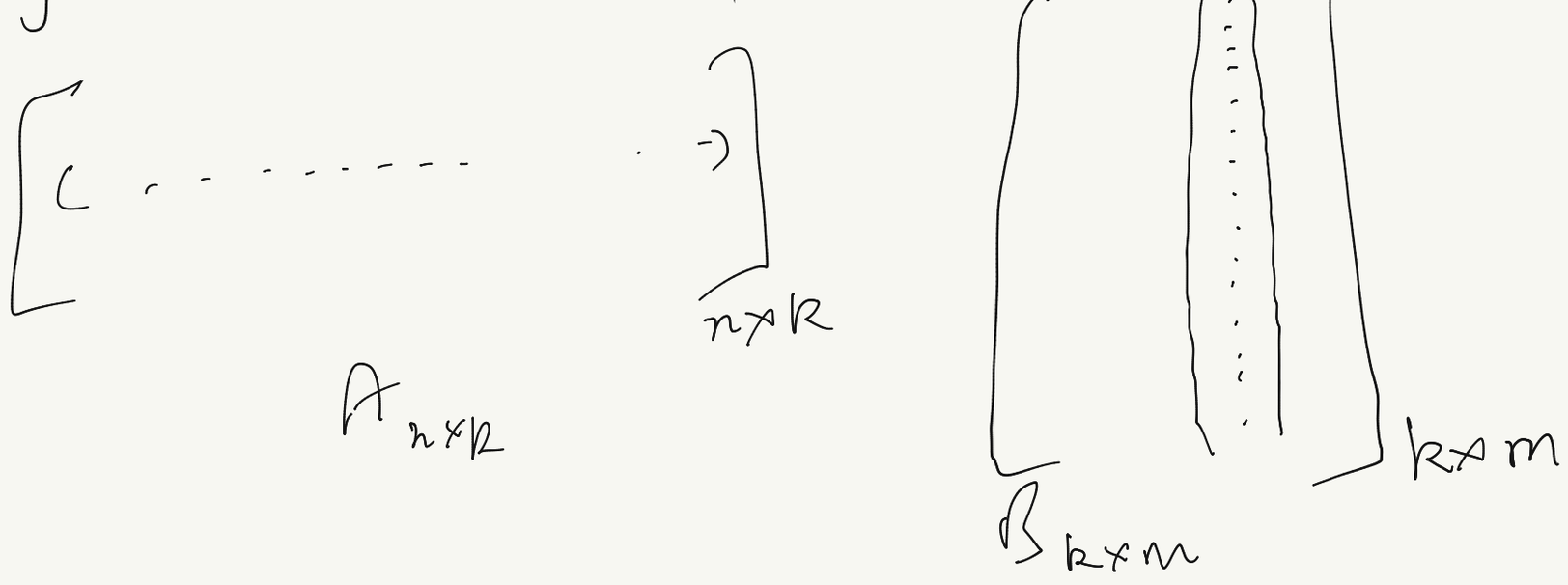
High performance computing (HPC) broadly refers to the use of multiple processors to solve challenging computing problems.

Learning about how to parallelize algorithms effectively is therefore a very useful skill in modern scientific computing.

In statistics, lots of opportunities for parallel computing.

Embarrassingly parallel problem: one where very little effort is required to separate the problem into a number of parallel tasks.

E.g. matrix multiplication $O(nmk^2)$ flops



Each element, C_{nm}, in matrix, may be computed independently
Easy to parallelize. If nm processors: k^2 + overhead for distributing and gathering results

Cost depends on # of processors
Speed / capacity of processors
requirements of parallelized computations
Communication overhead

Gains depend on size of problem + available computing
(Amdahl's bound, 1967)

E.g. if $\mathcal{L}(\underset{\sim}{\theta}; \underset{\sim}{x}) = \prod_{i=1}^{n} \mathcal{L}(\theta_i; x_i)$

$$l(\underset{\sim}{\theta}; \underset{\sim}{x}) = \sum_{i=1}^{n} l(\theta_i; x_i)$$ ← parallelize these calculations, each of which might be expensive

E.g. Simulation studies: see RMSE of estimate $\hat{\theta}$ as true value of θ varies.

$\theta_1 \rightarrow \widehat{RMSE}(\theta_1)$
$\vdots$
$\theta_K \rightarrow \widehat{RMSE}(\theta_K)$

} each on a different processor

Matlab & R have capabilities for simple parallelization
e.g. snow (simple network of workstations) package.

Graphical processing units (GPUs): huge number of processors, each w/ very limited capabilities.

When not embarrassingly parallel, can require a lot more creativity

Numerical Linear Algebra

Let real matrix $A \in \mathbb{R}^{m \times n}$ w/ $m \geq n$
$b \in \mathbb{R}^m$

Three main problems:

(1) Solving system of linear equations. Let $m = n$

Find $x \in \mathbb{R}^n$ s.t. $Ax = b$

E.g. matrix inversion in multivariate normal pdf

$$f(x; \mu, \Sigma) = (2\pi)^{-n/2} |\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right\}$$

In practice, should try to avoid taking inverses

(2) Least Squares.

Find $x \in \mathbb{R}^n = \underset{x \in \mathbb{R}^n}{\arg\min} \|Ax - b\|^2$

E.g. regression notation: find $\hat{\beta} = \underset{\beta \in \mathbb{R}^d}{\arg\min} \|y - X\beta\|^2$

where X: $n \times p$ matrix, p predictors, n data points

β: p-dimensional vector of reg. coefficients

y: n-dimensional " " responses

[In this case, simple calculus yields: $\hat{\beta} = (X^T X)^{-1} X^T y$

So inversion again. Very common in high-dimensional problems to have numerical issues when computing $(X^T X)^{-1}$. $X^T X$ is often "ill-conditioned"

If $p > n$: stat issue (non-invertible + not enough data to relate p predictors)

For large n, p: may be invertible in theory but numerical issues

(3) Eigenvalue problem: Let $m = n$

Find $(x, \lambda) \in \mathbb{R}^n \times \mathbb{R}$

$$Ax = \lambda x, \quad \|x\|_2^2 = 1$$

Eigendecompositions are central to principal components analysis.

E.g. dimension reduction via principal components analysis
If # of predictors is large, and many are correlated, use small # of linear combinations of variables that captures most of the variability of the data.

In statistics it is very common to have structured matrices. E.g. covariance matrices may be patterned.

Learn to take advantage of structures to speed up computation.

E.g. of methods for taking advantage of structured matrices.

Spatial or variance components model

$(X_1, \ldots, X_n)^T \sim N(\underset{\sim}{\mu}, \Sigma)$

where $\Sigma_{n\times n} = \sigma^2 \underset{\sim}{I} + H$, where

H can be written as $K_{n\times k} K^T_{k\times n}$ (this is common in spatial stats models)

To evaluate pdf, calculate Σ^{-1}

Cost: $O(n^3)$

Faster: use Sherman-Morrison-Woodbury formula

$A_{n\times n}$, $C_{k\times k}$: invertible Let U: $n\times k$ matrix

V: $k\times n$ matrix

$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$

↑ $k\times k$ matrix

Very useful if A^{-1} is easy e.g. A is diagonal matrix and $k \ll n$

Note: Sylvester's formula: determinant for same form of matrix.

Above e.g. $\Sigma^{-1} = (\sigma^2 I)^{-1} - (\sigma^2 I)^{-1} K \underbrace{(I^{-1} + K^T(\sigma^2 I)^{-1}K)^{-1}}_{\text{inversion of } j\times j \text{ matrix}} K(\sigma^2 I)^{-1}$

Dominant cost: inversion of $j\times j$ and matrix multiplications

Q. Calculate exact cost?

End of Thursday, Aug 30, 2018 lecture

Partitioned matrices

Let A be invertible matrix, partitioned as follows

$$A_{n\times n} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

E.g. when adding new rows (observations) in linear regr.
or " columns (variables) in " "

Rather than work w/ full A, use structure of matrix to get $A^{-1} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$ where

$$C_{11} = \left(A_{11} - A_{12}A_{22}^{-1}A_{21}\right)^{-1}$$
$$C_{22} = \left(A_{22} - A_{21}A_{11}^{-1}A_{12}\right)^{-1}$$
$$C_{12} = -A_{11}^{-1}A_{12}C_{22}$$
$$C_{21} = -A_{22}^{-1}A_{21}C_{11}$$

Note: Sherman-Woodbury-Morrison is a special case where $A^* = \begin{bmatrix} A & u \\ v & -C^{-1} \end{bmatrix}$

Matrix factorizations (stats-centric view)

Cholesky factorization: If $A \in \mathbb{R}^{n \times n}$ is positive definite, then there exists a unique lower triangular $C \in \mathbb{R}^{n \times n}$ w/ positive diagonal elements s.t. $A = CC^T$. (see Strang & Voss for proof)

C is a matrix "square root"

Some uses of Cholesky factorization:

[1] Simulating $\underset{\sim}{X} \sim N_n(\underset{\sim}{\mu}, \Sigma)$ $\underset{\sim}{\mu} \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ $\Sigma > 0$

(1) Assume we can simulate $Z_1, \ldots, Z_n \overset{iid}{\sim} N(0,1)$

$$\underset{\sim}{Z} = (Z_1, \ldots, Z_n)^T \sim N(0, I)$$

(2) Find cholesky factor of Σ, C s.t. $CC^T = \Sigma$

(3) $C\underset{\sim}{Z} \sim N(0, CC^T)$

So $C\underset{\sim}{Z} \sim N(0, \Sigma)$

(4) $\underset{\sim}{X} = C\underset{\sim}{Z} + \underset{\sim}{\mu} \sim N(\mu, \Sigma)$

Computational cost of above algorithm: dominated by Cholesky decomposition $O(n^3)$.

Related: Simulate $x \sim N(0, Q^{-1})$ where Q is a "precision matrix". Common in Markov random field models

Find C s.t. $CC^T = Q$. Solve: $Cy = z$, ← easy because C is lower triangular

where $\underset{\sim}{z} \sim N(0, I)$. Clearly, $y \sim N(0, Q^{-1})$

[2] Evaluating multivariate normal pdfs

$$f(x; \mu, \Sigma) = (2\pi)^{-n/2} |\Sigma|^{-1/2} \exp\left\{ -\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu) \right\}$$

(1) Find Cholesky factor of Σ: C

(2) Now, $(x-\mu)^T \Sigma^{-1} (x-\mu)$

$$= (x-\mu)^T (CC^T)^{-1} (x-\mu)$$

$$= (x-\mu)^T (C^T)^{-1} C^{-1} (x-\mu)$$

So, solve $Cz = (x-\mu)$ for z

Then $z = C^{-1}(x-\mu)$

Compute $z^T z$, which is $[C^{-1}(x-\mu)]^T [C^{-1}(x-\mu)]$

$$= (x-\mu)^T [C^{-1}]^T C^{-1} (x-\mu)$$

$$= (x-\mu)^T \Sigma^{-1} (x-\mu)$$

(3) To find $|\Sigma|^{-1/2}$, note that $|\Sigma|^{-1/2} = |CC^T|^{-1/2}$

where C was already computed above

Now $|CC^T|^{-1/2} = |C|^{-1}$

And $|C|$ is just a product of its diagonal elements since it is triangular.

Main computational cost is due to Cholesky factor again, $O(n^3)$.

[3] Avoiding matrix inversion

Want x s.t. $Ax = b$

That is, $x = A^{-1}b$

(1) write $A = LU$ cholesky factor

(2) Solve $Ly = b$ for y $(y = L^{-1}b)$

(3) Solve $Ux = y$ for x

$(x = U^{-1}y = U^{-1}L^{-1}b = A^{-1}b)$

Back to examples of structured matrices

Already discussed partitioned matrices and matrices amenable to Sherman-Woodbury-Morrison identity - faster inversion and determinants

Now: structured matrices for speeding up Cholesky factorization.

E.g. banded matrices

Consider AR(1) process, a 1st order Markov chain

$x_1 \sim N(0, \sigma^2)$, $x_n | x_{n-1} \sim N(\phi x_{n-1}, \sigma^2)$ for $n = 2, 3, \ldots$

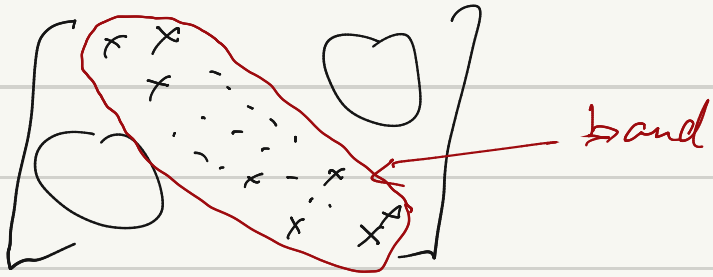
w/ $\phi \in (-1, 1)$, fixed constant

$\Sigma = Cov(x_1, x_2, \ldots, x_n)$ is a "dense" matrix.

That is, $\Sigma = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$

$\Sigma_{ij} > 0 \quad \forall i, j$

Interestingly, precision matrix $Q = \Sigma^{-1}$ is sparse (lots of zeroes). In fact it is a "banded" matrix.

$Q =$ 

This lends itself to much faster Choleski factorization algorithms, $O(n b_w^2)$ where b_w = bandwidth. Above $b_w = 1$

E.g. if $n = 5{,}000$

$n^2 = 1.25 \times 10^{11}$

$n b_w^2 = 5{,}000$ Even if $b_w = 10$, $n b_w^2 = 5 \times 10^5$

Enormous storage reductions as well.

For $b_w = 1$ only need to store $\leq 3n$

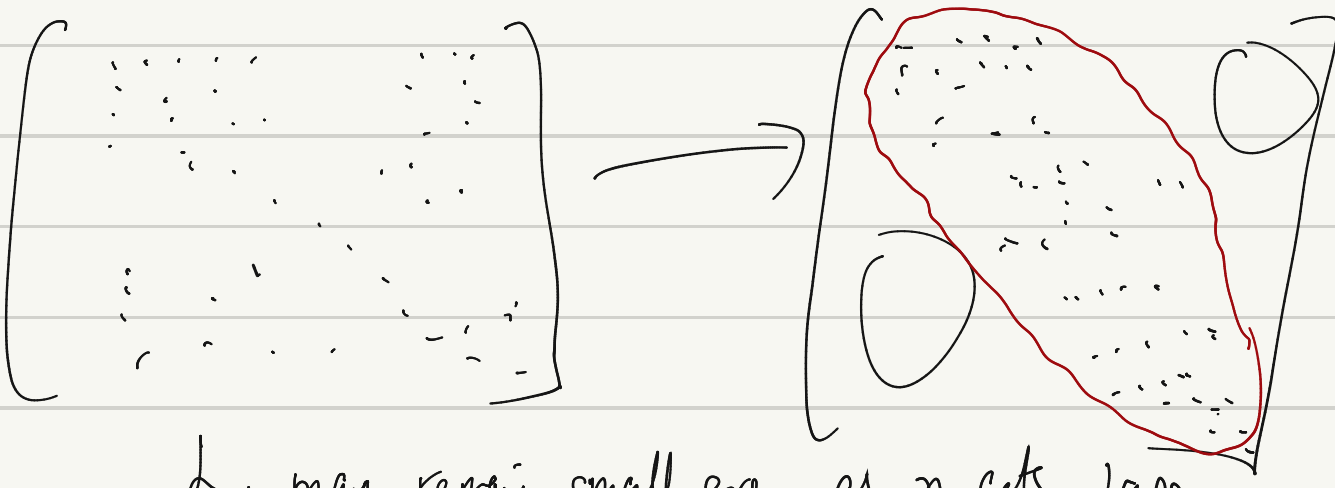
Above was a simple example but sparse and banded matrices are common in many fields e.g. graphical models, spatial models.

E.g. models where dependence is characterized mostly by conditional independence structure

Even if matrix is not banded, it might be sparse. Possible to reduce bandwidth using an algorithm e.g.

Gibbs-Poole-Stockmeyer-King / Cuthill-McKee

Fixed one-time cost



b_w may remain small even as n gets large.

stackoverflow: invert a matrix w/ reduced bandwidth

"Gibbs Poole Stockmeyer alg in R"

Popular structure in statistics: Kronecker product for covariances.

$A_{m\times n}$, $B_{p\times q}$, then Kronecker product $A \otimes B$ is $mp \times nq$ block matrix

$$C = A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}_{mp\times nq}$$

Matrix operations are much faster

$C^{-1} = \underbrace{B^{-1} \otimes A^{-1}}$, if C is invertible ($\Leftrightarrow$ B, A invertible)

dominated by cost of larger of B, A

e.g. $A_{m\times m}$, $B_{q\times q}$, $m > q$, then cost in $B^{-1}\otimes A^{-1}$ is dominated by m^3. Cost of direct C^{-1} is order $(mq)^3$ which may be much larger than m^3

$|C| = |A|^m |B|^n$ if $A_{n\times n}$, $B_{m\times m}$

Computational advantages + useful modeling framework

e.g. A describes one kind of dependence, while

B " another " " ",

$A \otimes B$ combines both

Eigendecomposition

Most popular use in stats is perhaps principal components analysis (PCA)

Let X have covariance matrix Σ

$\therefore \Sigma$ is symmetric, can be decomposed as follows

$$\Sigma = U D U^T$$

where D is diagonal matrix of eigenvalues of Σ and U is corresponding orthogonal matrix of eigenvectors

Suppose eigenvalues are distinct and ordered

$$0 \leq \lambda_1 < \lambda_2 \ldots < \lambda_m$$ then

columns $U_1, \ldots, U_m$ are unique up to sign (cf. K. Lange book)

R.v.s. $V_j = U_j^T X$, $j = 1, \ldots, m$

have covariance matrix $U^T \Sigma U = D$

$V_1, \ldots, V_m$ are principal components

uncorrelated, and increasing in variance.

HW exercises: (1) Refund package dti data set

(2) eigenvalue distr.

(3) Sim study

(4) Kernel rep : use of Sherman-W-M formula