

Introduction Computer Arithmetics

732A90
Computational Statistics

Krzysztof Bartoszek
(krzysztof.bartoszek@liu.se)

22 I 2019 (P42)
Department of Computer and Information Science
Linköping University

- Even simple data analysis (mean, variance) by hand is tedious
- Today: huge datasets, models capturing system complexity, interactions (between variables **and** observations)
- We will discuss:
 - Being careful with calculations—overflow
 - Generation of random variables including correlated ones
 - Numerically optimizing functions, esp. maximum likelihood
 - Computing confidence (credible) intervals for distributions when analytical ones are unobtainable

- Lectures
- Computer Labs
- Seminars
- Examination: Reports, seminars, final exam
- Final exam: computer based
- Answer in English.
- Electronic reports as .PDF.
- Disclose **ALL** collaborations and sources.
- Provide source code (if used).
- E-mail contact: krzysztof.bartoszek@liu.se

Course materials, software

Course contents

Examination

- Lecture slides
- 2016 lecture slides (732A38)
- Handouts, R code
- Various suggested www pages or articles
- Googling
- James E. Gentle “Computational Statistics”, Springer, 2009
- Geof H. Givens, Jennifer A. Hoeting “Computational Statistics”, Wiley, 2013
- R

- Recap: R
- Recap: Basic Statistics
- Computer Arithmetics (JG pages 85–105)
- Optimization (JG pages 241–272, handouts)
- Random Number Generation (JG pages 305–312, 325–328, handouts)
- Monte Carlo Methods (JG pages 312–318, 328 417–429, handouts)
- Numerical Model Selection and Hypothesis Testing (JG pages 52–56, 424, 435–467, handouts)
- Expectation Maximization Algorithm and Stochastic Optimization (JG pages 275–284, 296–298, 480–483, handout)

Pages are recommended reading for each lecture, **NOT** exact lecture content. The lectures will build up on this material.

Computer labs (need to be passed)

Presentation or opposition and attendance at seminars (see 732A90_ComputationalStatisticsVT2019.CourseInformation.pdf).

Computer exam points

A: [18, ∞), B: [16, 18), C: [14, 16), D: [12, 14), E: [10, 12), F: [0, 10)

Allowed aids for exam: printed books and own PDF document containing max 100 pages (see 732A90_ComputationalStatisticsVT2019.CourseInformation.pdf).

Computer Arithmetics

Computer Arithmetics: Examples

Data presentation

Computations can be affected by magnitudes of numbers.

```
x<-0.5^10000;y<-0.4^10000;x/(x+y)+y/(x+y)
x<-0.5^1000;y<-0.4^1000;x/(x|y)|y/(x|y)
x<-0.1^1000;y<-0.2^1000;x/(x+y)+y/(x+y)
```

```
t<-rnorm(5,10^18,1);t[3]-t[4];t[1]-t[2]
```

```
x<-10^800;sd<-10^400;y<-x/sd;y
```

- Computers store information in binary form
 - 0 1 0 1 1 0 0 1
- 1Byte=8bits (typical counting unit)
- 1Word=32 or 64bits (depending on architecture)
- 1KB=1024bytes
- 1MB=1024KB
- and so on

QUESTION: Why binary form?

And your are doing estimation under a nice, fancy model ...

For a detailed mathematical treatment see Ch. 4.2 in
D. E. Knuth (1998). The Art of Computer Programming, Volume 2, Addison-Wesley.

SHOULD YOU CARE?

Character encoding

Fixed-point system (integers)

Arithmetic operations

- ASCII (American Standard Code for Information Interchange)
 - 1 byte per character, 7 bits coding, 1 parity check or 0
 - $2^7 = 128$ characters can be encoded
 - “Usual” English letters, Arabic numerals, punctuation, i.e. “standard” keyboard
 - 1–31 control characters, 0: NULL **WHY?**
 - Design influenced by contemporary (1960) hardware
 - Extended ASCII: all 8 bits, 256 characters
- Unicode
 - 8, 16 or 32 bits encoding
 - “of more than 128,000 characters covering 135 modern and historic scripts, as well as multiple symbol sets” (Wikipedia)
- `read.csv()`, `read.table()` have `fileEncoding` argument

- We use the base-10 (decimal) system, e.g.
 $1234 = 1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$
- We could use base- m system for any m
- Computers: base-2 (binary) system
- Each integer represented as:
 $A = a_0 \cdot 2^0 + a_1 \cdot 2^1 + a_2 \cdot 2^2 + \dots$

EXERCISE: 5, 16, 17, 31, 32, 33, 255, 256 in binary

QUESTION: What is the range of a byte, word, double word?

- Negative numbers
 - **Leading bit:** first bit 0 if positive, 1 if negative
 - **Two's-complement:** sign bit 1, remaining bits to opposite value and then +1
 $e.g. 5 = 00000101, -5 = 11111011$
 - **QUESTION:** $5 + (-5) = ?$, try 12 and -12
 - Range: $[-2^{k-1}, 2^{k-1} - 1]$ on k bits **WHY?**

R operations on binary

- Addition, multiplication: base=2 instead of base=10
- Subtraction: $A - B = A + (-B)$ (*twos-complement*)
- Division: tedious, rounded towards 0 as `integer(17/3)`

- **Overflow:** adding two large numbers the sign bit can be treated as a high order bit and on some architectures results in a negative number

Floating-point system (rational, "real")

- How can we represent fractions (rational numbers)?
- Sign
- Exponent (**signed**, read standards if interested)
- Mantissa or Significand
- on 64bits:

sign	Exponent (1bit)	Mantissa (11bits)
0	0	0.100000000000000055511

$$\pm 0.d_1d_2\dots d_p \cdot b^e \quad b = 2, \quad p = 52$$

• Range: $\approx [-10^{300}, 10^{300}] \approx [-b^{e_{max}}, b^{e_{max}}]$

Floating-point system

- Rationals rounded towards the nearest computer float
- ```
options(digits=22) //max possible
0.1
[1] 0.1000000000000000055511
```

- EXAMPLE: Assume base  $b = 10$  and mantissa has 5 digits  $p = 5$ :  
 $1.2345 = +0.12345 \cdot 10^1$   
 $4.0000567 = +0.40000 \cdot 10^1$

- Problem remains whatever base ( $b$ ) is chosen

- EXERCISE: Try to convert some numbers

## Floating-point system

- Distribution of computer floats



- Dense from  $-1$  to  $1$
- Density decreases
- same number of points for each exponent:

$$\dots, \cdot 10^{-3}, \cdot 10^{-2}, \cdot 10^{-1}, \cdot 10^1, \cdot 10^2, \cdot 10^3, \dots$$

- What about integers?

$$5 = +0.50000 \cdot 10^1$$

```
options(digits=22)
9007199254740992
9007199254740993
9007199254740994
```

## Floating-point system, special "numbers"

- We do not discuss how the exponent is actually coded.
- Usually the maximum allowed number in the exponent is one unit less than possible.
- ±Inf: exponent is exp<sub>max</sub> + 1, mantissa is 0
- NaN: exponent is exp<sub>max</sub> + 1, mantissa is ≠ 0
- 0 WHY?

Overflow: number larger than can be represented

Underflow: loss of significant digits

```
10^200*10^200 = Inf
10^400/10^400 = NaN
10^-200/10^-200 = 0
10^-200*10^-200 =
0*10^400 =
x<-10^300;while(1){x<-x+1}
```

## Arithmetic operations

- Floats are rounded so usual mathematical laws do not hold — floating point arithmetic

### • Examples

```
1/3+1/3 = 0.6666667
options(digits=22)
1/3+1/3 = 0.666666666666666296592
10^(-200)/(10^(-200)+10^(-200)) = 0.5
10^(-200)/(10^(-200)+20^(-200)) = 1
```

- Software is designed to make operations as correct as possible

- Do we need to work with such extreme numbers?

## Arithmetic operations

- $X + Y, X \cdot Y$  can display overflow, underflow
- $A \neq B$  but  $X + A = B + X$
- $A + X = X$  but  $A + Y \neq Y$
- $A \cdot X = X$  but  $X - X / A$
- **COMPARING FLOATS IS TRICKY!**

```
options(digits=22)
x<-sqrt(2)
x*x
[1] 2.0000000000000044089
(x*x)==2
[1] FALSE
isTRUE(all.equal(x*x,2))
[1] TRUE
```

## Summation

## Potential solutions

## More on summing

Underflow problems can occur with any summation ( $x < x+1$ )

```
options(digits=22)
x<-1:1000000: sum(1/x); sum(1/rev(x))
[1] 14.39272672286572252176
[1] 14.39272672286572429812
```

### • WHICH ONE IS CORRECT?

### • WHICH ONE IS MORE ACCURATE?

#### Solution A:

- Sort the numbers ascending **CAN BE EXPENSIVE**
- Sum in this order

#### Solution B:

- Sum numbers pairwise, from  $n$  obtain  $n/2$  numbers  
**HOW TO CHOOSE PAIRS?**
- Continue until 1 number left

#### Example

- Computing exponent using Taylor series

$$e^x = 1 + x + x^2/2 + x^3/6 + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

## The exponential

## More on summing

## Can you explain why?

```
options(digits=22)
fTaylor<-function(x,N){1+sum(sapply(1:N,
 function(i,x){x^i/prod(1:i)},x=x,simplify=TRUE))}
exp(20) #fine
[1] 485165195.4097902774811
fTaylor(20,100)
[1] 485165195.4097902774811
fTaylor(20,100)-exp(20)
[1] 0
exp(-20) //problem
[1] 2.061153622438557869942e-09
fTaylor(-20,100)
[1] -3.853877217352419393137e-10
fTaylor(-20,200)
[1] -3.853877217352419393137e-10
```

#### Example

- Computing exponent using Taylor series

$$e^x = 1 + x + x^2/2 + x^3/6 + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

#### • WHY?

- Varying sign of terms
- CANCELLATION** adding two numbers of almost equal magnitude but of opposite sign
- Effects of cancellations accumulate
- SOLUTION:** Different algorithm ...

## Example due to Thomas Ericsson in his Numerical Analysis course at Chalmers

```
f1<-function(x){(x-1)^6}
f2<-function(x){1-6*x+15*x^2-20*x^3+3+15*x^4-6*x^5+x^6}
x<-seq(from=0.995,to=1.005,by=0.0001)
y1<-f1(x);y2<-f2(x)

plot(x,y1,pch=19,cex=0.5,ylim=c(-5*10^(-15),20
 *10^(-15)),main="Two ways to calculate e^(x-1)^6",xlab="x",ylab="y")
points(x,y2,pch=18,cex=0.8)
```

- Many problems (in Statistics, Numerical methods, e.t.c) can be reduced to solving

$$\mathbf{A}\vec{x} = \vec{b}$$

$\mathbf{A}$  (design) matrix  
 $\vec{x}$  vector of unknowns  
 $\vec{b}$  vector of scalars (data)

- Algorithm should be **numerically stable**

(small changes in  $\mathbf{A}$  or  $\vec{b}$  imply small changes in  $\vec{x}$ )

Minimize

$$RSS(\beta_0, \beta_1, \dots, \beta_m) = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_{i1} - \dots - \beta_m x_{im})^2$$

The system of equations

$$\frac{\partial RSS}{\partial \beta_0} = \frac{\partial RSS}{\partial \beta_1} = \dots = \frac{\partial RSS}{\partial \beta_m} = 0$$

can be written as

$$\mathbf{X}^T \mathbf{X} \vec{\beta} = \mathbf{X} \vec{y}$$

where

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1m} \\ 1 & x_{21} & \dots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{nm} \end{bmatrix}$$

- $\mathbf{A}\vec{x} = \vec{b}$  needs a stable numerical solution
- Computer arithmetics**

- Original system:  $\mathbf{A}\vec{x} = \vec{b}$
- Perturbed system:  $\mathbf{A}\vec{x}' = \vec{b}'$ ,  $\vec{x}' = \vec{x} + \delta\vec{x}$ ,  $\vec{b}' = \vec{b} + \delta\vec{b}$
- Stable: small perturbation in  $\vec{b}$ , small perturbations in  $\vec{x}$
- $\|\vec{b}'\| = \|\mathbf{A}\vec{x}'\| \leq \|\mathbf{A}\|\|\vec{x}'\|$  implies  $\|\vec{x}'\|^{-1} \leq \|\mathbf{A}\|\|\vec{b}'\|^{-1}$
- $\|\delta\vec{x}\| = \|\mathbf{A}^{-1}(\delta\vec{b})\| \leq \|\mathbf{A}^{-1}\|\|\delta\vec{b}\|$

$$\frac{\|\delta\vec{x}\|}{\|\vec{x}\|} \leq \|\mathbf{A}^{-1}\|\|\mathbf{A}\|\frac{\|\delta\vec{b}\|}{\|\vec{b}\|}$$

Condition number of a matrix

$$\kappa(\mathbf{A}) = \|\mathbf{A}^{-1}\| \|\mathbf{A}\|$$

- Large  $\kappa(\mathbf{A})$  is a bad sign, but does not imply ill-conditioning
- $L_2$  norm:  $\kappa(\mathbf{A})$  is the ratio of the maximum and minimum eigenvalues of  $\mathbf{A}$
- Under  $L_2$  norm

$$\kappa(\mathbf{A}^T \mathbf{A}) \geq \kappa(\mathbf{A})^2 \geq \kappa(\mathbf{A})$$

(regression setting)

Dealing with ill-conditioning

- Rescale the variables (columns)
- Use a different algorithm for solving e.g. QR, Cholesky, SVD

**Cholesky:**  $\mathbf{A}$  symmetric-positive-definite  $\mathbf{A}\vec{x} = \vec{b}$  is equivalent to  $\mathbf{L}\mathbf{L}^T \vec{x} = \vec{b}$  WHY?

- Solve  $\mathbf{L}\vec{y} = \vec{b}$
- Solve  $\mathbf{L}^T \vec{x} = \vec{y}$

- Computations can behave “differently” at different numerical ranges.

- Floating point system.

- Computer arithmetics is not the same as “usual” arithmetic.

- Summing series, solving linear systems (inversion?)

## Plan for today

## Optimization

## Optimization: Example

### Optimization

732A90  
Computational Statistics  
Krzysztof Bartoszek  
(krzysztof.bartoszek@liu.se)

24 I 2019 (P42)  
Department of Computer and Information Science  
Linköping University

- Introduction

- Mathematical definition of problem
- 1D optimization
- kD optimization
- R code examples

Nearly everything is optimization !

- Chemistry
- Physics
- Economics, Industry
- Engineering

ANY BIOLOGICAL  
ORGANISM

YOU

## Optimization: Example

## Optimization: Example

## Optimization: Example

### Statistics

#### Industry

#### Economics/Logistics

#### Maximize likelihood, model fitting

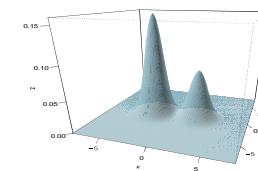
How to produce a cylindrical (WHY?) 0.5L beer can so it requires minimum material?

Given a certain product minimize e.g. material usage, production effort while still meeting consumer requirements.

- Travelling Salesman Problem

- Windmills

- Flight schedule (especially "cheap" airlines)



An i.i.d. sample  $(X_1, \dots, X_n)$  is drawn from a probability distribution  $P(X|\Theta)$ , where  $\Theta$  is an unknown parameter set.

The joint probability of all the observations is

$$P(X_1, \dots, X_n|\Theta) = \prod_{i=1}^n P(X_i|\Theta).$$

Find  $\Theta$  that maximizes  $P(X_1, \dots, X_n|\Theta)$ .

## Mathematical formulation

## Mathematical formulation

## Constraints examples

## Exercise

The goal is to minimize (maximize)

**Objective function:**  $f(\theta)$   
(reproduction, chances of survival, quality of life, cost, profit, likelihood, fit to data)

$$\min_{\theta \in \Theta} f(\theta) \text{ subject to } c_i(\theta) = 0, \quad i \in E \\ c_i(\theta) \geq 0, \quad i \in I$$

depending on

**Parameters or Unknowns  $\theta$**   
(reproduction strategy, resource utilization, consumer choices, height & diameter, production, raw material choice, service times, route, flight routes/times, parameters)

**QUESTION:** What should we do if we are interested in maximization instead of minimization?

**QUESTION:** What should we do if the constraints are  $c_i(x) \leq 0, i \in I$ ?

- Available environment

- Volume: 0.5l of can

**Production:** Factories ( $F_1, F_2$ ), retail outlets ( $R_1, R_2, R_3$ ), cost of shipping  $i \rightarrow j$ :  $c_{ij}$ , production  $a_i$  per week, requirement  $b_j$  per week **to optimize:**  $x_{ij}$  amount shipped  $i \rightarrow j$  per week

$$\begin{aligned} \min_{x \in \mathbb{R}^{3 \times 3}} & \sum_{i,j} c_{ij} x_{ij} && \text{minimize shipping costs} \\ \text{s.t.} & \sum_{j=1}^3 x_{ij} \leq a_i, i = 1, 2 && \text{production capacity} \\ & \sum_{i=1}^3 x_{ij} \geq b_j, j = 1, 2, 3 && \text{demand} \\ & \forall i, j: x_{ij} \geq 0 && \end{aligned}$$

**Question:** What would happen if we drop demand constraint?

**ML:** often no constraints

- Split into pairs/triplets/quadruples
- Think of some human anatomy part/organ:
  - What is its function?
  - What could it have been optimized for over the course of time?
  - Is it still under selection?
  - What constraints was and is it under?
- Think of a situation where optimization is needed in your own student/professional/personal/financial situation.
- State the problem in terms of
  - Objective function
  - Parameters
  - Constraints
  - Does it have a trivial solution?
- 10 minutes

## Optimization approaches

## 1D Optimization

## Golden section (minimization)

## 1D Optimization: Example

- Constrained optimization

- Lagrange multipliers, linear programming
- E.g. LASSO
- Not this lecture!

- Unconstrained optimization

- Steepest descent
- Newton method
- Quasi-Newton-Methods
- Conjugate gradients

- Function of a single parameter, find minimum
- What algorithm would you suggest?

- Golden-section search
- local minimum on  $[A, B]$  interval (constraint)

- Works by narrowing down the search interval with a constant reduction factor

$$1 - \alpha = \frac{\sqrt{5} - 1}{2} \approx 0.62$$

1:  $x_1 = A, x_3 = B,$

2: while  $|x_1 - x_3| > \epsilon$  do

3:    $\alpha = \alpha(x_3 - x_1)$

4:    $x_2 = x_1 + \alpha, x_4 = x_3 - \alpha$

5:   if  $f(x_4) > f(x_2)$  then

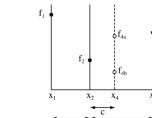
6:      $x_1 = x_2, x_3 = x_4$

7:   else

8:      $x_1 = x_2, x_3 = x_3$

9: end if [We know the value at 3 points!]

10: end while



732A90\_ComputationalStatisticsVT2019\_Lecture02codeSlide16.R

Why are there different methods?

Question: Does  $\alpha$  remind you of something?

$f$  has to be UNIMODAL

## Multi-dimensional optimization

Find

$$\min_{\vec{x} \in \mathbb{R}^n} f(\vec{x})$$

Using (known, or numerically evaluated)

$$\text{Gradient } \nabla f(\vec{x}) = \left( \frac{\partial f(\vec{x})}{\partial x_1}, \dots, \frac{\partial f(\vec{x})}{\partial x_n} \right)^T$$

$$\text{Hessian } \nabla^2 f(\vec{x}) = \left[ \frac{\partial^2 f(\vec{x})}{\partial x_i \partial x_j} \right]_{i,j=1}^n$$

## General strategy

- ❶ Provide a (good) starting point  $\vec{x}_0$ ,  $\vec{x} = \vec{x}_0$
- ❷ Choose a direction  $\vec{p}$  ( $\|\vec{p}\| = 1$ ) and step size  $a$
- ❸ Move to  $\vec{x} := \vec{x} + a\vec{p}$
- ❹ Repeat step 2 until convergence

## How to choose the direction?

### Taylor's theorem

$$f(\vec{x} + a\vec{p}) = f(\vec{x}) + [\alpha \vec{p}^T \cdot \nabla f(\vec{x})] + o(\alpha^2)$$

$\vec{p}$  s.t.  $\vec{p}^T \cdot \nabla f(\vec{x}) < 0$  is a *descent* direction.

### Steepest descent is

$$\vec{p} = -(\nabla f(\vec{x})) / \|\nabla f(\vec{x})\|$$

## How to choose the step size?

- ❶ **Expensive way:** find the global minimum in direction  $\vec{p}$
- ❷ **Trade-off way:** find a decrease which is *sufficient*

## BACKTRACKING

- 1: Choose (large)  $\alpha_0 > 0$ ,  $\rho \in (0, 1)$ ,  $c \in (0, 1)$ ,
- 2:  $\alpha = \alpha_0$
- 3: **repeat**
- 4:    $\alpha = \rho\alpha$
- 5:   **until**  $f(\vec{x} + a\vec{p}) \leq f(\vec{x}) + c\alpha \vec{p}^T \nabla f(\vec{x})$

## Newton's method

- ❶ Newton–Raphson method
- ❷ Hessian ignored in steepest descent
- ❸ If  $f$  is quadratic

$$f(p) = \frac{1}{2} \vec{p}^T \mathbf{A} \vec{p} + \vec{b}^T \vec{p} + c,$$

then minimum

$$\vec{p}^* = \mathbf{A}^{-1} \vec{b}.$$

- ❹ Taylor expansion of  $f$

$$f(\vec{x} + a\vec{p}) = f(\vec{x}) - \alpha \vec{p}^T \cdot \nabla f(\vec{x}) + \frac{\alpha^2}{2} \vec{p}^T \nabla^2 f(\vec{x}) \vec{p} + o(\alpha^3)$$

- ❺  $x := x + a\vec{p}$  where

$$\vec{p} = -(\nabla^2 f(\vec{x}))^{-1} \nabla f(\vec{x})$$

## Newton's method

- ❶  $(\nabla^2 f(\vec{x}))^{-1}$  is expensive to compute, there are quicker approaches, e.g. Cholesky decomposition
- ❷ Hessian should be **positive definite** for  $\vec{p}$  to be a descent direction (if not see book)
- ❸ Memory expensive — need to store  $O(n^2)$  elements

### BUT

- ❹ Method converges quickly esp. near optimum

## Quasi-Newton methods

- ❶  $k$  iteration number
  - ❷ Compute an approximation to the Hessian,  $\mathbf{B}$ , that will allow for efficient choice of  $\vec{p}$ .
  - ❸ **SECANT CONDITION:** (quasi-Newton condition)
- $$\mathbf{B}_{k+1} (\vec{x}_{k-1} - \vec{x}_k) = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$$
- BFGS Algorithm**
- 1: Choose  $\mathbf{B}_0 > 0$ ,  $\vec{x}_0$ ,  $k = 0$
  - 2: **repeat**
  - 3:    $\vec{p}_k$  is solution of  $\mathbf{B}_k \vec{p}_k = \nabla f(\vec{x}_k)$
  - 4:   find suitable  $\alpha_k$
  - 5:    $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k$
  - 6:   calculate  $\mathbf{B}_{k+1}$  (next slice)
  - 7:    $k = k + 1$
  - 8: **until** convergence of  $\vec{x}_k$  at minimum

- ❶ We want  $\mathbf{B}_{k+1}$  and  $\mathbf{B}_k$  to be close to each other

$$\begin{aligned} & \min_{\mathbf{B}} \|\mathbf{B} - \mathbf{B}_k\| \\ & \text{s.t. } \mathbf{B} = \mathbf{B}^T, \text{ secant condition} \end{aligned}$$

- ❷  $\vec{y}_k = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$
- ❸  $\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \vec{y}_k \vec{y}_k^T \mathbf{B}_k}{\vec{y}_k^T \mathbf{B}_k \vec{y}_k} + \frac{\vec{x}_k \vec{x}_k^T}{\vec{y}_k^T \mathbf{B}_k \vec{y}_k}$
- ❹ Closed form Sherman–Morrison formula for  $\mathbf{B}_{k+1}^{-1}$
- ❺ We have to store  $\mathbf{B}_k^{-1}$

## BFGS

- ❶ BFGS: Broyden–Fletcher–Goldfarb–Shanno
- ❷ More iterations than Newton's method (uses approximation)
- ❸ Each iteration quicker, no numeric inversion
- ❹ Good for large scale problems
- ❺ Choice of  $\mathbf{B}_0$ ?

## Conjugate Gradient method—quadratic case

- Minimize
- $$f(\vec{x}) = \frac{1}{2} \vec{x}^T \mathbf{A} \vec{x} + \vec{b}^T \vec{x}$$
- for  $\mathbf{A}$  symmetric positive definite.

Gradient:

$$\nabla f(\vec{x}) = \mathbf{A} \vec{x} + \vec{b} = r(\vec{x})$$

Two vectors  $\vec{p}$  and  $\vec{q}$  are **conjugate** with respect to  $\mathbf{A}$  if

$$\vec{p}^T \mathbf{A} \vec{q} = 0.$$

*IDEA:*  $\vec{p}$  and  $\vec{q}$  are orthogonal w.r.t. to an inner product associated with  $\mathbf{A}$ . Use this to find a basis that will allow for easy finding of  $\vec{x}$ .

## Conjugate Gradient method

- ❶  $\vec{p}_0 = \vec{r}_0$
- ❷  $\vec{p}_{k+1} = -\vec{r}_k + \beta_{k+1} \vec{p}_k$

**Exercise: check this**

- ❸ Convergence in  $\dim(\mathbf{A})$  steps (or unless cutoff for  $\vec{r}_k$ )

- ❶ If  $f(\cdot)$  general, use  $\triangleright f(\cdot)$  instead of  $r(\cdot)$

- 1: Choose  $\vec{x}_0$ ,  $\vec{p}_0 = -\nabla f(\vec{x}_0)$ ,  $k = 0$
- 2: **while**  $\nabla f(\vec{x}_k) \neq 0$  **do**
- 3:   find suitable  $\alpha_k$
- 4:    $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k$  {and now update step}
- 5:    $\vec{p}_{k+1} = (\nabla^T f(\vec{x}_{k+1}) \nabla f(\vec{x}_{k+1})) / (\nabla^T f(\vec{x}_k) \nabla f(\vec{x}_k))$  {Fletcher–Reeves update, other possible}
- 6:    $\vec{p}_{k+1} = -\nabla f(\vec{x}_{k+1}) + \beta_{k+1} \vec{p}_k$
- 7:    $k = k + 1$
- 8: **end while**

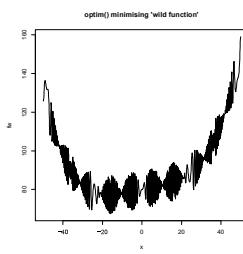
## Nonlinear CG method

## kD Optimization: Example

## kD Optimization: Example

## Summary

- ❶ Local minimum convergence
- ❷ But this is true of all methods that cannot “jump out” of descent path
- ❸ Faster than steepest descent
- ❹ Slower than Newton and Quasi-Newton but significantly less memory



732A90\_ComputationalStatisticsVT2019\_Lecture02codeSlide31.R

- ❶ Optimization is everywhere
- ❷ Numerical methods for finding minimum
- ❸ 1D: Golden section (unimodal), `optimize()`
- ❹ kD: choose step size and direction (gradient), `optim()`

**Random Number Generation**

732A90  
Computational Statistics  
  
Krzysztof Bartoszek  
(krzysztof.bartoszek@liu.se)  
  
30 I 2019 (P42)  
Department of Computer and Information Science  
Linköping University

**Pseudorandom numbers**

- A computer is a deterministic machine
- Congruential generators
- Functions of time
- Be careful with respect to application

**First step: Generating  $\text{Unif}[0, 1]$**

Linear congruential generator

Define a sequence of integers according to

$$x_{k+1} = (a \cdot x_k + c) \mod m, \quad k \geq 0$$

$x_0$  is seed, e.g. based on time

$\mod m$ : remainder after division by  $m$

- $x_k \in \{0, \dots, m-1\}$  and integer
- $x_k/m \sim \text{Unif}[0, 1]$
- $a, c \in [0, m)$  need to be carefully selected

**Generated numbers will get into a loop with a certain period**

$$x_{k+1} = (a \cdot x_k + c) \mod m, \quad k \geq 0$$

$x_0 = a = c = 7, m = 10$

- $x_1 = (7 \cdot 7 + 7) \mod 10 = 56 \mod 10 = 6$
- $x_2 = (7 \cdot 6 + 7) \mod 10 = 49 \mod 10 = 9$
- $x_3 = (7 \cdot 9 + 7) \mod 10 = 70 \mod 10 = 0$
- $x_4 = (7 \cdot 0 + 7) \mod 10 = 7 \mod 10 = 7$
- $x_5 = (7 \cdot 7 + 7) \mod 10 = 56 \mod 10 = 6$
- ...

**First step: Generating  $\text{Unif}[0, 1]$**

See also D. E. Knuth (1998). The Art of Computer Programming, Volume 2, Addison-Wesley, Ch. 3.3.4

```
fthreebits<-function(k,s,L,N){
X0<-s+1;X<-8*x0+5;m<-2^L;X<-X0
for (i in 1:N){
 print(c(X,rev(intToBits(X)[1:5])))
 X<- (a*X)%Ź m #a=0
}
}

source("crogen.R");rthreebits(k,s+1,1-a,N,10)
{1} 13 0 1 1 0 1
{1} 13 0 1 1 0 1
{1} 181 0 0 1 0 1
{1} 181 0 0 1 0 1
{1} 233 1 1 1 0 1
{1} 193 0 0 0 0 1
{1} 193 0 0 0 0 1
{1} 131 1 1 0 0 1
{1} 131 1 1 0 0 1
{1} 131 1 1 0 0 1
```

Last three bits change between 001 and 101  
Discard less significant bits

**First step: Generating  $\text{Unif}[0, 1]$**

**First step: Generating  $\text{Unif}[0, 1]$**

- Period is  $\leq m$  by definition
- $a, c, m$  (large) have to be chosen carefully
  - $a$  and  $m$  have to be relatively prime (no common divisors other than 1)
  - $a \equiv 1 \pmod p$  for every prime divisor  $p$  of  $m$
  - $a \equiv 1 \pmod 4$  if 4 divides  $m$
  - Then full period  $m$  reached (what about  $a = c = 1$ ?)
- Seed defines the random sequence — same seed, same sequence
- Be careful when re-opening an R workspace
- Other methods (not in this course)

**Second step: Generating  $\text{Unif}[a, b]$**

- $U \sim \text{Unif}[0, 1]$  can be transformed into  $X \sim \text{Unif}[a, b]$  as
$$X = a + U \cdot (b - a)$$

**Second step: Generating  $\text{Unif}[a, b]$**

- $U$  can also be transformed into discrete uniform distribution on integers  $\in \{1, \dots, n\}$  as  $\lfloor \cdot \rfloor$ , integer part
$$X = \lfloor nU \rfloor + 1$$

**Questions**

- Why +1?
- How can  $U$  be transformed into  $Y$ , where  $Y$  is discrete uniform on integers  $\{50, 55, 60\}$ ?

**Second step: Generating nonuniform random numbers**

- $U \sim \text{Unif}(0, 1)$
- Let  $F_Y$  be the cumulative distribution function (CDF) of  $Y$

$$F_Y(u) = P(Y \leq u) = \begin{cases} 0 & u < 0 \\ u & 0 < u \leq 1 \\ 1 & 1 < u \end{cases}$$

- The probability distribution function (PDF) of  $Y$

**Inverse CDF method**

Let  $X$  be a random variable with CDF  $X \sim F_X$  ( $F_X$  strictly increasing)

Consider  $Y = F_X^{-1}(U)$ , where  $U \sim \text{Unif}(0, 1)$

$$\begin{aligned} F_Y(y) &= P(Y \leq y) = P(F_X^{-1}(U) \leq y) \\ &= P(F_X(F_X^{-1}(U)) \leq F_X(y)) \\ &= P(U \leq F_X(y)) = F_U(F_X(y)) = F_X(y) \end{aligned}$$

$Y$  has same probability distribution as  $X$

**Inverse CDF method**

If we can generate  $U \sim \text{Unif}(0, 1)$ , then we can generate  $X \sim F_X$  as

$$X = F_X^{-1}(U)$$

Provided we can calculate  $F_X^{-1}$ ...

**Inverse CDF method: Example**

Find  $F_X^{-1}$

$$\begin{aligned} y &= 1 - e^{-\lambda x} \\ e^{-\lambda x} &= 1 - y \\ x &= -\frac{1}{\lambda} \ln(1 - y) \\ F_X^{-1}(y) &= -\frac{1}{\lambda} \ln(1 - y) \end{aligned}$$

Hence, if  $U \sim \text{Unif}(0, 1)$ , then

$$-\frac{1}{\lambda} \ln(1 - U) = X \sim \exp(\lambda)$$

**Inverse CDF method**

- When  $F_X^{-1}$  can be derived: **EASY**
- When NOT: numerical solution time-consuming
- numerical errors ?

Situation 2 is common ... e.g.  $\mathcal{N}(0, 1)$

**Generating discrete RVs**

- Define distribution  $P(X = x_i) = p_i$
- Generate  $U \sim \text{Unif}(0, 1)$
- If  $U \leq p_0$ , set  $X = x_0$
- Else if  $U \leq p_0 + p_1$ , set  $X = x_1$
- ...

**Generating  $\mathcal{N}(0, 1)$**

Assume

- $\theta \in \text{Unif}(0, 2\pi)$
- $D \sim \text{Unif}(0, 1)$

1: Generate  $\theta, D$   
2: Generate  $X_1$  and  $X_2$  as

$$\begin{aligned} X_1 &= \sqrt{-2 \ln D} \cos \theta \\ X_2 &= \sqrt{-2 \ln D} \sin \theta \end{aligned}$$

$X_1$  and  $X_2$  are independent and normally distributed

But finding such transformations is not easy

## Acceptance/rejection methods

- IDEA: generate  $Y \sim f_Y$  similar to some known PDF  $f_X$
  - IDEA:  $f_Y$  is easy to generate from
  - REQUIREMENT: there exists a constant  $c$
- $\forall_x c f_Y(x) \geq f_X(x)$
- $f_Y$ : majorizing density, proposal density
  - $f_X$ : target density
  - $c$ : majorizing constant

## Generating multivariate normal

Generate  $\mathcal{N}(\vec{\mu}, \Sigma) \in \mathbb{R}^n$ 

- Generate  $n$  i.i.d.  $\mathcal{N}(0,1)$  r.v.s.  $\vec{X} = (X_1, \dots, X_n)$   
(We know how to do this, see slide 16)
- Compute Cholesky decomposition (a.k.a. matrix square root) of  $\Sigma$ , i.e. find  $\mathbf{A}$ , lower triangular s.t.  $\mathbf{A}\mathbf{A}^T = \Sigma$ ,  
(in R: `chol()`)
- $\vec{Y} = \vec{\mu} + \mathbf{A}\vec{X}$

## QUESTION:

what is the expectation and variance-covariance of  $\vec{Y}$ ?

## Acceptance/rejection methods

## Acceptance/rejection methods

## Acceptance/rejection methods: Example

## Acceptance/rejection methods:

```

1: while X not generated do
2: Generate $Y \sim f_Y$
3: Generate $U \sim \text{Unif}(0,1)$
4: if $U < f_X(Y)/(c f_Y(Y))$ then
5: $X = Y$
6: Set X is generated
7: end if
8: end while

```

•  $X \sim f_X$  CHECK THIS

- Larger  $c$ : larger rejection rates— $c$  as small as possible  
number of draws  $\sim$  Geometric( $1/c$ ) mean:  $c$
- Can work in higher dimensions—but high rejection rate

```

Generate beta(2,7)
y<-dbeta(seq(0.2,0.0001),2,7)
<--max(y)*c
[1] 3.172554

1: while X not generated do
2: Generate $Y \sim \text{Unif}(0,1)$
3: Generate $U \sim \text{Unif}(0,1)$
4: if $U \leq dbeta(Y, 2, 7)/(c - 1)$ then
5: $X = Y$
6: Set X is generated
7: end if
8: end while

```

QUESTION:  
Compare acceptance and rejection regions (of  $Y$ ) for different  $c$ .

- Acceptance/rejection is difficult to apply
- Difficult to find majorizing density
  - can always take  $\sup(f_X) \cdot \text{Unif}(0,1)$
  - but what is the problem?

## Random numbers in R

## Summary

- ➊ `ddistribution name()`: density of distribution
- ➋ `pdistribution name()`: CDF of distribution
- ➌ `qdistribution name()`: quantiles of distribution
- ➍ `rdistribution name()`: simulate from distribution

- Computers generate pseudo-random numbers
- We draw from pseudo-uniform and transform to desired distribution
- Analytical methods for transforming exist but are distribution specific

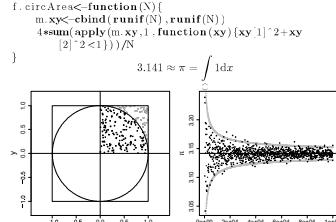
## What is the area of the unit circle?

### Monte Carlo Methods

732A90  
Computational Statistics

Krzysztof Bartoszek  
(krzysztof.bartoszek@liu.se)

5 II 2019 (P42)  
Department of Computer and Information Science  
Linköping University



## Monte Carlo methods: outline

- Monte Carlo methods are a class of computational algorithms that use repeated random sampling to compute their results.

- Monte Carlo methods for random number generation
  - Metropolis-Hastings algorithm
  - Gibbs sampler

- Monte Carlo methods for statistical inference
  - Estimate integrals (we already did!)
  - Variance estimation
  - Variance reduction: importance sampling, control variates

## Markov Chain Monte Carlo

- Previous lecture: Generate
  - univariate distributions (inverse CDF, acceptance/rejection)
  - multivariate normal

but general multivariate distribution?

## MCMC

## Bayesian inference: Recap

A dataset  $D$  is obtained by sampling from a distribution  $f(\cdot | \theta)$ .  
How to estimate  $\theta$ ?

- Frequentists:**  $\theta$  is an unknown but fixed parameter, compute likelihood  $L(D|\theta)$  and find  $\theta$  that maximizes it.

- Bayesians:**  $\theta$  is a random variable with **prior** probability law  $p(\theta)$  before observing  $D$

- After observing  $D$ , Bayes' theorem gives

$$p(\theta | D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)d\theta}$$

## Bayesian inference: Recap

$$\boxed{p(\theta | D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)d\theta}}$$

We know:  $p(D|\theta)$  (the model),  $p(\theta)$  (the prior)  
We need: simulate from  $p(\theta | D)$  (the posterior)

- General (multivariate) type distribution
- Integral can be impossible to compute

- MCMC solves this
- Not needed (given  $D$  it is constant)

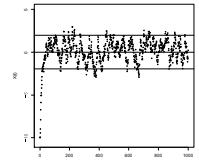
## Markov Chains: Recap

- A Markov chain is a sequence  $X_0, X_1, \dots$  of random variables such that the distribution of the next value depends only on the current one (and parameters).

$P(X_{t+1}|X_t)$  is called a **transition kernel**. Assume it does not depend on  $t$  (**time homogeneous**).

- A Markov chain is **stationary**, with stationary distribution  $\Phi$ , if  $\forall_k X_k \sim \Phi$
- One shows (not trivial in general) that under *certain* conditions a Markov chain will converge to the stationary distribution in the limit.

$$X(t+1) = e^{-1}X(t) + \epsilon, \epsilon \sim \mathcal{N}(0, \frac{1}{2} \cdot (1 - e^{-2}))$$



Discard first  $K - 1$  samples: **burn-in period**

## MCMC: Example

Linear regression with residual normally/student/etc. distributed

$$Y = \beta X + \epsilon$$

How to find credible interval for  $\beta$  if we know  $\text{Var}[\epsilon] = \sigma^2$ ?

$$P(Y|X, \beta) = \prod_{i=1}^N f(Y_i | \text{mean} = \beta X_i, \text{var} = \sigma^2)$$

- Obtain  $P(\beta|Y, X)$  by drawing from  $P(Y|X, \beta)P(\beta)$  in a clever way.

- The prior?

- Use the MCMC sample to obtain quantiles.

Normal residual: analytical solution

## Metropolis–Hastings algorithm

We have

- A PDF  $\pi(x)$  that we want to sample from.
- A proposal distribution  $q(\cdot | X_t)$  that has a regular form w.r.t. to  $\pi(\cdot)$   
E.g.  $q(\cdot | X_t)$  is normal with mean  $X_t$  and given variance
- Regular form: suffices that the proposal has the same support as  $\pi$ .

## Metropolis–Hastings Sampler

$$\alpha(X_t, Y) = \min \left\{ 1, \frac{\pi(Y)q(X_t | Y)}{\pi(X_t)q(Y | X_t)} \right\}$$

```

1: Initialize chain to X_0 , $t = 0$
2: while $t < t_{\max}$ do
3: Generate a candidate point $Y \sim q(\cdot | X_t)$
4: Generate $U \sim U_{[0,1]}$
5: if $U < \alpha(X_t, Y)$ then
6: $X_{t+1} = Y$
7: else
8: $X_{t+1} = X_t$
9: end if
10: $t = t + 1$
11: end while

```

- Informally: "The chain  $(X_t)_{t=0}^\infty$  will converge to  $\pi(\cdot)$ ."

- The chain might not move sometimes.

- The values of the chain are dependent.

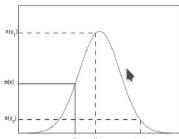
- If  $q(X_t | Y) = q(Y | X_t)$  (i.e. symmetric proposal) we get **Random-walk Metropolis Carlo**:

$$\alpha(X_t, Y) = \min \left\{ 1, \frac{\pi(Y)}{\pi(X_t)} \right\}$$

## Choice of proposal distribution

- In Random-walk Monte Carlo

If  $\pi(Y) \geq \pi(X)$ , the chain moves to the next point, otherwise only with some probability.

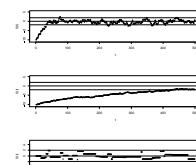


## Choice of proposal dist.: target: $\pi(\cdot) = \mathcal{N}(0, 1)$

732A90\_ComputationalStatisticsVT2019\_Lecture04codeSlide14.R

## Choice of proposal distribution

$q$  normal with sd: props= 0.5, 0.1 and 20



We want to generate from a distribution on  $\mathbb{R}^d$ .

```

1: Initialize chain to $X_0 = (X_{0,1}, \dots, X_{0,d})$, $t = 0$
2: while $t < t_{\max}$ do
3: for $i = 1, \dots, d$ do
4: Generate
 $X_{t+1,i} \sim f(\cdot | X_{t+1,1}, \dots, X_{t+1,i-1}, X_{t+1,i+1}, \dots, X_{t+1,d})$
5: end for
6: $t = t + 1$
7: end while

```

## Gibbs sampler

## Gibbs sampler: target: $d$ -dim $\mathcal{N}(\mu, \Sigma)$

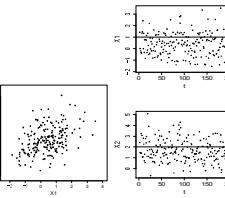
- At each iteration inside the `for` loop univariate random numbers are generated.
- Only one element is updated.
- WE NEED TO KNOW THE CONDITIONAL MARGINAL DISTRIBUTIONS.**
- Convergence may be slow.
- Can be useful in high dimensions (i.e. proposal density may be difficult to find in another way).

732A90\_ComputationalStatisticsVT2019\_Lecture04codeSlide18.R

## Gibbs sampler: Example (code: see R scripts)

Generate from

$$\mathcal{N}([1 \ 2]^T, \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix})$$



- When should we stop the chain? When are we (nearly) at the stationary distribution?

- Typically such a sample is generated to make further inference.

## Convergence monitoring: Gelman–Rubin method

We want to estimate  $v(\theta)$ .

- Generate  $k$  sequences of length  $n$  with different starting points.
  - Compute between- and within- sequence variances:
- $$B = \frac{n}{k-1} \sum_{i=1}^k (\bar{\pi}_i - \bar{\pi}_.)^2 \quad W = \sum_{i=1}^k \frac{s_i^2}{n} \quad s_i^2 = \sum_{j=1}^n (\bar{\pi}_{ij} - \bar{\pi}_i)^2$$

Overall variance estimate:  $\text{Var}[v] = \frac{n-1}{n}W + \frac{1}{n}B$

Gelman–Rubin factor:

$$\sqrt{R} = \sqrt{\frac{\text{Var}[v]}{W}}$$

Values much larger than 1 indicate lack of convergence

See `?coda::gelman.diag`

## Gibbs sampler

```
library(coda)
f1<-mcmc.list(); f2<-mcmc.list(); n<-100; k<-20
X<-matrix(rnorm(n*k), ncol=k, nrow=n)
X<-X+(apply(X, 2, cumsum)*(matrix(rep(1:n, k), ncol=k)^2))
for (i in 1:k) {f1[[i]]<-as.mcmc(X[, i]); f2[[i]]<-as.mcmc(X[, i])}
print(gelman.diag(f1))
Potential scale reduction factors:
Point est., Upper C.I.
#f1.i 1.82 2.38
print(gelman.diag(f2))
Potential scale reduction factors:
Point est., Upper C.I.
#f2.i 1.01 1.01
```

## MC for inference

Estimation of a definite integral

$$\theta = \int_D f(x) dx \quad \left( \text{recall } \pi = \int \mathbb{1} dx \right)$$

Decompose into:

$$f(x) = g(x)p(x) \quad \text{where } \int_D p(x)dx = 1$$

Then, if  $X \sim p(\cdot)$

$$\theta = \mathbb{E}[g(X)] = \int_D g(x)p(x)dx$$

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^n g(x_i), \quad \forall x_i \sim p(\cdot)$$

- Decomposition is not unique, some will be better (lower variance) others worse.  $p(x) \propto |f(x)|$ : minimal
- Can we easily generate from  $p(\cdot)$ ?
- Bayesian inference: use MCMC samples from  $p(\theta|D)$  to obtain a point estimator

$$\theta^* = \int \theta p(\theta|D) d\theta \approx \frac{1}{n} \sum_{i=1}^n \theta_i$$

- $\hat{\theta}$  depends on  $n$  and  $g(X)$ , how variable will it be?
- MCMC: estimator biases as chain correlated, use longer chain and batch mean instead of  $x_i$ .

## Summary

- Generating data from a general multivariate distribution
- Markov Chain Monte Carlo:  
Metropolis–Hastings algorithm, Gibbs sampling
- Convergence: Gelman–Rubin method
- Estimation of integral

## Model selection

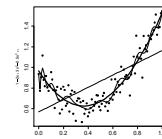
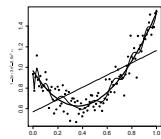
## Model selection

## Hypothesis testing: Recap

### Model Selection and Hypothesis Testing

732A90  
Computational Statistics  
Krzysztof Bartoszek  
(krzysztof.bartoszek@liu.se)

II 2019 ()  
Department of Computer and Information Science  
Linköping University



Tools for model selection

- Comparing different models
- Information criteria (not this course)
- Cross-validation
- Hypothesis testing
- Uncertainty estimation
- Confidence intervals

- Assume a probabilistic model  
State a null hypothesis ( $H_0$  e.g. no difference) and alternative ( $H_1$  difference)
- Observe data  $X$
- Calculate a test statistic e.g.  $T(X) = (\bar{X})/(sd(\bar{X}))$   
(different statistics will have different efficiency (power, ability to distinguish between hypotheses) associated with them)
- Under  $H_0$   $T(X)$  has "known" distribution
- Decision: Is the value of  $T(X)$  surprising (in the critical region)? If so reject  $H_0$  in favour of  $H_1$ .

### Hypothesis testing: Example

```
x<-rnorm(10,mean=4,sd=1)
```

Hypotheses:  
 $H_0 : \mu = 4, X \sim N(\mu, \sigma^2)$   
 $H_1 : \mu \neq 4, X \sim N(\mu, \sigma^2)$

```
x<-rnorm(10,mean=4,sd=1)
```

Hypotheses:  
 $H_0 : \mu = 4, X \sim N(\mu, \sigma^2)$   
 $H_1 : \mu \neq 4, X \sim N(\mu, \sigma^2)$

Test statistic

$$T(x) = \frac{\bar{x} - \mu}{s/\sqrt{n}} \sim t(n-1)$$

```
t<-c(-(mean(x)-4)/(sqrt(var(x)/length(x))),
t<-qt(0.975,df=length(x)-1)
(tx>0) | | (tx< (-t0)) ## reject if TRUE
```

### Hypothesis testing: Example

```
x<-rnorm(10,mean=4,sd=1)
```

Hypotheses:  
 $H_0 : \mu = 4, X \sim N(\mu, \sigma^2)$   
 $H_1 : \mu \neq 4, X \sim N(\mu, \sigma^2)$

Test statistic

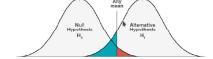
$$T(x) = \frac{\bar{x} - \mu}{s/\sqrt{n}} \sim t(n-1)$$

```
t<-c(-(mean(x)-4)/(sqrt(var(x)/length(x))),
t<-qt(0.975,df=length(x)-1)
(tx>0) | | (tx< (-t0)) ## reject if TRUE
```

### Hypothesis testing: Power

How does one compares different statistics?  
**POWER**  
Power = 1 - Type II error

Ability to correctly identify surprise,  
i.e. indicate  $H_1$ .



How to compute power?

- Analytically (?)
- Generate data samples that satisfy  $H_1$   
Compute percent of correct rejections

### Monte Carlo Hypothesis testing

We may use "any" test statistic.  
We do not need to know its distribution.

$H_0 : \mu = 4, X \sim N(\mu, \sigma^2)$   
 $H_1 : \mu \neq 4, X \sim N(\mu, \sigma^2)$

We may use "any" test statistic.  
We do not need to know its distribution.

$H_0 : \mu = 4, X \sim N(\mu, \sigma^2)$   
 $H_1 : \mu \neq 4, X \sim N(\mu, \sigma^2)$

Test statistic

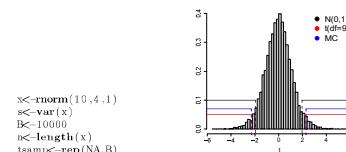
$$T(x) = \frac{\bar{x} - \mu}{s/\sqrt{n}} \sim t(n-1)$$

```
1: for i = 1 to B do
2: Generate Y_1, \dots, Y_n i.i.d. from H_0 , i.e. $\mathcal{N}(4, \sigma^2)$
3: Compute t_i from Y_1, \dots, Y_n
4: end for
```

5: Use  $t_1, \dots, t_B$  to construct a histogram

6: Use the histogram as the distribution of  $T(x)$  under  $H_0$

### Monte Carlo Hypothesis testing

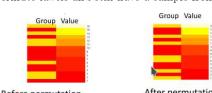


### Permutation tests

- A, k, a. randomization tests
- One solution if we do not know the distribution under  $H_0$
- Computationally expensive
- Any sample size
- Two sample problem:
  - Population 1 distributed as  $F'$
  - Population 2 distributed as  $G$
  - $H_0: F' = G$
  - $H_1: F' \neq G$

### Permutation tests: mouse data

**IDEA:** If  $F = G$  then group label does not matter  
We may permute labels and still have a sample from  $F$  (or  $G$ )



Do the values differ significantly between control and treatment groups?

Test statistic:

$$T(X) = \text{mean}(\text{values}|\text{group} = z) - \text{mean}(\text{values}|\text{group} = y)$$

### Permutation test: scheme

1:  $T(X)$  value of statistic from observed data

2: Create permutations  $g_1^*, \dots, g_B^*$  of group variable  
{If the number of permutations is too large, sample  $B$  randomly without replacement. E.g. generate random permutations and keep only unique ones.}

3: Evaluate test statistic on each permutation

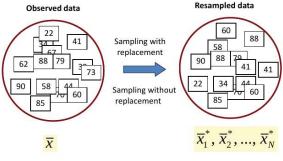
4: Estimate p-value:  $\hat{p} = \#\{T(X_{g_i^*}) \geq T(X)\}/B$   
5: If test is two-sided:  $\hat{p} = \#\{|T(X_{g_i^*})| \geq |T(X)|\}/B$

### Permutation tests

Do we reject the null?

```
B=1000
stat=numeric(B)
stat.dim(mouse)[1]
for(b in 1:B){
 Gbsample(mouse$Group, n)
 stat[b]=mean(mouse$Value[Gbsample=="z"])-mean(mouse$Value[Gbsample=="y"])}
stat=mean(mouse$Value[mouse$Group=="z"])-mean(mouse$Value[mouse$Group=="y"])
print(c(stat, mean(stat>stat0)))
/I/ 30.63492 0.12700
```

## Resampling methods



## Bootstrap in R

```
library("boot")
stat1<-function(data,vn){
 data<-as.data.frame(data[,vn])
 res<-lm(Response~Predictor,data)
 res$coefficients[2]
}
x<-rnorm(100);data<-cbind(Predictor=x,Response=3+2*x
)
res<-boot(data,stat1,R=1000)
print(hoot.ci(res))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates
#Intervals :
#Level Normal Basic
#95% (1.932, -2.164) (1.935, -2.162)
#Level Percentile BCa
#95% (1.934, -2.161) (1.936, -2.166)
```

## Summary

- Why are some models better than others?
- Hypothesis testing
- Monte Carlo hypothesis testing
- Resampling methods (permutations, jackknife, bootstrap)
- Simulation methods (parametric bootstrap)

| Resampling methods                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Jackknife and bootstrap                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Uncertainty estimation: confidence intervals                                                                                                                                                                                                                                                                                                                      | Uncertainty estimation: variance of estimator |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| <p>Theory different, coding similar<br/>Data (i.i.d.) <math>X \sim F(\cdot, w)</math></p> <p>1: Observed data: <math>D = (X_1, \dots, X_n)</math>, estimator <math>\hat{w} = T(D)</math><br/>2: <b>for</b> <math>i = 1, \dots, B</math> { Jackknife <math>B \leq n</math>} <b>do</b><br/>3:   Generate<br/>    <math>D_i^* = (X_1^*, \dots, X_n^*)</math> by sampling with replacement<br/>    {Nonparametric Bootstrap, <math>F</math> unknown}<br/>4:   <math>D_i^* = X[-i]</math> {Jackknife, <math>F</math> unknown}<br/>5:   <math>D_i^* = (X_1^*, \dots, X_n^*)</math> by generating from <math>F(\cdot, \hat{w})</math><br/>    {Parametric Bootstrap, <math>F</math> known}<br/>6: <b>end for</b><br/>7: Distribution of <math>\hat{w}</math> is estimated by <math>T(D_1^*), \dots, T(D_B^*)</math><br/>    {The histogram based on resampled values is used in place of the true density}</p> | <p>Estimate <math>100(1 - \alpha)\%</math> percentile confidence interval for <math>w</math><br/><math>se(\cdot)</math> is the square root of estimated variance (computationally heavy)<br/><b>NOT</b> by jackknife <b>TOO DEPENDENT!!</b></p> <p>1: Compute <math>T(D_1^*), \dots, T(D_B^*)</math><br/>2: Sort in ascending order, obtaining <math>y_1, \dots, y_B</math><br/>    {percentile method} OR<br/>        Compute <math>y_i = (T(D_i^*) - T(D)) / se(T(D_i^*))</math> <math>i = 1, \dots, B</math><br/>    {t method}<br/>3: Define <math>A_1 = \lceil (Ba/2) \rceil</math>, <math>A_2 = \lfloor (B - Ba/2) \rfloor</math><br/>4: Confidence interval is given by<br/>    <math>(y_{A_1}, y_{A_2})</math> {percentile method} OR<br/>    <math>(T(D_{A_1}^*) - se(T(D^*)) \cdot y_{A_1}, T(D) + se(T(D^*)) \cdot y_{A_2})</math><br/>    {t method}</p> <p>Hypothesis testing: does statistic from observed data fall into CI (<math>H_0</math>) or not (<math>H_1</math>)</p> | <p><b>Bootstrap</b></p> $\text{Var}[\widehat{T}(\cdot)] = \frac{1}{B-1} \sum_i^B (T(D_i^*) - \overline{T}(D^*))^2$ <p><b>Jackknife (<math>n = B</math>)</b></p> $\widehat{\text{Var}[T(\cdot)]} = \frac{1}{n(n-1)} \sum_{i=1}^n (T_i^* - \overline{T}(D))^2,$ <p>where</p> $T_i^* = nT(D) - (n-1)T(D_i^*) \quad \overline{T}(D) = \frac{1}{n} \sum_{i=1}^n T_i^*$ |                                               |

## Bootstrap bias correction

### Comments

```
1: Observed data: $D = (X_1, \dots, X_n)$, estimator $\hat{w} = T(D)$
2: for $i = 1, \dots, B$ do
3: Generate
 $D_i^* = (X_1^*, \dots, X_n^*)$ by sampling with replacement.
4: Calculate $T_i^* = T(D_i^*)$.
5: end for
6: Bias corrected estimator is

$$T_1 := 2T(D) - \frac{1}{B} \sum_i^B T_i^*.$$

```

Jackknife also has a bias correction method (see 2016 slides).

## Comments

- Jackknife overestimate variance
- Bootstrap-t method is more accurate than percentile
- Permutations: sampling **without** replacement, bootstrap **with**
- Permutation p-value exact if all permutations used, bootstrap always approximate
- Bootstrap may be used for a wider class of problems
- Nonparametric bootstrap works badly for small samples ( $n < 40$ )
- Parametric bootstrap can work for small samples
- Bias corrections
- Methods do not require distributional assumptions

## Permutation tests for model selection

Data predictors:  $\mathbf{x}_{\cdot, c(V_1, V_2)}$ , response:  $Y$   
Model  $M$  relating  $Y$  and  $X$

**Competing models**  
 $H_0$  variables  $V_1$  should not be in  $M$  (smaller model)  
 $H_1$  all variables are significant  
Test statistic:  $T(M)$

**Permutation test**

- 1: **for**  $i = 1 \dots B$  **do**
- 2:   Obtain  $V_1^*$  by permuting order of columns in  $V_1$ , fit model  $Y \times \mathbf{x}_{\cdot, c(V_1^*, V_2)}$
- 3:   Compute test statistic  $T_i$  for this model
- 4: **end for**
- 5: Compute p-value using above distribution of  $T$

## Stochastic and combinatorial optimization

## Stochastic and combinatorial optimization

## Simulated annealing

### EM Algorithm, Stochastic Optimization

732A90  
Computational Statistics

Krzysztof Bartoszek  
(krzysztof.bartoszek@liu.se)

28 II 2018 (A37)  
Department of Computer and Information Science  
Linköping University

- So far: Unconstrained optimization

- Predictor variables are continuous
- Response function is differentiable

- We discussed Steepest descent, Newton, BFGS, CG

- But: predictors can be discrete  
(scheduling problems, travelling salesman)
- But: outcome can be discrete, noisy or multi-modal

Given a (large) set of states  $S$ , find

$$\min_{s \in S} f(s)$$

- Exhaustive search (shortest path algorithm)
- Often exhaustive search is NP-hard (TSP)
- Alternative: stochastic methods

random search

Motivation from physics: cooling of metal

- Parameters:  
Energy of metal  
(decreasing, but not strictly monotonic)  
Temperature (decreasing)

- Aim: find global minimum energy

### Simulated annealing

```

0. Set $k = 1$ and initialize state s .
1. Compute the temperature $T(k)$.
2. Set $i = 0$ and $j = 0$.
3. Generate a new state r and compute $\delta f = f(r) - f(s)$.
4. Based on δf , decide whether to move from state s to state r .
 accept state r ;
 otherwise
 accept state r with a probability $P(\delta f, T(k))$.
 If state r is accepted, set $s = r$ and $i = i + 1$.
5. If i is equal to the limit for the number of successes at a given temperature,
 go to step 3.
6. Set $j = j + 1$. If j is less than the limit for the number of iterations at
 given temperature, go to step 3.
7. If $i = 0$,
 deliver s as the optimum; otherwise,
 if $k < k_{max}$,
 set $k = k + 1$ and go to step 1;
 otherwise
 issue message that
 'algorithm did not converge in k_{max} iterations'.

```

### Simulated annealing

- [https://www.youtube.com/watch?v=1aq\\_Fpr4XZc](https://www.youtube.com/watch?v=1aq_Fpr4XZc)
- Generating new state:
  - Continuous: choose a new point  $a$  (random) distance from the current one
  - Discrete: similar or some rearrangement
- Selection probability: e.g.  $\exp(-\delta f(x)/T)$ : decreasing with  $f(x)$ , increasing with  $T$
- Temperature function: constant, proportional to  $k$ , or
 
$$T(k+1) = b(k)T(k), \quad b(k) = (\log(k))^{-1}$$

**Remember:** A smaller value is better than one on the path to the global minimum! Always keep track of smallest found.

### Simulated annealing: TSP example

Assume constant temperature

```

1: Choose initial configuration ($Town_1, \dots, Town_n$)
2: $k = 1$
3: while $k < k_{max} + 1$ do
4: Generate new configuration by rearrangement.
 (1, 2, 3, 4, 5, 6, 7, 8, 9) → (1, 6, 5, 4, 3, 2, 7, 8, 9)
 (1, 2, 3, 4, 5, 6, 7, 8, 9) → (1, 7, 8, 2, 3, 4, 5, 6, 9)
5: Measure difference in path length (δf) between old and new configuration
6: if shorter path found then
7: accept it
8: else
9: accept it with probability $P(\delta f)$
10: end if
11: $k + +$
12: end while

```

### Genetic algorithm

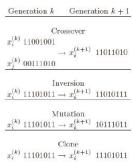
- Inspiration from evolutionary theory: survival of the fittest
  - Variables=genotypes
  - Observation organism, characterized by genetic code
  - State space=population of organisms
  - Objective function=fitness of organism
- New points are obtained from old points by crossover and mutation, the population only retains the fittest organisms (with better objective function).
- [https://en.wikipedia.org/wiki/List\\_of\\_genetic\\_algorithm\\_applications](https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications)

### Genetic algorithm

Encoding points  
 1. Enumerate each element of the state space,  $S$

- Code for observation  $i$  is binary representation of  $i$  (or something else)

Mutation and recombination rules



- Determine a representation of the problem, and define an initial population,  $x_1^{(0)}, \dots, x_m^{(0)}$ . Set  $k = 0$ .
- Compute the objective function (the "fitness") for each member of the population,  $f(x_i^{(0)})$  and assign probabilities  $p_i$  to each item in the population, perhaps proportional to its fitness.
- Choose (with replacement) a probability sample of size  $m \leq n$ . This is the reproducing population.
- Randomly form a new population  $x_1^{(k+1)}, \dots, x_m^{(k+1)}$  from the reproducing population, using various mutation and recombination rules (see Table 6.2). This may be done using random selection of the rule for each individual or pair of individuals.
- If convergence criteria are met, stop, and deliver  $\arg \min_{x_i^{(k+1)}} f(x_i^{(k+1)})$  as the optimum; otherwise, set  $k = k + 1$  and go to step 1.

### Genetic algorithm

Encoding and crossover  
 • Encode tours as  $A_1, \dots, A_n$  but

### Genetic algorithm: TSP example

Parent 1: FAB|ECGD Parent 2: DEA|CGBF  
Child: FAB|CGBF Child: DEA|ECGD

Instead

- Remove FAB from DEACGBF → DECG.  
Child becomes FARDECC.
- Second child will be by taking prefix from Parent 2: DEAFBCG

### Genetic algorithm: Mutations

- If a population is small and only crossover: the input domain becomes limited and may converge to a local minimum.
- Large initial populations are computationally heavy.
- Mutations allow one to explore more of  $S$ : jump out of local minimum.
- In TSP: mutation move a city in the tour to another position.
- Reproduction: Among  $m$  tours selected at step 2, two best are selected for reproduction: two worst replaced by children.
- If  $m$  is large, some tours might never be parents, global solution may be missed. Random clause of reproduction?
- Mutation probability is usually small (unless you want to jump wildly)

### EM algorithm

Fundamental algorithm of computational statistics!

Model depends on the data which are observed (known)  $\mathbf{Y}$  and latent (unobserved) data  $\mathbf{Z}$ .

The data's (both  $\mathbf{Y}$ 's and  $\mathbf{Z}$ 's) distribution depends on some parameters  $\theta$ .

AIM: Find MLE of  $\theta$ .

- All data is known: Apply unconstrained optimization (discussed in Lecture 2)
- Unobserved data
  - Sometimes it is possible to look at the marginal distribution of the observed data.
  - Otherwise: EM algorithm

### EM algorithm

Let

$$Q(\theta, \theta^k) = \int \log p(\mathbf{Y}, \mathbf{Z}|\theta)p(\mathbf{Z}|\mathbf{Y}, \theta^k)d\mathbf{Z} = E[\log(p(\mathbf{Y}, \mathbf{Z}|\theta^k), \mathbf{Y})]$$

```

1: $k = 0, \theta^0 = \theta^0$
2: while Convergence not attained and $k < k_{max} + 1$ do
3: E-step: Derive $Q(\theta, \theta^k)$
4: M-step: $\theta^{k+1} = \operatorname{argmax}_{\theta^k} Q(\theta, \theta^k)$
5: $k + +$
6: end while

```

**Example:** Normal data with missing values (but here analytical approach is also possible)

### EM algorithm: R

```

> Y<-rnorm(100)
> Y[sample(1:length(Y),20,replace=FALSE)]<-NA
> EM.Norm(Y,0.0001,100)
[1] 1.0000 0.1000 -997.5705
[1] 0.1341894 1.3227095 -28.2789837
[1] -0.03897274 1.38734070 -126.86036252
[1] -0.07360517 1.39307050 -126.86081589
[1] -0.08053165 1.39392861 -126.86593837
[1] -0.08191695 1.39408871 -126.86585537
> mean(Y,na.rm=TRUE)
[1] -0.08226328
> var(Y,na.rm=TRUE)
[1] 1.411775

```

### EM algorithm: R

Notice: can be done by studying marginal distribution of observed data.

**Mixture models**  $Z$  is a latent variable,  $P(Z = k) = \pi_k$

- Mixed data comes from different sources (e.g. for regression, classification)

• Clustering

- Density in each cluster is normally distributed.
- Cluster label is latent (we do not know what are the classes an observation is from the given cluster)

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x | \mu_k, \Sigma_k) \quad (\text{informally})$$

Direct MLE leads to numerical problems.

Introduce latent class variables and use EM.

1. Initialize the means  $\mu_0$ , covariances  $\Sigma_0$  and evaluate the initial value of the log likelihood.

2. **E-step:** Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)}{\sum_l \pi_l \mathcal{N}(x_n | \mu_l, \Sigma_l)} \quad (9.23)$$

3. **M step:** Re-estimate the parameters using the current responsibilities

$$\mu_i^{(n)} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) x_n \quad (9.24)$$

$$\Sigma_i^{(n)} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (x_n - \mu_i^{(n)}) (x_n - \mu_i^{(n)})^T \quad (9.25)$$

$$\pi_i^{(n)} = \frac{N_k}{N} \quad (9.26)$$

where

$$N_k = \sum_{n=1}^N \gamma(z_{nk}) \quad (9.27)$$

4. Evaluate the log likelihood

$$\ln p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{k=1}^K \left\{ \sum_{n=1}^N \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k) \right\} \quad (9.28)$$

$$Ez_{nk} = \gamma(z_{nk})$$

Random walk over the state space in search of minimum

- Follow decreasing path

- **BUT** with a certain probability go to higher values, to avoid local minima traps.

- **Never forget** best found conformation!

- Simulated annealing, Genetic algorithm, **EM algorithm**, Stochastic gradient descent (see 2016 slides)