

Section 1

INTRODUCTION

Introduction to Scientific Computing

Fall 2013

Ilya Safro
Clemson University

Course: Scientific Computing,
CP SC 481/681

Instructor: Ilya Safro

Office hours: 228 McAdams Hall, TBD

Email: isafro@clemson.edu

- Do you have a plan, Mr. Fix?
- Do I have a plan? Yes, I have a plan!

Movie "80 Days Around the World"

Type of work	How many?	Time	Points	<i>Class Attendance Mandatory</i>
Theoretical homework	≤ 8	2 weeks	20	
Practical homework	≤ 8	2 weeks	20	
Oral presentation or Mid term exam	1 1	2 weeks -	20	
Final home project or Final exam	1 1	2 weeks -	40	

Grading: A(≥ 92), B(80-91), C(65-79), D(55-64), F(0-54)



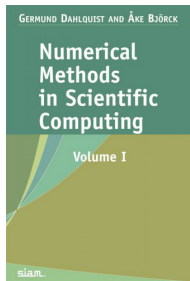
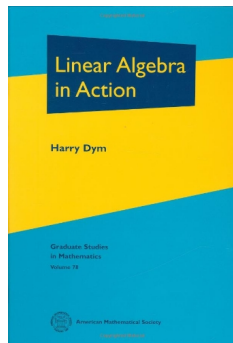
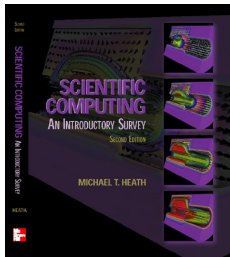
Extra work in class - up to 10 points. We do not want to miss the next Turing and Fields laureates, so any submitted journal paper or TR written during and as a result of this course - 100 points, and new interesting ideas - up to 100 points (both are based on instructor's subjective judgement)

TYPES OF HOMEWORK

- ▶ (I)mplementation. You should implement the solution and submit code, input, and output. You can use one of these: C/C++, Java, Fortran, Python, Perl, Matlab, Scilab. Using C/C++/Java/Fortran scientific libraries (such as LAPACK, BLAS, GNU Scientific Library) is a plus and bonus. **Recommendation: Python + SciPy, NumPy.**
- ▶ (T)heoretical. You should prove or compute something without implementing it. Must be submitted in pdf or word formats.
- ▶ (T_{ns})heoretical. Do not submit.
- ▶ (R)eading. You should read the handout or paper. You don't submit anything but I can ask something similar at final/midterm exam.

Section 2

LITERATURE



Section 3

TITAN

TITAN SUPERCOMPUTER

AT OAK RIDGE NATIONAL LABORATORY



FLOPs = Floating-point operations per sec
 iPad: 1-2 GFlops
 Desktop: 5-10GFlops (can be ≈ 100 GFlops)
 Giga = 10^9 , Tera = 10^{12} , Peta = 10^{15} , Exa = 10^{18}

TITAN SPECS

PEAK PERFORMANCE

20⁺
 PETAFLOPS

299,008
 OPTERON CORES



NVIDIA TESLA
 K20 GPU ACCELERATORS

18,688

GPUs

TOTAL SYSTEM MEMORY

710

TERABYTES

COMPUTE NODES

18,688

32GB + 6GB



Memory Per Node

GEMINI
 INTERCONNECT

4,352 sqft

FLOOR SPACE

Section 4

MATHEMATICAL SOFTWARE

DESIRABLE CHARACTERISTICS OF MATHEMATICAL SOFTWARE

- ▶ **Reliability:** works correctly for easy problems
- ▶ **Robustness:** usually works for hard problems, but fails gracefully and informatively when it does fail
- ▶ **Accuracy:** produces results as accurate as warranted by the problem and input data, preferably with an estimate of the accuracy achieved
- ▶ **Efficiency:** requires execution time and storage that are close to the minimum possible for the problem being solved
- ▶ **Maintainability:** is easy to understand and modify
- ▶ **Portability:** adapts with little or no change to new computing environments
- ▶ **Usability:** has a convenient and well-documented user interface
- ▶ **Applicability:** solves a broad range of problems

Section 5

COMPUTER ARITHMETIC

Subsection 1

COMPUTER ARITHMETIC

FLOATING-POINT NUMBERS

In computers the reals (\mathbb{R}) are usually represented by a floating-point number systems. For example $12345 = 1.2345 \times 10^4$, and $0.12345 = 1.2345 \times 10^{-1}$. Any floating point number has the form

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}} \right) \beta^E, \text{ where}$$

$0 \leq d_i \leq \beta - 1$ are integers (called mantissa), and

β base

p precision

$E \in [L, U]$ exponent within range $[L, U]$.

Sign, exponent, and mantissa are stored in separate fields of fixed width in one structure (called f-p word). Usually $\beta = 2$, however, history knows other examples such as $\beta = 8$, and 16.

Typical precision: IEEE single-, double-, and quadruple-precision.

FLOATING-POINT NUMBERS

A floating-point system is normalized if $d_0 \neq 0$ unless the number is zero. Advantages of normalization?

FLOATING-POINT NUMBERS

A floating-point system is normalized if $d_0 \neq 0$ unless the number is zero. Advantages of normalization?

- ▶ unique representation
- ▶ no wasted digits
- ▶ for $\beta = 2$ the leading bit is always 1 (better precision with extra bit)

ROUNDING

Machine numbers are those that can be exactly represented by f-p. If $x \in \mathbb{R}$ cannot be exactly represented by f-p then it is *approximated* by some nearby f-p $\mathbf{fl}(x)$.

- ▶ Chop rounding rule (old): round to the next f-p number towards zero.
- ▶ Round to nearest rule (IEEE standard, more accurate and more expensive): nearest f-p (tie \rightarrow round to even).

Example 1.6 Rounding Rules. Rounding the following decimal numbers to two digits using each of the rounding rules gives the following results

Number	Chop	Round to nearest	Number	Chop	Round to nearest
1.649	1.6	1.6	1.749	1.7	1.7
1.650	1.6	1.6	1.750	1.7	1.8
1.651	1.6	1.7	1.751	1.7	1.8
1.699	1.6	1.7	1.799	1.7	1.8

Machine precision: $\epsilon_{\text{mach}} = \beta^{1-p}$ (chop), $\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p}$ (rounding to nearest).

Relative error is bounded $\left| \frac{\mathbf{fl}(x) - x}{x} \right| \leq \epsilon_{\text{mach}}$.

Potential sources of error: rounding, dec-bin/bin-dec conversion

FLOATING-POINT ARITHMETIC, SOURCE OF ERROR

Example of addition: $3.25 \times 10^3 + 2.63 \times 10^{-1}$

- ▶ Align decimal points
- ▶ Add
- ▶ Normalize (in this example already normalized)

$$\begin{array}{r}
 3.250000 \times 10^3 \\
 + 0.000263 \times 10^3 \\
 \hline
 = 3.250263 \times 10^3
 \end{array}$$

what if $p = 2$?

FLOATING-POINT ARITHMETIC, SOURCE OF ERROR

Example of addition: $3.25 \times 10^3 + 2.63 \times 10^{-1}$

- ▶ Align decimal points
- ▶ Add
- ▶ Normalize (in this example already normalized)

$$\begin{array}{r}
 3.250000 \times 10^3 \\
 + 0.000263 \times 10^3 \\
 \hline
 = 3.250263 \times 10^3
 \end{array}$$

what if $p = 2$?

Example of multiplication: $3.0 \times 10^1 \cdot 0.5 \times 10^2$

- ▶ Multiply mantissas
- ▶ Add exponents

$$\begin{array}{r}
 * \quad \left| \begin{array}{c|c|c} 3.0 & \times & 10^1 \\ 0.5 & \times & 10^2 \end{array} \right. \\
 \hline
 = \quad \left| \begin{array}{c|c|c} 1.5 & \times & 10^3 \end{array} \right.
 \end{array}$$

ONE MORE EXAMPLE OF ϵ_{MACH} PROBLEM

Consider harmonic series which are divergent

$$\sum_{n=1}^{\infty} \frac{1}{n}.$$

What happens if we compute this series? Are they finite because $\frac{1}{n}$ will eventually underflow or the partial sum will overflow?

ONE MORE EXAMPLE OF ϵ_{MACH} PROBLEM

Consider harmonic series which are divergent

$$\sum_{n=1}^{\infty} \frac{1}{n}.$$

What happens if we compute this series? Are they finite because $\frac{1}{n}$ will eventually underflow or the partial sum will overflow?

No. The sum will be finite before either of these occurs. The partial sum ceases to change once $1/n$ becomes negligible relative to the partial sum, i.e., when

$$1/n < \epsilon_{\text{mach}} \sum_{k=1}^{n-1} \frac{1}{k}.$$

F-P IS COMMUTATIVE BUT NOT ALWAYS ASSOCIATIVE

Ideally, $x \text{ flop } y = \text{fl}(x \text{ op } y)$. Indeed, this happens withing ranges of IEEE f-p standard. However, some laws are not necessarily valid.

Consider ϵ slightly smaller than ϵ_{mach}

$$(1 + \epsilon) + \epsilon = 1, \text{ but } 1 + (\epsilon + \epsilon) > 1.$$

USE MATHEMATICALLY EQUIVALENT BUT SIMPLER FORMULA

Given finite sequence of $x_i \in \mathbb{R}$,

$$\text{Mean } \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i; \quad \text{Standard Deviation } \sigma = \left(\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{1/2}$$

σ is computed by two passes for \bar{x} and $\sigma \Rightarrow$ error is accumulated.

USE MATHEMATICALLY EQUIVALENT BUT SIMPLER FORMULA

Given finite sequence of $x_i \in \mathbb{R}$,

$$\text{Mean } \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i; \quad \text{Standard Deviation } \sigma = \left(\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{1/2}$$

σ is computed by two passes for \bar{x} and $\sigma \Rightarrow$ error is accumulated. Thus for better efficiency one can use

$$\sigma = \left(\frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - n\bar{x}^2 \right) \right)^{1/2}$$

which can be computed in one pass.

BIG \mathcal{O} NOTATION (LANDAU'S SYMBOL)

- ▶ Analysis of algorithms: given problem of size n , and $T(n) = 5n^3 + 2n^2 + n$, we say $T(n)$ grows at the order on n^3 or $T(n) \in \mathcal{O}(n^3)$.
- ▶ Error term in approximation: in $e^x = 1 + x + x^2/2 + \mathcal{O}(x^3)$, where $x \rightarrow 0$ we express that the error is smaller in $|\cdot|$ than cx^3 (c is constant).

BIG \mathcal{O} NOTATION (LANDAU'S SYMBOL)

- ▶ Analysis of algorithms: given problem of size n , and $T(n) = 5n^3 + 2n^2 + n$, we say $T(n)$ grows at the order on n^3 or $T(n) \in \mathcal{O}(n^3)$.
- ▶ Error term in approximation: in $e^x = 1 + x + x^2/2 + \mathcal{O}(x^3)$, where $x \rightarrow 0$ we express that the error is smaller in $|\cdot|$ than cx^3 (c is constant).

Definition.

Let $f(x)$, and $g(x)$ are defined on \mathbb{R} . We say that

$$f(x) \in \mathcal{O}(g(x))$$

iff $\exists N, C$, s.t. $|f(x)| \leq C|g(x)|$ for all $x > N$.

Intuition: f does not grow faster than g .

Edmund Landau 1877-1938



BIG \mathcal{O}/o NOTATIONS (LANDAU'S SYMBOLS)

Definition.

If $a \in \mathbb{R}$ we write $f(x) \in \mathcal{O}(g(x))$ for $x \rightarrow a$ iff $\exists d > 0, C$ s.t.

$$|f(x)| \leq C|g(x)| \text{ for all } x \text{ s.t. } |x - a| < d.$$

Definition.

We write $f(x) \in o(g(x))$ for $x \rightarrow \infty$ iff $\forall C > 0 \exists N$ s.t. $\forall x > N$ we have

$$|f(x)| < C|g(x)|$$

$f(x) \in o(g(x))$ means that f grows much slower than g and is insignificant in comparison.

MORE NOTATIONS FOR COMPARING FUNCTIONS

Notation	Definition	Analogy
$f(n) \in \mathcal{O}(g(n))$...	\leq
$f(n) \in o(g(n))$...	$<$
$f(n) \in \Theta(g(n))$	$g(n) \in \mathcal{O}(f(n))$	\geq
$f(n) \in \omega(g(n))$	$g(n) \in o(f(n))$	$>$
$f(n) \in \Theta(g(n))$	$f(n) \in \mathcal{O}(g(n))$ and $g(n) \in \mathcal{O}(f(n))$	$=$

Homework 1 (T).

1. Prove or disprove that $T(n) = 8n^4 + 2n^2 + 10 \in \mathcal{O}(n^2)$.
2. Prove or disprove that $T(n) = 3n^3 + 20n^2 + 5n \in \mathcal{O}(n^3)$.
3. Prove or disprove that $T(n) = n^2 + 10n \in \Theta(n^2)$.
4. Prove or disprove that $T(n) = n^7 \in o(n^8)$.
5. Prove or disprove that $T(n) = n^3 + n^2 \in \omega(n^2)$.
6. Calculate the complexity of any matrix-matrix multiplication algorithm for square matrices.

Section 6

SYSTEMS OF LINEAR EQUATIONS

SYSTEMS OF LINEAR EQUATIONS

A system of linear equations is represented by

$$Ax = b, \text{ where } A \in \mathcal{F}^{m \times n} \text{ (usually field } \mathcal{F} \text{ will be either } \mathbb{R} \text{ or } \mathbb{C}\text{)}$$

An $n \times n$ matrix A is called nonsingular if it satisfies any of these conditions:

1. $\exists A^{-1}$ s.t. $AA^{-1} = A^{-1}A = I$
2. $\det(A) \neq 0$
3. $\text{rank}(A) = n$ (rank is the maximum number of linearly independent rows or columns)
4. $\forall z \in \mathcal{F}^n$ if $z \neq 0$ then $Az \neq 0$

Otherwise it is singular. Existence and uniqueness is summarized in:

- ▶ Unique solution: A nonsingular, b arbitrary
- ▶ Infinitely many solutions: A is singular, $b \in \mathbf{span}(A)$
- ▶ No solution: A singular, $b \notin \mathbf{span}(A)$

$$\mathbf{span}(A) = \{Ax : x \in \mathbb{R}^n\}$$

Subsection 1

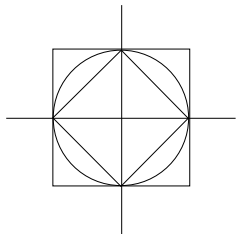
SENSITIVITY AND CONDITIONING

VECTOR NORMS

We define p -norm and ∞ -norm of vector x , $p > 0$ by

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}$$

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$



Unit 2-dim spheres in various norms

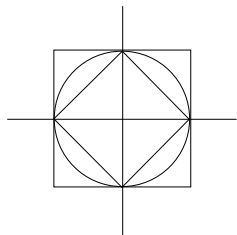
What norms do we see here?

VECTOR NORMS

We define p -norm and ∞ -norm of vector x , $p > 0$ by

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}$$

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$



Unit 2-dim spheres in various norms

What norms do we see here?

$\forall x, y \in \mathcal{F}^n$ and p -norm the following hold

1. $\|x\| > 0$ if $x \neq 0$
2. $\forall \gamma \in \mathcal{F} \quad \|\gamma x\| = |\gamma| \cdot \|x\|$
3. $\|x + y\| \leq \|x\| + \|y\|$
(triangle inequality)
4. $|\|x\| - \|y\|| \leq \|x - y\|$

Some useful facts

1. $\|x\|_1 \geq \|x\|_2 \geq \|x\|_\infty$
2. $\|x\|_1 \leq \sqrt{n} \|x\|_2$
3. $\|x\|_2 \leq \sqrt{n} \|x\|_\infty$
4. $\|x\|_1 \leq n \|x\|_\infty$

MATRIX NORMS

Matrix norms are defined in terms of vector norms, namely,

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

MATRIX NORMS

Matrix norms are defined in terms of vector norms, namely,

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

Matrix norm measures the maximum stretching the matrix does to vectors. Examples of matrix norms for $A \in \mathcal{F}^{m \times n}$:

$$\|A\|_1 = \max_j \sum_{i=1}^m |a_{ij}| \quad \text{and} \quad \|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}| \quad (1)$$

Check that matrix norms for $A \in \mathcal{F}^{n \times 1}$ agree with vector norms.

Homework 2 (T).

Choose one equality in (1) and prove it.

PROPERTIES OF MATRIX NORMS

For any two matrices A and B the following properties hold

1. $\|A\| > 0$ if $A \neq 0$
2. $\forall \gamma \quad \|\gamma A\| = |\gamma| \cdot \|A\|$
3. $\|A + B\| \leq \|A\| + \|B\|$
4. $\|AB\| \leq \|A\| \cdot \|B\|$ (this norm is called multiplicative)
5. $\forall x \quad \|Ax\| \leq \|A\| \cdot \|x\|$

Note that in general case a matrix norm is any function that satisfies the first three properties. Properties 4 and 5 work for p -norms but not necessarily for other norms.

Another useful norm is the Frobenius norm $\|A\|_F$

$$\|A\|_F = \left(\sum_{i=1}^n \sum_{j=1}^m |a_{ij}|^2 \right)^{1/2}$$

Homework 3 (I).

Generate matrices with random values form 0 to 100. Calculate their 1-, ∞ -, and F -norms.

SENSITIVITY OF MATRICES / SMALL PERTURBATIONS

Main idea: if A is invertible and B is close to A , i.e., $\|A - B\|$ is small enough wrt some multiplicative norm, then B is also invertible.

Theorem 1.

Let $A, B \in \mathbb{C}^{n \times n}$ and A is invertible, such that $\|A^{-1}\| \leq \gamma$ and $\|B - A\| < 1/\gamma$ wrt to some multiplicative norm for some scalar $\gamma > 0$. Then:

1. B is invertible

$$2. \|B^{-1}\| \leq \frac{\gamma}{1 - \gamma\|B - A\|}.$$

Homework 4 (IT).

Given $A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 5 \\ 5 & 6 & 0 \end{pmatrix}$, $B_1 = \begin{pmatrix} 1.01 & 2 & 3 \\ 0 & 1 & 5.01 \\ 5 & 6 & 0 \end{pmatrix}$, and $B_2 = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 5 \\ 0.5 & 1.5 & 4 \end{pmatrix}$. Calculate

A^{-1} , and check if B_1 and B_2 are invertible by using Theorem 1. If one of them is invertible compute an upper bound for $\|B^{-1}\|$.

SENSITIVITY OF MATRICES

Even if the difference between norms after some tiny perturbation is small the difference between the solutions of systems of equations can be large.

$$\begin{pmatrix} 52 & 78 & 86 \\ 36 & 11 & 76 \\ 30 & 44 & 50 \end{pmatrix} \cdot x = \begin{bmatrix} 63 \\ 72 \\ 9 \end{bmatrix}$$

$$\implies x \approx 10^4 \cdot \begin{bmatrix} 3.4884 \\ -0.5994 \\ -1.56555 \end{bmatrix}$$

$$\|\cdot\|_1 = 212$$

$$\begin{pmatrix} 52 & 78 & 86 \\ 36 & 11 & 76 \\ 30 & 44 & 50.0006 \end{pmatrix} \cdot x = \begin{bmatrix} 63 \\ 72 \\ 9 \end{bmatrix}$$

$$\implies x \approx 10^4 \cdot \begin{bmatrix} 2.6123 \\ -0.4489 \\ -1.1723 \end{bmatrix}$$

$$\|\cdot\|_1 = 212.0006$$

Can we estimate how sensitive is our matrix? In particular, can the matrix stretch or shrink the vector?

MATRIX CONDITION NUMBER

Definition.

The condition number of a nonsingular square matrix A wrt to a given norm is

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\| = \left(\max_{x \neq 0} \frac{\|Ax\|}{\|x\|} \right) \cdot \left(\min_{x \neq 0} \frac{\|Ax\|}{\|x\|} \right)^{-1}$$

Matlab example: `condition_number.m`

MATRIX CONDITION NUMBER

Definition.

The condition number of a nonsingular square matrix A wrt to a given norm is

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\| = \left(\max_{x \neq 0} \frac{\|Ax\|}{\|x\|} \right) \cdot \left(\min_{x \neq 0} \frac{\|Ax\|}{\|x\|} \right)^{-1}$$

Matlab example: `condition_number.m`

MCN measures the amount of maximum stretching of vectors vs maximum shrinking. In the example we demonstrated the amount of distortion of unit sphere under transformation.

- | | |
|---|--|
| <ul style="list-style-type: none"> ▶ $\text{cond}(A) \geq 1$ ▶ $\text{cond}(I) = 1$ | <ul style="list-style-type: none"> ▶ $\text{cond}(\alpha A) = \alpha \text{cond}(A)$ ▶ $D = \text{diag}(d_i) \Rightarrow \text{cond}(D) = \max d_i / \min d_i$ |
|---|--|

MCN is a measure of how close a matrix is to being singular.

SENSITIVITY OF LINEAR SYSTEMS

Homework 5 (R).

Read about condition estimation (SC pp. 58-59).

Let x and \hat{x} be the solutions of nonsingular $Ax = b$ and $A\hat{x} = b + \Delta b$, respectively. If $\Delta x = \hat{x} - x$ then $Ax + A\Delta x = b + \Delta b$, and $\Delta x = A^{-1}\Delta b$.

$$\begin{aligned} \|b\| = \|Ax\| &\leq \|A\| \cdot \|x\| &\Rightarrow \|x\| &\geq \|b\|/\|A\| \\ \Delta x = A^{-1}\Delta b &&\Rightarrow \|\Delta x\| &\leq \|A^{-1}\| \cdot \|\Delta b\|, \end{aligned}$$

and

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A^{-1}\| \cdot \|\Delta b\| \cdot \frac{\|A\|}{\|b\|} = \mathbf{cond}(A) \frac{\|\Delta b\|}{\|b\|}, \quad (2)$$

i.e., given relative change Δb we can bound maximum relative change in the solution.

Similar result holds for matrices. If $(A + E)\hat{x} = b$ then

$$\frac{\|\Delta x\|}{\|\hat{x}\|} \leq \mathbf{cond}(A) \frac{\|E\|}{\|A\|} \quad (3)$$

... CAN WE USE CALCULUS?

We define $A(t) = A + tE$ and $b(t) = b + t\Delta b$, where $t \in \mathbb{R}$ is a parameter. Solve the linear system $A(t)x(t) = b(t)$ (*).

Differentiate (*)

$$A'(t)x(t) + A(t)x'(t) = b'(t)$$

... CAN WE USE CALCULUS?

We define $A(t) = A + tE$ and $b(t) = b + t\Delta b$, where $t \in \mathbb{R}$ is a parameter. Solve the linear system $A(t)x(t) = b(t)$ (*).

$$\begin{array}{ll} \text{Differentiate (*)} & A'(t)x(t) + A(t)x'(t) = b'(t) \\ \text{since } A'(t) = (A + tE)' = E \text{ we have} & Ex(t) + A(t)x'(t) = \Delta b, \end{array}$$

... CAN WE USE CALCULUS?

We define $A(t) = A + tE$ and $b(t) = b + t\Delta b$, where $t \in \mathbb{R}$ is a parameter.

Solve the linear system $A(t)x(t) = b(t)$ (*).

Differentiate (*)

since $A'(t) = (A + tE)' = E$ we have

i.e.,

$$A'(t)x(t) + A(t)x'(t) = b'(t)$$

$$Ex(t) + A(t)x'(t) = \Delta b,$$

$$x'(t) = A(t)^{-1}(\Delta b - Ex(t)),$$

... CAN WE USE CALCULUS?

We define $A(t) = A + tE$ and $b(t) = b + t\Delta b$, where $t \in \mathbb{R}$ is a parameter. Solve the linear system $A(t)x(t) = b(t)$ (*).

Differentiate (*)

since $A'(t) = (A + tE)' = E$ we have

i.e.,

and,

$$A'(t)x(t) + A(t)x'(t) = b'(t)$$

$$Ex(t) + A(t)x'(t) = \Delta b,$$

$$x'(t) = A(t)^{-1}(\Delta b - Ex(t)),$$

$$x'(0) = A^{-1}(\Delta b - Ex(0)).$$

... CAN WE USE CALCULUS?

We define $A(t) = A + tE$ and $b(t) = b + t\Delta b$, where $t \in \mathbb{R}$ is a parameter. Solve the linear system $A(t)x(t) = b(t)$ (*).

Differentiate (*)

since $A'(t) = (A + tE)' = E$ we have

i.e.,

and,

By Taylor's theorem we have

$$A'(t)x(t) + A(t)x'(t) = b'(t)$$

$$Ex(t) + A(t)x'(t) = \Delta b,$$

$$x'(t) = A(t)^{-1}(\Delta b - Ex(t)),$$

$$x'(0) = A^{-1}(\Delta b - Ex(0)).$$

$$x(t) = x(0) + tx'(0) + \mathcal{O}(t^2), \text{ i.e.,}$$

... CAN WE USE CALCULUS?

We define $A(t) = A + tE$ and $b(t) = b + t\Delta b$, where $t \in \mathbb{R}$ is a parameter. Solve the linear system $A(t)x(t) = b(t)$ (*).

Differentiate (*)

since $A'(t) = (A + tE)' = E$ we have

i.e.,

and,

By Taylor's theorem we have

$$A'(t)x(t) + A(t)x'(t) = b'(t)$$

$$Ex(t) + A(t)x'(t) = \Delta b,$$

$$x'(t) = A(t)^{-1}(\Delta b - Ex(t)),$$

$$x'(0) = A^{-1}(\Delta b - Ex(0)).$$

$$x(t) = x(0) + tx'(0) + \mathcal{O}(t^2), \text{ i.e.,}$$

If $x \equiv x(0)$ we obtain from

$$x(t) - x(0) = tx'(0) + \mathcal{O}(t^2) = tA^{-1}(\Delta b - Ex(0)) + \mathcal{O}(t^2)$$

... CAN WE USE CALCULUS?

We define $A(t) = A + tE$ and $b(t) = b + t\Delta b$, where $t \in \mathbb{R}$ is a parameter. Solve the linear system $A(t)x(t) = b(t)$ (*).

Differentiate (*)

since $A'(t) = (A + tE)' = E$ we have

i.e.,

and,

By Taylor's theorem we have

$$A'(t)x(t) + A(t)x'(t) = b'(t)$$

$$Ex(t) + A(t)x'(t) = \Delta b,$$

$$x'(t) = A(t)^{-1}(\Delta b - Ex(t)),$$

$$x'(0) = A^{-1}(\Delta b - Ex(0)).$$

$$x(t) = x(0) + tx'(0) + \mathcal{O}(t^2), \text{ i.e.,}$$

If $x \equiv x(0)$ we obtain from

$$x(t) - x(0) = tx'(0) + \mathcal{O}(t^2) = tA^{-1}(\Delta b - Ex(0)) + \mathcal{O}(t^2)$$

$$\frac{\|x(t) - x\|}{\|x\|} \leq \|A^{-1}\| \left(\frac{\|\Delta b\|}{\|x\|} + \|E\| \right) |t| + \mathcal{O}(t^2)$$

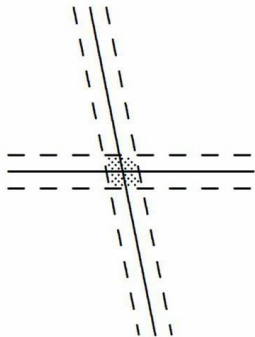
$$\leq \mathbf{cond}(A) \left(\frac{\|\Delta b\|}{\|A\| \cdot \|x\|} + \frac{\|E\|}{\|A\|} \right) |t| + \mathcal{O}(t^2) \quad (4)$$

$$\leq \mathbf{cond}(A) \left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|E\|}{\|A\|} \right) |t| + \mathcal{O}(t^2)$$

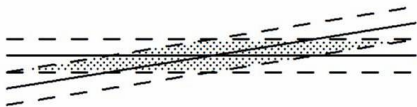
relative change in the
solution

relative change in the problem data

GEOMETRIC INTERPRETATION



well-conditioned



ill-conditioned

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \mathbf{cond}(A) \cdot \epsilon_{mach}$$

(illustration from SC book)

RESIDUAL

Definition.

The residual of an approximate solution \hat{x} to $Ax = b$ is the difference

$$r = b - A\hat{x}$$

RESIDUAL

Definition.

The residual of an approximate solution \hat{x} to $Ax = b$ is the difference

$$r = b - A\hat{x}$$

In theory if A is nonsingular then $\|\Delta x\| = \|\hat{x} - x\| = 0$ iff $\|r\| = 0$. In practice, the situation is more complicated.

Example (Scaling).

Multiplying $Ax = b$ by α gives the same solution but $r = \alpha(b - A\hat{x})$, i.e., r depends on the scaling of the problem.

RESIDUAL

Definition.

The residual of an approximate solution \hat{x} to $Ax = b$ is the difference

$$r = b - A\hat{x}$$

In theory if A is nonsingular then $\|\Delta x\| = \|\hat{x} - x\| = 0$ iff $\|r\| = 0$. In practice, the situation is more complicated.

Example (Scaling).

Multiplying $Ax = b$ by α gives the same solution but $r = \alpha(b - A\hat{x})$, i.e., r depends on the scaling of the problem.

Definition.

Relative residual is defined to be $\frac{\|r\|}{\|A\| \cdot \|\hat{x}\|}$.

The error is $\|\Delta x\| = \|\hat{x} - x\| = \|A^{-1}(A\hat{x} - b)\| \leq \|A^{-1}\| \cdot \|r\|$

$\Rightarrow \|\Delta x\|/\|\hat{x}\| \leq \mathbf{cond}(A) \frac{\|r\|}{\|A\| \cdot \|\hat{x}\|}$, i.e., small relative residual implies small relative error when A is well-conditioned.

What if the relative residual is large? Let \hat{x} be the approx sol of $Ax = b$

i.e., it satisfies $(A + E)\hat{x} = b$ for some E

$$\|r\| = \|b - A\hat{x}\| = \|E\hat{x}\| \leq \|E\| \cdot \|\hat{x}\| \Rightarrow \frac{\|r\|}{\|A\| \cdot \|\hat{x}\|} \leq \frac{\|E\|}{\|A\|},$$

i.e., ... ?

What if the relative residual is large? Let \hat{x} be the approx sol of $Ax = b$

i.e., it satisfies $(A + E)\hat{x} = b$ for some E

$$\|r\| = \|b - A\hat{x}\| = \|E\hat{x}\| \leq \|E\| \cdot \|\hat{x}\| \Rightarrow \frac{\|r\|}{\|A\| \cdot \|\hat{x}\|} \leq \frac{\|E\|}{\|A\|},$$

i.e., ... ? large relative residual implies a large error in the matrix.

Example (Small Residual).

$$Ax = \begin{pmatrix} 0.913 & 0.659 \\ 0.457 & 0.33 \end{pmatrix} \cdot x = b = (0.2540, 1.27)^T$$

$$\hat{x}_1 = (-0.0827, 0.5)^T, \quad \text{and} \quad \hat{x}_2 = (0.999, -1.001)^T$$

$$\|r_1\|_1 = 2.1 \cdot 10^{-4}, \quad \text{and} \quad \|r_2\|_1 = 2.4 \cdot 10^{-2}$$

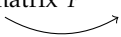
$\|r_1\|$ is smaller than $\|r_2\|$ but the exact solution is $(1, -1)$. A is ill-conditioned, i.e., large condition number and small residual do not imply small solution.

PROBLEM TRANSFORMATIONS

Can we find an easier system to compute (instead of the original one) which has the same solution?

$$Ax = b \Rightarrow M \cdot Ax = M \cdot b, \text{ } M \text{ is nonsingular}$$

Transformation 1: Permutation. Rows of A can be reordered without changing x .

permutation matrix P 

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} v_3 \\ v_1 \\ v_2 \end{pmatrix}$$

Homework 6 (T_{ns}).

Prove that for permutation matrix P the following holds $P^{-1} = P^T$.

We will use this fact in the next example.

Example (Permutation Matrix).

$$A = \begin{pmatrix} 0 & 0 & 3 & 5 \\ 0 & 9 & 1 & 0 \\ 0 & 0 & 0 & 4 \\ 1 & 0 & 2 & 3 \end{pmatrix}, Ax = b \implies$$

triangular matrix,
easy to solve

$$PAz = Pb, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} Az = \begin{pmatrix} 1 & 0 & 2 & 3 \\ 0 & 9 & 1 & 0 \\ 0 & 0 & 3 & 5 \\ 0 & 0 & 0 & 4 \end{pmatrix} \cdot z = Pb$$

$$z = (AP)^{-1}b = P^{-1}A^{-1}b = P^T A^{-1}b = P^T x \implies x = Pz$$

Example (Diagonal Scaling).

D is diagonal $\implies DA$ does row scaling of A . Can affect accuracy of numerical solution.

FACTOR-SOLVE METHODS

To solve $Ax = b$, we rewrite A as

$$A = A_1 \cdot A_2 \cdot \dots \cdot A_k,$$

where each A_i is easy to solve, i.e., we solve $(A_1 \cdot A_2 \cdot \dots \cdot A_k)x = b$.

$$\begin{aligned} A_1 y_1 &= b \\ A_2 y_2 &= y_1 \\ &\dots \\ A_{k-1} y_{k-1} &= y_{k-2} \\ A_k x &= y_{k-1} \end{aligned}$$

Complexity: total flops = flops(factorization) + flops(solve k systems)
 Usually flops(factorization) \gg flops(solve k systems), i.e., same system of equations can be solved much faster for different b .

TRIANGULAR MATRICES

An upper triangular matrix is a matrix U with $u_{ij} = 0$ for $i > j$.

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

Some properties of upper (and lower) triangular matrices

- ▶ $U_1 + U_2$, $U_1 \cdot U_2$, U_1^{-1} are upper triangular
- ▶ If $U_0 = U_1 U_2$ then $u_{0(ii)} = u_{1(ii)} \cdot u_{2(ii)}$
- ▶ $u_{ii}^{-1} = 1/u_{ii}$
- ▶ $Ux = b$ is easy to solve by recursive back substitution in $\mathcal{O}(n^2)$ flops.

$$x_n = b_n/u_{nn}, \quad x_i = (b_i - \sum_{k=i+1}^n u_{ik}x_k)/u_{ii}, \quad i = n-1 : 1$$

LU FACTORIZATION: ELEMENTARY MATRICES

Elementary matrix = I + one off-diagonal entry. Example

$$E_{21} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Let us apply Gaussian elimination $[E_{21}|I] \rightarrow [I|E_{21}^{-1}]$, i.e.,

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 3 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{\text{row}_2 - 3\text{row}_1} \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -3 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

In general, E_{ij}^{-1} is easy to compute by negating (i, j) of E_{ij} . Left multiplication on E_{ij} is equivalent to an elementary row operation.

$$\text{Example: } E_{21}A = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \text{row}_1 \\ \text{row}_2 \\ \text{row}_3 \end{bmatrix} = \begin{bmatrix} \text{row}_1 \\ 3\text{row}_1 + \text{row}_2 \\ \text{row}_3 \end{bmatrix}$$

LU FACTORIZATION

Claim: We can obtain upper triangular form by multiplying A by sequence of elementary matrices E_{ij} . Each E_{ij} corresponds to one operation of Gaussian elimination.

Homework 7 (T).

Let $A = \begin{bmatrix} 5 & -1 & 2 \\ 10 & 3 & 7 \\ 15 & 17 & 19 \end{bmatrix}$. Find sequence of elementary matrices to obtain an upper triangular form.

Since we can find a sequence of elementary matrices to obtain an upper triangular form $E_{i_1j_1}E_{i_2j_2}\dots E_{i_nj_n}A = U$ and for each E_{ij} we can find an inverse E_{ij}^{-1} we can find a factorization of A into multiplication of lower (L) and upper (U) triangular forms:

$$\begin{aligned} E_{i_1j_1}E_{i_2j_2}\dots E_{i_nj_n}A &= U \\ (E_{i_1j_1}E_{i_2j_2}\dots E_{i_nj_n})^{-1}E_{i_1j_1}E_{i_2j_2}\dots E_{i_nj_n}A &= (E_{i_1j_1}E_{i_2j_2}\dots E_{i_nj_n})^{-1}U \\ A &= E_{i_nj_n}^{-1}\dots E_{i_2j_2}^{-1}E_{i_1j_1}^{-1}U \\ A &= LU \quad \Leftarrow (E_{i_1j_1}E_{i_2j_2}\dots E_{i_nj_n})^{-1} = L \text{ has lower triangular form} \end{aligned}$$

LU FACTORIZATION

Given non-singular A ...

Theorem: If row swaps are not needed to solve the linear system $Ax = b$ by Gaussian elimination, then A has the LU factorization.

LU FACTORIZATION

Given non-singular A ...

Theorem: If row swaps are not needed to solve the linear system $Ax = b$ by Gaussian elimination, then A has the LU factorization.

Theorem: There is a permutation matrix P such that there exists LU factorization of PA than can be carried without row swaps $PA = LU$. For fixed P the factorization is unique.

Two steps of the algorithm for solving $Ax = b$

- ▶ Factorization $PA = LU$.
- ▶ Solution of the systems $Ly = Pb$ and $Ux = y$.

LU FACTORIZATION: GENERAL CASE

In the general case when $A \in R^{m \times n}$ of $\text{rank}(A) = r \leq \min(m, n)$, it can be shown that matrix $P_r A P_c \in R^{m \times n}$ can be factored into a product of a unit lower trapezoidal matrix $L \in R^{m \times r}$ and an upper trapezoidal matrix $U \in R^{r \times n}$. Here P_r and P_c are permutation matrices performing the necessary row and column permutations, respectively. The factorization can be written in block form as

$$P_r A P_c = LU = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} (U_{11} U_{12}),$$

where the matrices L_{11} and U_{11} are triangular and nonsingular. Note that the block L_{21} is empty if the matrix A has full row rank, i.e. $r = m$; the block U_{12} is empty if the matrix A has full column rank, i.e. $r = n$.

LU FACTORIZATION: GENERAL CASE

To solve the system

$$P_r A P_c (P_c^T x) = LU\tilde{x} = P_r b = \tilde{b}, \quad x = P_c \tilde{x},$$

Using the factorization we set $y = Ux$ and solve $\begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} y = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix}$ which determines y as the solution of $L_{11}y = \tilde{b}_1$, i.e., the system is consistent iff $L_{21}y = \tilde{b}_2$. Next $U\tilde{x} = y$, i.e.,

$$(U_{11}U_{12}) \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = y$$

For an arbitrary \tilde{x}_2 this system uniquely determines \tilde{x}_1 as the solution to the triangular system $U_{11}\tilde{x}_1 = y - U_{12}\tilde{x}_2$. Thus, if consistent the system has a unique solution only if A has full column rank.

PIVOTING

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

If $\epsilon \neq 1$ then the system has a unique solution $x_1 = -x_2 = -1/(1 - \epsilon)$. If $\epsilon = 10^{-16}$ then if we multiply row 1 by 10^{16} and subtract from the second row then $(1 - 10^{16})x_2 = -10^{16}$ and $x_2 = 1$. If we use this x_2 then $x_1 = 0$.

```

1: int main () {
2:   double x;
3:   std::cout.precision(64);
4:   x = -1.0 * pow(10.0,16.0)/(1 - pow(10.0,16.0));
5:   cout << x << endl;
6:   return 0;
7: }
```

It is important to perform row (and/or column) interchanges not only when a pivotal element is zero, but also when it is small.

PIVOTING (MARKOWITZ)

Partial pivoting. The pivot is taken as the largest element in magnitude in the unreduced part of the k th column, i.e., at the start of the k th stage choose interchange rows k and r , where r is the smallest integer for which

$$|a_{rk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|.$$

Complete pivoting. The pivot is taken as the largest element in magnitude in the whole unreduced part of the matrix, i.e., at the start of the k th stage interchange rows k and r and columns k and s , where r and s are the smallest integers for which

$$|a_{rs}^{(k)}| = \max_{k \leq i, j \leq n} |a_{ij}^{(k)}|.$$

Complete pivoting requires overhead in the complexity. Partial pivoting is a standard choice in many solvers.

Homework 8 (T_{ns}).

Handout: LU factorization with pivoting. Error analysis. SC pp 74-76.

MINIMUM DEGREE PIVOTING

- ▶ If both a_{ji} and a_{ij} are nonzeros for all i, j then A has symmetric sparsity structure.
- ▶ If A is diagonally dominant and has symmetric sparse structure then MDP (special case of Markowitz) reduces the number of fill-ins in LU.
- ▶ Matrix A can be represented as a graph $G = (V, E)$.
- ▶ Elimination algorithm for v : 1) remove v from V ; 2) remove all $uv \in E$ (incident edges); 3) create clique from all nodes that were adjacent to v
- ▶ The added edges correspond to newly created nonzeros when row v is eliminated

Main algorithm:

1. for all $v \in V$
2. choose $v \in V$ that has minimum degree
3. eliminate v

SUPERLU

SuperLU is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines. The library is written in C and is callable from either C or Fortran. The library routines will perform an LU decomposition with partial pivoting and triangular system solves through forward and back substitution. The LU factorization routines can handle non-square matrices but the triangular solves are performed only for square matrices. The matrix columns may be reordered (before factorization) either through library or user supplied routines. This reordering for sparsity is completely separate from the factorization. Working precision iterative refinement subroutines are provided for improved backward stability. Routines are also provided to equilibrate the system, estimate the condition number, calculate the relative backward error, and estimate error bounds for the refined solutions.

- ▶ <http://crd-legacy.lbl.gov/~xiaoye/SuperLU/>
- ▶ run example

COMPLEXITY

Solving k systems of linear equations for the same A by

Gaussian elimination	$k \cdot \mathcal{O}(n^3)$
LU factorization	$\mathcal{O}(n^3) + k \cdot \mathcal{O}(n^2)$
Inverse ($x = A^{-1}b$)	$\mathcal{O}(n^3) + k \cdot \mathcal{O}(n^2)$
Cramer's rule ($x_i = \frac{\det A_i}{\det A}$)	$k \cdot \mathcal{O}(n^3)$ by LU or $\mathcal{O}(n!)$ by Laplace

WHAT IF WE MODIFY A ? IN SOME CASES RE-FACTORIZATION CAN BE AVOIDED.

Example: rank-one modification, i.e., the new matrix is $A + uv^T$.

Reminder: A has rank 1 $\iff A = uv^T$.

Simple case: only entry (j, k) changes from a_{jk} to \tilde{a}_{jk} , i.e., $A \leftarrow A - \alpha e_j e_k^T$.

The Sherman-Morrison formula,

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

i.e., if we need to solve $(A - uv^T)x = b$ the S-M formula gives

$$x = (A - uv^T)^{-1}b = A^{-1}b + A^{-1}u(1 - v^T A^{-1}u)^{-1}v^T A^{-1}b.$$

Algorithm: rank-one updating of solution (requires only $\mathcal{O}(n^2)$ work)

- 1: solve $Az = u$ for z , i.e., $z = A^{-1}u$
- 2: solve $Ay = b$ for y , i.e., $y = A^{-1}b$
- 3: $x = y + ((v^T y)/(1 - v^T z))z$

For rank- k modification use Woodbury formula $(A - UV^T)^{-1} = A^{-1} + A^{-1}U(I - V^T A^{-1}U)^{-1}V^T A^{-1}$

S-M FORMULA

Homework 9 (I).

- ▶ Generate or download an invertible sparse (at least 5% sparse) square matrix A of size at least 1000×1000 .
- ▶ Generate one vector b (at least 25% sparse).
- ▶ Generate 100 random rank-1 changes.
- ▶ Solve $(A + E_i)x = b$ for all random changes by using full Gaussian Elimination or full LU scheme
- ▶ Solve $(A + E_i)x = b$ for all random changes by using S-M formula and keeping A^{-1}
- ▶ Draw graph of total running time (y-axis) for all respective iterations (x-axis). In the time of the first S-M iteration include $time(A^{-1})$
- ▶ If A is too large for your computer's performance try to decrease it's size.

IMPROVING BY ITERATIVE REFINEMENT

Suppose we computed an approximate solution x_0 by LU. The residual is

$$r_0 = b - Ax_0$$

We can use the same LU factors to compute

$$As_0 = r_0$$

and improve $x_1 = x_0 + s_0$ because

$$Ax_1 = A(x_0 + s_0) = Ax_0 + As_0 = (b - r_0) + r_0 = b.$$

Note, that computing residual may require better precision.

Section 7

SPECIAL TYPES OF LINEAR SYSTEMS

SPECIAL TYPES OF MATRICES

- ▶ Symmetric: $A = A^T$ (or conjugate if A is complex)
- ▶ Positive definite: $x^T Ax > 0$ for all $x \neq 0$ ($A \succ 0$)
- ▶ Positive semi-definite: $x^T Ax \geq 0$ for all $x \neq 0$ ($A \succeq 0$)
- ▶ Banded: $a_{ij} = 0$ for all $|i - j| > \beta$ (bandwidth of A)
- ▶ Sparse: most entries of A are 0

For these matrices one can save both space and time in comparison to dense general matrices. Examples of applications include machine learning, stochastic processes, Jacobians, Hessians, graphs, optimization, etc.

Examples of sparse matrices UFL collection

SPARSE MATRIX STORAGE

Matrices can be

- ▶ huge (tera-, peta-, exa-, zetta-, yotta-, ... bytes)
- ▶ structurally different at different resolutions
- ▶ collected in parallel, i.e., the data is mixed
- ▶ noisy, irregular, etc.

SPARSE MATRIX STORAGE

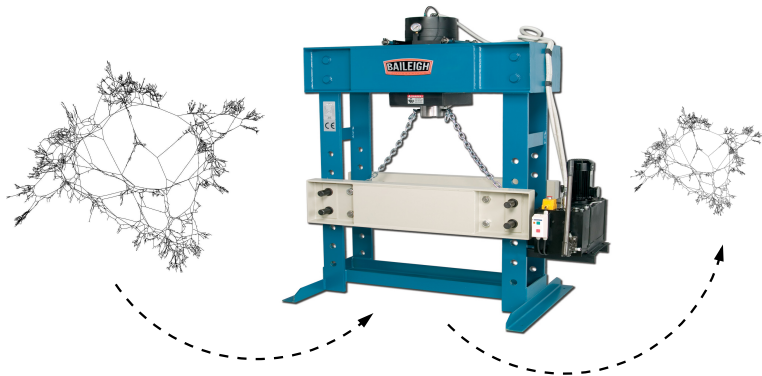
Matrices can be

- ▶ huge (tera-, peta-, exa-, zetta-, yotta-, ... bytes)
- ▶ structurally different at different resolutions
- ▶ collected in parallel, i.e., the data is mixed
- ▶ noisy, irregular, etc.

Challenges

- ▶ How to store the network efficiently?
- ▶ How to design an extremely fast access to nodes and links?
- ▶ How to minimize the number of cache misses?

Find a compressed representation of a network.



MATRIX REPRESENTATION: COMPRESSED ROW FORMAT

Rows	Sorted list of non-zero entries (with possible additional information)
1	2, 5, 6, 12, 18, 23, 103
...	...
1584	1585, 1592, 1600

Given a sorted list of non-zeros (x_1, x_2, x_3, \dots) , represent it by a list of differences $(x_1, x_2 - x_1, x_3 - x_1, \dots)$ or $(x_1, x_2 - x_1, x_3 - x_2, \dots)$.

MATRIX REPRESENTATION: COMPRESSED ROW FORMAT

Rows	Sorted list of non-zero entries (with possible additional information)
1	2, 5, 6, 12, 18, 23, 103
...	...
1584	1585, 1592, 1600

Given a sorted list of non-zeros (x_1, x_2, x_3, \dots) , represent it by a list of differences $(x_1, x_2 - x_1, x_3 - x_1, \dots)$ or $(x_1, x_2 - x_1, x_3 - x_2, \dots)$.

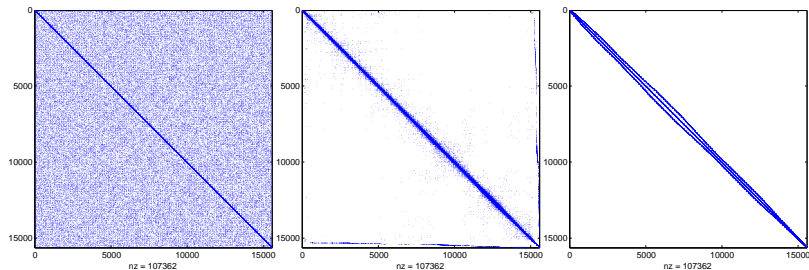
Rows	Sorted list of non-zero entries (with possible additional information)
1	1, 4, 5, 11, 17, 22, 102
...	...
1584	1, 8, 16

THE MINIMUM LOGARITHMIC ARRANGEMENT

MLogA: minimize over all possible $\pi \in s(n)$

$$\sum_{ij \in E} \lg |\pi(i) - \pi(j)|.$$

MLogA is NP-hard.



COMBINATORIAL SCIENTIFIC COMPUTING (CSC)

... is a research in an interdisciplinary field that spans scientific computing and algorithmic computer science. Research in CSC comprises three key components. The first component involves identifying a problem in scientific computing and building an appropriate combinatorial model of the problem, in order to make the computation effective and efficient. The second component involves the design, analysis, and implementation of algorithms to solve the combinatorial subproblem. The emphasis in this step is on practical algorithms that are efficient for large-scale problems; an algorithm with a time complexity quadratic in the input size could be too slow to be useful, if the worst-case behavior is realized. The algorithm could compute an exact, approximate, or heuristic solution to the problem, and it should run quickly within the context of the other computational steps in the scientific computation. The third component involves developing software.

Hendrickson, Pothen “Combinatorial Scientific Computing: The Enabling Power of Discrete Algorithms in Computational Science”

POSITIVE DEFINITE SYSTEMS

Examples of positive definite matrices:

- ▶ covariance matrix ($\Sigma_{i,j} = \text{cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$), optimization, machine learning.
- ▶ If $A = A^T$, $a_{ii} > 0$, and $a_{ii} > \sum_{j \neq i} |a_{ij}|$ then A is pd
- ▶ If for $x \neq 0$ $Ax \neq 0$ then $A^T A$ spd ($x^T A^T Ax = (x^T A^T)(Ax) > 0$).

SYMMETRIC POSITIVE DEFINITE SYSTEMS

Let A be any $n \times n$ matrix.

- ▶ A principal submatrix of A is any $m \times m$ submatrix obtained by deleting $n - m$ rows and corresponding columns.
- ▶ A leading principal submatrix of A of order m is obtained by deleting the last $n - m$ rows and columns.

SYMMETRIC POSITIVE DEFINITE SYSTEMS

Let A be any $n \times n$ matrix.

- ▶ A principal submatrix of A is any $m \times m$ submatrix obtained by deleting $n - m$ rows and corresponding columns.
- ▶ A leading principal submatrix of A of order m is obtained by deleting the last $n - m$ rows and columns.

Theorems

- ▶ If A is SPD then every principal submatrix is SPD.
- ▶ Diagonal entries of SPD are positive.
- ▶ **Cholesky factorization.** If A is SPD then LU factorization can be arranged as $A = LL^T$, i.e., $U = L^T$. Matrix A can also be factorized as LDL^T where L is unit lower triangular, and D is diagonal matrix.

2x2 EXAMPLE OF CHOLESKY FACTORIZATION

$$\begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \cdot \begin{pmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{pmatrix}$$

$$l_{11} = \sqrt{a_{11}}, \quad l_{21} = a_{21}/l_{11}, \quad l_{22} = \sqrt{a_{22} - l_{21}^2}$$

SYMMETRIC POSITIVE DEFINITE SYSTEMS

```

for  $j = 1$  to  $n$ 
    { for each column  $j$  }
    for  $k = 1$  to  $j - 1$ 
        { loop over all prior columns  $k$  }
        for  $i = j$  to  $n$ 
            { subtract a multiple of
              column  $k$  from column  $j$  }
             $a_{ij} = a_{ij} - a_{ik} \cdot a_{jk}$ 
        end
    end
     $a_{jj} = \sqrt{a_{jj}}$ 
    for  $k = j + 1$  to  $n$ 
        { scale column  $j$  by square
          root of diagonal entry }
         $a_{kj} = a_{kj} / a_{jj}$ 
    end
end
end

```

A number of facts about the Cholesky factorization algorithm make it very attractive and popular for symmetric positive definite matrices:

- The n square roots required are all of positive numbers, so the algorithm is well-defined.
- No pivoting is required for numerical stability.
- Only the lower triangle of \mathbf{A} is accessed, and hence the upper triangular portion need not be stored.
- Only about $n^3/6$ multiplications and a similar number of additions are required.

Homework 10 (I).

Find SPD matrix at the UFL collection. Implement or run existing Cholesky factorization on it.

Section 8

LINEAR LEAST SQUARES

DEFINITIONS

If $A \in \mathcal{F}^{m \times n}$ is not square then $Ax = b$ has usually either no solution or many solutions.

- ▶ if $m > n$ then A is overdetermined (i.e., more equations than unknowns)
- ▶ if $m < n$ then A is underdetermined

Data fitting is one of the most common sources of overdetermined systems. Instead of looking for an exact solution we will minimize some norm of the residual

$$r = b - Ax, \text{ i.e., we seek } x \text{ such that } Ax \approx b.$$

Example.

One need to predict price for stocks X, Y, and Z. Fundamental analysis recommends $X = 10$, $Y = 20$, $Z = 30$. In previous years we found a linear dependences $X + 8 = Y$, and $X + 23 = Z$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \approx \begin{pmatrix} 10 \\ 20 \\ 30 \\ -8 \\ -23 \end{pmatrix}$$

$$\Rightarrow X = 9.75, Y = 18.875, Z = 31.375$$

DATA FITTING

Given m data points (t_i, y_i) we need to find the best model function $f(t, x) : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ in the lsq sense, where $x = (x_1, \dots, x_n)$ are the parameters of f

$$\min_x \sum_{i=1}^m (y_i - f(t_i, x))^2$$

The fitting function f is linear if it is linear in x

$$f(t, x) = x_1\phi_1(t) + \dots + x_n\phi_n(t).$$

Example of linear f : $f(t, x) = x_1t^0 + x_2t^1 + \dots + x_nt^{n-1}$.

Example of nonlinear f : $f(t, x) = x_1^1t^0 + x_2^2t^1 + \dots + x_n^nt^{n-1}$.

If f is linear we can set A with $a_{ij} = \phi_j(t_i)$, b with $b_i = y_i$ and linear least squares problems will be

$$Ax \approx b$$

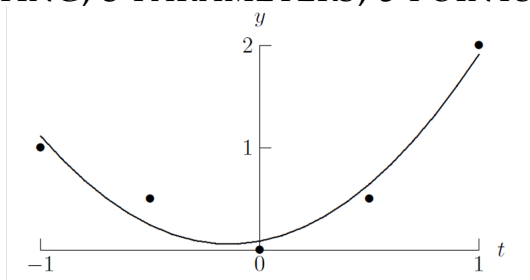
VANDERMONDE MATRIX

Vandermonde systems have several attractive properties. Applications in FFT, interpolation, approximation, etc. Square systems of equations can be solved in $\mathcal{O}(n \log n)$ flops. Determinant of square system is $\prod_{i < j} (\alpha_j - \alpha_i)$.

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \\ 1 & t_4 & t_4^2 \\ 1 & t_5 & t_5^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \mathbf{b}.$$

A matrix \mathbf{A} of this particular form, whose columns (or rows) are successive powers of some independent variable, is called a *Vandermonde matrix*.

DATA FITTING, 3 PARAMETERS, 5 POINTS



t	-1.0	-0.5	0.0	0.5	1.0
y	1.0	0.5	0.0	0.5	2.0

$$\mathbf{A} \mathbf{x} = \begin{bmatrix} 1 & -1.0 & 1.0 \\ 1 & -0.5 & 0.25 \\ 1 & 0.0 & 0.0 \\ 1 & 0.5 & 0.25 \\ 1 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \approx \begin{bmatrix} 1.0 \\ 0.5 \\ 0.0 \\ 0.5 \\ 2.0 \end{bmatrix} = \mathbf{b}$$

From M. Heath "Intro to SC"

$$p(t) = 0.086 + 0.4t + 1.4t^2$$

NORMAL EQUATIONS

We can solve LSQ by minimizing 2nd norm of the residual

$$\phi(x) = \|r\|_2^2 = r^T r = (b - Ax)^T (b - Ax) = b^T b - 2x^T A^T b + x^T A^T A x$$

Necessary condition for minimizing $\phi(x)$ is $\nabla\phi(x) = 0$, where

$$\nabla\phi(x) = 2A^T A x - 2A^T b$$

Sufficient condition for minimizer x is that the Hessian ($2A^T A$) is pod.

Theorem 2.

$A^T A$ is pod iff columns of A are linearly independent, i.e., $\text{rank}(A) = n$.

Example.

One need to predict price for stocks X, Y, and Z. Fundamental analysis recommends $X = 10$, $Y = 20$, $Z = 30$. In previous years we found a linear dependences $X + 8 = Y$, and $X + 23 = Z$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \approx \begin{pmatrix} 10 \\ 20 \\ 30 \\ -8 \\ -23 \end{pmatrix}$$

$$A^T A x = \begin{pmatrix} 3 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 2 \end{pmatrix} x = \begin{pmatrix} -21 \\ 28 \\ 53 \end{pmatrix} = A^T b$$

$$\Rightarrow X = 9.75, Y = 18.875, Z = 31.375$$

$$\Rightarrow \|r\|_2^2 = 6.375$$

Example.

One need to predict price for stocks X , Y , and Z . Fundamental analysis recommends $X = 10$, $Y = 20$, $Z = 30$. In previous years we found a linear dependences $X + 8 = Y$, and $X + 23 = Z$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \approx \begin{pmatrix} 10 \\ 20 \\ 30 \\ -8 \\ -23 \end{pmatrix}$$

$$A^T A x = \begin{pmatrix} 3 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 2 \end{pmatrix} x = \begin{pmatrix} -21 \\ 28 \\ 53 \end{pmatrix} = A^T b$$

$$\Rightarrow X = 9.75, Y = 18.875, Z = 31.375$$

$$\Rightarrow \|r\|_2^2 = 6.375$$

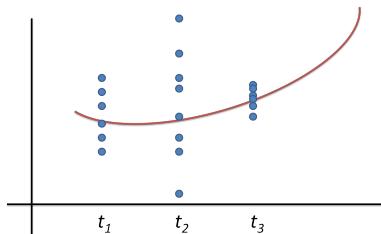
Homework 11.

Find overdetermined problem at UFL collection, generate random right hand side b , and solve LSQ problem with normal equations.

WEIGHTED LINEAR LEAST SQUARES

Given m data points (t_i, y_i) we need to find the best model function $f(t, x) : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ in the weighted lsq sense, where $x = (x_1, \dots, x_n)$ are the parameters of f

$$\min_x \sum_{i=1}^m w_i (y_i - f(t_i, x))^2$$



- ▶ advantages: different interpretations of intervals for estimation, prediction, calibration and optimization, factors of importance
- ▶ disadvantage: requires knowledge about w_i , almost never the case in real applications

Subsection 1

QR

ORTHOGONALITY AND PROJECTIONS

Recall $\cos(\Theta) = \frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|} = \frac{\sum_i (v_1)_i (v_2)_i}{\sqrt{\sum_i (v_1)_i^2} \sqrt{\sum_i (v_2)_i^2}}$. Vectors v_1 and v_2 are orthogonal if $v_1^T v_2 = 0$.

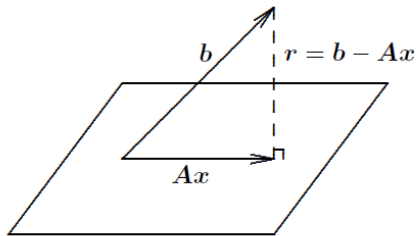


Figure 3.2: Geometric depiction of a linear least squares problem.

For LSQ problem $Ax \approx b$ for $m > n$ usually $b \notin \text{span}(A)$, so $r = Ax - b$ is orthogonal to each column of A , i.e., $0 = A^T r = A^T (b - Ax)$ or $A^T Ax = A^T b$.

PROJECTORS

Definition.

Matrix P is called projector (or idempotent) if $P^2 = P$. If $P = P^T$ it is called orthogonal projector.

In general, for any projector P , any $v \in \text{range}(P)$ is projected onto itself, i.e., $v = Px$ for some x then $Pv = P(Px) = P^2x = Px = v$.

Example.

$P = \begin{pmatrix} c^2 & cs \\ cs & s^2 \end{pmatrix}$, where $c = \cos(\Theta)$, and $s = \sin(\Theta)$.

$$\begin{pmatrix} c^2 & cs \\ cs & s^2 \end{pmatrix} \cdot \begin{pmatrix} c^2 & cs \\ cs & s^2 \end{pmatrix} = \begin{pmatrix} c^2(c^2 + s^2) & cs(c^2 + s^2) \\ cs(c^2 + s^2) & s^2(c^2 + s^2) \end{pmatrix}$$

PROJECTORS

Definition.

Matrix P is called projector (or idempotent) if $P^2 = P$. If $P = P^T$ it is called orthogonal projector.

In general, for any projector P , any $v \in \text{range}(P)$ is projected onto itself, i.e., $v = Px$ for some x then $Pv = P(Px) = P^2x = Px = v$.

Example.

$P = \begin{pmatrix} c^2 & cs \\ cs & s^2 \end{pmatrix}$, where $c = \cos(\Theta)$, and $s = \sin(\Theta)$.

$$\begin{pmatrix} c^2 & cs \\ cs & s^2 \end{pmatrix} \cdot \begin{pmatrix} c^2 & cs \\ cs & s^2 \end{pmatrix} = \begin{pmatrix} c^2(c^2 + s^2) & cs(c^2 + s^2) \\ cs(c^2 + s^2) & s^2(c^2 + s^2) \end{pmatrix}$$

Matrix $I - P$ is called complementary projector, namely,

$$(I - P)^2 = (I - P)(I - P) = I - IP - PI + P^2 = I - P.$$

If P is an orthogonal projector then $P_{\perp} = I - P$ is an orthogonal projector onto $\text{span}(P)^{\perp}$

Any $v \in \mathbb{R}^m$ can be expressed as $v = (P + (I - P))v = Pv + P_{\perp}v$.

Any $v \in \mathbb{R}^m$ can be expressed as $v = (P + (I - P))v = Pv + P_{\perp}v$.

Let P be an orthogonal projector onto $\mathbf{span}(A)$ then

$$\begin{aligned}\|b - Ax\|_2^2 &= \|P(b - Ax) + P_{\perp}(b - Ax)\|_2^2 \\ &= \|P(b - Ax)\|_2^2 + \|P_{\perp}(b - Ax)\|_2^2 \\ &= \|Pb - Ax\|_2^2 + \|P_{\perp}b\|_2^2 \quad (\text{since } PA = A, P_{\perp}A = 0),\end{aligned}$$

i.e., the LSQ solution is given by ...

Any $v \in \mathbb{R}^m$ can be expressed as $v = (P + (I - P))v = Pv + P_{\perp}v$.
 Let P be an orthogonal projector onto $\mathbf{span}(A)$ then

$$\begin{aligned} \|b - Ax\|_2^2 &= \|P(b - Ax) + P_{\perp}(b - Ax)\|_2^2 \\ &= \|P(b - Ax)\|_2^2 + \|P_{\perp}(b - Ax)\|_2^2 \\ &= \|Pb - Ax\|_2^2 + \|P_{\perp}b\|_2^2 \quad (\text{since } PA = A, P_{\perp}A = 0), \end{aligned}$$

i.e., the LSQ solution is given by ... $Ax = Pb$ because we need to minimize only first term (no x in the second one). Multiplying both sides by A^T gives

$$A^T Ax = A^T b$$

because $A^T P = A^T P^T = (PA)^T = A^T$.

One way to obtain P (when $A^T A$ is nonsingular, i.e., A has full column rank) is

$$P = A(A^T A)^{-1} A^T, \text{ i.e., } y = Pb = A(A^T A)^{-1} b = Ax$$

Example: Matlab `lsq_example.m`

SENSITIVITY AND CONDITIONING

Definition.

If A has full column rank, i.e., $A^T A$ is nonsingular then its pseudoinverse is given by

$$A^+ = (A^T A)^{-1} A^T, \text{ i.e., } A^+ A = I.$$

SENSITIVITY AND CONDITIONING

Definition.

If A has full column rank, i.e., $A^T A$ is nonsingular then its pseudoinverse is given by

$$A^+ = (A^T A)^{-1} A^T, \text{ i.e., } A^+ A = I.$$

Definition.

The condition number of $m \times n$ matrix with $\text{rank}(A) = n$ is

$$\mathbf{cond}(A) = \|A\|_2 \cdot \|A^+\|_2.$$

If $\text{rank}(A) < n$ then $\mathbf{cond}(A) = \infty$. The condition number of nonsquare matrix measures closeness to rank deficiency.

- ▶ If b lies near $\mathbf{span}(A)$ then a small perturbation in b changes $y = Pb$ relatively little.
- ▶ If b is almost orthogonal to $\mathbf{span}(A)$ then $y = Pb$ will be small and any change in b can cause large perturbation in y .
 \Rightarrow large residual can be a reason of more sensitive system.

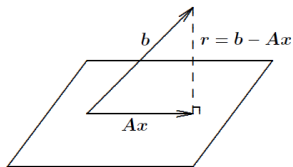


Figure 3.2: Geometric depiction of a linear least squares problem.

- ▶ If b lies near $\mathbf{span}(A)$ then a small perturbation in b changes $y = Pb$ relatively little.
- ▶ If b is almost orthogonal to $\mathbf{span}(A)$ then $y = Pb$ will be small and any change in b can cause large perturbation in y .
 \Rightarrow large residual can be a reason of more sensitive system.

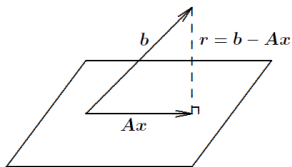


Figure 3.2: Geometric depiction of a linear least squares problem.

We measure closeness of b to $\mathbf{span}(A)$ by ratio $\frac{\|Ax\|_2}{\|b\|_2} = \cos(\Theta)$.

Example.

One need to predict price for stocks X , Y , and Z . Fundamental analysis recommends $X = 10$, $Y = 20$, $Z = 30$. In previous years we found linear dependencies $X + 8 = Y$, and $X + 23 = Z$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \approx \begin{pmatrix} 10 \\ 20 \\ 30 \\ -8 \\ -23 \end{pmatrix}$$

Example: Matlab pseudoinverse `lsq_example.m`

LSQ: EXAMPLE OF INACCURATE RESULTS

Given $A = \begin{pmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{pmatrix}$, where $\epsilon < \sqrt{\epsilon_{mach}}$. When we compute A^+ we need to calculate

$$A^T A = \begin{pmatrix} 1 + \epsilon^2 & 1 \\ 1 & 1 + \epsilon^2 \end{pmatrix}.$$

which is in f-p arithmetic singular because $A^T A = \mathbf{fl}(A^T A) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

AUGMENTED SYSTEMS

Based on the definition of r we need to find x s.t.

$$\begin{aligned}r + Ax &= b \\ A^T r &= 0\end{aligned}$$

which can be written and solved as the augmented systems

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} \alpha I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r/\alpha \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

α is a scaling parameter which brings the gap between r and A .

TWO METHODS FOR NORMAL EQUATIONS

Normal Equations $A^T A x = A^T b$

- ▶ solve NE with Cholesky factorization (faster)
- ▶ QR factorization (more stable numerically)

Recall complexity of Cholesky:

- ▶ $C = A^T A$ (mn^2 flops)
- ▶ Cholesky $C = LL^T$ ($\frac{1}{3}n^3$ flops)
- ▶ $y = A^T b$ (mn flops)
- ▶ solve $Lz = y$ (n^2 flops)
- ▶ solve $L^T x = z$ (n^2 flops)

Complexity = $mn^2 + \frac{1}{3}n^3$ flops

QR FACTORIZATION

Given $A \in \mathbb{R}^{m \times n}$ left-invertible \Rightarrow it can be factored $A = QR$, where $R \in \mathbb{R}^{n \times n}$ is upper triangular with $r_{ii} > 0$, and $Q \in \mathbb{R}^{m \times n}$ is orthogonal ($Q^T Q = I$).

Complexity: $\mathcal{O}(mn^2)$

$$A^T A x = A^T b$$

$$R^T Q^T Q R x = R^T Q^T b$$

$$R^T R x = R^T Q^T b \quad (Q \text{ is orthogonal } \Rightarrow Q^T Q = I)$$

$$R x = Q^T b \quad (R \text{ is nonsingular})$$

1. QR factorization: $A = QR$ ($\mathcal{O}(mn^2)$ flops)
2. form $d = Q^T b$ ($\mathcal{O}(mn)$ flops)
3. solve $Rx = d$ ($\mathcal{O}(n^2)$ flops)

Homework 12 (R).

Read about Gram-Schmidt orthogonalization (chapters 3.4.5 and 3.5.3)

Homework 13 (I).

Find large-scale overdetermined problem at the UFL collection, generate random right hand side b , and solve LSQ problem with QR and NE. Check what method is faster.

Subsection 2

SVD

SINGULAR VALUE DECOMPOSITION

Definition.

The singular value decomposition of $A \in \mathbb{R}^{m \times n}$ has the form

$$A = U\Sigma V^T,$$

where $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices (their columns are called left and right singular vectors), and $\Sigma \in \mathbb{R}^{m \times n}$ is diagonal with $\sigma_{ii} \geq 0$ (called singular values and ordered in descending order).

SVD: EXAMPLE

Example 4.16 Singular Value Decomposition. The singular value decomposition of

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

is given by $U\Sigma V^T =$

$$\begin{bmatrix} 0.141 & 0.825 & -0.420 & -0.351 \\ 0.344 & 0.426 & 0.298 & 0.782 \\ 0.547 & 0.028 & 0.664 & -0.509 \\ 0.750 & -0.371 & -0.542 & 0.079 \end{bmatrix} \begin{bmatrix} 25.5 & 0 & 0 \\ 0 & 1.29 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.504 & 0.574 & 0.644 \\ -0.761 & -0.057 & 0.646 \\ 0.408 & -0.816 & 0.408 \end{bmatrix}.$$

Thus, we have $\sigma_1 = 25.5$, $\sigma_2 = 1.29$, and $\sigma_3 = 0$. A singular value of zero indicates that the matrix is rank-deficient; in general, the rank of a matrix is equal to the number of nonzero singular values, which in this example is two.

COMPRESSED REPRESENTATION

Given $A \in \mathbb{R}^{m \times n}$ with $\mathbf{rank}(A) = n$

$$A = U\Sigma V^T = [U_1 U_2] \begin{bmatrix} \Sigma_1 \\ 0 \end{bmatrix} V^T = U_1 \Sigma_1 V^T,$$

the solution of LSQ problem $Ax \approx b$ is given by

$$Ax = U_1 \Sigma_1 V^T x = b \Rightarrow x = V \Sigma_1^{-1} U_1^T b.$$

COMPRESSED REPRESENTATION

Given $A \in \mathbb{R}^{m \times n}$ with $\mathbf{rank}(A) = n$

$$A = U\Sigma V^T = [U_1 U_2] \begin{bmatrix} \Sigma_1 \\ 0 \end{bmatrix} V^T = U_1 \Sigma_1 V^T,$$

the solution of LSQ problem $Ax \approx b$ is given by

$$Ax = U_1 \Sigma_1 V^T x = b \Rightarrow x = V \Sigma_1^{-1} U_1^T b.$$

For A of any shape and rank the solution of LSQ problem is given by

$$x = \sum_{\sigma_i \neq 0} \frac{u_i^T b}{\sigma_i} v_i \quad (5)$$

SVD is very useful for ill-conditioned, (nearly) rank-deficient systems because small σ_i can be dropped from (5).

SVD: CONNECTIONS TO $\|\cdot\|$ AND **cond()**

SVD has many useful properties such as

- ▶ $\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sigma_{\max}$
- ▶ $\mathbf{cond}(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$
- ▶ $\mathbf{rank}(A) = |\{\sigma_i \neq 0\}_i|$. Example of threshold/precision problem with $\mathbf{rank}(A) = 2 \Rightarrow \mathbf{rank}(A) = 1$.

$$\begin{pmatrix} 0.913 & 0.659 \\ 0.780 & 0.564 \\ 0.457 & 0.330 \end{pmatrix} = \begin{pmatrix} -0.71042 & 0.60854 & -0.35353 \\ -0.60730 & -0.78393 & -0.12902 \\ -0.35565 & 0.12304 & 0.92648 \end{pmatrix} \cdot \begin{pmatrix} 1.5850 & 0.0000 \\ 0.0000 & 0.00062107 \\ 0.0000 & 0.0000 \end{pmatrix} \cdot \begin{pmatrix} -0.81065 & -0.58554 \\ 0.58554 & -0.81065 \end{pmatrix}$$

- ▶ Pseudoinverse $A^+ = V\Sigma^+U^T$, where $1/\sigma_i = 0$ if $\sigma_i = 0$

UNDERSTANDING SVD $A = U\Sigma V^T$

- ▶ The columns of U corresponding to $\sigma_i \neq 0$ form orthonormal basis for $\text{span}(A)$
- ▶ The columns of U corresponding to $\sigma_i = 0$ form orthogonal basis for $\text{span}(A)^\perp$
- ▶ The columns of V corresponding to $\sigma_i = (\neq)0$ form orthogonal basis for $\text{null}(A)$ ($\text{null}(A)^\perp$)

Low-rank approximation: $A = U\Sigma V^T = \sigma_1 E_1 + \sigma_2 E_2 + \dots + \sigma_n E_n$, where

Theorem 3 (Eckart-Young).

Given SVD $A = U\Sigma V^T = \sum_{i=1}^n \sigma_i E_i$ with $\text{rank}(A) = r \leq p = \min(m, n)$ and

$$A_k = \sum_{i=1}^k \sigma_i E_i \text{ then } \min_{\text{rank}(B)=k} \|A - B\|_F = \|A - A_k\|_F = \sigma_{k+1}^2 + \dots + \sigma_p^2.$$

In other words, A_k is the optimal approximation in terms of the approximation error measured by the F-norm, among all matrices of rank k .

SVD: LATENT-SEMANTIC INDEXING

	d1	d2	d3	d4	d5	d6	d7	d8
t1 car	1	3	1	1	0	0	0	0
t2 truck	3	1	0	1	0	0	0	0
t3 speed	0	2	6	1	0	0	0	0
t4 price	1	2	3	0	0	0	0	0
t5 insurance	4	1	1	1	2	0	0	0
t6 accident	0	0	0	1	1	3	1	2
t7 pain	0	0	0	0	1	0	1	1
t8 head	0	0	0	0	3	1	3	1
t9 doctor	0	0	0	0	1	1	0	2
t10 drug	0	0	0	0	4	4	1	1

Matlab example: svdsample.m

SVD EXAMPLES

Image compression

- ▶ ImageCompressionSVD.cdf
- ▶ ColorImageCompressionHOSVD.cdf
- ▶ Recommendation Systems



Section 9

EIGENVALUES

EFFECTS OF TRANSFORMATION, EIGENPROBLEMS

We study several complex effects of transformation on vectors

- ▶ rotation
- ▶ permutation
- ▶ reflection
- ▶ scalar multiplication

Some of the simple effects can serve as approaches to determine stability of numerical methods, upper and lower bounds, convergence, etc.

Definition.

Given $A \in \mathcal{F}^{n \times n}$ matrix. If $Ax = \lambda x$ then $x \in \mathcal{F}^n$ and $\lambda \in \mathcal{F}$ are called eigenvector and eigenvalue of A , respectively. All eigenvalues of A denoted by $\lambda(A)$ are called spectrum of A and $\rho(A) = \max\{|\lambda| : \lambda \in \lambda(A)\}$ is called spectral radius.

More precisely x is a right eigenvector. We can define left eigenvector with $y^T A = \lambda y^T$. Note that left evec of $A =$ right evec of A^T .

EIGENPROBLEMS

Eigenvalues/eigenvectors determine

- ▶ direction of only expanding/shrinking the vectors
- ▶ expansion factor
- ▶ behavior of linear transformation by decomposing it into simpler components

Example: [evalsdemo.m](#), [Wiki](#)

Example.

Newton's second law: the sum of the external forces acting on an object at any instant in time is equal to the product of the object's mass and acceleration (*). We consider the system of springs with stiffness coefficients k_i and masses m_i . According to (*) $My'' + Ky = 0$, where

$$M = \text{diag}(m_1, m_2, m_3), \text{ and } K = \begin{pmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{pmatrix}$$

Harmonic motion of the system is given by $y_k(t) = x_k e^{i\omega t}$, where ω is frequency, and x_k is amplitude. Since $y_k''(t) = -\omega^2 x_k e^{i\omega t}$ we obtain

$$Kx = \omega^2 Mx \text{ or } Ax = \lambda x \text{ with } A = M^{-1}K, \lambda = \omega^2 \Rightarrow$$

frequency and amplitude are determined by solving eigenproblem.

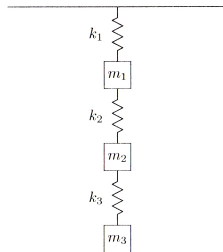


Figure 4.1: Spring-mass system.

BASIC FACTS ABOUT EVECS/EVALS

- ▶ $Ax = \lambda x$ is equivalent to $(A - \lambda I)x = 0$ which has a nonzero solution iff $A - \lambda I$ is singular, i.e., when $p(\lambda) = \det(A - \lambda I) = 0$. $p(\lambda)$ is called the characteristic polynomial of A and its roots are evals.
- ▶ According to the Fundamental Theorem of Algebra (root existence thm) $p(\lambda) = \alpha \prod_{i=1}^n (\lambda - \lambda_i) \Rightarrow A$ always has n eigenvalues but some of the can be complex and may not be distinct.

BASIC FACTS ABOUT EVECS/EVALS

- ▶ $Ax = \lambda x$ is equivalent to $(A - \lambda I)x = 0$ which has a nonzero solution iff $A - \lambda I$ is singular, i.e., when $p(\lambda) = \det(A - \lambda I) = 0$. $p(\lambda)$ is called the characteristic polynomial of A and its roots are evals.
- ▶ According to the Fundamental Theorem of Algebra (root existence thm) $p(\lambda) = \alpha \prod_{i=1}^n (\lambda - \lambda_i) \Rightarrow A$ always has n eigenvalues but some of the can be complex and may not be distinct.
- ▶ Abel's Impossibility Theorem. In general, polynomial equations higher than 4th degree are incapable of algebraic solution in terms of a finite number of arithmetic operations, and root extractions \Rightarrow we will compute evals by iterative algorithms.

Niels H. Abel 1802-1829



MORE DEFINITIONS

- ▶ Algebraic multiplicity (AM) of λ is its multiplicity as a root of char. poly.
- ▶ Geometric multiplicity (GM) of λ is the number of its corresponding eigenvectors.
- ▶ $GM \leq AM$

MORE DEFINITIONS

- ▶ Algebraic multiplicity (AM) of λ is its multiplicity as a root of char. poly.
- ▶ Geometric multiplicity (GM) of λ is the number of its corresponding eigenvectors.
- ▶ $GM \leq AM$
- ▶ If A has n independent evecs x_1, \dots, x_n corresponding to $\lambda_1, \dots, \lambda_n$ then if $D = \text{diag}(x_1, \dots, x_n)$ and $X = [x_1 \dots x_n]$ then X is nonsingular and

$$AX = XD \Rightarrow X^{-1}AX = D$$

and A is called diagonalizable.

MORE DEFINITIONS

- ▶ Algebraic multiplicity (AM) of λ is its multiplicity as a root of char. poly.
- ▶ Geometric multiplicity (GM) of λ is the number of its corresponding eigenvectors.
- ▶ $GM \leq AM$
- ▶ If A has n independent evecs x_1, \dots, x_n corresponding to $\lambda_1, \dots, \lambda_n$ then if $D = \text{diag}(x_1, \dots, x_n)$ and $X = [x_1 \dots x_n]$ then X is nonsingular and

$$AX = XD \Rightarrow X^{-1}AX = D$$

and A is called diagonalizable.

- ▶ Scaling: $Ax = \lambda x \Rightarrow \gamma x$ is also an eigenvector of A .
- ▶ $S_\lambda = \{x | Ax = \lambda x\}$ is called eigenspace corresponding to λ .

LOCALIZING EIGENVALUES

- ▶ $|\lambda| \leq \|A\|$ for any vector induced matrix norm.

LOCALIZING EIGENVALUES

- ▶ $|\lambda| \leq \|A\|$ for any vector induced matrix norm.
- ▶ More general localization is given by Gershgorin's Theorem:

Theorem 4.

All eigenvalues of $n \times n$ matrix A are contained within the union of n disks, with the k th disk centered at a_{kk} and having radius $\sum_{j \neq k} |a_{kj}|$.

Matlab demo: gershgorin.m

LOCALIZING EIGENVALUES

- ▶ $|\lambda| \leq \|A\|$ for any vector induced matrix norm.
- ▶ More general localization is given by Gershgorin's Theorem:

Theorem 4.

All eigenvalues of $n \times n$ matrix A are contained within the union of n disks, with the k th disk centered at a_{kk} and having radius $\sum_{j \neq k} |a_{kj}|$.

Matlab demo: gershgorin.m

Intuition: Let λ be eval with its normalized vec $\|x\|_\infty = 1$. If x_k is an entry of x s.t. $|x_k| = 1$ then from $Ax = \lambda x$ follows that

$$(\lambda - a_{kk})x_l = \sum_{j \neq l} a_{kj}x_j \Rightarrow |\lambda - a_{kk}| \leq \sum_{j \neq k} |a_{kj}| \cdot |x_j| \leq \sum_{j \neq k} |a_{kj}|$$

- ▶ Example of conclusion: what happens with strictly diagonally dominant matrices? Nonsingular! (Levy-Desplanques thm)

SENSITIVITY AND CONDITIONING OF EIGENPROBLEMS

We need to quantify the sensitivity of eigenvalues to small changes in matrix.

Given A with n linearly independent evecs $X = [x_1, \dots, x_n]$ (X is nonsingular), i.e., A is diagonalizable

$$X^{-1}AX = D = \text{diag}(\lambda_1, \dots, \lambda_n)$$

We introduce a perturbation $A + E$, and $F = X^{-1}EX$ then

$$X^{-1}(A + E)X = X^{-1}AX + X^{-1}EX = D + F,$$

so $D + F$ and $A + E$ have the same evals. If μ is one of these evals, i.e.,

$$(D + F)v = \mu v = Dv + Fv \Rightarrow (\mu I - D)v = Fv \Rightarrow v = (\mu I - D)^{-1}Fv$$

Taking norms and dividing by $\|v\|_2$ we have $\|(\mu I - D)^{-1}\|_2^{-1} \leq \|F\|_2$
 $\Rightarrow \dots \Rightarrow |\mu - \lambda_k| \leq \mathbf{cond}_2(X)\|E\|_2$, where λ_k is eval of D closest to μ

SENSITIVITY AND CONDITIONING OF EIGENPROBLEMS

$$|\mu - \lambda_k| \leq \mathbf{cond}_2(X) \|E\|_2$$

Conclusion: the eigenvalues are sensitive if the eigenvectors are nearly linearly dependent.

SENSITIVITY AND CONDITIONING OF EIGENPROBLEMS

$$|\mu - \lambda_k| \leq \mathbf{cond}_2(X) \|E\|_2$$

Conclusion: the eigenvalues are sensitive if the eigenvectors are nearly linearly dependent.

Since this bound depends on all of the evects it can overestimate the sensitivity of a particular eval. Let x , and y be right and left evects of a simple eval λ .

Consider a perturbed matrix $A + E$ eigen-problem

$$(A + E)(x + \Delta x) = (\lambda + \Delta\lambda)(x + \Delta x) \xrightarrow{\text{remove small terms}} A\Delta x + Ex \approx \Delta\lambda x + \lambda\Delta x$$

$$\xrightarrow{y^H \cdot (\cdot)} y^H A \Delta x + y^H E x \approx \Delta\lambda y^H x + \lambda y^H \Delta x \xrightarrow{y^H A = \lambda y^H} y^H E x \approx \Delta\lambda y^H x$$

$$\Rightarrow \Delta\lambda \approx \frac{y^H E x}{y^H x} \Rightarrow |\Delta\lambda| \leq \frac{\|y\|_2 \|x\|_2}{|y^H x|} \|E\|_2 = \frac{1}{\cos(\Theta)} \|E\|_2,$$

Θ is an angle between x and y . Example: evals of Hermitian matrices are always well-conditioned because left and right evects are the same, i.e.,

$$\cos(\Theta) = 1.$$

EXAMPLE FROM SC BOOK

$$A = \begin{pmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{pmatrix}, \text{spec}(A) = \{1, 2, 3\}$$

A has different eigenvalues, i.e., A is diagonalizable but not normal
 $A^T A \neq A A^T$.

$$X = \begin{pmatrix} 0.31623 & -0.40406 & -0.13914 \\ -0.94868 & 0.90914 & 0.97398 \\ 0.0000 & 0.10102 & -0.17889 \end{pmatrix}, Y = \begin{pmatrix} 0.68103 & -0.67627 & -0.68825 \\ 0.22526 & -0.22542 & -0.22942 \\ 0.69675 & -0.70132 & -0.68825 \end{pmatrix}$$

$$\text{cond}(X) = 1.2889e + 03$$

$x(:, 1)^T \cdot y(:, 1) = 0.0017$, $x(:, 2)^T \cdot y(:, 2) = -0.0025$, $x(:, 3)^T \cdot y(:, 3) = -0.0046$, i.e., right evects are ill-conditioned, and right and left evects are almost orthogonal.

If we change $a_{22} = 180.01$ then $\text{spec}(A) = \{0.2073, 3.5019, 2.3008\}$.

PROBLEM TRANSFORMATION

What happens with evals if we ...

- ▶ Shift. If $Ax = \lambda x$ then $(A - \sigma I)x = (\lambda - \sigma)x$.
- ▶ Inversion. If A is nonsingular, and $Ax = \lambda x$ then $A^{-1}x = \frac{1}{\lambda}x$.
- ▶ Powers. If $Ax = \lambda x$ then $A^k x = \lambda^k x$. Also if A is diagonalizable $A = X^T D X$ then $A^k = X^T D^k X$.
- ▶ Polynomials. Let $p(A) = \sum_{i=0}^k c_i A^i$, and $Ax = \lambda x$ then $p(A)x = p(\lambda)x$.

What is the interpretation of powers of A ?

- ▶ B is *similar* to A if there is nonsingular T s.t. $B = T^{-1}AT$ then

$$By = \lambda y \Rightarrow T^{-1}ATy = \lambda y \Rightarrow ATy = \lambda Ty,$$

i.e., A and B have the same eigenvalues and A has evects $x = Ty$.

SCHUR DECOMPOSITION

Not all matrices are diagonalizable with similarity transformation. (We can find Jordan decomposition but it is not very useful.) If $A = (a_{ij}) \in \mathbb{C}^{m \times n}$ the adjoint of A is denoted by $A^H = (\bar{a}_{ji})$. A square matrix A is called unitary if $A^H A = I$.

Theorem 5 (The Schur Normal Form).

Given $A \in \mathbb{C}^{n \times n}$ there exists a unitary matrix $U \in \mathbb{C}^{n \times n}$ s.t.

$$U^H A U = T = D + N,$$

where T is upper triangular, N is strictly upper triangular, $D = \text{diag}(\lambda_1, \dots, \lambda_n)$ with λ_i are evals of A . U can be chosen so that the evals appear in arbitrary order in D .

The advantage of the Schur normal form is that it can be obtained using a numerically stable unitary transformation. The columns of U are Schur vectors but they are not evecs.

SCHUR DECOMPOSITION

Theorem 6 (The Schur Real Form).

Given $A \in \mathbb{R}^{n \times n}$ there exists a real orthogonal matrix $Q \in \mathbb{R}^{n \times n}$ s.t.

$$Q^T A Q = T = D + N,$$

where T is real block upper triangular, D is block diagonal with 1×1 and 2×2 blocks, and where all the 2×2 blocks have complex conjugate evals.

Normal matrix A ($A^H A = A A^H$) can be unitarily diagonalized, i.e., $U^H A U = D = \text{diag}(\lambda_1, \dots, \lambda_n)$. In particular, any Hermitian A

$$A = U \Lambda U^H = \sum_{i=1}^n \lambda_i u_i u_i^H,$$

where λ_i are real.

Section 10

Computing Eigenvalues and Eigenvectors

POWER ITERATION

Direct computation of evals and evecs is computationally difficult for large-scale matrices. One way to solve the problem efficiently is to use iterative methods.

POWER ITERATION

Direct computation of evals and evecs is computationally difficult for large-scale matrices. One way to solve the problem efficiently is to use iterative methods.

A single largest eval can be computed with PI

- 1: $x_0 \leftarrow$ random nonzero vector
- 2: **for** $k = 1$ **to** K **do**
- 3: $x_k = Ax_{k-1}$
- 4: **end for**

POWER ITERATION

Direct computation of evals and evecs is computationally difficult for large-scale matrices. One way to solve the problem efficiently is to use iterative methods.

A single largest eval can be computed with PI

- 1: $x_0 \leftarrow$ random nonzero vector
- 2: **for** $k = 1$ **to** K **do**
- 3: $x_k = Ax_{k-1}$
- 4: **end for**

Let us express $x_0 = \sum_{j=1}^n \alpha_j v_j$, where v_j are evecs of A . Then

$$\begin{aligned}
 x_k &= Ax_{k-1} = A^2x_{k-2} = \cdots = A^kx_0 \\
 &= A^k \sum_j \alpha_j v_j = \sum_j \alpha_j A^k v_j = \sum_j \lambda_j^k \alpha_j v_j \\
 &= \lambda_1^k (\alpha_1 v_1 + \sum_{j=2}^n (\frac{\lambda_j}{\lambda_1})^k \alpha_j v_j) \quad \text{note that } (\frac{\lambda_j}{\lambda_1})^k \rightarrow 0
 \end{aligned}$$

NORMALIZED POWER ITERATION

Problems with power iteration:

- ▶ more than one maximum eval and PI converges to a linear combination of them (see proof)
- ▶ given real matrix and real vector it cannot converge to complex vector
- ▶ possible numerical problems (like fast growth of the vector, see example)

NORMALIZED POWER ITERATION

Problems with power iteration:

- ▶ more than one maximum eval and PI converges to a linear combination of them (see proof)
- ▶ given real matrix and real vector it cannot converge to complex vector
- ▶ possible numerical problems (like fast growth of the vector, see example)

1: $x_0 \leftarrow$ random nonzero vector

2: **for** $k = 1$ **to** K **do**

3: $x_k = Ax_{k-1}$

4: $x_k = x_k / \|x_k\|_\infty$

5: **end for**

GEOMETRIC INTERPRETATION

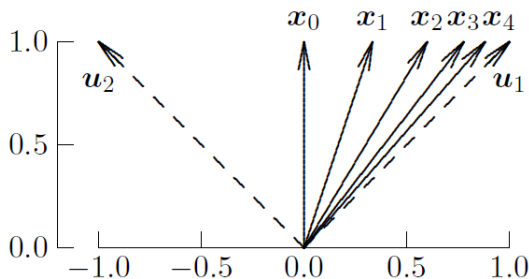


Figure 4.1: Geometric interpretation of the power method.

The convergence rate of power iteration depends on $|\lambda_2/\lambda_1|$. Smaller ratio leads to faster convergence.

NORMALIZED INVERSE ITERATION

Eigenvalues of A^{-1} are the reciprocals of those of A . If we need a smallest eval then we can use this fact. However, instead of applying PI to A^{-1} we solve equivalent system (with LU or Cholesky).

- 1: $x_0 \leftarrow$ random nonzero vector
- 2: **for** $k = 1$ **to** K **do**
- 3: Solve $Ay_k = x_{k-1}$
- 4: $x_k = y_k / \|y_k\|_\infty$
- 5: **end for**

RAYLEIGH QUOTIENT ITERATION

If x is an approximate evec for a real matrix A , then finding λ can be considered as LSQ problem $x\lambda \approx Ax$ which can be solved by normal equation $x^T x \lambda = x^T Ax$, i.e., the LSQ solution is

$$\lambda = \frac{x^T Ax}{x^T x} \quad \text{called Rayleigh quotient.}$$

In particular RQ $\frac{x_k^T Ax_k}{x_k^T x_k}$ at iter k gives a better approximation than NPI.

RAYLEIGH QUOTIENT ITERATION

If x is an approximate evec for a real matrix A , then finding λ can be considered as LSQ problem $x\lambda \approx Ax$ which can be solved by normal equation $x^T x \lambda = x^T Ax$, i.e., the LSQ solution is

$$\lambda = \frac{x^T Ax}{x^T x} \quad \text{called Rayleigh quotient.}$$

In particular RQ $\frac{x_k^T Ax_k}{x_k^T x_k}$ at iter k gives a better approximation than NPI.

- 1: $x_0 \leftarrow$ random nonzero vector
- 2: **for** $k = 1$ **to** K **do**
- 3: $\sigma_k = x_{k-1}^T Ax_{k-1} / x_{k-1}^T x_{k-1}$
- 4: Solve $(A - \sigma_k I)y_k = x_{k-1}$
- 5: $x_k = y_k / \|y_k\|_\infty$
- 6: **end for**

For complex matrices RQ is $\frac{x_k^H Ax_k}{x_k^H x_k}$. The number of correct digits in the approx evec is at least doubled.

DEFLATION, COMPUTING THE SECOND EVAL

If x_1, λ_1 are computed then we can compute λ_2 by deflation, which removes the known eigenvalue. Intuitively, the process is analogous to obtaining $p(\lambda)/(\lambda - \lambda_1)$. Simple case: operation that eliminates eval λ_i in real square diagonalizable matrix

$$A'_i = A - \lambda_i x_i y_i^T,$$

where x_i , and y_i are the right and left eigenvectors, so $\lambda(A) = \lambda_i \cup \lambda(A_i)$.

SIMULTANEOUS ITERATION

Let us consider computing of several evals/evects at once. Let

$X_0 = [x_1^{(0)}, \dots, x_p^{(0)}]$ be the matrix of p linearly independent columns, i.e., $\text{rank}(X_0) = p$. Power iteration on X_0 :

- 1: $X_0 \leftarrow$ any $n \times p$ matrix of rank p
- 2: **for** $k = 1$ **to** K **do**
- 3: $X_k = AX_{k-1}$
- 4: **end for**

SIMULTANEOUS ITERATION

Let us consider computing of several evals/evecs at once. Let

$X_0 = [x_1^{(0)}, \dots, x_p^{(0)}]$ be the matrix of p linearly independent columns, i.e., $\text{rank}(X_0) = p$. Power iteration on X_0 :

- 1: $X_0 \leftarrow$ any $n \times p$ matrix of rank p
- 2: **for** $k = 1$ **to** K **do**
- 3: $X_k = AX_{k-1}$
- 4: **end for**

Let $S_0 = \mathbf{span}(X_0)$, and let S be the invariant subspace spanned by evecs v_1, \dots, v_p corresponding to p largest evals $\lambda_1, \dots, \lambda_p$. If no nonzero vector in S is orthogonal to S_0 then for any $k > 0$, the columns of $X_k = A^k X_0$ form a basis for p -dim space $S_k = A^k S_0$. One can show that S_k converges to S .

SIMULTANEOUS ITERATION

Let us consider computing of several evals/evects at once. Let

$X_0 = [x_1^{(0)}, \dots, x_p^{(0)}]$ be the matrix of p linearly independent columns, i.e., $\text{rank}(X_0) = p$. Power iteration on X_0 :

- 1: $X_0 \leftarrow$ any $n \times p$ matrix of rank p
- 2: **for** $k = 1$ **to** K **do**
- 3: $X_k = AX_{k-1}$
- 4: **end for**

Let $S_0 = \text{span}(X_0)$, and let S be the invariant subspace spanned by evects v_1, \dots, v_p corresponding to p largest evals $\lambda_1, \dots, \lambda_p$. If no nonzero vector in S is orthogonal to S_0 then for any $k > 0$, the columns of $X_k = A^k X_0$ form a basis for p -dim space $S_k = A^k S_0$. One can show that S_k converges to S .

Problems: 1) normalization to avoid of overflow; 2) since converges to a multiple of dominant evect of A , the columns of X_k are increasingly ill-conditioned which requires orthonormalization by QR .

Homework 14 (R).

QR Iteration for eigenvectors, SC book Section 4.5.6

SIMULTANEOUS ITERATION

- 1: $X_0 \leftarrow$ any $n \times p$ matrix of rank p
- 2: $Q_0 R_0 = QR(X_0)$
- 3: **for** $k = 1$ **to** K **do**
- 4: $W = A Q_{k-1}$
- 5: $Q_k R_k = W$
- 6: **end for**

COMPUTING EVALS/EVECS FOR LARGE MATRICES

- ▶ Computing all evals/evecs exactly is very difficult, usually unrealistic.
- ▶ Very large matrices are almost always sparse. If the matrix is not sparse then it is very difficult to store its entries. Example: size of double-precision matrix of size $10^6 \times 10^6$ is approximately 8TB.
- ▶ Approaches that are based on the similarity transformation can hardly work because they cause fill-in and after several transformations the matrix will be dense.
- ▶ Solution: use mat-vec multiplications because their complexity is proportional to the number of non-zeros. Building Krylov sequence x, Ax, A^2x, \dots is easy.
- ▶ Moreover, if a subroutine Ax is provided for any x then there is no need to store the matrix. In this case we can use Krylov subspaces.

KRYLOV SUBSPACE METHODS

Let A be an $n \times n$ matrix, x_0 an arbitrary nonzero vector. We define Krylov sequence $x_k = Ax_{k-1}$ (i.e., power iterations). We define the $n \times k$ Krylov matrix

$$K_k = [x_0 \ x_1 \ \dots \ x_{k-1}] = [x_0 \ Ax_0 \ \dots \ A^{k-1}x_0]$$

and the Krylov subspace is defined as $\mathcal{K}_k = \mathbf{span}(K_k)$. For $k = n$

$$AK_n = [Ax_0 \ \dots \ Ax_{n-2} \ Ax_{n-1}] = [x_1 \ \dots \ x_{n-1} \ x_n] \quad (6)$$

$$= K_n [e_2 \ \dots \ e_n \ K_n^{-1}x_n] = K_n C_n, \quad (7)$$

i.e., $K_n^{-1}AK_n = C_n$, where C_n is the the Hessenberg matrix. It can be reduced to a triangular form through QR and deflation steps.

KRYLOV SUBSPACE METHODS

Unfortunately, the convergence of columns of K_k to the dominant evec of A means that K_n is likely to be ill-conditioned basis to span \mathcal{K}_n but we can apply QR, i.e., $Q_n R_n = K_n$, and the columns of Q_n form orthonormal basis for \mathcal{K}_n .

$$Q_n^H A Q_n = (K_n R_n^{-1})^{-1} A K_n R_n^{-1} = R_n K_n^{-1} A K_n R_n^{-1} = R_n C_n R_n^{-1} \equiv H,$$

i.e., H is a Hessenberg matrix that unitarily similar to A . The columns of Q_n can be computed one at a time.

$$A Q_n = Q_n H \Rightarrow A q_k = h_{1k} q_1 + \dots + h_{kk} q_k + h_{k+1,k} q_{k+1},$$

i.e., q_k is computed by using the preceding q_1, \dots, q_{k-1} , where

$$h_{jk} = q_j^H A q_k.$$

ARNOLDI ITERATION

Algorithm 4.9 Arnoldi Iteration

```

 $x_0$  = arbitrary nonzero starting vector
 $q_1 = x_0 / \|x_0\|_2$  { normalize }
for  $k = 1, 2, \dots$ 
     $u_k = Aq_k$  { generate next vector }
    for  $j = 1$  to  $k$  { subtract from new vector
         $h_{jk} = q_j^H u_k$  its components in all
         $u_k = u_k - h_{jk}q_j$  preceding vectors }
    end
     $h_{k+1,k} = \|u_k\|_2$ 
    if  $h_{k+1,k} = 0$  then stop { stop if matrix is reducible }
     $q_{k+1} = u_k / h_{k+1,k}$  { normalize }
end

```

HOW TO APPROXIMATE EVALS/EVECS

At a given step k of the Arnoldi process we can approximate evals/evecs by using the Rayleigh quotient. Given $Q_k = [q_1 \dots q_k]$ with k Arnoldi vectors, and let $U_k = [q_{k+1} \dots q_n]$ be the matrix with the noncomputed Arnoldi vectors, i.e., $Q_n = [Q_k U_k]$.

$$H = Q_n^H A Q_n = [Q_k U_k]^H A [Q_k U_k] = \begin{pmatrix} H_k & M \\ \tilde{H}_k & N \end{pmatrix},$$

where M, N are not computed yet. The eigenvalues of H_k are called Ritz values and vectors $Q_k y$, where y is evec of H_k are called Ritz vectors. One can prove that Ritz vals/vecs converge to evals/evecs of A .

HOW TO APPROXIMATE EVALS/EVECS

At a given step k of the Arnoldi process we can approximate evals/evecs by using the Rayleigh quotient. Given $Q_k = [q_1 \dots q_k]$ with k Arnoldi vectors, and let $U_k = [q_{k+1} \dots q_n]$ be the matrix with the noncomputed Arnoldi vectors, i.e., $Q_n = [Q_k U_k]$.

$$H = Q_n^H A Q_n = [Q_k U_k]^H A [Q_k U_k] = \begin{pmatrix} H_k & M \\ \tilde{H}_k & N \end{pmatrix},$$

where M, N are not computed yet. The eigenvalues of H_k are called Ritz values and vectors $Q_k y$, where y is evec of H_k are called Ritz vectors.

One can prove that Ritz vals/vecs converge to evals/evecs of A .

- ▶ Complexity: each new k th iteration requires mat-vec multiplication Ax , and $\mathcal{O}(kn)$ work to orthogonalize vector against all previous.
- ▶ Symmetric matrices: Lanczos iteration.

COMPUTING SVD

Singular Value Decomposition of A is given by

$$A = U\Sigma V^T$$

where U and V are square orthogonal matrices, and Σ is diagonal matrix with singular values of A .

Theorem 7.

The singular values of A are nonnegative square roots of eigenvalues of $A^T A$ and columns of U and V are orthonormal vecs of AA^T and $A^T A$, respectively.

If $A^T A$ is given one can compute singular values A . Otherwise, SVD can be computed by a variant of QR iterations/Jacobi/...

Complexity: $mn^2 + n^3$.

SOFTWARE REVIEW

- ▶ BLAS
- ▶ ARPACK
- ▶ LAPACK, LAPACK++
- ▶ SCALAPACK

COMPUTING EIGENVALUES

Homework 15 (I).

- ▶ Download an $n \times n$ matrix at UFL matrix collection, $n > 1000$.
- ▶ Download and compile last version of LAPACK (<http://www.netlib.org/lapack/>, including LAPACKE, and BLAS). You can also use LAPACK++ interface.
- ▶ Browse the manual, see what functions for computing evals/evecs are implemented in LAPACK. What data structures?
- ▶ Compute eigenvalues and eigenvectors.
- ▶ Bonus: compute singular values of the matrix.

Section 11

NONLINEAR EQUATIONS

DEFINITIONS

Definition.

A general system of m nonlinear equations in n unknowns has the form

$$f(x) = 0,$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. We seek n -vector x s.t. all m component functions of $f(x)$ are zero simultaneously.

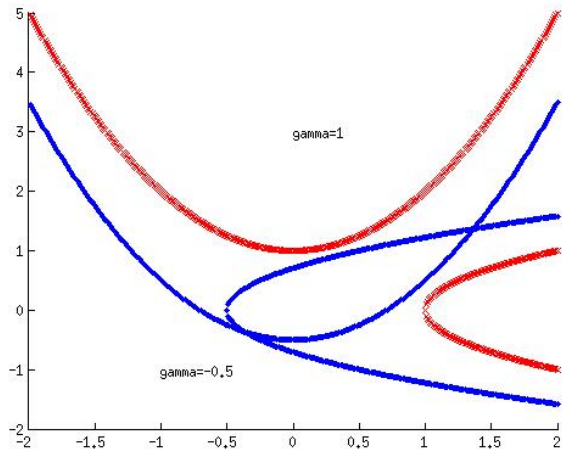
- ▶ The system can be overdetermined when $m > n$ (usually no exact solution, only approximate).
- ▶ The system can be underdetermined when $m < n$ (usually has infinitely many solutions))
- ▶ A solution x s.t. $f(x) = 0$ is called a root of the equation

Example.

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} x_1^2 + 2x_2 + 3 \\ -2x_1^3 + x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

EXAMPLE, 2-DIM

red - no solution, blue - 2 solutions, $f(x) = \begin{bmatrix} x_1^2 - x_2 + \gamma \\ -x_1 + x_2^2 + \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$



EXAMPLES, 1-DIM, BASIC THEOREMS

- ▶ $e^x + 1 = 0$ - no solution
- ▶ $e^{-x} - x = 0$ - 1 solution
- ▶ $x^2 - 4 \sin(x) = 0$ - 2 solutions
- ▶ $x^3 + 6x^2 + 11x = 0$ - 3 solutions
- ▶ $\sin(x) = 0$ - infinitely many solutions

Theorem 8 (Intermediate Value Theorem).

Let $f(x)$ be a function which is continuous on closed interval $[a, b]$ and let y_0 be a real number lying between $f(a)$ and $f(b)$, i.e. with $f(a) \leq y_0 \leq f(b)$ or $f(b) \leq y_0 \leq f(a)$. Then there is at least one c with $a \leq c \leq b$ such that $y_0 = f(c)$.

Conclusion: if $f(a)$ and $f(b)$ differ in signs then there is at least one root within $[a, b]$. This interval is called a *bracket* for a solution of 1-dim nonlin equation $f(x) = 0$. Can be generalized for n -dim.

BASIC THEOREMS (CONT.)

Definition.

- ▶ *The Jacobian matrix of f is defined by $\{J_f(x)\}_{ij} = \partial f_i(x) / \partial x_j$.*

BASIC THEOREMS (CONT.)

Definition.

- ▶ The Jacobian matrix of f is defined by $\{J_f(x)\}_{ij} = \partial f_i(x) / \partial x_j$.
- ▶ A function $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is contractive on a set $S \subset \mathbb{R}^n$ if $\exists \gamma$, with $0 < \gamma < 1$, s.t. $\|g(x) - g(z)\| \leq \gamma \|x - z\|$ for all $x, z \in S$.

BASIC THEOREMS (CONT.)

Definition.

- ▶ The Jacobian matrix of f is defined by $\{J_f(x)\}_{ij} = \partial f_i(x) / \partial x_j$.
- ▶ A function $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is contractive on a set $S \subset \mathbb{R}^n$ if $\exists \gamma$, with $0 < \gamma < 1$, s.t. $\|g(x) - g(z)\| \leq \gamma \|x - z\|$ for all $x, z \in S$.
- ▶ A fixed point of g is any x s.t. $g(x) = x$.

BASIC THEOREMS (CONT.)

Definition.

- ▶ The Jacobian matrix of f is defined by $\{J_f(x)\}_{ij} = \partial f_i(x) / \partial x_j$.
- ▶ A function $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is contractive on a set $S \subset \mathbb{R}^n$ if $\exists \gamma$, with $0 < \gamma < 1$, s.t. $\|g(x) - g(z)\| \leq \gamma \|x - z\|$ for all $x, z \in S$.
- ▶ A fixed point of g is any x s.t. $g(x) = x$.
- ▶ If f is a function whose domain is X and range is Y then f is invertible if there exists function $g : Y \rightarrow X$ such that

$$f(x) = y \iff g(y) = x$$

BASIC THEOREMS (CONT.)

Theorem 9 (Inverse Function Theorem).

For a continuously differentiable f if its $J(f)$ is nonsingular at point x^ then there is a neighborhood of $f(x^*)$ in which f is locally invertible, i.e., $f(x) = y$ has a solution for any y in the that neighborhood.*

BASIC THEOREMS (CONT.)

Theorem 9 (Inverse Function Theorem).

For a continuously differentiable f if its $J(f)$ is nonsingular at point x^ then there is a neighborhood of $f(x^*)$ in which f is locally invertible, i.e., $f(x) = y$ has a solution for any y in the that neighborhood.*

Theorem 10 (Contraction Mapping).

If g is contractive on a closed set $S \subset \mathbb{R}^n$ and $g(S) \subset S$ then g has a unique fixed point in S , i.e., if f has the form $f(x) = x - g(x)$, where g is contractive on S , with $g(S) \subset S$, then $f(x) = 0$ has a unique solution in S , i.e., the fixed point of g .

SENSITIVITY AND CONDITIONING

- ▶ We define the condition number of the problem to be the ratio of of the relative change in the solution to the relative change in the input,i.e.,

$$\text{CN} = \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|} = \frac{\text{relative forward error}}{\text{relative backward error}}$$

SENSITIVITY AND CONDITIONING

- ▶ We define the condition number of the problem to be the ratio of of the relative change in the solution to the relative change in the input, i.e.,

$$\text{CN} = \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|} = \frac{\text{relative forward error}}{\text{relative backward error}}$$

- ▶ Absolute forward error = $f(x + \Delta x) - f(x) \approx f'(x)\Delta x$
- ▶ Relative forward error = $\frac{f(x + \Delta x) - f(x)}{f(x)} \approx \frac{f'(x)\Delta x}{f(x)}$

SENSITIVITY AND CONDITIONING

- ▶ We define the condition number of the problem to be the ratio of of the relative change in the solution to the relative change in the input, i.e.,

$$\text{CN} = \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|} = \frac{\text{relative forward error}}{\text{relative backward error}}$$

- ▶ Absolute forward error = $f(x + \Delta x) - f(x) \approx f'(x)\Delta x$
- ▶ Relative forward error = $\frac{f(x + \Delta x) - f(x)}{f(x)} \approx \frac{f'(x)\Delta x}{f(x)}$
- ▶ $\text{CN} \approx \left| \frac{f'(x)\Delta x/f(x)}{\Delta x/x} \right| = \left| \frac{xf'(x)}{f(x)} \right|$

SENSITIVITY AND CONDITIONING

- ▶ We define the condition number of the problem to be the ratio of of the relative change in the solution to the relative change in the input, i.e.,

$$\text{CN} = \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|} = \frac{\text{relative forward error}}{\text{relative backward error}}$$

- ▶ Absolute forward error = $f(x + \Delta x) - f(x) \approx f'(x)\Delta x$
- ▶ Relative forward error = $\frac{f(x + \Delta x) - f(x)}{f(x)} \approx \frac{f'(x)\Delta x}{f(x)}$
- ▶ $\text{CN} \approx \left| \frac{f'(x)\Delta x/f(x)}{\Delta x/x} \right| = \left| \frac{xf'(x)}{f(x)} \right|$

Example.

$f(x) = \sqrt{x}, f'(x) = 1/(2\sqrt{x}) \Rightarrow \text{CN} \approx \left| \frac{xf'(x)}{f(x)} \right| = \left| \frac{x/(2\sqrt{x})}{\sqrt{x}} \right| = 1/2$, i.e., a relative change in input causes a relative change in the output of about half that size.

SENSITIVITY AND CONDITIONING

- ▶ In root-finding problems $f(x)$ at the solution is 0, so we will use absolute CN instead of the relative CN for evaluating f that is given by $\frac{|\Delta y|}{|\Delta x|} \approx |f'(x^*)|$.
- ▶ The root-finding absolute CN is opposite to the evaluation of f , i.e., it is given by $1/|f'(x^*)| \Rightarrow$ if for approximation \hat{x} we have $|f(\hat{x})| \leq \epsilon$ then the error $|\hat{x} - x^*|$ in the solution may be as large as $\frac{\epsilon}{|f'(x^*)|}$.
- ▶ The corresponding absolute CN in n-dim are $\|J_f(x^*)\|$, and $\|J_f^{-1}(x^*)\|$, for f evaluation and root-finding, respectively

SENSITIVITY AND CONDITIONING

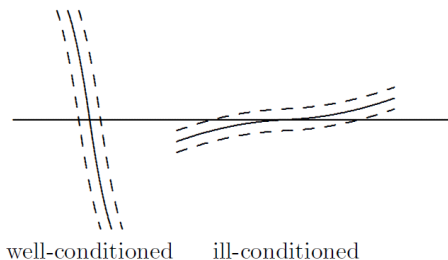
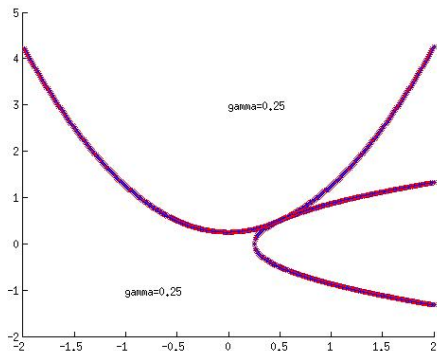


Figure 5.2: Conditioning of roots of nonlinear equations.

- ▶ lines are curved, we seek for intersection with x-axis
- ▶ dashed curves - regions of uncertainty, the root should be between dashed curves
- ▶ small (large) interval of uncertainty - steep (shallow) slope

EXAMPLE OF ILL-CONDITIONED SYSTEM

$$f(x) = \begin{bmatrix} x_1^2 - x_2 + \gamma \\ -x_1 + x_2^2 + \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad J_f(x) = \begin{bmatrix} 2x_1 & -1 \\ -1 & 2x_2 \end{bmatrix}$$



For $\gamma = 0.25$ there is a unique solution $x^* = (0.5, 0.5)^T$, and J_f is singular. Indeed, for smaller γ there are 2 solutions, and for larger γ there is no solution.

CONVERGENCE RATES, STOPPING CRITERIA

Nonlinear equations are usually solved by iterative methods that produce increasingly accurate approximations. The iterations are terminated when the solution is accurate enough.

- ▶ $e_k = x_k - x^*$ is an error at iteration k
- ▶ iterative method converges with rate r is

$$\lim_{k \rightarrow \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = C, \text{ where } C > 0 \text{ is finite}$$

Interesting cases are

- ▶ $r = 1$, and $C < 1$ - convergence rate is linear
 - ▶ $r > 1$ - convergence is superlinear
 - ▶ $r = 2, 3, \dots$ - convergence is quadratic, cubic, etc.
- ▶ often $\|e_k\|$ is difficult to know/compute, so a reasonable surrogate is the relative change in successive iterates $\|x_{k+1} - x_k\|/\|x_k\|$. Example: $f(x_1) = 0, f(x_2) = \epsilon$, and $\|x_1 - x_2\|$ is large.

Section 12

Nonlinear Equations in One Dimension

INTERVAL BISECTION

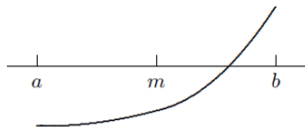
If one cannot find x^* s.t. $f(x^*) \stackrel{\text{exactly}}{=} 0$ (one reason - machine precision) then an alternative is to seek a short interval where f changes the sign.

Initial input: a function f , an interval $[a, b]$ such that $\text{sign}(f(a)) \neq \text{sign}(f(b))$, and an error tolerance tol .

```

while ((b - a) > tol) do
  m = a + (b - a)/2
  if sign(f(a)) = sign(f(m)) then
    a = m
  else
    b = m
  end
end
end

```



Easy to implement; sometimes hard to estimate the correct interval; can be slow; $r = 1$, $C = 0.5$

INTERVAL BISECTION

$$f(x) = x^2 - 4 \sin(x) = 0$$

a	$f(a)$	b	$f(b)$
1.000000	-2.365884	3.000000	8.435520
1.000000	-2.365884	2.000000	0.362810
1.500000	-1.739980	2.000000	0.362810
1.750000	-0.873444	2.000000	0.362810
1.875000	-0.300718	2.000000	0.362810
1.875000	-0.300718	1.937500	0.019849
1.906250	-0.143255	1.937500	0.019849
1.921875	-0.062406	1.937500	0.019849
1.929688	-0.021454	1.937500	0.019849
1.933594	-0.000846	1.937500	0.019849
1.933594	-0.000846	1.935547	0.009491
1.933594	-0.000846	1.934570	0.004320
1.933594	-0.000846	1.934082	0.001736
1.933594	-0.000846	1.933838	0.000445
1.933716	-0.000201	1.933838	0.000445
1.933716	-0.000201	1.933777	0.000122
1.933746	-0.000039	1.933777	0.000122
1.933746	-0.000039	1.933762	0.000041
1.933746	-0.000039	1.933754	0.000001
1.933750	-0.000019	1.933754	0.000001
1.933752	-0.000009	1.933754	0.000001
1.933753	-0.000004	1.933754	0.000001

FIXED-POINT ITERATION

Reminder: given $g : \mathbb{R} \rightarrow \mathbb{R}$, a value x , s.t. $x = g(x)$ is called a fixed point.

Fixed-point problems $x = g(x)$ often arise in practice. Moreover, nonlinear equations can usually be recast as f-p. For example in many iterative scheme

$$x_{k+1} = g(x_k),$$

where g is a function chosen so that fixed points are the solutions for $f(x) = 0$. Such f-p representations of $f(x) = 0$ can be different. Not all of them are iteration schemes.

Example: $f(x) = x^2 - x - 2$ has roots $x = 2$, $x = -1$. Some f-p representations are

- ▶ $g(x) = x^2 - 2, g(x) = x = x^2 - 2 \Rightarrow f(x)$
- ▶ $g(x) = \sqrt{x + 2}, \dots$
- ▶ $g(x) = 1 + 2/x, \dots$
- ▶ $g(x) = (x^2 + 2)/(2x - 1), \dots$

the idea is to release x out of $f(x)$

FIXED-POINT ITERATION, EXAMPLE

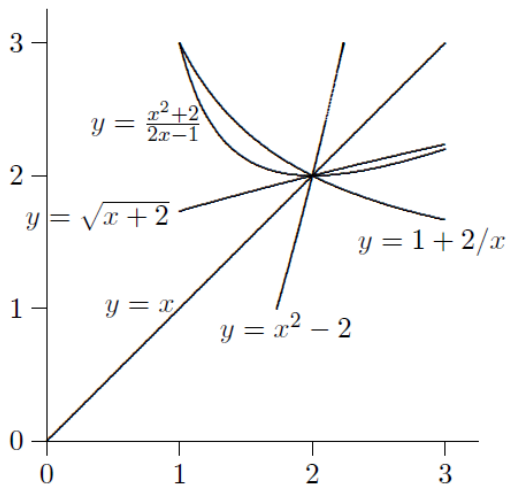
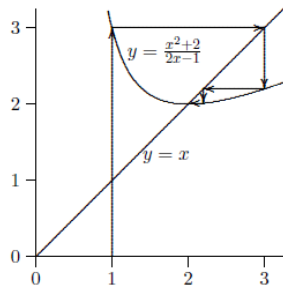
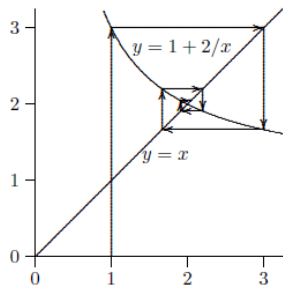
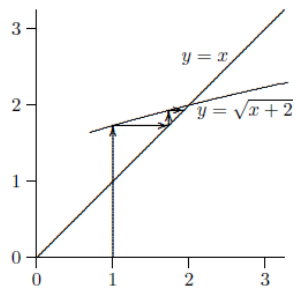
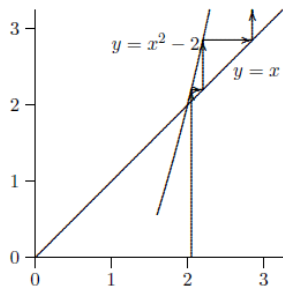


Figure 5.3: A fixed point of some nonlinear functions.

FIXED-POINT ITERATION, EXAMPLE



MEAN VALUE THEOREM

Theorem 11.

If $f : D \rightarrow \mathbb{R}$ is continuous on closed $D = [a, b]$, and differentiable on the open interval (a, b) then $\exists c \in (a, b)$ such that

$$f'(c) = \frac{f(b) - f(a)}{b - a}.$$

FIXED-POINT ITERATION

Derivative of smooth g at x^* is one way to characterize the iterative scheme (IS)

$$x_{k+1} = g(x)$$

- ▶ If $x^* = g(x^*)$ and $|g'(x^*)| < 1$ then the IS is *locally convergent*, i.e., x^* is in some interval $[a, b]$, where f-p iteration converges.

FIXED-POINT ITERATION

Derivative of smooth g at x^* is one way to characterize the iterative scheme (IS)

$$x_{k+1} = g(x)$$

- ▶ If $x^* = g(x^*)$ and $|g'(x^*)| < 1$ then the IS is *locally convergent*, i.e., x^* is in some interval $[a, b]$, where f-p iteration converges.
- ▶ If $|g'(x^*)| > 1$ then g diverges for any starting point ($\neq x^*$)

FIXED-POINT ITERATION

Derivative of smooth g at x^* is one way to characterize the iterative scheme (IS)

$$x_{k+1} = g(x)$$

- ▶ If $x^* = g(x^*)$ and $|g'(x^*)| < 1$ then the IS is *locally convergent*, i.e., x^* is in some interval $[a, b]$, where f-p iteration converges.
- ▶ If $|g'(x^*)| > 1$ then g diverges for any starting point ($\neq x^*$)

Sketch of Proof: if x^* is a fixed point then the error at k th iter is

$$e_{k+1} = x_{k+1} - x^* = g(x_k) - g(x^*)$$

By the Mean Value Theorem there exists a_k between x_k and x^* s.t.

$$g(x_k) - g(x^*) = g'(a_k)(x_k - x^*) \Rightarrow e_{k+1} = g'(a_k)e_k.$$

a_k is not known, but if $|g'(x^*)| < 1$ then by starting iterations close enough to x^* , we know that $\exists C$ s.t. $|g'(a_k)| \leq C < 1$, for $k = 0, 1, \dots \Rightarrow$

$$|e_{k+1}| \leq C|e_k| \leq \dots \leq C^k|e_0| \stackrel{C \leq 1}{\Rightarrow} C^k \rightarrow 0, |e_k| \rightarrow 0.$$

F-P ITERATION, EXAMPLE

Function $f(x) = x^2 - x - 2$ has roots $x = 2$, $x = -1$. Some f-p representations are

- ▶ $g(x) = x^2 - 2, g'(x) = 2x \Rightarrow g'(2) = 4$, i.e., g diverges
- ▶ $g(s) = \sqrt{x+2}, g'(x) = 1/(2\sqrt{x+2}) \Rightarrow g'(2) = 1/4$, i.e., g converges with $C = 1/4$. The positive sign of $g'(2)$ causes the iterates to approach the f-p from one side.
- ▶ $g(x) = 1 + 2/x, g'(x) = -2/x^2 \Rightarrow g'(2) = -1/2$, i.e., g converges with $C = 1/2$, sides of convergence are alternating.
- ▶ $g(x) = (x^2 + 2)/(2x - 1),$
 $g'(x) = (2x^2 - 2x - 4)/(2x - 1)^2 \Rightarrow g'(2) = 0$, i.e., g converges quadratically (when $g'(f-p) = 0$).

F-P ITERATION, EXAMPLE

Function $f(x) = x^2 - x - 2$ has roots $x = 2$, $x = -1$. Some f-p representations are

- ▶ $g(x) = x^2 - 2$, $g'(x) = 2x \Rightarrow g'(2) = 4$, i.e., g diverges
- ▶ $g(s) = \sqrt{x+2}$, $g'(x) = 1/(2\sqrt{x+2}) \Rightarrow g'(2) = 1/4$, i.e., g converges with $C = 1/4$. The positive sign of $g'(2)$ causes the iterates to approach the f-p from one side.
- ▶ $g(x) = 1 + 2/x$, $g'(x) = -2/x^2 \Rightarrow g'(2) = -1/2$, i.e., g converges with $C = 1/2$, sides of convergence are alternating.
- ▶ $g(x) = (x^2 + 2)/(2x - 1)$,
 $g'(x) = (2x^2 - 2x - 4)/(2x - 1)^2 \Rightarrow g'(2) = 0$, i.e., g converges quadratically (when $g'(f-p) = 0$).

Homework 16.

Find two different equivalent fixed-point problems for $f(x) = x^2 - 3x + 2 = 0$. Analyze the convergence properties. Confirm the analysis by implementing each of them and verifying the convergence.

TAYLOR SERIES

Theorem 12.

A function $f(x)$, that has infinitely many derivatives at x_0 , can be represented by the power series

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n$$

for all x within the interval of convergence of the series. If $x_0 = 0$ then TS are known to be Maclauren series.

TS have many theoretical and practical uses. If function is analytical then everything is simpler, however in practice many functions are not infinitely differentiable, and any actual computation can only include a finite number of terms. Thus, in practice a small number of derivatives is used. Therefore, it is important to know how accurately a function can be represented by its truncated TS. An error estimation (Taylor reminder)

$$R_n(x) = f(x) - T_n(x)$$

where $T_n(x)$, the n th partial sum of the Taylor series of f centered at x_0 .

Example: <http://www.mathworks.com/help/symbolic/taylor.html>
+ multivar handout

NEWTON'S METHOD

A nonlinear function can be approximated by the truncated Taylor series

$$f(x + h) \approx f(x) + f'(x)h$$

that is a linear function of h that approximates f near given x . Zero of this function is easy to compute when $h = -f(x)/f'(x)$ if $f'(x) \neq 0$. “Zeros” of f and its approximation are not exactly equal, so we will repeat the iteration. In other words, we approximate f near x_k by the tangent line at $f(x_k)$.

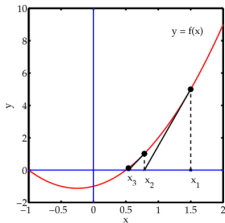
NEWTON'S METHOD

A nonlinear function can be approximated by the truncated Taylor series

$$f(x+h) \approx f(x) + f'(x)h$$

that is a linear function of h that approximates f near given x . Zero of this function is easy to compute when $h = -f(x)/f'(x)$ if $f'(x) \neq 0$. “Zeros” of f and its approximation are not exactly equal, so we will repeat the iteration. In other words, we approximate f near x_k by the tangent line at $f(x_k)$.

- 1: $x_0 \leftarrow$ initial guess
- 2: **for** $k = 1$ **to** K **do**
- 3: $x_{k+1} = x_k - f(x_k)/f'(x_k)$
- 4: **end for**



NEWTON'S METHOD, EXAMPLE

$$f(x) = x^2 - 4 \sin(x) = 0, f'(x) = 2x - 4 \cos(x), \text{ initial } x_0 = 3$$

$$x_{k+1} = x_k - \frac{x_k^2 - 4 \sin(x_k)}{2x_k - 4 \cos(x_k)}$$

x	$f(x)$	$f'(x)$	h
3.000000	8.435520	9.959970	-0.846942
2.153058	1.294772	6.505771	-0.199019
1.954039	0.108438	5.403795	-0.020067
1.933972	0.001152	5.288919	-0.000218
1.933754	0.000000	5.287670	0.000000

The change at each iteration is $h_k = -f(x_k)/f'(x_k)$

NEWTON'S METHOD

We can view Newton's method as a way of transforming $f(x) = 0$ into f-p problem $x = g(x)$, where

$$g(x) = x - f(x)/f'(x) \text{ (because } h_k \rightarrow 0)$$

We need to estimate the convergence of the scheme with

$$g'(x) = f(x)f''(x)/(f'(x))^2.$$

If x^* is a simple root ($f(x^*) = 0$, and $f'(x^*) \neq 0$) then $g'(x^*) = 0 \Rightarrow$ the convergence is quadratic, $r = 2 \Rightarrow$ the number of correct digits is doubled at each iteration for simple root problem. For multiple roots the convergence is linear (check at home!).

SECANT METHOD

- ▶ Possible problem with Newton's method: both f and f' must be evaluated at each iteration; f' can be expensive to evaluate

SECANT METHOD

- ▶ Possible problem with Newton's method: both f and f' must be evaluated at each iteration; f' can be expensive to evaluate
- ▶ One way to cope with this problem is to replace f' with finite difference approximation

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \Leftarrow \text{requires two evaluations of } f$$

SECANT METHOD

- ▶ Possible problem with Newton's method: both f and f' must be evaluated at each iteration; f' can be expensive to evaluate
- ▶ One way to cope with this problem is to replace f' with finite difference approximation

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \Leftrightarrow \text{requires two evaluations of } f$$

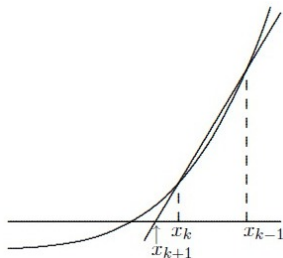
▶ Secant Method

1: $x_0, x_1 \leftarrow$ initial guess

2: **for** $k = 1$ **to** K **do**

$$3: \quad x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{(f(x_k) - f(x_{k-1}))}$$

4: **end for**



SECANT METHOD, EXAMPLE

$f(x) = x^2 - 4 \sin(x) = 0$, h_k is the change at x_k at each iteration.

x	$f(x)$	h
1.000000	-2.365884	
3.000000	8.435520	-1.561930
1.438070	-1.896774	0.286735
1.724805	-0.977706	0.305029
2.029833	0.534305	-0.107789
1.922044	-0.061523	0.011130
1.933174	-0.003064	0.000583
1.933757	0.000019	-0.000004
1.933754	0.000000	0.000000

SECANT METHOD, CONVERGENCE

It can be shown that $\lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k| \cdot |e_{k-1}|} = c > 0$, i.e., the sequence is locally convergent, and the rate is superlinear. We define for each k

$s_k = |e_{k+1}|/|e_k|^r$, r is the convergence rate that we need to determine

$$\Rightarrow |e_{k+1}| = s_k |e_k|^r = s_k (s_{k-1} |e_{k-1}|^r)^r = s_k s_{k-1}^r |e_{k-1}|^{r^2},$$

$$\frac{|e_{k+1}|}{|e_k| \cdot |e_{k-1}|} = \frac{s_k s_{k-1}^r |e_{k-1}|^{r^2}}{s_{k-1} |e_{k-1}|^r |e_{k-1}|} = s_k s_{k-1}^{r-1} |e_{k-1}|^{r^2-2-1}.$$

$\frac{|e_{k+1}|}{|e_k| \cdot |e_{k-1}|} \rightarrow c$, and on the other hand $|e_k| \rightarrow 0$ then $r^2 - r - 1 = 0$, i.e., $r = 1.618$.

Both Newton's and Secant methods must be started close enough to the solution in order to converge. The Secant method must have 2 starting points.

INVERSE INTERPOLATION

Secant method uses *linear* interpolation to approximate function when the solution is close enough. A better convergence rate can be obtained by using higher-degree polynomial interpolation.

Example: Muller's method with convergence rate $r \approx 1.839$.

INVERSE INTERPOLATION

Secant method uses *linear* interpolation to approximate function when the solution is close enough. A better convergence rate can be obtained by using higher-degree polynomial interpolation.

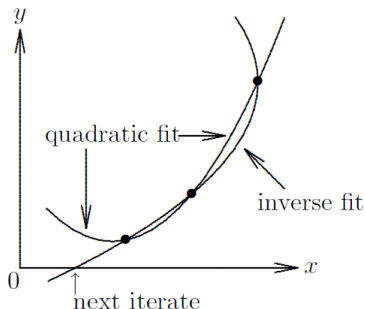
Example: Muller's method with convergence rate $r \approx 1.839$.

Possible problems with polynomial interpolation

- ▶ interpolation polynomial may not have real roots
- ▶ not all roots are easy to compute (in particular, for high-order polynomials)
- ▶ multiple roots, choice of root can be hard

Inverse Interpolation: we fit the values x_k as a function of the values $y_k = f(x_k)$ by a polynomial $p(y)$, so that the next approximate solution is simply $p(0)$.

INVERSE INTERPOLATION



For quadratic interpolation we need 3 approximate values a, b, c with corresponding evaluations f_a, f_b, f_c . By using Lagrangian interpolation we fix $u = f_b/f_c$, $v = f_b/f_a$, $w = f_a/f_c$.

$$p = v(w(u-w)(c-b) - (1-u)(b-a)), \quad q = (w-1)(u-1)(v-1) \Rightarrow x_{k+1} = b + \frac{p}{q}.$$

INVERSE INTERPOLATION, EXAMPLE

$f(x) = x^2 - 4 \sin(x) = 0$, $a = 1$, $b = 2$, $c = 3$, h_k is the change in x_k at each iteration.

x	$f(x)$	h
1.000000	-2.365884	
2.000000	0.362810	
3.000000	8.435520	
1.886318	-0.244343	-0.113682
1.939558	0.030786	0.053240
1.933742	-0.000060	-0.005815
1.933754	0.000000	0.000011

Homework 17 (R).

Linear Fractional Interpolation, SC book, Section 5.5.6

Section 13

Systems of Nonlinear Equations

FIXED-POINT ITERATION

Given $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ then a f-p iteration for g is to find $x \in \mathbb{R}^n$ s.t.

$x = g(x)$, and $x_{k+1} = g(x_k)$ is the corresponding f-p iteration.

FIXED-POINT ITERATION

Given $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ then a f-p iteration for g is to find $x \in \mathbb{R}^n$ s.t.

$x = g(x)$, and $x_{k+1} = g(x_k)$ is the corresponding f-p iteration.

- ▶ In 1-dim the convergence is determined by $|g'(x^*)|$
- ▶ In n -dim the analogous condition is related to the spectral radius

$$\rho(G(x^*)) < 1, \tag{8}$$

where $G(x)$ is the Jacobian matrix of g at x , $\{G(x)\}_{ij} = \frac{\partial g_i(x)}{\partial x_j}$. If (8) is satisfied then the f-p iteration converges if started close enough to the solution. Smaller ρ gives faster convergence.

FIXED-POINT ITERATION

Given $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ then a f-p iteration for g is to find $x \in \mathbb{R}^n$ s.t.

$x = g(x)$, and $x_{k+1} = g(x_k)$ is the corresponding f-p iteration.

- ▶ In 1-dim the convergence is determined by $|g'(x^*)|$
- ▶ In n -dim the analogous condition is related to the spectral radius

$$\rho(G(x^*)) < 1, \quad (8)$$

where $G(x)$ is the Jacobian matrix of g at x , $\{G(x)\}_{ij} = \frac{\partial g_i(x)}{\partial x_j}$. If (8) is satisfied then the f-p iteration converges if started close enough to the solution. Smaller ρ gives faster convergence.

- ▶ Often we don't need to compute all evals, for example $\rho(A) \leq \|A\|$.
- ▶ If $G(x^*) = 0$ the convergence is quadratic or better.

NEWTON'S METHOD

Not all methods for 1-dim can be generalized for n -dim. Given $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ the truncated Taylor series is

$$f(x + s) = f(x) + J_f(x)s,$$

Reminder: $f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}$

where J_f is the Jacobian $\{J_f(x)\}_{ij} = \partial f_i(x) / \partial x_j$. If s satisfies the linear system $J_f(x)s = -f(x)$ then $x + s$ is an approximate zero of f .

NEWTON'S METHOD

Not all methods for 1-dim can be generalized for n -dim. Given

$f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ the truncated Taylor series is

$$f(x + s) = f(x) + J_f(x)s,$$

Reminder: $f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}$

where J_f is the Jacobian $\{J_f(x)\}_{ij} = \partial f_i(x) / \partial x_j$. If s satisfies the linear system $J_f(x)s = -f(x)$ then $x + s$ is an approximate zero of f .

NM: replaces the system of nonlinear equations with the system of linear equations at each iteration.

- 1: $x_0 \leftarrow$ initial guess
- 2: **for** $k = 1$ **to** K **do**
- 3: Solve $J_f(x_k)s_k = -f(x_k)$ for s_k
- 4: $x_k = x_k + s_k$
- 5: **end for**

Example: newton2ddemo.m

NEWTON'S METHOD

- *Convergence.* If the corresponding f-p iterator is smooth then we can differentiate it and evaluate its Jacobian

$$g(x) = x - J_f(x)^{-1}f(x)$$

$$G(x^*) = I - J_f(x^*)^{-1}J_f(x^*) + \sum_{i=1}^n f_i(x^*)H_i(x^*) = 0,$$

where $H_i(x)$ is a component matrix of the derivative of $J_f(x)^{-1}$ (which is a tensor), i.e., the convergence is usually quadratic (given nonsingular J_f at x^*)

NEWTON'S METHOD

- *Convergence.* If the corresponding f-p iterator is smooth then we can differentiate it and evaluate its Jacobian

$$g(x) = x - J_f(x)^{-1}f(x)$$

$$G(x^*) = I - J_f(x^*)^{-1}J_f(x^*) + \sum_{i=1}^n f_i(x^*)H_i(x^*) = 0,$$

where $H_i(x)$ is a component matrix of the derivative of $J_f(x)^{-1}$ (which is a tensor), i.e., the convergence is usually quadratic (given nonsingular J_f at x^*)

- *Complexity.* **(1)** Computing $J_f(x_k)$ (either in closed form or in finite differences) can be done in $\mathcal{O}(m)$ operations, where m is a number of nonzero elements in J_f based on its sparsity structure (n^2 for dense).

NEWTON'S METHOD

- *Convergence.* If the corresponding f-p iterator is smooth then we can differentiate it and evaluate its Jacobian

$$g(x) = x - J_f(x)^{-1}f(x)$$

$$G(x^*) = I - J_f(x^*)^{-1}J_f(x^*) + \sum_{i=1}^n f_i(x^*)H_i(x^*) = 0,$$

where $H_i(x)$ is a component matrix of the derivative of $J_f(x)^{-1}$ (which is a tensor), i.e., the convergence is usually quadratic (given nonsingular J_f at x^*)

- *Complexity.* **(1)** Computing $J_f(x_k)$ (either in closed form or in finite differences) can be done in $\mathcal{O}(m)$ operations, where m is a number of nonzero elements in J_f based on its sparsity structure (n^2 for dense). **(2)** Computing $J_f(x_k)s_k = -f(x_k)$ by LU factorization (or similar) costs $\mathcal{O}(n^3)$ for dense structures.

NEWTON'S METHOD

- *Convergence.* If the corresponding f-p iterator is smooth then we can differentiate it and evaluate its Jacobian

$$g(x) = x - J_f(x)^{-1}f(x)$$

$$G(x^*) = I - J_f(x^*)^{-1}J_f(x^*) + \sum_{i=1}^n f_i(x^*)H_i(x^*) = 0,$$

where $H_i(x)$ is a component matrix of the derivative of $J_f(x)^{-1}$ (which is a tensor), i.e., the convergence is usually quadratic (given nonsingular J_f at x^*)

- *Complexity.* **(1)** Computing $J_f(x_k)$ (either in closed form or in finite differences) can be done in $\mathcal{O}(m)$ operations, where m is a number of nonzero elements in J_f based on its sparsity structure (n^2 for dense). **(2)** Computing $J_f(x_k)s_k = -f(x_k)$ by LU factorization (or similar) costs $\mathcal{O}(n^3)$ for dense structures.
- When partial derivatives are difficult to evaluate then use Secant Updating Method. See handout Section 5.6.3

ROBUST NEWTON-LIKE METHODS

Newton's method may fail to converge when started far from a solution.

- ▶ 1-dim hybrid fail-safe method: bisection and Newton (hard to generalize for n -dim)

ROBUST NEWTON-LIKE METHODS

Newton's method may fail to converge when started far from a solution.

- ▶ 1-dim hybrid fail-safe method: bisection and Newton (hard to generalize for n -dim)
- ▶ Damped Newton Method: when s_k is computed then

$$x_{k+1} = x_k + \alpha_k s_k,$$

where α_k is a scalar. One way to choose α_k is to monitor $\|f(x_k)\|_2$ and even minimize $\|f(x_k + \alpha_k s_k)\|$. Note that α_k can vary for different k .

SOFTWARE

- ▶ <http://www.netlib.org/toms/> (555, 617, 652, 777, 666, 681)
- ▶ <http://www.neos-guide.org/content/nonlinear-equations>

Section 14

INTRODUCTION TO OPTIMIZATION

INTRODUCTION

- ▶ Optimization problems arise in all areas of science and engineering. Any design problem usually involves optimizing some figure of merit such as cost or efficiency. Among all configurations in the design space the goal is to find one or more with the best objective.
- ▶ The objectives can be multiple, different, and mutually contradictory.
- ▶ Optimization problems can include elements of randomness.
- ▶ Can contain many degrees of freedom
- ▶ Almost any scientific computing problem can be (re)formulated as an optimization problem.

EXAMPLE, ISING MODEL

The Ising model is a math model of ferromagnetism in statistical mechanics.

- ▶ The model consists of discrete variables that represent magnetic dipole moments of atomic spins that can be in one of two states (+1 or -1). The spins are arranged in a graph, usually, a lattice, allowing each spin to interact with its neighbors.
- ▶ The model allows the identification of phase transitions.

EXAMPLE, ISING MODEL

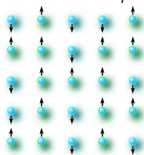
The Ising model is a math model of ferromagnetism in statistical mechanics.

- ▶ The model consists of discrete variables that represent magnetic dipole moments of atomic spins that can be in one of two states (+1 or 1). The spins are arranged in a graph, usually, a lattice, allowing each spin to interact with its neighbors.
- ▶ The model allows the identification of phase transitions. We consider a set of d -dim lattice states. In graph representation each node j correspond to discrete variable $\sigma_j \in \{1, -1\}$. A spin configuration $\sigma = \{\sigma_i\}$ is an assignment of spin value to each node of the lattice.
- ▶ For any two adjacent nodes i and j we define interaction J_{ij} , and each node i has an external magnetic field h_i .

EXAMPLE, ISING MODEL

The Ising model is a math model of ferromagnetism in statistical mechanics.

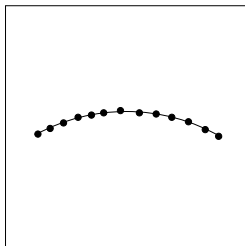
- ▶ The model consists of discrete variables that represent magnetic dipole moments of atomic spins that can be in one of two states (+1 or -1). The spins are arranged in a graph, usually, a lattice, allowing each spin to interact with its neighbors.
- ▶ The model allows the identification of phase transitions. We consider a set of d -dim lattice states. In graph representation each node j correspond to discrete variable $\sigma_j \in \{1, -1\}$. A spin configuration $\sigma = \{\sigma_i\}$ is an assignment of spin value to each node of the lattice.
- ▶ For any two adjacent nodes i and j we define interaction J_{ij} , and each node i has an external magnetic field h_i . The energy of σ is given by the Hamiltonian $H(\sigma) = - \sum_{i,j} J_{ij} \sigma_i \sigma_j - \mu \sum_i h_i \sigma_i$, where μ is magnetic moment.



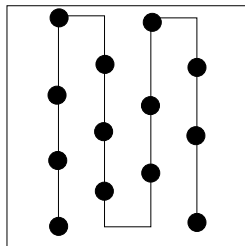
EXAMPLE, LAYOUT PROBLEM

Find an optimal layout of 2D objects such that

1. the total length of the given connections between these objects will be minimal
2. the two-dimensional space will be well utilized and
3. the overlapping between objects will be as little as possible



(a)



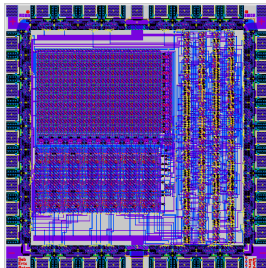
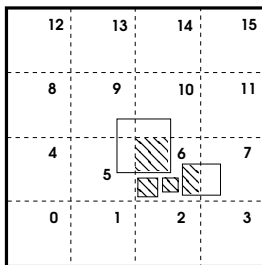
(b)

TWO-DIMENSIONAL LAYOUT PROBLEM: APPLICATIONS

- ▶ **Facility location problem.** In this class of problems the goal is to locate a number of facilities within a minimized distance from the clients. In many industrial versions of the problem there exist additional demands, such as the minimization of the routing between the facilities and various space constraints (e.g., the factory planning problem).could be located.
- ▶ **Network visualization**
- ▶ **Wireless networks and coverage problems.** These have a broad range of applications in the military, surveillance, environmental monitoring, and health care fields. In these problems, having a limited number of resources (like antennae or sensors), one has to cover the area on which many demand points are distributed and have to be serviced.
- ▶ **Engineering VLSI systems.**

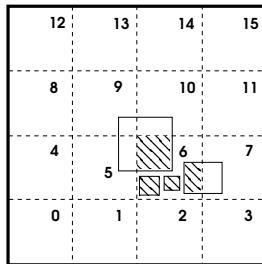
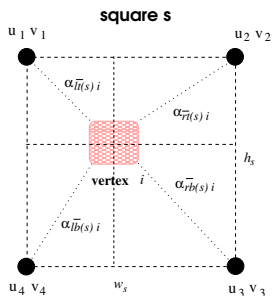
TWO-DIMENSIONAL LAYOUT PROBLEM

minimize Total edge length (*quadratic objective*)
subject to \forall small squares s the amount of the material inside s is less than its area (*linear inequality constraints*).

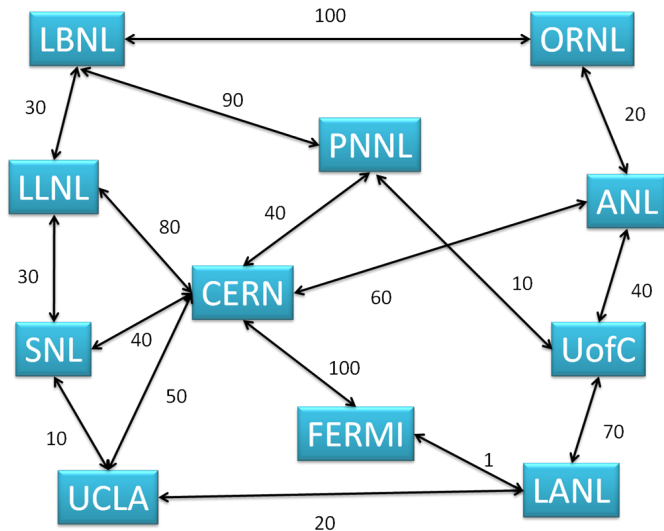


MATERIAL MOVEMENT PROBLEM FORMULATION

$$\min_{u,v} \frac{1}{2} \sum_{ij \in E} w_{ij} \left[\left(\tilde{x}_i + \sum_{p \in c(i)} \alpha_{pi} u_p - \tilde{x}_j - \sum_{p \in c(j)} \alpha_{pj} u_p \right)^2 + \left(\tilde{y}_i + \sum_{p \in c(i)} \alpha_{pi} v_p - \tilde{y}_j - \sum_{p \in c(j)} \alpha_{pj} v_p \right)^2 \right]$$

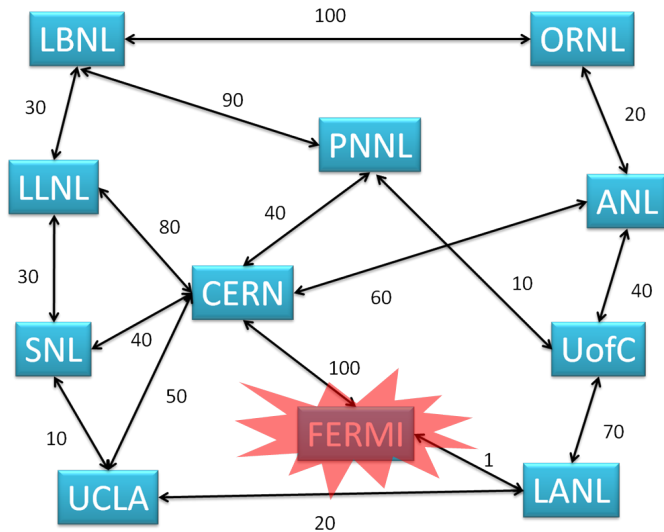


RESPONSE TO EPIDEMICS, CYBER ATTACKS



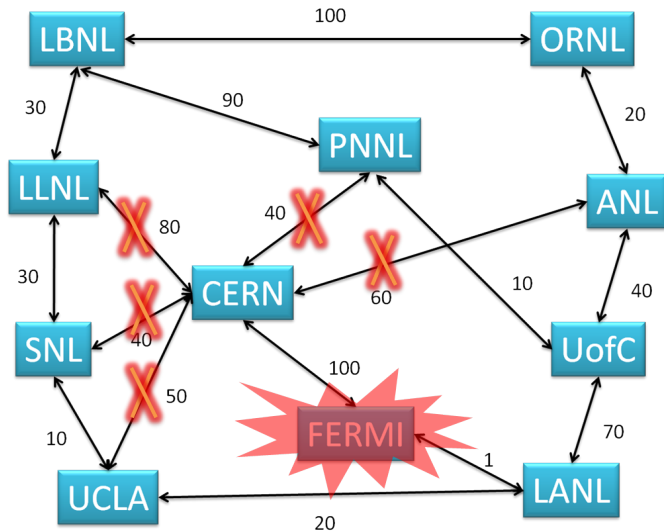
Open Science Grid: collaboration network example

RESPONSE TO EPIDEMICS, CYBER ATTACKS



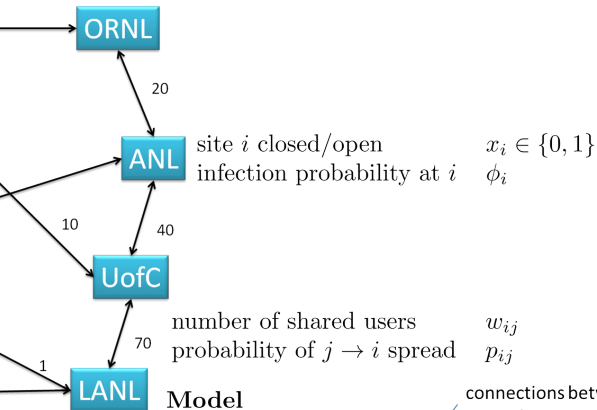
Open Science Grid: collaboration network example

RESPONSE TO EPIDEMICS, CYBER ATTACKS



Open Science Grid: collaboration network example

RESPONSE TO EPIDEMICS, CYBER ATTACKS



Model

maximize
 x

subject to

infection at node i is less
than some constant

connections between
open sites

$$\sum_{ij \in E} w_{ij} x_i x_j$$

$$x_i - \prod_{j \in N(i)} (1 - p_{ij} \phi_j x_j) \leq t_i \quad \forall i \in V$$

$$x \in \{0, 1\}^n$$

DEFINITIONS

An optimization problem can be expressed as the problem of determining an argument for which some function f has an extreme value (min or max) on a given domain.

A general continuous optimization problem has the form

$$\min_x f(x) \text{ subject to } g(x) = 0 \text{ and } h(x) \leq 0,$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$, and $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$.

- ▶ $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$, and $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$ are called the constraints (i.e., some set S called feasible set)
- ▶ any x that satisfies constraints (i.e., $x \in S$) is called feasible point
- ▶ x^* is called minimizer if for all $x \in S$ $f(x^*) \leq f(x)$

MAIN CLASSES OF OPTIMIZATION PROBLEMS

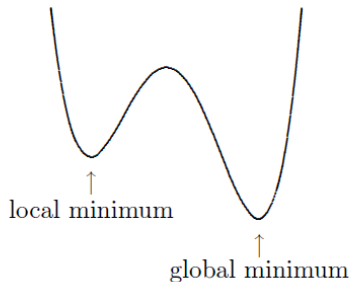
- ▶ Linear programming (LP): The objective function and constraints are linear. The decision variables involved are scalar and continuous.
- ▶ Nonlinear programming (NLP): The objective function and/or constraints are nonlinear. The decision variables are scalar and continuous.
- ▶ Integer programming (IP): The decision variables are scalars and integers.
- ▶ Mixed integer linear programming (MILP): The objective function and constraints are linear. The decision variables are scalar; some of them are integers whereas others are continuous variables.
- ▶ Mixed integer nonlinear programming (MINLP): A nonlinear programming problem involving integer as well as continuous decision variables.

MAIN CLASSES OF OPTIMIZATION PROBLEMS (CONT)

- ▶ Discrete optimization: Problems involving discrete (integer) decision variables. This includes IP, MILP, and MINLPs.
- ▶ Optimal control: The decision variables are vectors.
- ▶ Stochastic programming or stochastic optimization: Also termed optimization under uncertainty. In these problems, the objective function and/or the constraints have uncertain (random) variables. Often involves the above categories as subcategories.
- ▶ Multiobjective optimization: Problems involving more than one objective. Often involves the above categories as subcategories.

CLASSIFICATION OF OP PROPERTIES

- ▶ if $f, g,$ and h are all linear then OP is a linear programming.
- ▶ if any of them is nonlinear then OP is a nonlinear programming.
- ▶ the minimum can be global or local (within some neighborhood around x^*)



- ▶ under some restrictions on $f, g,$ and h OP can be easier to solve (for example convex problems)

EXISTENCE AND UNIQUENESS

Some basic theorems

- ▶ If f is continuous on a closed and bounded set $S \subset \mathbb{R}^n$, then f has a global minimum on S . If S is not closed or is unbounded then f may not have local and global minimum. Example: $f(x) = x$ on (a, b) .

EXISTENCE AND UNIQUENESS

Some basic theorems

- ▶ If f is continuous on a closed and bounded set $S \subset \mathbb{R}^n$, then f has a global minimum on S . If S is not closed or is unbounded then f may not have local and global minimum. Example: $f(x) = x$ on (a, b) .
- ▶ A continuous f is called coercive on unbounded $S \subseteq \mathbb{R}^n$ if

$$\lim_{\|x\| \rightarrow \infty} f(x) = +\infty,$$

i.e., $f(x)$ must be large whenever $\|x\|$ is large \Rightarrow if f is coercive on a closed unbounded S then f has a global minimum on S .

Examples: $f(x) = x^2$ is coercive on \mathbb{R} - has global minimum;

$f(x) = x^3$ is not coercive on \mathbb{R} - has non global minimum.

- ▶ A level set for $f : S \rightarrow \mathbb{R}$ is the set of all points in S for which f has some given constant value. In \mathbb{R}^2 they are contours (like in maps). We are interested in the interior regions whose boundary is a level set, i.e., $f(x \in IR) \leq c$

CONVEXITY

For general optimization problem it is difficult to say something about uniqueness of solutions, and local-vs-global solutions relationships without some assumptions.

CONVEXITY

For general optimization problem it is difficult to say something about uniqueness of solutions, and local-vs-global solutions relationships without some assumptions.

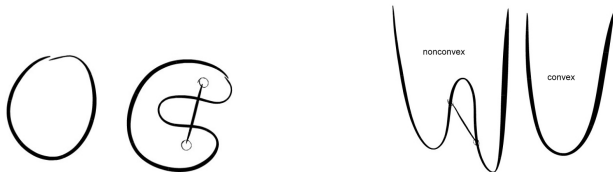
Definition.

A set $S \subseteq \mathbb{R}^n$ is convex if it contains the line segment between any two of its points, i.e.,

$$\forall x, y \in S \quad \{\alpha x + (1 - \alpha)y \mid 0 \leq \alpha \leq 1\} \subseteq S.$$

A function $f : S \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ is convex on convex set S if its graph along any line segment in S lies on or below the chord connecting the function values at the endpoints of the segment, i.e., if

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y) \quad \forall \alpha \in [0, 1], \text{ and } \forall x, y \in S.$$



CONVEXITY

- ▶ If f is convex on convex S then f is necessarily continuous at any interior point of S .

CONVEXITY

- ▶ If f is convex on convex S then f is necessarily continuous at any interior point of S .
- ▶ Any sublevel set of a convex function is convex.

CONVEXITY

- ▶ If f is convex on convex S then f is necessarily continuous at any interior point of S .
- ▶ Any sublevel set of a convex function is convex.
- ▶ Any local minimum of convex f on convex set S is a global minimum on S . If f is strictly convex then the global minimum is unique.

CONVEXITY

- ▶ If f is convex on convex S then f is necessarily continuous at any interior point of S .
- ▶ Any sublevel set of a convex function is convex.
- ▶ Any local minimum of convex f on convex set S is a global minimum on S . If f is strictly convex then the global minimum is unique. Existence of minimum is not guaranteed. If S is closed and bounded then existence is already assured.

CONVEXITY

- ▶ If f is convex on convex S then f is necessarily continuous at any interior point of S .
- ▶ Any sublevel set of a convex function is convex.
- ▶ Any local minimum of convex f on convex set S is a global minimum on S . If f is strictly convex then the global minimum is unique. Existence of minimum is not guaranteed. If S is closed and bounded then existence is already assured.
- ▶ If f is strictly convex on an unbounded $S \subseteq \mathbb{R}^n$ then f has a minimum on S iff f is coercive on S .

UNCONSTRAINED OPTIMALITY CONDITIONS

If $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable then the vector-valued function

$\nabla f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defined by

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

is called the gradient of f .

- ▶ If f is continuously differentiable then $(-\nabla f)$ points toward points with higher (lower) values than $f(x)$. (Intuition: by Taylor's theorem for any $s \in \mathbb{R}^n$ $f(x + s) = f(x) + \nabla f(x + \alpha s)^T s$ for some $\alpha \in (0, 1)$. Taking $s = -\nabla f(x)$ we can see that f decreases along $-\nabla f(x)$ if $\nabla f(x) \neq 0$.)
- ▶ If x^* is a local minimum then there is no downhill direction, so $\nabla f(x^*) = 0$. Such an x^* is called critical point (or stationary point, or equilibrium point).

UNCONSTRAINED OPTIMALITY CONDITIONS

The first-order necessary condition for a minimum:

If $f : S \rightarrow \mathbb{R}$ is continuously differentiable and x^* is an interior point of S at which f has local minimum, then x^* must be a critical point, i.e., $\nabla f(x) = 0$.

If f is convex then any CP is a global minimum. However, in general it could be min/max/saddle point, so we need a criterion for classifying CP for their optimality.

UNCONSTRAINED OPTIMALITY CONDITIONS

The first-order necessary condition for a minimum:

If $f : S \rightarrow \mathbb{R}$ is continuously differentiable and x^* is an interior point of S at which f has local minimum, then x^* must be a critical point, i.e., $\nabla f(x) = 0$.

If f is convex then any CP is a global minimum. However, in general it could be min/max/saddle point, so we need a criterion for classifying CP for their optimality.

If f is twice differentiable, then matrix $H_f = \left(\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)_{ij}$ is called the

Hessian of f , i.e., the Jacobian of ∇f . H is symmetric if second partial derivatives are continuous. Let's add one more term for Taylor's Expansion

$$f(x^* + s) = f(x^*) + \nabla f(x^*)^T s + \frac{1}{2} s^T H_f(x^* + \alpha s) s, \quad \alpha \in (0, 1)$$

Since x^* is CP $\rightarrow \nabla f(x^*) = 0$, so the H -term determines if $f(x^* + s) \lesseqgtr f(x^*)$.

UNCONSTRAINED OPTIMALITY CONDITIONS

The second-order sufficient condition for a minimum:

If H_f is positive definite at CP x^* then x^* is a local minimum.

UNCONSTRAINED OPTIMALITY CONDITIONS

The second-order sufficient condition for a minimum:

If H_f is positive definite at CP x^* then x^* is a local minimum.

Classification of critical points: if $\nabla f(x^*) = 0$ and $H_f(x^*)$ is

- ▶ positive definite $\Rightarrow x^*$ is a minimum of f .
- ▶ negative definite $\Rightarrow x^*$ is a maximum of f .
- ▶ indefinite $\Rightarrow x^*$ is a saddle point of f .
- ▶ singular \Rightarrow various situations can occur.

UNCONSTRAINED OPTIMALITY CONDITIONS

The second-order sufficient condition for a minimum:

If H_f is positive definite at CP x^* then x^* is a local minimum.

Classification of critical points: if $\nabla f(x^*) = 0$ and $H_f(x^*)$ is

- ▶ positive definite $\Rightarrow x^*$ is a minimum of f .
- ▶ negative definite $\Rightarrow x^*$ is a maximum of f .
- ▶ indefinite $\Rightarrow x^*$ is a saddle point of f .
- ▶ singular \Rightarrow various situations can occur.

Test for convexity: if $H_f(x)$ is positive definite at every $x \in S \Rightarrow f$ is convex on S .

Reminder: How to test pod? 1. Cholesky factorization works iff pod;
2. Compute inertia of the matrix using LDL^T .

EXAMPLE

$$f(x) = 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6, \nabla f(x) = \begin{bmatrix} 6x_1^2 + 6x_1 + 12x_2 \\ 12x_1 + 6x_2 - 6 \end{bmatrix}$$

Solutions of $\nabla f(x) = 0$ are $[1, -1]$ and $[2, -3]$ (by any method for solving nonlinear equations).

$$\text{Test for minimums: } H_f(x) = \begin{bmatrix} 12x_1 + 6 & 12 \\ 12 & 6 \end{bmatrix}$$

$$H_f(1, -1) = \begin{bmatrix} 12 & 12 \\ 12 & 6 \end{bmatrix}, \quad H_f(2, -3) = \begin{bmatrix} 30 & 12 \\ 12 & 6 \end{bmatrix}$$

$H_f(1, -1)$ is not pos (saddle point), $H_f(2, -3)$ is pos (local minimum).

CONSTRAINED OPTIMALITY CONDITIONS

- ▶ In the unconstrained problems the minima are found in the interior points of a feasible set.
- ▶ In the constrained problems the minima can often be found at the boundary.

CONSTRAINED OPTIMALITY CONDITIONS

- ▶ In the unconstrained problems the minima are found in the interior points of a feasible set.
- ▶ In the constrained problems the minima can often be found at the boundary.

If S is a feasible set, a nonzero s is a feasible direction at a point $x^* \in S$ if $\exists r > 0$ s.t. $x^* + \alpha s \in S \forall \alpha \in [0, r]$. Then **the first-order necessary optimality condition** is that for any feasible direction s

$$\nabla f(x^*)^T s \geq 0 \text{ i.e., } f \text{ is non-decreasing near } x^*.$$

The second-order necessary optimality condition is

$$s^T H_f(x^*) s \geq 0 \text{ for any feasible direction } s, H_f \text{ is posd in } s$$

LAGRANGE MULTIPLIERS

Consider a nonlinear optimization problem

$$\min_x f(x) \text{ subject to } g(x) = 0, f : \mathbb{R}^n \rightarrow R, g : \mathbb{R}^n \rightarrow \mathbb{R}^m \quad (9)$$

A necessary condition for a feasible x^* to be a solution is that the negative gradient of f lies in span of the constraint normals, i.e., $-\nabla f(x^*) = J_g^T(x^*)\lambda^*$ for some λ^* , the Lagrange multipliers. We redefine (9) with Lagrangian function $\mathcal{L} : \mathbb{R}^{n+m} \rightarrow R$

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x)$$

with gradient and Hessians

$$\nabla \mathcal{L}(x, \lambda) = \begin{bmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ \nabla_\lambda \mathcal{L}(x, \lambda) \end{bmatrix} = \begin{bmatrix} \nabla f(x) + J_g^T(x)\lambda \\ g(x) \end{bmatrix}, \text{ and } H_{\mathcal{L}}(x, \lambda) = \begin{bmatrix} B(x, \lambda) & J_g^T(x) \\ J_g(x) & 0 \end{bmatrix}$$

where $B(x, \lambda) = \nabla_{xx} \mathcal{L}(x, \lambda) = H_f(x) + \sum_{i=1}^m \lambda_i H_{g_i}(x)$.

LAGRANGE MULTIPLIERS

... we are looking for a critical point of the Lagrangian function

$$\nabla \mathcal{L}(x, \lambda) = 0$$

However, $H_{\mathcal{L}}$ is symmetric but not positive definite (even if B is pod).

How to cope with this problem ...

- ▶ \mathcal{L} can be augmented with the “penalty”, so that $H_{\mathcal{L}}$ is pod.

LAGRANGE MULTIPLIERS

... we are looking for a critical point of the Lagrangian function

$$\nabla \mathcal{L}(x, \lambda) = 0$$

However, $H_{\mathcal{L}}$ is symmetric but not positive definite (even if B is pod).

How to cope with this problem ...

- ▶ \mathcal{L} can be augmented with the “penalty”, so that $H_{\mathcal{L}}$ is pod.
- ▶ Sufficient condition for a constrained minimum is that $B(x^*, \lambda^*)$ at the critical point be pod on the tangent space to the constraint surface, i.e., the $\text{Null}(J_g(x^*))$.

EXAMPLE

Given $f(x_1, x_2) = 2\pi x_1(x_1 + x_2)$ and $g(x_1, x_2) = \pi x_1^2 x_2 - C \Rightarrow$

$$\nabla f(x) = 2\pi \begin{bmatrix} 2x_1 + x_2 \\ x_1 \end{bmatrix} \quad \text{and} \quad J_g(x) = \pi \begin{bmatrix} 2x_1 x_2 & x_1^2 \end{bmatrix}.$$

$$\nabla \mathcal{L}(x, \lambda) = \pi \begin{bmatrix} 2(2x_1 + x_2 + x_1 x_2 \lambda) \\ 2x_1 x_1^2 \lambda \\ x_1^2 x_2 - C/\pi \end{bmatrix} = 0 \Rightarrow x_1 = 5.4, x_2 = 10.8, \lambda = -0.37$$

To confirm the optimality of this the solution that can be obtained by any method for nonlinear systems we need to compute H_f , and H_g , and

B . Let's take $C = 10^3$ then $B = \begin{pmatrix} -12.6 & -6.3 \\ -6.3 & 0 \end{pmatrix} \Rightarrow B$ is not pod (check

evals!) \Rightarrow basis for 1d null space of $J_g(x^*) = [369 \ 92.3]$, i.e.,

$z = [-0.243 \ 0.97]^T \Rightarrow z^T B z = 2.23 > 0$, i.e., the solution is minimum.

INEQUALITY CONSTRAINTS

$\min_x f(x)$ subject to $g(x) = 0$, $h(x) \leq 0$, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$.

- ▶ Constraint $h_i(x) \leq 0$ is said to be *active* at feasible x if $h_i(x) = 0$ (otherwise it is inactive)
- ▶ Karush-Kuhn-Tucker first-order necessary conditions:
 1. $\nabla_x \mathcal{L}(x^*, \lambda^*) = 0$
 2. $g(x^*) = 0$
 3. $h(x^*) \leq 0$
 4. $\lambda_i^* \geq 0, i = m + 1, \dots, m + p$
 5. $h_i(x^*) \lambda_i^* = 0, i = m + 1, \dots, m + p$

x^* is a constrained local minimum, if there exist Lagrange multipliers λ^* such that 1-5 hold.

EXAMPLE, INEQUALITY CONSTRAINTS, PART 1

$$f(x) = 0.5x_1^2 + 2.5x_2^2 \quad \text{s.t.} \quad h(x) = x_2 - x_1 + 1 \leq 0$$

- ▶ the unconstrained infeasible minimum is at $(0,0) \Rightarrow$ the constraint is active

EXAMPLE, INEQUALITY CONSTRAINTS, PART 1

$$f(x) = 0.5x_1^2 + 2.5x_2^2 \quad \text{s.t.} \quad h(x) = x_2 - x_1 + 1 \leq 0$$

- ▶ the unconstrained infeasible minimum is at $(0,0) \Rightarrow$ the constraint is active
- ▶ the Lagrangian is

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T h(x) = 0.5x_1^2 + 2.5x_2^2 + \lambda(x_2 - x_1 + 1), \quad (\lambda \text{ is scalar})$$

EXAMPLE, INEQUALITY CONSTRAINTS, PART 1

$$f(x) = 0.5x_1^2 + 2.5x_2^2 \quad \text{s.t.} \quad h(x) = x_2 - x_1 + 1 \leq 0$$

- ▶ the unconstrained infeasible minimum is at $(0,0) \Rightarrow$ the constraint is active

- ▶ the Lagrangian is

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T h(x) = 0.5x_1^2 + 2.5x_2^2 + \lambda(x_2 - x_1 + 1), \quad (\lambda \text{ is scalar})$$

- ▶ $\nabla f(x) = (x_1 \quad 5x_2)^T$, and $J_h(x) = [-1 \quad 1] \Rightarrow$

$$\nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) + J_h^T(x) \lambda = \begin{pmatrix} x_1 \\ 5x_2 \end{pmatrix} + \lambda \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

EXAMPLE, INEQUALITY CONSTRAINTS, PART 2

Let's check for optimality conditions.

$$\blacktriangleright \begin{pmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ h(x) \end{pmatrix} = 0 \Rightarrow \begin{pmatrix} 1 & 0 & -1 \\ 0 & 5 & 1 \\ -1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

$$x_1 = 0.833, x_2 = -0.167, \lambda = 0.833$$

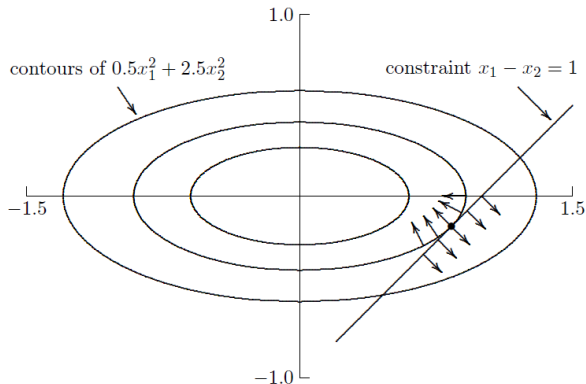


Figure 6.7: Solution to constrained optimization problem.

SENSITIVITY AND CONDITIONING

Homework 18 (R).

Chapter 6.3, Sensitivity and Conditioning

OPTIMIZATION IN 1-DIM

Definition.

A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called unimodal on $[a, b]$ if there is unique value $x^* \in [a, b]$, s.t.

- ▶ $f(x^*)$ is the minimum on $[a, b]$, and
- ▶ for any $x_1, x_2 \in [a, b]$ with $x_1 < x_2$

$$x_1 < x^* \Rightarrow f(x_1) > f(x_2) \text{ and } x_1 > x^* \Rightarrow f(x_1) < f(x_2)$$

GOLDEN SECTION SEARCH

Initial input: a function f , an interval $[a, b]$ on which f is unimodal, and an error tolerance tol .

$$\tau = (\sqrt{5} - 1)/2$$

$$x_1 = a + (1 - \tau)(b - a)$$

$$f_1 = f(x_1)$$

$$x_2 = a + \tau(b - a)$$

$$f_2 = f(x_2)$$

while $((b - a) > tol)$ do

 if $(f_1 > f_2)$ then

$$a = x_1$$

$$x_1 = x_2$$

$$f_1 = f_2$$

$$x_2 = a + \tau(b - a)$$

$$f_2 = f(x_2)$$

 else

$$b = x_2$$

$$x_2 = x_1$$

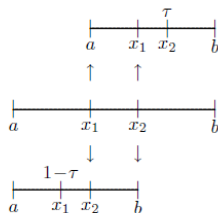
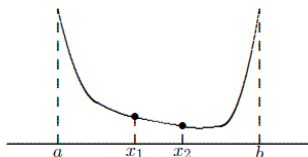
$$f_2 = f_1$$

$$x_1 = a + (1 - \tau)(b - a)$$

$$f_1 = f(x_1)$$

 end

end



Golden section search is safe but slowly convergent. Specifically, it is linearly convergent, with $r = 1$ and $C \approx 0.618$.

GOLDEN SECTION SEARCH

$$f(x) = 0.5 - xe^{-x^2}$$

x_1	f_1	x_2	f_2
0.764	0.074	1.236	0.232
0.472	0.122	0.764	0.074
0.764	0.074	0.944	0.113
0.652	0.074	0.764	0.074
0.584	0.085	0.652	0.074
0.652	0.074	0.695	0.071
0.695	0.071	0.721	0.071
0.679	0.072	0.695	0.071
0.695	0.071	0.705	0.071
0.705	0.071	0.711	0.071

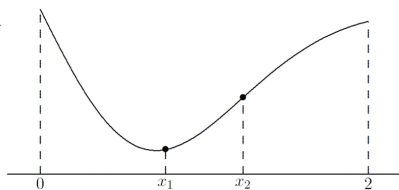


Figure 6.2: First iteration of golden section search for example problem.

SUCCESSIVE PARABOLIC INTERPOLATION

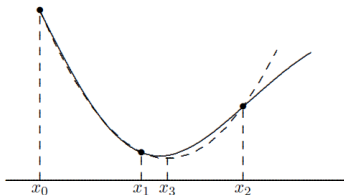
Basic idea: the function f to be minimized is evaluated at 3 points and a quadratic polynomial is fit to these points. The minimum of the parabola is taken and then one of the points is dropped and the process repeated until convergence (not guaranteed, but usually with $r = 1.324$).

- ▶ Given 3 points u, v, w with corresponding f_u, f_v , and f_w .
- ▶ The minimum of the parabola is at $v + p/q$

$$p = \pm(v - u)^2(f_v - f_w) - (v - w)^2(f_v - f_u)$$

$$q = \mp 2((v - u)(f_v - f_w) - (v - w)(f_v - f_u)).$$

- ▶ $u \leftarrow w, w \leftarrow v$, and $v \leftarrow v + p/q$



NEWTON'S METHOD

This is another method to compute local quadratic approximation. We use a truncated Taylor series

$$f(x+h) \approx f(x) + f'(x)h + \frac{1}{2}f''(x)h^2, \text{ the minimum is at } h = -f'(x)/f''(x)$$

- 1: $x_0 \leftarrow$ initial guess
- 2: **for** $k = 1$ **to** K **do**
- 3: $x_{k+1} = x_k - f'(x_k)/f''(x_k)$
- 4: **end for**

NEWTON'S METHOD

This is another method to compute local quadratic approximation. We use a truncated Taylor series

$$f(x+h) \approx f(x) + f'(x)h + \frac{1}{2}f''(x)h^2, \text{ the minimum is at } h = -f'(x)/f''(x)$$

- 1: $x_0 \leftarrow$ initial guess
- 2: **for** $k = 1$ **to** K **do**
- 3: $x_{k+1} = x_k - f'(x_k)/f''(x_k)$
- 4: **end for**

Hybrid algorithms: slow-but-sure methods are combined with fast-but-risky methods. Fast method is done at each iteration but then it should be confirmed by bracketing.

EXAMPLE OF NM

$$f(x) = 0.5 - xe^{-x^2} \Rightarrow f'(x) = (2x^2 - 1)e^{-x^2}, f''(x) = 2x(3 - 2x^2)e^{-x^2}$$

The iteration is $x_{k+1} = x_k - \frac{2x_k^2 - 1}{2x_k(3 - 2x_k^2)}$

x_k	$f(x_k)$
1.000	0.132
0.500	0.111
0.700	0.071
0.707	0.071

Subsection 1

MULTIDIMENSIONAL OPT, UNCONSTRAINED

DIRECT SEARCH

- ▶ Similar to Golden Section Search in 1d
- ▶ No convergence guarantee (in contrast to GSS)
- ▶ Nelder-Mead method implemented in Matlab as *fminsearch*

Example:

```
>> banana = @(x)100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
```

```
>> [x,foal] = fminsearch(banana, [-1.2, 1])
```

```
x = [1, 1]
```

STEEPEST DESCENT

Basic idea: the negative gradient points downhill for any x s.t.

$\nabla f(x) \neq 0$. Usually can improve at each step with linear convergence.

- 1: $x_0 \leftarrow$ initial guess
- 2: **for** $k = 1$ **to** K **do**
- 3: $s_k = -\nabla f(x_k)$
- 4: choose α_k to minimize $f(x_k + \alpha_k s_k)$ // line search
- 5: $x_{k+1} = x_k + \alpha_k s_k$
- 6: **end for**

EXAMPLE OF SLOW ZIG-ZAG CONVERGENCE

$$f(x) = 0.5x_1^2 + 2.5x_2^2 \Rightarrow \nabla f(x) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$$

x_k		$f(x_k)$	$\nabla f(x_k)$	
5.000	1.000	15.000	5.000	5.000
3.333	-0.667	6.667	3.333	-3.333
2.222	0.444	2.963	2.222	2.222
1.481	-0.296	1.317	1.481	-1.481
0.988	0.198	0.585	0.988	0.988
0.658	-0.132	0.260	0.658	-0.658
0.439	0.088	0.116	0.439	0.439
0.293	-0.059	0.051	0.293	-0.293
0.195	0.039	0.023	0.195	0.195
0.130	-0.026	0.010	0.130	-0.130

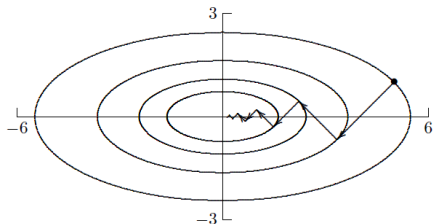


Figure 6.5: Convergence of steepest descent.

Ellipses represent contours on which f has the same constant value.

NEWTON'S METHOD

Local quadratic approximation can be obtained from the truncated Taylor series

$$f(x + s) \approx f(x) + \nabla f(x)^T s + \frac{1}{2} s^T H_f(x) s,$$

where H_f is a Hessian matrix. The quadratic function in s is minimized when $H_f(x)s = -\nabla f(x)$. The convergence is usually quadratic if has good starting point.

- 1: $x_0 \leftarrow$ initial guess
- 2: **for** $k = 1$ **to** K **do**
- 3: Solve $H_f(x_k)s_k = -\nabla f(x_k)$ for s_k
- 4: $x_{k+1} = x_k + s_k$
- 5: **end for**

TRUST-REGION METHODS

Sec 6.5.3: An alternative to a line search is a trust region method, in which an estimate is maintained of the radius of a region in which the quadratic model is sufficiently accurate for the computed Newton step to be reliable, and thus the next approximate solution is constrained to lie within the trust region. If the current trust radius is binding, minimizing the quadratic model function subject to this constraint may modify the direction as well as the length of the Newton step. The accuracy of the quadratic model at a given step is assessed by comparing the actual decrease in the objective function value with that predicted by the quadratic model. Once near a solution, the trust radius should be large enough to permit full Newton steps, yielding

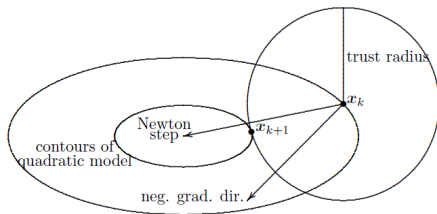


Figure 6.6: Modification of Newton step by trust region method.

rapid local convergence.

QUASI-NEWTON METHODS

One significant disadvantage of Newton's method is a substantial amount of work per iteration. For dense Hessians it requires $\mathcal{O}(n^2)$ scalar function evaluations and $\mathcal{O}(n^3)$ operations the linear system Newton step. Q-N methods make an attempt to reduce the complexity and improve reliability (fast convergence when in a good initial point only). They have a form

$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f(x_k),$$

where B_k is some approximation of Hessian. Many Q-N methods are faster and more robust than the original Newton's method.

CONJUGATE GRADIENT METHOD

... is an effective alternative to Newton's Method that does not require second derivatives. Moreover, we don't need to store approximation to H_f , i.e., the method is suitable for large-scale problems.

- 1: $x_0 \leftarrow$ initial guess
- 2: $g_0 = \nabla f(x_0)$
- 3: $s_0 = -g_0$
- 4: **for** $k = 1$ **to** K **do**
- 5: Choose α_k to minimize $f(x_k + \alpha_k s_k)$
- 6: $x_{k+1} = x_k + \alpha_k s_k$
- 7: $g_{k+1} = \nabla f(x_{k+1})$
- 8: $\beta_{k+1} = (g_{k+1}^T g_{k+1}) / (g^T g_k)$
- 9: $s_{k+1} = -g_{k+1} + \beta_{k+1} s_k$
- 10: **end for**

The method avoids repeatedly searching the same directions by modifying new gradient at each step to remove components in previous direction. The info on H_f is accumulated in conjugate implicitly.

CGM EXAMPLE

$$f(x) = 0.5x_1^2 + 2.5x_2^2 \Rightarrow \nabla f(x) = \begin{pmatrix} x_1 \\ 5x_2 \end{pmatrix}.$$

- ▶ Starting with $x_0 = (5 \ 1)^T$ the initial search direction is the negative gradient $s_0 = -g_0 = -\nabla f(x_0) = (-5 \ -5)^T$
- ▶ the exact minimum along this dir is given by $\alpha_0 = 1/3 \Rightarrow x_1 = (3.33 \ -0.667)^T$
- ▶ compute new gradient $g_1 = \nabla f(x_1) = (3.33 \ -3.333)$
- ▶ (here is the difference with steepest descent) new search direction is given by computing β_1 and then

$$s_1 = -g_1 + \beta_1 s_0 = \begin{pmatrix} -3.333 \\ 3.333 \end{pmatrix} + 0.444 \begin{pmatrix} -5 \\ 5 \end{pmatrix} = \begin{pmatrix} -5.556 \\ 1.111 \end{pmatrix}$$

- ▶ the minimum along this direction is given by $\alpha_1 = 0.6$ which gives the solution

MORE METHODS

Homework 19 (R).

Chapters 6.5.5, 6.5.7 on Secant Updating and Inexact Newton Methods

NONLINEAR LEAST SQUARES

Given data points (t_i, y_i) we need to find vector $x \in \mathbb{R}^n$ of parameters that gives the best fit in the LSQ sense to model $f(t, x) : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$. So far we considered only linear model functions.

NONLINEAR LEAST SQUARES

Given data points (t_i, y_i) we need to find vector $x \in \mathbb{R}^n$ of parameters that gives the best fit in the LSQ sense to model $f(t, x) : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$. So far we considered only linear model functions.

Define residual function $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is $r_i(x) = y_i - f(t_i, x)$, $i = 1..m$,

NONLINEAR LEAST SQUARES

Given data points (t_i, y_i) we need to find vector $x \in \mathbb{R}^n$ of parameters that gives the best fit in the LSQ sense to model $f(t, x) : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$. So far we considered only linear model functions.

Define residual function $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is $r_i(x) = y_i - f(t_i, x)$, $i = 1..m$, and minimize the function

$$\phi(x) = 1/2r(x)^T r(x) \text{ (i.e., sum of squares of residual components)}$$

NONLINEAR LEAST SQUARES

Given data points (t_i, y_i) we need to find vector $x \in \mathbb{R}^n$ of parameters that gives the best fit in the LSQ sense to model $f(t, x) : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$. So far we considered only linear model functions.

Define residual function $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is $r_i(x) = y_i - f(t_i, x)$, $i = 1..m$, and minimize the function

$$\phi(x) = 1/2r(x)^T r(x) \text{ (i.e., sum of squares of residual components)}$$

The gradient and Hessian of ϕ are

$$\nabla\phi(x) = J^T(x)r(x), \text{ and } H_\phi(x) = J^T(x)J(x) + \sum_{i=1}^m r_i(x)H_{r_i}(x),$$

where $J(x)$ is the Jacobian of $r(x)$, and H_{r_i} is the Hessian of $r_i(x)$.

NONLINEAR LEAST SQUARES

Now we can apply Newton's method to compute x_k with the direction s_k defined by linear system

$$(J^T(x_k)J(x_k) + \sum_{i=1}^m r_i(x_k)H_{r_i}(x_k))s_k = -J^T(x_k)r(x_k). \quad (10)$$

GAUSS-NEWTON METHOD FOR NLSQ

Each of H_{r_i} in H_ϕ (Eq 10) is multiplied by small (if the data fits well) term r_i . In Gauss-Newton method for NLSQ, H_{r_i} terms are dropped.

GAUSS-NEWTON METHOD FOR NLSQ

Each of H_{r_i} in H_ϕ (Eq 10) is multiplied by small (if the data fits well) term r_i . In Gauss-Newton method for NLSQ, H_{r_i} terms are dropped.

$$(J^T(x_k)J(x_k))s_k = -J^T(x_k)r(x_k) \text{ is an approx Newton step}$$

This system is normal equations for $m \times n$ linear LSQ problem

$$J(x_k)s_k \approx -r(x_k)$$

that can be solved by orthogonal factorization or by something else. The iterative solution will be

$$x_{k+1} = x_k + s_k,$$

i.e., we replace NLSQ by a sequence of LLSQ.

Homework 20.

The concentration of a drug in the blood-stream is expected to diminish exponentially with time. Fit the model function

$$y = f(t, x) = x_1 e^{x_2 t}$$

to the following data

t	0.5	1	1.5	2	2.5	3	3.5	4
y	6.8	3	1.5	0.75	0.48	0.25	0.2	0.15

1. Perform exponential fit using NLSQ
2. Take logarithm of the model function. It will now be linear in x_2 . The exponential fit can now be done using linear LSQ. (Do not forget to take logs of data points). Use linear LSQ to compute x_1 , and x_2 . Check if the values agree with those in (1).

Subsection 2

Constrained Optimization

LINEAR PROGRAMMING

One of several standard forms of LP is

$$\min_x f(x) = c^T x \text{ subject to } Ax = b \text{ and } x \geq 0$$

where $m < n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c, x \in \mathbb{R}^n$.

- ▶ The feasible region for LP is convex polyhedron in \mathbb{R}^n , and the global minimum occur at one of its vertices.
- ▶ The standard method for LP is called the *simplex method*. It systematically examines a sequence of these vertices to find the minimum. This is the most popular and practically efficient method for LP.
- ▶ Another method: interior-point method which gives better worst case complexity.

LP EXAMPLE

$$\min_{\mathbf{x}} f(\mathbf{x}) = \mathbf{c}^T \mathbf{x} = -8x_1 - 11x_2$$

subject to the linear inequality constraints

$$5x_1 + 4x_2 \leq 40, \quad -x_1 + 3x_2 \leq 12, \quad x_1 \geq 0, \quad x_2 \geq 0.$$

The feasible region, which is bounded by the coordinate axes and the other two straight lines, is shaded in Fig. 6.8. Contour lines of the objective function are drawn, with corresponding values of the objective function shown along the bottom of the graph. The minimum value necessarily occurs at one of the vertices of the feasible region, in this case the point $x_1 = 3.79$, $x_2 = 5.26$, where the objective function has the value -88.2 .

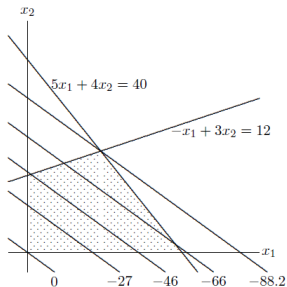


Figure 6.8: Linear programming problem from Example 6.11.

OPTIMIZATION SOFTWARE

- ▶ Important to distinguish: constrained/unconstrained, 1-dim/n-dim, types of derivatives for n-dim
- ▶ AMPL (see Wiki)
- ▶ List of software (see Wiki)

Section 15

INTERPOLATION

INTRODUCTION

Given the set of data points (t_i, y_i) , $i = 1, \dots, m$ such as results of some experiments, and scientific observations. We might want to ...

- ▶ draw a smooth curve through the data points
- ▶ infer the data values between these points
- ▶ determine parameters of functions that describe the data
- ▶ approximate its derivative, integral, etc.

We need to be able to represent the discrete data in terms of relatively simple functions. For example, LSQ methods. Instead of capturing the "trend" of the data we will design functions that match data points. This is not always suitable (for example when the data points are noisy).

INTRODUCTION

Given the set of data points (t_i, y_i) , $i = 1, \dots, m$ such as results of some experiments, and scientific observations. We might want to ...

- ▶ draw a smooth curve through the data points
- ▶ infer the data values between these points
- ▶ determine parameters of functions that describe the data
- ▶ approximate its derivative, integral, etc.

We need to be able to represent the discrete data in terms of relatively simple functions. For example, LSQ methods. Instead of capturing the "trend" of the data we will design functions that match data points. This is not always suitable (for example when the data points are noisy).

The simplest interpolation problem in 1d: for given data (t_i, y_i) , where $t_1 < t_2 < \dots < t_m$ we seek a function $f : \mathbb{R} \rightarrow \mathbb{R}$ such that

$$f(t_i) = y_i, \quad \forall i$$

f is called interpolant.

EXISTENCE, UNIQUENESS, AND CONDITIONING OF INTERPOLANT

Intuition: Can we match the parameters in the interpolant to the data points to be fit:

- ▶ too few parameters - the interpolant doesn't exist
- ▶ too many parameters - the interpolant is not unique

EXISTENCE, UNIQUENESS, AND CONDITIONING OF INTERPOLANT

Intuition: Can we match the parameters in the interpolant to the data points to be fit:

- ▶ too few parameters - the interpolant doesn't exist
- ▶ too many parameters - the interpolant is not unique

Definition.

For a given set of data points $(t_i, y_i)_{i=1}^m$ an interpolant is chosen from the space of functions spanned by a suitable set of **basis functions** $\phi_1(t), \dots, \phi_n(t)$, and f is expressed as linear combination

$$f(t) = \sum_{j=1}^n x_j \phi_j(t),$$

where x_j are the parameters to be determined.

Requirement to interpolant f is ...

Requirement to interpolant f is ... interpolation of all data points, i.e.,

$$f(t_i) = \sum_{j=1}^n x_j \phi_j(t) = y_i.$$

This we can rewrite as a system of linear equations $Ax = y$, where $a_{ij} = \phi_j(t_i)$.

- ▶ number of basis functions $n =$ number of data points $m \Rightarrow$ square lin system and all data points can fit exactly if A is nonsingular

Requirement to interpolant f is ... interpolation of all data points, i.e.,

$$f(t_i) = \sum_{j=1}^n x_j \phi_j(t) = y_i.$$

This we can rewrite as a system of linear equations $Ax = y$, where $a_{ij} = \phi_j(t_i)$.

- ▶ number of basis functions $n =$ number of data points $m \Rightarrow$ square lin system and all data points can fit exactly if A is nonsingular
- ▶ in LSQ approximation the number of bf and parameters is smaller than number of data points \Rightarrow more equations than unknowns \Rightarrow usually cannot fit the data exactly

Requirement to interpolant f is ... interpolation of all data points, i.e.,

$$f(t_i) = \sum_{j=1}^n x_j \phi_j(t) = y_i.$$

This we can rewrite as a system of linear equations $Ax = y$, where $a_{ij} = \phi_j(t_i)$.

- ▶ number of basis functions $n =$ number of data points $m \Rightarrow$ square lin system and all data points can fit exactly if A is nonsingular
- ▶ in LSQ approximation the number of bf and parameters is smaller than number of data points \Rightarrow more equations than unknowns \Rightarrow usually cannot fit the data exactly
- ▶ sometimes underdetermined systems are important if we need to choose the parameter with particular properties

POLYNOMIAL INTERPOLATION

Denote by \mathbb{P}_k the set of all polynomials of degree at most k on some interval. $(+, \times, \mathbb{P}_k)$ is a vector space, $\dim(\mathbb{P}_k) = k + 1$. The choice of the basis for \mathbb{P}_k is very important for complexity, performance, sensitivity, etc.

To interpolate n points we need $k = n - 1$

- natural basis with n monomials $\phi_j(t) = t^{j-1}, j = 1, \dots, n$, i.e., polynomial $p_{n-1} \in \mathbb{P}_{n-1}$ has the form

$$p_{n-1}(t) = x_1 + x_2 t + \dots + x_n t^{n-1}$$

So, we solve the Vandermonde system (nonsingular if all t_i distinct)

$$Ax = \begin{pmatrix} 1 & t_1 & \cdots & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & t_n & \cdots & t_n^{n-1} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix} = y$$

MONOMIAL BASIS, EXAMPLE

We need to interpolate 3 data points $(-2, -27)$, $(0, -1)$, $(1, 0)$. There is a unique polynomial interpolating 3 points $\{(t_i, y_i)\}_{i=1}^3$

$$p_2(t) = x_1 + x_2t + x_3t^2 \Rightarrow Ax = \begin{pmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = y$$

$$\begin{pmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -27 \\ 1 \\ 0 \end{pmatrix} \Rightarrow x = (-1 \ 5 \ -4)^T \Rightarrow p(t) = -1 + 5t - 4t^2.$$

MONOMIALS, VANDERMONDE SYSTEM

It is possible to solve VS in $\mathcal{O}(n^2)$ complexity. However, it is likely that the system will be ill-conditioned on high-degree polynomials.

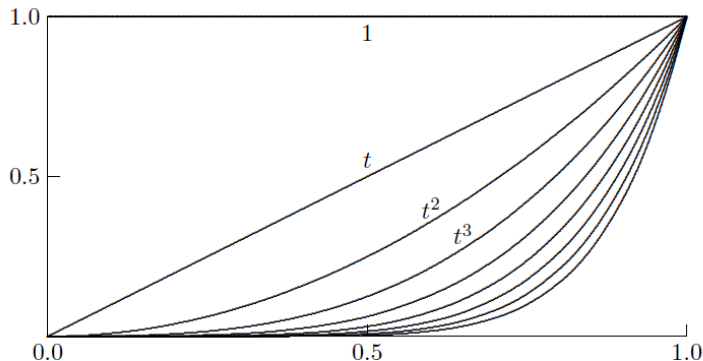


Figure 7.1: Monomial basis functions.

High-degree monomials are very close to each other.

MONOMIALS, VANDERMONDE SYSTEM

The conditioning of the monomial basis can be improved by shifting and scaling

$$\phi_j(t) = \left(\frac{t-c}{d}\right)^{j-1}, \text{ where } c = \frac{t_1 + t_n}{2}, \text{ and } d = \frac{t_n - t_1}{2}$$

Now the variable will be in interval $[-1, 1]$. In monomial basis

$p_{n-1}(t) = \sum x_i t^{i-1}$ can be evaluated efficiently by *nested evaluation*

$$p_{n-1}(t) = x_1 + t(x_2 + t(x_3 + t(\dots(x_{n-1} + x_n t)\dots)))$$

with n additions/multiplications. Same principle can be applied in forming VS.

LAGRANGE INTERPOLATION

Lagrange basis functions for \mathbb{P}_{n-1} , called *fundamental polynomials*, are given by

$$l_j(t) = \frac{\prod_{k=1, k \neq j}^n (t - t_k)}{\prod_{k=1, k \neq j}^n (t_j - t_k)}, \quad j = 1, \dots, n$$

LAGRANGE INTERPOLATION

Lagrange basis functions for \mathbb{P}_{n-1} , called *fundamental polynomials*, are given by

$$l_j(t) = \frac{\prod_{k=1, k \neq j}^n (t - t_k)}{\prod_{k=1, k \neq j}^n (t_j - t_k)}, \quad j = 1, \dots, n$$

From definition it follows that

$$l_j(t_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

LAGRANGE INTERPOLATION

Lagrange basis functions for \mathbb{P}_{n-1} , called *fundamental polynomials*, are given by

$$l_j(t) = \frac{\prod_{k=1, k \neq j}^n (t - t_k)}{\prod_{k=1, k \neq j}^n (t_j - t_k)}, \quad j = 1, \dots, n$$

From definition it follows that

$$l_j(t_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \Rightarrow l_j(t_i) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \Rightarrow$$

for this basis the matrix of system $Ax = y$ is I , i.e., the polynomial interpolating $\{(t_i, y_i)\}_i$ is

$$p_{n-1}(t) = y_1 l_1(t) + y_2 l_2(t) + \dots + y_n l_n(t).$$

LAGRANGE INTERPOLATION

The resulting parameter are conditioned much better. However, it is more expensive for evaluation compared to the monomials.

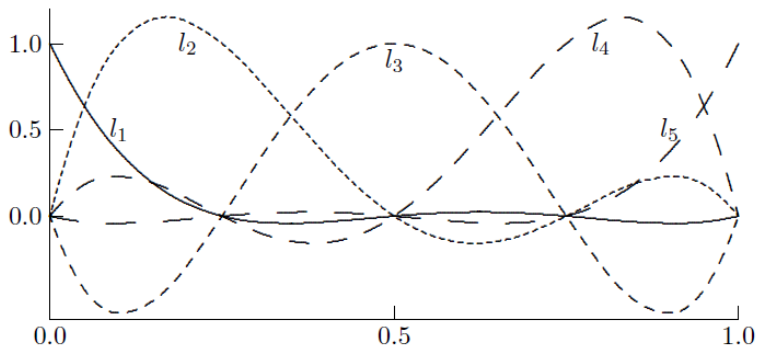


Figure 7.2: Lagrange basis functions.

LAGRANGE INTERPOLATION, EXAMPLE

Example 7.2 Lagrange Interpolation. We illustrate Lagrange interpolation by using it to find the interpolating polynomial for the three data points $(-2, -27)$, $(0, -1)$, $(1, 0)$ of Example 7.1. The Lagrange form for the polynomial of degree two interpolating three points (t_1, y_1) , (t_2, y_2) , (t_3, y_3) is

$$p_2(t) = y_1 \frac{(t - t_2)(t - t_3)}{(t_1 - t_2)(t_1 - t_3)} + y_2 \frac{(t - t_1)(t - t_3)}{(t_2 - t_1)(t_2 - t_3)} + y_3 \frac{(t - t_1)(t - t_2)}{(t_3 - t_1)(t_3 - t_2)}.$$

For this particular set of data, this formula becomes

$$\begin{aligned} p_2(t) &= -27 \frac{(t - 0)(t - 1)}{(-2 - 0)(-2 - 1)} + (-1) \frac{(t - (-2))(t - 1)}{(0 - (-2))(0 - 1)} + 0 \frac{(t - (-2))(t - 0)}{(1 - (-2))(1 - 0)} \\ &= -27 \frac{t(t - 1)}{6} + \frac{(t + 2)(t - 1)}{2}. \end{aligned}$$

Depending on the use to be made of it, the polynomial can be evaluated in this form for any t , or it can be simplified to produce the same result as we saw previously for the same data using the monomial basis (as expected, since the interpolating polynomial is unique).

NEWTON INTERPOLATION

Two previous methods were very different in terms of matrix creation, complexity. NI is between these extremes. The Newton basis functions for \mathbb{P}_{n-1} are

$$\phi_j(t) = \prod_{k=1}^{j-1} (t - t_k), \quad j = 1, \dots, n,$$

i.e., A is lower triangular because $\phi_j(t_i) = 0$ for $i < j$. The polynomial has a form

$$p_{n-1}(t) = x_1 + x_2(t - t_1) + x_3(t - t_1)(t - t_2) + \dots + x_n(t - t_1)(t - t_2)\dots(t - t_{n-1}).$$

NEWTON INTERPOLATION

Two previous methods were very different in terms of matrix creation, complexity. NI is between these extremes. The Newton basis functions for \mathbb{P}_{n-1} are

$$\phi_j(t) = \prod_{k=1}^{j-1} (t - t_k), \quad j = 1, \dots, n,$$

i.e., A is lower triangular because $\phi_j(t_i) = 0$ for $i < j$. The polynomial has a form

$$p_{n-1}(t) = x_1 + x_2(t - t_1) + x_3(t - t_1)(t - t_2) + \dots + x_n(t - t_1)(t - t_2)\dots(t - t_{n-1}).$$

Efficient evaluation form

$$p_{n-1}(t) = x_1 + (t - t_1)(x_2 + (t - t_2)(x_3 + (t - t_3)(\dots(x_{n-1} + x_n(t - t_{n-1}))))))$$

NEWTON INTERPOLATION

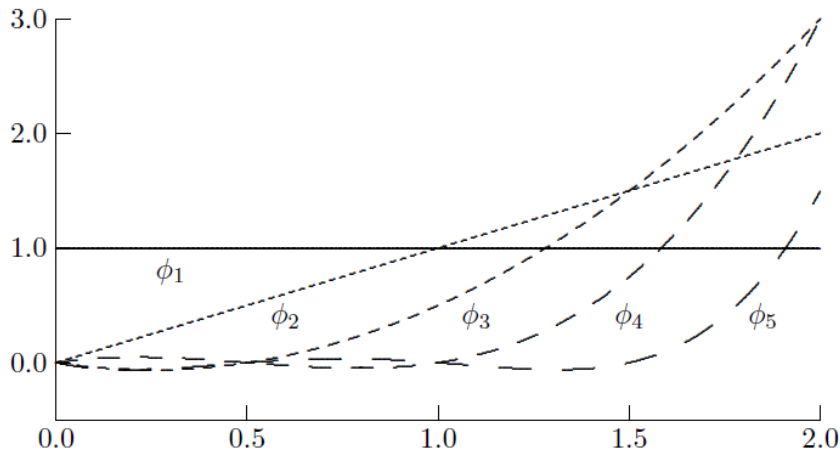


Figure 7.3: Newton basis functions.

NEWTON INTERPOLATION, EXAMPLE

Example 7.3 Newton Interpolation. We illustrate Newton interpolation by using it to find the interpolating polynomial for the three data points $(-2, -27)$, $(0, -1)$, $(1, 0)$ of Example 7.1. With the Newton basis, we have the triangular linear system

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & t_2 - t_1 & 0 \\ 1 & t_3 - t_1 & (t_3 - t_1)(t_3 - t_2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}.$$

For this particular set of data, this system becomes

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix},$$

whose solution, obtained by forward-substitution, is $x = [-27 \ 13 \ -4]^T$. Thus, the interpolating polynomial is

$$p(t) = -27 + 13(t + 2) - 4(t + 2)t,$$

which reduces to the same polynomial we obtained earlier by the other two methods.

INCREMENTAL NEWTON INTERPOLATION

When successive data points are added (like in data streaming algorithms). If $p_j(t)$ is a polynomial of degree $j - 1$ interpolating current j points, then for any constant x_{j+1}

$$p_{j+1}(t) = p_j(t) + x_{j+1}\phi_{j+1}(t)$$

has degree j and it interpolates the same j points. The free parameter x_{j+1} can be chosen so that $p_{j+1}(t)$ interpolates $(j + 1)$ st point, namely, y_{j+1} . Specifically,

$$x_{j+1} = \frac{y_{j+1} - p_j(t_{j+1})}{\phi_{j+1}(t_{j+1})}$$

INCREMENTAL NEWTON INTERPOLATION, EXAMPLE

Example 7.4 Incremental Newton Interpolation. We illustrate by building the Newton interpolant for the previous example incrementally as new data points are added. We begin with the first data point, $(t_1, y_1) = (-2, -27)$, which is interpolated by the constant polynomial

$$p_1(t) = y_1 = -27.$$

Adding the second data point, $(t_2, y_2) = (0, -1)$, we modify the previous polynomial so that it interpolates the new data point as well:

$$\begin{aligned} p_2(t) &= p_1(t) + x_2\phi_2(t) = p_1(t) + \frac{y_2 - p_1(t_2)}{\phi_2(t_2)}\phi_2(t) \\ &= p_1(t) + \frac{y_2 - y_1}{t_2 - t_1}(t - t_1) = -27 + 13(t + 2). \end{aligned}$$

Finally, we add the third data point, $(t_3, y_3) = (1, 0)$, modifying the previous polynomial so that it interpolates the new data point as well:

$$\begin{aligned} p_3(t) &= p_2(t) + x_3\phi_3(t) = p_2(t) + \frac{y_3 - p_2(t_3)}{\phi_3(t_3)}\phi_3(t) \\ &= p_2(t) + \frac{y_3 - p_2(t_3)}{(t_3 - t_1)(t_3 - t_2)}(t - t_1)(t - t_2) \\ &= -27 + 13(t + 2) - 4(t + 2)t. \end{aligned}$$

NEWTON INTERPOLATION, DIVIDED DIFFERENCES

Given a set of data points (t_i, y_i) , $i = 1, \dots, n$, an alternative method for computing the coefficients x_k of the Newton polynomial interpolant is via quantities known as *divided differences*, which are usually denoted by $f[\]$ and are defined recursively by the formula

$$f[t_1, t_2, \dots, t_k] = \frac{f[t_2, t_3, \dots, t_k] - f[t_1, t_2, \dots, t_{k-1}]}{t_k - t_1},$$

where the recursion begins with $f[t_k] = y_k$, $k = 1, \dots, n$. It turns out that the coefficient of the j th basis function in the Newton interpolant is given by $x_j = f[t_1, t_2, \dots, t_j]$. Like forward-substitution, use of this recursion also requires $\mathcal{O}(n^2)$ arithmetic operations to compute the coefficients of the Newton interpolant, but it is less prone to overflow or underflow than is direct formation of the entries of the triangular Newton basis matrix.

NEWTON INTERPOLATION, DIVIDED DIFFERENCES

Example 7.5 Divided Differences. We illustrate divided differences by using this approach to derive the Newton interpolant for the same data points as in the previous examples.

$$\begin{aligned}
 f[t_1] &= y_1 = -27, & f[t_2] &= y_2 = -1, & f[t_3] &= y_3 = 0, \\
 f[t_1, t_2] &= \frac{f[t_2] - f[t_1]}{t_2 - t_1} = \frac{-1 - (-27)}{0 - (-2)} = 13, \\
 f[t_2, t_3] &= \frac{f[t_3] - f[t_2]}{t_3 - t_2} = \frac{0 - (-1)}{1 - 0} = 1, \\
 f[t_1, t_2, t_3] &= \frac{f[t_2, t_3] - f[t_1, t_2]}{t_3 - t_1} = \frac{1 - 13}{1 - (-2)} = -4.
 \end{aligned}$$

Thus, the Newton polynomial is given by

$$\begin{aligned}
 p(t) &= f[t_1]\phi_1(t) + f[t_1, t_2]\phi_2(t) + f[t_1, t_2, t_3]\phi_3(t) \\
 &= f[t_1] + f[t_1, t_2](t - t_1) + f[t_1, t_2, t_3](t - t_1)(t - t_2) \\
 &= -27 + 13(t + 2) - 4(t + 2)t.
 \end{aligned}$$

ORTHOGONAL POLYNOMIALS

Homework 21 (R).

Section 7.3.4

INTERPOLATING CONTINUOUS FUNCTIONS

We have considered discrete data points without worrying about the behavior of interpolant between these points. If the data points represent continuous function then it is important to know how close the interpolant to the real function in all points. For the polynomial interpolation we can use the following result.

Theorem 13.

If f is smooth and p_{n-1} is an interpolant for f at n points $\{t_i\}_{i=1}^n$ then for each $t \in [t_1, t_n]$ there is $\Theta \in (t_1, t_n)$ such that

$$f(t) - p_{n-1}(t) = \frac{f^{(n)}(\Theta)}{n!} (t - t_1)(t - t_2) \dots (t - t_n).$$

Θ is usually not known but in many cases we can bound the derivative. The results becomes very practical if we know that $|f^{(n)}(t)| \leq M$ for all $t \in [t_1, t_n]$, and $h = \max\{t_{i+1} - t_i\}$, then

$$\max_{t \in [t_1, t_n]} |f(t) - p_{n-1}(t)| \leq \frac{Mh^n}{4n}.$$

HIGH-DEGREE POLYNOMIALS, PROBLEMS

High-degree polynomials are expensive to determine and evaluate. Moreover, in some bases the coefficients of the polynomial may be poorly determined as a result of ill-conditioning of the linear system to be solved. A high-degree polynomial necessarily has lots of "wiggles" which may bear no relation to the data to be fit. Although the polynomial goes through the required data points, it may oscillate wildly between data points and thus be useless for many purposes.

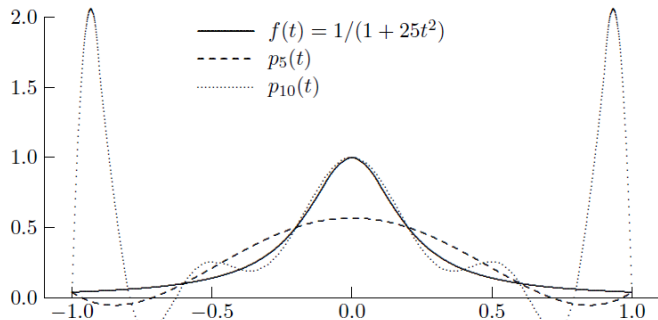


Figure 7.5: Interpolation of Runge's function at equally spaced points.

HIGH-DEGREE POLYNOMIALS, PROBLEMS

Why does this happen?

HIGH-DEGREE POLYNOMIALS, PROBLEMS

Why does this happen? HDP's derivative has $n - 1$ zeros, i.e., HDP has $n - 1$ extrema or inflection points. One possible problem is equally spaced data points. Solution: use Chebyshev points that distribute the error more evenly.

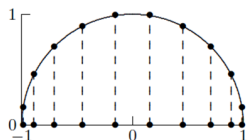


Figure 7.6: Chebyshev points for interpolation.

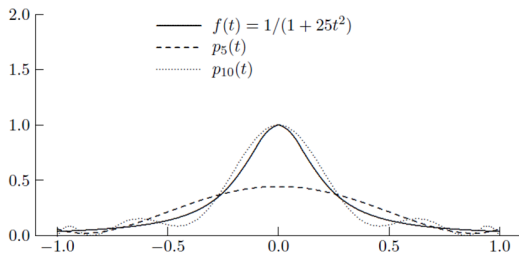


Figure 7.7: Interpolation of Runge's function at the Chebyshev points.

PIECEWISE POLYNOMIAL INTERPOLATION

- ▶ Problems with polynomial interpolation: complexity, oscillating behavior
- ▶ Possible solution: fit large number of data points with many low degree polynomials
- ▶ For given points (t_i, y_i) with $t_1 < \dots < t_n$, a different polynomial is used at each subinterval. The simplest example is linear interpolation with straight lines.
- ▶ Cubic Splines: formulate and solve linear system of equations with fitting polynomial coefficients and also satisfying 1st and 2nd derivative smoothness constraints.

Section 16

FAST FOURIER TRANSFORM

INTRODUCTION

Many functions represent cyclic behavior. In modeling cyclic data it is better to use trigonometric basis functions instead of polynomials, splines, etc. Resulting function will be represented as linear combination of sin and cos which decomposes the function into its components of various frequencies. This trigonometric interpolation can be done efficiently using fast Fourier transform.

- ▶ Given a function of period T $f(t) = f(t + T)$ choose N and sample $f(t)$ within the interval $0 \leq t \leq T$, at N equally spaced points, $n\Delta t$, $n = 0, 1, \dots, N - 1$, and $\Delta t = T/N$.
- ▶ The result is discrete vector $f_n = f(n\Delta t) = [f_0, f_1, \dots, f_{N-1}]^T$.
- ▶ Inner product of two discrete functions is defined as

$$\langle f, g \rangle = \sum_{n=0}^{N-1} f_n^* \cdot g_n$$

INTRODUCTION

Reminder: complex exponential notation by Euler's identity

$$e^{i\Theta} = \cos \Theta + i \sin \Theta,$$

where $i = \sqrt{-1}$. Since $e^{-i\Theta} = \cos \Theta - i \sin \Theta$ easy to see that

$$\cos(2\pi kt) = \frac{e^{2\pi ikt} + e^{-2\pi ikt}}{2} \quad \text{and} \quad \sin(2\pi kt) = i \frac{e^{-2\pi ikt} - e^{2\pi ikt}}{2}$$

INTRODUCTION

For a given integer n we will use the notation

$$\omega_n = \cos(2\pi/n) - i \sin(2\pi/n) = e^{-2\pi i/n}$$

for a given n th root of unity, i.e., $\omega_n^n = 1$. These roots of unity are called *twiddle factors*, i.e., ω_n^k and ω_n^{-k} .

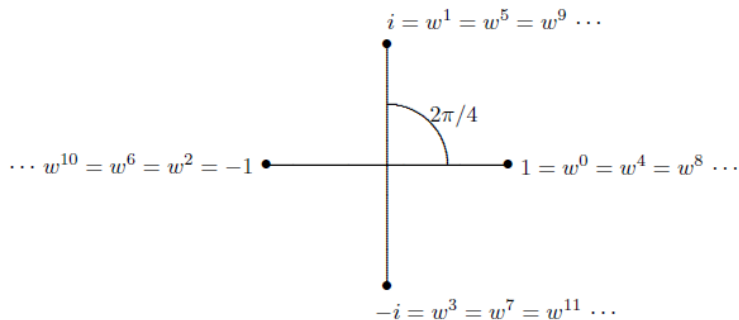


Figure 12.1: Roots of unity $w^k = e^{2\pi i k/n}$ in the complex plane for $n = 4$.

DISCRETE FOURIER TRANSFORM

Definition.

Given a sequence $x = [x_0, \dots, x_{n-1}]^T$, its discrete Fourier transform (DFT) is the sequence $y = [y_0, \dots, y_{n-1}]^T$ given by

$$y_m = \sum_{k=0}^{n-1} x_k \omega_n^{mk} \quad \text{for } m = 0, 1, \dots, n-1.$$

DISCRETE FOURIER TRANSFORM

Definition.

Given a sequence $x = [x_0, \dots, x_{n-1}]^T$, its discrete Fourier transform (DFT) is the sequence $y = [y_0, \dots, y_{n-1}]^T$ given by

$$y_m = \sum_{k=0}^{n-1} x_k \omega_n^{mk} \quad \text{for } m = 0, 1, \dots, n-1.$$

We can write the DFT in matrix notation $y = F_n x$, where the Fourier matrix is given by $\{F_n\}_{mk} = \omega_n^{mk}$. Example of F_4 :

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

Note that F_n is complex Vandermonde matrix but not Hermetian.

DFT

If f is not discretized (i.e., original signal) then it can be represented in a space of trigonometric functions as

$$F(j\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t} dt$$

DFT

If f is not discretized (i.e., original signal) then it can be represented in a space of trigonometric functions as

$$F(j\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t} dt = f[0]e^{-j0} + f[1]e^{-j\omega T} + \dots + f[k]e^{-j\omega kT} + \dots + f[N-1]e^{-j\omega(N-1)T}$$

$$\text{i.e., } F(j\omega) = \sum_{k=0}^{N-1} f[k]e^{-j\omega kT}.$$

We could in principle evaluate this for any ω , but with only N data points to start with, only N final outputs will be significant.

DFT PERIOD

Hence the sequence shown below in Fig. 7.1(a) is considered to be one period of the periodic sequence in plot (b).

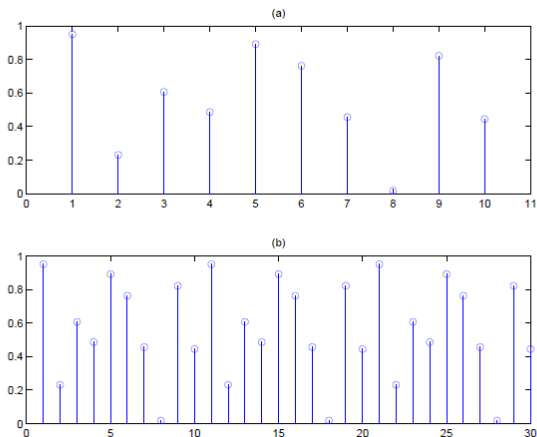


Figure 7.1: (a) Sequence of $N = 10$ samples. (b) implicit periodicity in DFT.

DFT EXAMPLE

Let the continuous signal be

$$f(t) = \underbrace{5}_{\text{dc}} + \underbrace{2 \cos(2\pi t - 90^\circ)}_{1\text{Hz}} + \underbrace{3 \cos 4\pi t}_{2\text{Hz}}$$

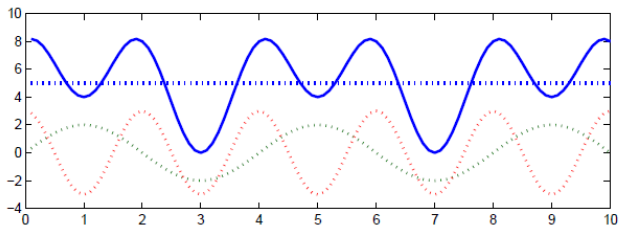


Figure 7.2: Example signal for DFT.

DFT EXAMPLE

Let's sample f 4 times per sec from $t = 0$ to $t = 3/4$. The values of the discrete samples are given by $f[k] = 5 + 2 \cos(\pi/2 \cdot k - 90) + 3 \cos \pi k$, with $t = kT = k/4$

i.e., $f[0] = 8, f[1] = 4, f[2] = 8, f[3] = 0 \Rightarrow F[n] = \sum_{k=0}^3 f[k] e^{-j\pi/2 \cdot nk}$.

$$\begin{pmatrix} F[0] \\ F[1] \\ F[2] \\ F[3] \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \begin{pmatrix} f[0] \\ f[1] \\ f[2] \\ f[3] \end{pmatrix} = \begin{pmatrix} 20 \\ -j4 \\ 12 \\ j4 \end{pmatrix}$$

The magnitude of the DFT coefficients is shown below in Fig. 7.3.

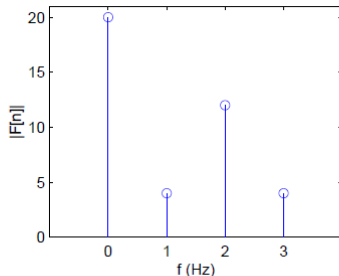


Figure 7.3: DFT of four point sequence.

DFT

The inverse of F_n is easy to find

$$\frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w^{-1} & w^{-2} & w^{-3} \\ 1 & w^{-2} & w^{-4} & w^{-6} \\ 1 & w^{-3} & w^{-6} & w^{-9} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w^1 & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

i.e., $F_n^{-1} = \frac{1}{n} F_n^H$. Thus, the inverse DFT is given by

$$x_k = \frac{1}{n} \sum_{m=0}^{n-1} y_m \omega_n^{-mk}.$$

Conclusion: if the components of x are sample values of function, then DFT solves the trigonometric interpolation problem with mat-vec multiplication in $\mathcal{O}(n^2)$ steps. Fast FT can do better, namely, in $\mathcal{O}(n \log_2 n)$ steps.

FAST FOURIER TRANSFORM

Consider the case $n = 4$.

$$y_m = \sum_{k=0}^3 x_k w^{mk}, \quad m = 0, \dots, 3.$$

Writing out the four equations in full, we have

$$\begin{aligned} y_0 &= x_0 w^0 + x_1 w^0 + x_2 w^0 + x_3 w^0, \\ y_1 &= x_0 w^0 + x_1 w^1 + x_2 w^2 + x_3 w^3, \\ y_2 &= x_0 w^0 + x_1 w^2 + x_2 w^4 + x_3 w^6, \\ y_3 &= x_0 w^0 + x_1 w^3 + x_2 w^6 + x_3 w^9. \end{aligned}$$

Noting from Fig. 12.1 that

$$w^0 = w^4 = 1, \quad w^2 = w^6 = -1, \quad w^9 = w^1,$$

and regrouping, we obtain the four equations

$$\begin{aligned} y_0 &= (x_0 + w^0 x_2) + w^0 (x_1 + w^0 x_3), \\ y_1 &= (x_0 - w^0 x_2) + w^1 (x_1 - w^0 x_3), \\ y_2 &= (x_0 + w^0 x_2) + w^2 (x_1 + w^0 x_3), \\ y_3 &= (x_0 - w^0 x_2) + w^3 (x_1 - w^0 x_3). \end{aligned}$$

\Rightarrow we have 8 additions and 6 multiplications (instead of 12 and 16).

FFT

Conclusion: Computing DFT of original 4 point sequence has been reduced to computing the DFT of its 2 point even and odd subsequences. This property holds in general for n point sequences. Let's see its pattern

define the Fourier matrix F_n by $\{F_n\}_{mk} = w^{mk}$. Thus, for example,

$$F_1 = 1, \quad F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}.$$

Let P_4 be the permutation matrix

$$P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

and D_2 be the diagonal matrix

$$D_2 = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}.$$

FFT

Then we have

$$F_4 P_4 = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & -1 & i & -i \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & -i & i \end{array} \right] = \begin{bmatrix} F_2 & D_2 F_2 \\ F_2 & -D_2 F_2 \end{bmatrix},$$

i.e., F_4 can be rearranged so that each block is a diagonally scaled version of F_2 . Such a hierarchical splitting can be carried out at each level, provided the number of points is even. In general, P_n is the permutation that groups the even-numbered columns of F_n before the odd-numbered columns, and $D_{n/2} = \text{diag}(1, \dots, w^{n/2})$. Thus, to apply F_n to a sequence of length n , we need merely apply $F_{n/2}$ to its even and odd subsequences and scale the results, where necessary, by $D_{n/2}$.

FFT

```

procedure fft ( $x, y, n, w$ )
  if  $n = 1$  then
     $y[0] = x[0]$ 
  else
    for  $k = 0$  to  $(n/2) - 1$ 
       $p[k] = x[2k]$            { split into even and
       $s[k] = x[2k + 1]$        odd subsequences }
    end
    fft ( $p, q, n/2, w^2$ )
    fft ( $s, t, n/2, w^2$ )
    for  $k = 0$  to  $n - 1$ 
       $y[k] = q[k \bmod (n/2)] + w^k t[k \bmod (n/2)]$ 
    end
  end

```

FFT

- There are $\log_2 n$ levels of recursion, each of which involves $\mathcal{O}(n)$ arithmetic operations, so the total cost is $\mathcal{O}(n \log_2 n)$. If the weights w^k are precomputed, the total number of *real* floating-point operations required by the FFT algorithm is $5n \log_2 n$, compared with $8n^2$ real floating-point operations for an ordinary complex matrix-vector product.
- For clarity, separate arrays are used for the subsequences, but in fact the transform can be computed in place using no additional storage.
- The input sequence is assumed to be complex. If the input sequence is real, then additional symmetries in the DFT can be exploited to reduce the storage and operation count by half.
- The output sequence is not produced in the natural order. Most FFT routines automatically allow for this by appropriately rearranging either the input or output sequence. This additional step does not affect the overall computational complexity, because the necessary rearrangement (analogous to a sort) also costs $\mathcal{O}(n \log_2 n)$.
- The FFT algorithm can be formulated using iteration rather than recursion, which is often desirable for greater efficiency or when using a programming language that does not support recursion.

FFT

Despite its name, the fast Fourier transform is an algorithm, not a transform. It is a particular way (or family of ways) of computing the discrete Fourier transform of a sequence in a very efficient manner. As we have seen, the DFT is defined in terms of a matrix-vector product, whose straightforward evaluation would appear to require $\mathcal{O}(n^2)$ arithmetic operations. Use of the FFT algorithm reduces the work to only $\mathcal{O}(n \log_2 n)$, which makes an enormous practical difference in the time required to transform large sequences, as illustrated in Table 12.1.

Table 12.1: Complexity of FFT algorithm versus matrix-vector multiplication

n	$n \log_2 n$	n^2
64	384	4096
128	896	16384
256	2048	65536
512	4608	262144
1024	10240	1048576

FAST POLYNOMIAL MULTIPLICATION

Suppose we are given two polynomials:

$$\begin{aligned} p(x) &= a_0 + a_1x + \cdots + a_{N-1}x^{N-1}, \\ q(x) &= b_0 + b_1x + \cdots + b_{N-1}x^{N-1}. \end{aligned}$$

Their product is defined by

$$p(x) \cdot q(x) = c_0 + c_1x + \cdots + c_{2N-2}x^{2N-2}$$

where

$$c_i = \sum_{\max\{0, i-(N-1)\} \leq k \leq \min\{i, N-1\}} a_k b_{i-k}.$$

Complexity: every a_i is multiplied with every b_j in $\mathcal{O}(N^2)$ operations.

DFT FOR POLYNOMIALS

Let us start with introducing the discrete Fourier transform (DFT) problem. Denote by ω_N an N th complex root of 1, that is, $\omega_N = e^{i\frac{2\pi}{N}}$, where $i^2 = -1$. DFT is the mapping between two vectors:

$$\mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix} \mapsto \hat{\mathbf{a}} = \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \vdots \\ \hat{a}_{N-1} \end{pmatrix}$$

such that

$$\hat{a}_j = \sum_{k=0}^{N-1} a_k \omega_N^{jk}, \quad j = 0, \dots, N-1.$$

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)^2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix} = \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \vdots \\ \hat{a}_{N-1} \end{pmatrix}$$

The matrix above is a Vandermonde matrix and denoted by V_N .

Essentially, DFT evaluates the polynomial

$$p(x) = a_0 + a_1x + \cdots + a_{N-1}x^{N-1}$$

at N points $\omega_N^0, \omega_N^1, \dots, \omega_N^{N-1}$; in other words, $\hat{a}_k = p(\omega_N^k)$ for $0 \leq k \leq N-1$.

FAST POLYNOMIAL MULTIPLICATION

1. Evaluate $p(x)$ and $q(x)$ at $2N$ points $\omega_{2N}^0, \dots, \omega_{2N}^{2N-1}$ using DFT. This step takes time $\Theta(N \log N)$.
2. Obtain the values of $p(x)q(x)$ at these $2N$ points through pointwise multiplication

$$\begin{aligned}
 (p \cdot q)(\omega_{2N}^0) &= p(\omega_{2N}^0) \cdot q(\omega_{2N}^0), \\
 (p \cdot q)(\omega_{2N}^1) &= p(\omega_{2N}^1) \cdot q(\omega_{2N}^1), \\
 &\vdots \\
 (p \cdot q)(\omega_{2N}^{2N-1}) &= p(\omega_{2N}^{2N-1}) \cdot q(\omega_{2N}^{2N-1}).
 \end{aligned}$$

This step takes time $\Theta(N)$.

3. Interpolate the polynomial $p \cdot q$ at the product values using inverse DFT to obtain coefficients $c_0, c_1, \dots, c_{2N-2}$. This last step requires time $\Theta(N \log N)$.

Complexity: $\mathcal{O}(N \cdot \log N)$ operations.

Homework 22 (I).

Gauss analyzed the orbit of the asteroid Pallas based on the observational data

θ	0	30	60	90
x	408	89	-66	10
θ	120	150	180	210
x	338	807	1238	1511
θ	240	270	300	330
x	1583	1462	1183	804

where θ is the ascension in degrees and x is the declination in minutes.

(a) Fit the given data to the function

$$f(\theta) = a_0 + \sum_{k=1}^5 [a_k \cos(2\pi k\theta/360) + b_k \sin(2\pi k\theta/360)] + a_6 \cos(2\pi 6\theta/360),$$

where a_k , $k = 0, \dots, 6$, and b_k , $k = 1, \dots, 5$ are parameters to be determined by the fit. Since there are twelve parameters and twelve data points, the linear system to be solved is square, and the result should interpolate the data. As a matter of historical interest, Gauss performed this computation by a divide-and-conquer method closely related to the FFT algorithm; see [100].

(b) Plot the original data points along with a smooth curve of the function determined by the parameters computed in part a.

(c) Use an FFT routine to compute the DFT \mathbf{y} of the sequence \mathbf{x} .

(d) What relationship can you determine between the real and imaginary parts of \mathbf{y} and the parameters a_k and b_k computed in part a? (*Hint:* You may need to scale by the sequence length or its square root, depending on the particular FFT routine you use.)

Section 17

ITERATIVE METHODS FOR LINEAR SYSTEMS AND MULTIGRID

STATIONARY ITERATIVE METHODS

So far we learned direct methods which can be prohibitive for very large systems of equations. Like IM for nonlinear systems, and for optimization, IM for linear systems begin with initial guess and successfully improve it until the solution is accurate enough.

Simplest type of IM for $Ax = b$ has the form

$$x_{k+1} = Gx_k + c,$$

where G and c are chosen so that fixed point of the function $g(x) = Gx + c$ is a solution to $Ax = b$. Such methods are called *stationary* if G and c are constants for all iterations.

One way to obtain G is called *splitting*

$$A = M - N,$$

where M is nonsingular. Then we can rewrite $G = M^{-1}N$ and $c = M^{-1}b$, so that

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b \text{ or } Mx_{k+1} = Nx_k + b$$

In other words at each iteration we solve a linear system with M . This splitting scheme is a f-p iteration

$$g(x) = M^{-1}Nx + M^{-1}b$$

with Jacobian $G(x) = M^{-1}N$.

Theorem 14.

This iteration scheme is convergent if the spectral radius

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

and smaller $\rho(G)$ the faster the convergence.

Solution is always a tradeoff. Complexity per iteration = complexity of solving ls with M . Extreme example $M = A$.

EXAMPLE: ITERATIVE REFINEMENT OF GAUSSIAN ELIMINATION SOLUTION

Iterative refinement has a form

$$x_{k+1} = x_k + B^{-1}(b - Ax_k),$$

where ideally B^{-1} is an inverse of A . This can be rewritten as

$$x_{k+1} = (I - B^{-1}A)x_k + B^{-1}b.$$

This is a SIM with $G = I - B^{-1}A$ and $c = B^{-1}b$. The scheme converges if

$$\rho(I - B^{-1}A) < 1.$$

Iterative refinement can stabilize "fast but risky" algorithms.

JACOBI METHOD

In the matrix splitting $A = M - N$, the simplest choice for M is diagonal, specifically the diagonal of A . If $D = \text{diag}(A)$, L and U are lower and upper triangular parts of A then the splitting is

$$M = D, N = -(L + U).$$

The Jacobi method is defined as

$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)}) \text{ or } x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}}{a_{ii}},$$

i.e., at each point the solution is the average of previous neighbors. In other words the guess is diffused until the stable point is reached. Doesn't always converge. Converges for diagonally row dominant matrices. Convergence is very slow.

GAUSS-SEIDEL METHOD

Let's use the latest information available at each point at the moment of its computation

$$x^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}}{a_{ii}}, \text{ or}$$

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

The splitting is $M = D + L$, and $N = -U$. In addition to faster convergence, another benefit of the Gauss-Seidel method is that duplicate storage is not needed for the vector x , since the newly computed component values can overwrite the old ones immediately. On the other hand, the updating of the unknowns must now be done successively, in contrast to the Jacobi method, in which the unknowns can be updated in any order or even simultaneously.

SUCCESSIVE OVER-RELAXATION

The convergence rate of the Gauss-Seidel method can be accelerated by a technique called SOR, which the step to the next Gauss-Seidel iterate as a search direction, but with a fixed search parameter denoted by ω .

$$x^{(k+1)} = x^{(k)} + \omega(x_{GS}^{(k+1)} - x^{(k)})$$

We can think of SOR as taking a weighted average of the current iterate and the next GS iterate

$$x^{(k+1)} = (1 - \omega)x^{(k)} + \omega x_{GS}^{(k+1)}.$$

In either case, ω is a fixed relaxation parameter chosen to accelerate convergence. A value $\omega > 1$ gives over-relaxation, whereas $\omega < 1$ gives under-relaxation. We always have $0 < \omega < 2$ (otherwise the method diverges), but choosing a specific value of ω to attain the best possible convergence rate is difficult.

SOR

In matrix notation

$$\begin{aligned}x^{(k+1)} &= x^{(k)} + \omega(D^{-1}(b - Lx^{(k+1)} - Ux^{(k)}) - x^{(k)}) \\ &= (D + \omega L)^{-1}((1 - \omega)D - \omega U)x^{(k)} + \omega(D + \omega L)^{-1}b,\end{aligned}$$

so the splitting is

$$M = \frac{1}{\omega}D + L, \quad N = \left(\frac{1}{\omega} - 1\right)D - U$$

CONJUGATE GRADIENT METHOD

CGM is one of the methods based on optimization. If A is symmetric and positive definite then the minimizer of

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

solves $Ax = b$, i.e., we can search for a direction s_k , so that $x_{k+1} = x_k + \alpha s_k$.

CONJUGATE GRADIENT METHOD

CGM is one of the methods based on optimization. If A is symmetric pod then minimizer of

$$\phi(x) = 1/2x^T Ax - x^T b$$

solves $Ax = b$, i.e., we can search for a direction s_k , so that $x_{k+1} = x_k + \alpha s_k$.

Note special features of QP optimization

- ▶ negative gradient is a residual $-\nabla\phi(x) = b - Ax = r$
- ▶ for any search direction s_k , we need not perform a line search, because the optimal choice for α can be determined analytically. Specifically, the minimum over α occurs when the new residual is orthogonal to the search direction:

$$0 = \frac{d}{d\alpha}\phi(x_{k+1}) = \nabla\phi(x_{k+1})^T \frac{d}{d\alpha}x_{k+1} = -r_{k+1}^T s_k$$

if we express new residual in terms of old residual we get

$$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha s_k) = (b - Ax_k) - \alpha As_k = r_k - \alpha As_k$$

i.e., $\alpha = (r_k^T s_k)/(s_k^T As_k)$.

CGM ALGORITHM

Algorithm 11.1 Conjugate Gradient Method for Linear Systems

 $\mathbf{x}_0 =$ initial guess

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$$

$$\mathbf{s}_0 = \mathbf{r}_0$$

for $k = 0, 1, 2, \dots$

$$\alpha_k = \mathbf{r}_k^T \mathbf{r}_k / \mathbf{s}_k^T \mathbf{A} \mathbf{s}_k \quad \{ \text{compute search parameter} \}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k \quad \{ \text{update solution} \}$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{s}_k \quad \{ \text{compute new residual} \}$$

$$\beta_{k+1} = \mathbf{r}_{k+1}^T \mathbf{r}_{k+1} / \mathbf{r}_k^T \mathbf{r}_k$$

$$\mathbf{s}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{s}_k \quad \{ \text{compute new search direction} \}$$

end

CGM WITH PRECONDITIONER

Algorithm 11.2 Preconditioned Conjugate Gradient Method

 $\mathbf{x}_0 =$ initial guess $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ $\mathbf{s}_0 = \mathbf{M}^{-1}\mathbf{r}_0$ for $k = 0, 1, 2, \dots$ $\alpha_k = \mathbf{r}_k^T \mathbf{M}^{-1} \mathbf{r}_k / \mathbf{s}_k^T \mathbf{A} \mathbf{s}_k$ { compute search parameter } $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$ { update solution } $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{s}_k$ { compute new residual } $\beta_{k+1} = \mathbf{r}_{k+1}^T \mathbf{M}^{-1} \mathbf{r}_{k+1} / \mathbf{r}_k^T \mathbf{M}^{-1} \mathbf{r}_k$ $\mathbf{s}_{k+1} = \mathbf{M}^{-1} \mathbf{r}_{k+1} + \beta_{k+1} \mathbf{s}_k$ { compute new search direction }end

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{1,2} \\ u_{2,2} \end{bmatrix} = \begin{bmatrix} u_{0,1} + u_{1,0} \\ u_{3,1} + u_{2,0} \\ u_{0,2} + u_{1,3} \\ u_{3,2} + u_{2,3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Jacobi

k	x_1	x_2	x_3	x_4
0	0.000	0.000	0.000	0.000
1	0.000	0.000	0.250	0.250
2	0.062	0.062	0.312	0.312
3	0.094	0.094	0.344	0.344
4	0.109	0.109	0.359	0.359
5	0.117	0.117	0.367	0.367
6	0.121	0.121	0.371	0.371
7	0.123	0.123	0.373	0.373
8	0.124	0.124	0.374	0.374
9	0.125	0.125	0.375	0.375

Gauss-Seidel

k	x_1	x_2	x_3	x_4
0	0.000	0.000	0.000	0.000
1	0.000	0.000	0.250	0.312
2	0.062	0.094	0.344	0.359
3	0.109	0.117	0.367	0.371
4	0.121	0.123	0.373	0.374
5	0.124	0.125	0.375	0.375
6	0.125	0.125	0.375	0.375

SOR

k	x_1	x_2	x_3	x_4
0	0.000	0.000	0.000	0.000
1	0.000	0.000	0.268	0.335
2	0.072	0.108	0.356	0.365
3	0.119	0.121	0.371	0.373
4	0.123	0.124	0.374	0.375
5	0.125	0.125	0.375	0.375

CGM

k	x_1	x_2	x_3	x_4
0	0.000	0.000	0.000	0.000
1	0.000	0.000	0.333	0.333
2	0.125	0.125	0.375	0.375

Subsection 1

Multiscale Methods, Multigrid

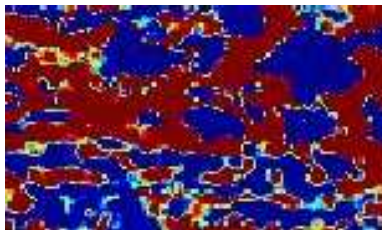
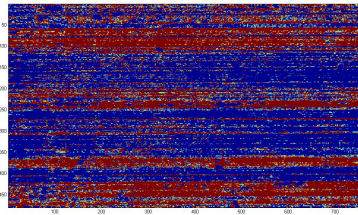
In many cases, a big scale gap can be observed between micro- and macroscopic scales of problem because of the difference in physical (social, biological, mathematical, etc.) models and/or laws at different scales.





In many problems, notwithstanding the fact that elementary parts of the system have a complicated (and even nondeterministic) behavior, their ensembles represent much more structured systems.

low resolution



high resolution

Even when differences between models at different scales are not observed, an efficient approximation of the microscopic scale can be achieved by looking at the macroscopic scale with its substantially smaller number of elementary objects (such as variables and particles).



THE MULTISCALE METHOD: A COARSE VIEW

Multiscale \approx **Multilevel** \approx **Multigrid** \approx **Multiresolution**

- ▶ The **Multiscale method** is a class of algorithmic techniques for solving efficiently large-scale computational and optimization problems.

THE MULTISCALE METHOD: A COARSE VIEW

Multiscale \approx **Multilevel** \approx **Multigrid** \approx **Multiresolution**

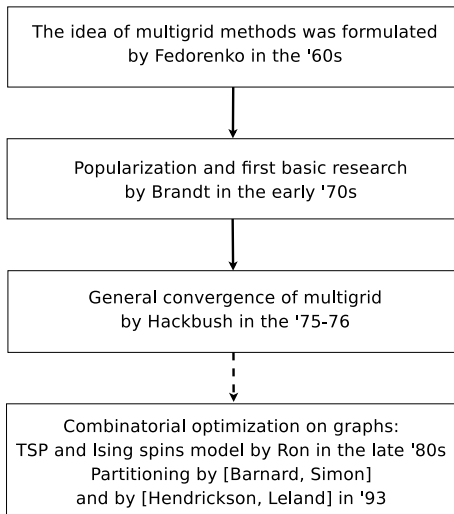
- ▶ The **Multiscale method** is a class of algorithmic techniques for solving efficiently large-scale computational and optimization problems.
- ▶ A multivariable problem defined in some space can have an approximate description at any given length scale of that space: a continuum problem can be discretized at any given resolution, multiparticle system can be represented at any given characteristic length, etc.

THE MULTISCALE METHOD: A COARSE VIEW

Multiscale \approx **Multilevel** \approx **Multigrid** \approx **Multiresolution**

- ▶ The **Multiscale method** is a class of algorithmic techniques for solving efficiently large-scale computational and optimization problems.
- ▶ A multivariable problem defined in some space can have an approximate description at any given length scale of that space: a continuum problem can be discretized at any given resolution, multiparticle system can be represented at any given characteristic length, etc.
- ▶ The multiscale algorithm recursively constructs a sequence of such descriptions at increasingly larger (coarser) scales, and combines *local* processing at each scale with various *inter-scale* interactions.

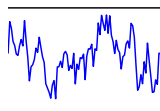
HISTORY OF MULTIGRID METHODS



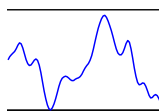
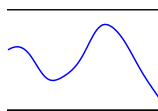
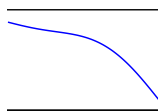
THE BASIC OBSERVATION

Observation 17.1.

A suitable relaxation can always reduce the information content of the error (by smoothing it), and quickly make it approximable by far fewer variables (which are related to the smooth error modes).



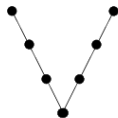
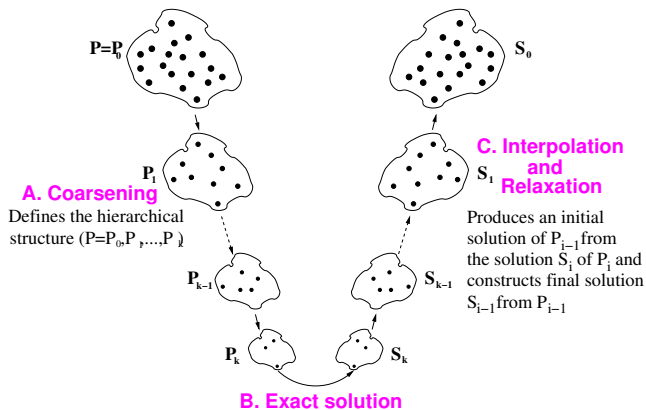
original error

 $k = 5$  $k = 10$  $k = 500$

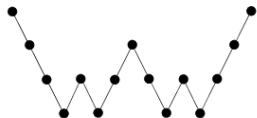
Reminder: Iterative relaxation for solving $Ax = b$

$$x^{(k+1)} = Tx^{(k)} + v$$

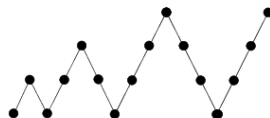
CYCLES



V-cycle



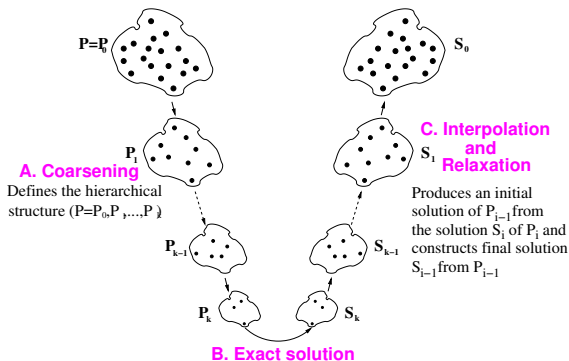
W-cycle



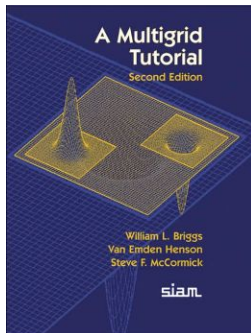
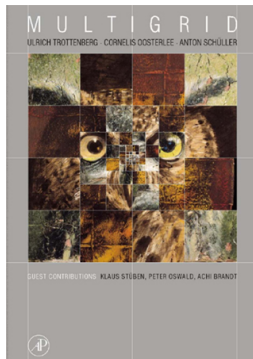
FMG-scheme

COMPUTATIONAL WORK

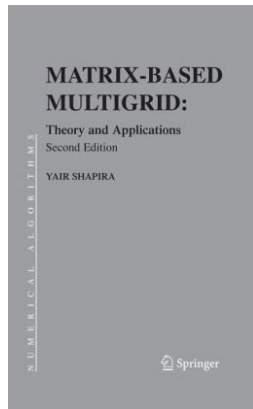
Total complexity, $\sum_i c_i m_i \approx H_{m_0}$, is **linear**, where c_i is computational work per variable and m_i is a number of variables at level i .



LITERATURE



(we will follow this book)



and many papers, surveys, and guides by Achi Brandt are available at
<http://www.wisdom.weizmann.ac.il/~achi>

TWO-LEVEL ALGORITHM

If \hat{x} is an approximate solution to $Ax = b$, with residual $r = b - A\hat{x}$, then the error $e = x - \hat{x}$ satisfies the equation $Ae = r$. Thus, in improving the approximate solution we can work with just this “residual equation” involving the error and the residual, rather than the solution and original right-hand side. One advantage of the residual equation is that zero is a reasonable starting guess for its solution. A two-grid method then takes the following form:

1. On the fine grid, use a few iterations of a smoother to compute an approximate solution \hat{x} for the system $Ax = b$.
2. Compute the residual $r = b - A\hat{x}$.
3. Restrict the residual to the coarse grid.
4. On the coarse grid, use a few iterations of a smoother on the residual equation to obtain a coarse grid approximation to the error.
5. Interpolate the coarse grid correction to the fine grid to obtain an improved approximate solution on the fine grid.
6. Apply a few iterations of a smoother to the corrected solution on the fine grid.

TWO-LEVEL ALGORITHM

Algorithm 2.1: MG-V($\mathbf{v}^h, \mathbf{f}^h$)

comment: A multigrid V-cycle.

if Ω^h is the coarsest grid

then Solve for \mathbf{v}^h in $\mathbf{A}^h \mathbf{v}^h = \mathbf{f}^h$ using a direct solver

else $\left\{ \begin{array}{l} \text{Perform } \mu \text{ smoothing iterations on } \mathbf{A}^h \mathbf{v}^h = \mathbf{f}^h \\ \text{Restrict the residual: } \mathbf{f}^{2h} \leftarrow \mathbf{R}^h(\mathbf{f}^h - \mathbf{A}^h \mathbf{v}^h) \\ \text{Set the initial guess for } \Omega^{2h}: \mathbf{v}^{2h} \leftarrow \mathbf{0} \\ \text{Recurse using } \Omega^{2h}: \mathbf{v}^{2h} \leftarrow \text{MG-V}(\mathbf{v}^{2h}, \mathbf{f}^{2h}) \\ \text{Prolong } \mathbf{v}^{2h} \text{ and correct } \mathbf{v}^h: \mathbf{v}^h \leftarrow \mathbf{v}^h + \mathbf{P}^h \mathbf{v}^{2h} \\ \text{Perform } \mu \text{ smoothing iterations on } \mathbf{A}^h \mathbf{v}^h = \mathbf{f}^h \end{array} \right.$

return (\mathbf{v}^h)

GENERAL IDEA OF THE ALGEBRAIC MULTIGRID

Brandt, McCormick, Rudge, "Algebraic Multigrid (AMG) for automatic multigrid solution with application to geodetic computations", 1982

- ▶ Given : $A \in \mathbb{R}^{n \times n}$ positive definite, symmetric.
- ▶ Goal : solve $Ax = b$.
- ▶ Claim : If A is positive definite, then

$$x \text{ minimizes } P(x) = \frac{1}{2}x^T Ax - x^T b \text{ iff } Ax = b.$$

- ▶ \tilde{x} - current approximation
- ▶ $e(rror) = x - \tilde{x}$ (hard to estimate)
- ▶ $b - A\tilde{x} = r(esidual) = A(x - \tilde{x}) = Ae$

GENERAL IDEA OF THE ALGEBRAIC MULTIGRID

At all levels : solve $Ae = r$, where $e(\text{error}) = x - \tilde{x}$ and
 $r(\text{residual}) = b - A\tilde{x}$

$$\min \frac{1}{2} e^T A e - e^T r =$$

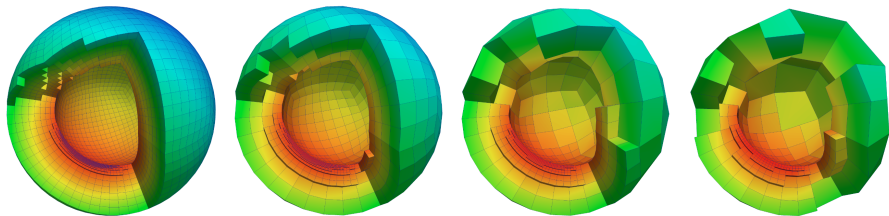
$$\min \frac{1}{2} (\tilde{e} + \uparrow_c^f e^c)^T A (\tilde{e} + \uparrow_c^f e^c) - (\tilde{e} + \uparrow_c^f e^c)^T r \leftrightarrow \dots \leftrightarrow$$

$$\min \frac{1}{2} (e^c)^T \left[(\uparrow_c^f)^T A \uparrow_c^f \right] e^c - (e^c)^T (\uparrow_c^f)^T (r - A\tilde{e}) =$$

$$\min \frac{1}{2} (e^c)^T A^c e^c - (e^c)^T r^c$$

- ▶ \tilde{e} - initial fine level error
- ▶ e^c - coarse level error
- ▶ \uparrow_c^f - coarse-to-fine interpolation operator

WHAT WE MEAN BY A GRID: GEOMETRIC CASE



Variables are defined at known spatial locations (grid points)

WHAT WE MEAN BY A GRID: ALGEBRAIC CASE

$$A = \begin{bmatrix} X & X & & X & X & & \\ X & X & X & X & & & \\ & & X & X & X & X & X \\ X & X & X & X & & & \\ X & & & & X & X & \\ & & X & & X & X & \end{bmatrix}$$

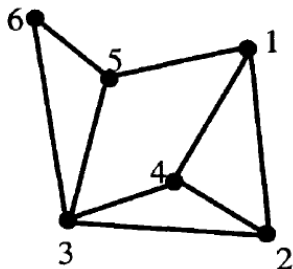


Figure 8.1: The nonzero structure of A , where X indicates a nonzero entry, is shown on the left. The resulting undirected adjacency graph appears on the right.

HOW TO CHOOSE A GRID

Now that we can represent the fine grid, how do we select a coarse grid? With standard multigrid methods, smooth functions are geometrically or physically smooth; they have a low spatial frequency. In these cases, we assume that relaxation smooths the error and we select a coarse grid that represents smooth functions accurately. We then choose intergrid operators that accurately transfer smooth functions between grids.

With AMG, the approach is different. We first select a relaxation scheme that allows us to determine the nature of the smooth error. Because we do not have access to a physical grid, the sense of smoothness must be defined algebraically. The next step is to use this sense of smoothness to select coarse grids, which will be subsets of the unknowns. A related issue is the choice of intergrid transfer operators that allow for effective coarsening. Finally, we select the coarse-grid versions of the operator A , so that coarse-grid correction has the same effect that it has in geometric multigrid: it must eliminate the error components in the range of the interpolation operator.

ALGEBRAIC SMOOTHNESS

We also assume that A is a symmetric M -matrix: it is symmetric, pod, has positive diagonal entries, and nonpositive off-diagonal entries. These properties are shared by matrices arising from the discretization of many scalar elliptic differential equations (not necessary for AMG to work). We solve $Au = f$.

- ▶ Weighted point Jacobi $v \leftarrow v + \omega D^{-1}(f - Av)$
- ▶ Error propagation is $e \leftarrow (I - \omega D^{-1}A)e$
- ▶ WJ has slow convergence; When we reach an iteration at which no progress is observed we say that current error is algebraically smooth, i.e., $0 < e_i - e_{i+1} < \epsilon$.
- ▶ More correct way to define it is by $\|e\|_A = (Ae, e)^{\frac{1}{2}}$, i.e., we say that current error is algebraically smooth if $\|(I - \omega D^{-1}A)e\|_A \approx \|e\|_A$.
- ▶ It can be shown that WJ gives $r_i \ll a_{ii}|e_i|$ and thus $Ae \rightarrow$ (or \approx) 0 . Same can be done with GS, JAC, and other relaxations.

INFLUENCE AND DEPENDENCE

Important implication: if error is smooth it can be well approximated by its neighbors, i.e.,

$$a_{ii}e_i \approx - \sum_{j \neq i} a_{ij}e_j.$$

This principle is used by many different multiscale algorithms.

Second fundamental concept is **strong dependence or strong influence**.

Which u_j are most important in the i th equation in determining u_i ?

INFLUENCE AND DEPENDENCE

Important implication: if error is smooth it can be well approximated by its neighbors, i.e.,

$$a_{ii}e_i \approx - \sum_{j \neq i} a_{ij}e_j.$$

This principle is used by many different multiscale algorithms.

Second fundamental concept is **strong dependence or strong influence**.

Which u_j are most important in the i th equation in determining u_i ?

Answer in classical AMG approach: if the coefficient, a_{ij} , which multiplies u_j in the i th equation, is large relative to the other coefficients in the i th equation, then a small change in the value of u_j has more effect on the value of u_i than a small change in other variables in the i th equation.

Definition.

Given $0 < \Theta \leq 1$, the variable u_i strongly depend on the variable u_j if

$$-a_{ij} \geq \Theta \max_{k \neq i} \{-a_{ik}\}.$$

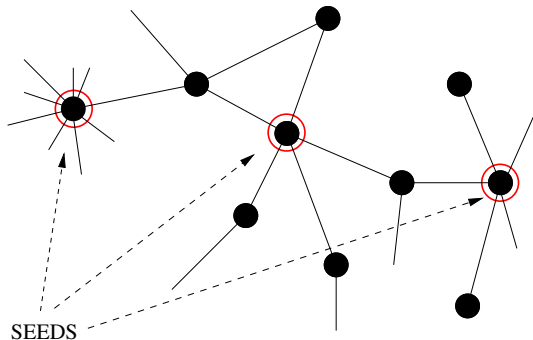
INFLUENCE AND DEPENDENCE

In other words, grid point i strongly depends on grid point j if the coefficient a_{ij} is comparable in magnitude to the largest off-diagonal coefficient in the i th equation.

With the concepts of smooth error and strong influence/dependence in hand, we can return to the task of defining the multigrid components for AMG. We begin by defining a two-grid algorithm, then proceed to multigrid by recursion. Having defined the relaxation scheme, we have several tasks before us:

- ▶ select a coarse grid so that the smooth components can be represented accurately;
- ▶ define an interpolation operator so that the smooth components can be accurately transferred from the coarse grid to the fine grid; and
- ▶ define a restriction operator and a coarse-grid version of A using the variational properties.

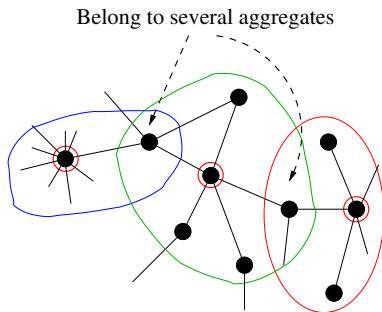
COARSE VARIABLES: C-POITS SELECTION



- ▶ Choose a dominating set $C \subset V$ s.t. all others from $F = V \setminus C$ are “strongly coupled” to C .
- ▶ Each chosen variable will be a *seed* of a coarse aggregate.

COARSE VARIABLES: THE INTERPOLATION WEIGHTS

$$(\uparrow_c^f)_{iJ} = \begin{cases} \frac{a_{ij}}{\left(\sum_{k \in N(i)} a_{ik}\right)} & i \in F, j \in N(i) \\ 1 & i \in C, j = i \\ 0 & \text{otherwise} \end{cases}$$



- ▶ Define the interpolation weights of all variables
- ▶ In some sense, the interpolation weights are the probabilities of a variable to share a common property with the aggregates it belongs to.

INTERPOLATION OPERATOR

Precisely speaking we need to define interpolation operator \uparrow_{2h}^h such that the i th component will be

$$(\uparrow_{2h}^h e)_i = \begin{cases} e_i & \text{if } i \in C \\ \sum_{j \in C_N(i)} w_{ij} e_j & \text{if } i \in F. \end{cases}$$

Recall that the main characteristic of smooth error is that the residual is small: $r \approx 0$. We can write the i th component of this condition as

$$a_{ii}e_i \approx - \sum_{j \in N_i} a_{ij}e_j$$

Splitting the sum into its component sums over the coarse interpolatory set, C_i , the fine-grid points with strong influence, D_i^s , and the weakly connected neighbors, D_i^w , we have

$$a_{ii}e_i \approx - \sum_{j \in C_i} a_{ij}e_j - \sum_{j \in D_i^s} a_{ij}e_j - \sum_{j \in D_i^w} a_{ij}e_j$$

INTERPOLATION OPERATOR

Interpolation weights are given by

$$w_{ij} = - \frac{a_{ij} + \sum_{m \in D_i^s} \left(\frac{a_{im} a_{mj}}{\sum_{k \in C_i} a_{mk}} \right)}{a_{ii} + \sum_{n \in D_i^w} a_{in}}$$

- ▶ **Selecting coarse grid.** By examining the suitability of each grid point to be a point of one of the C_i sets, we make an initial partitioning of the grid points into C - and F -points. Then, as the interpolation operator is constructed, we make adjustments to this partitioning, changing points initially chosen as F -points to be C -points in order to ensure that the partitioning conforms to certain heuristic rules. Denote by S_i the set of points that strongly influence i .
- ▶ Rule 1: For each F -point i , every point $j \in S_i$ that strongly influences i either should be in the coarse interpolatory set C_i or should strongly depend on at least one point in C_i .
- ▶ Rule 2: The set of coarse points C should be a maximal subset of all points with the property that no C -point strongly depends on another C -point.

COARSE GRID

Although physical grids may not be present, they are usually denoted by: h for fine-grid quantities; and $2h$ for coarse-grid quantities. Once the coarse grid is chosen and the interpolation operator \uparrow_{2h}^h is constructed, the restriction operator is defined using the usual variational property

$$\uparrow_h^{2h} = (\uparrow_{2h}^h)^T$$

The coarse-grid operator is constructed using the Galerkin condition

$$A^{2h} = \uparrow_h^{2h} A^h \uparrow_{2h}^h .$$

AMG Two-Grid Correction Cycle

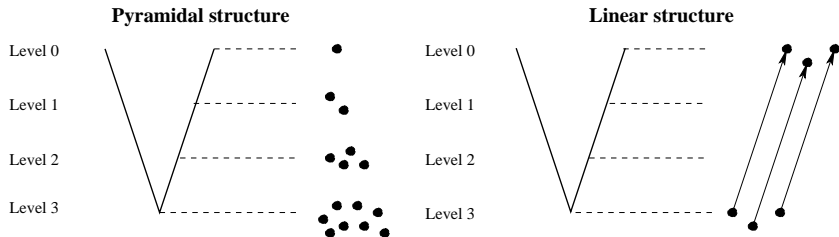
$$\mathbf{v}^h \leftarrow \text{AMG}(\mathbf{v}^h, \mathbf{f}^h).$$

- Relax ν_1 times on $A^h \mathbf{u}^h = \mathbf{f}^h$ with initial guess \mathbf{v}^h .
- Compute the fine-grid residual $\mathbf{r}^h = \mathbf{f}^h - A^h \mathbf{v}^h$ and restrict it to the coarse grid by $\mathbf{r}^{2h} = I_h^{2h} \mathbf{r}^h$.
- Solve $A^{2h} \mathbf{e}^{2h} = \mathbf{r}^{2h}$ on Ω^{2h} .
- Interpolate the coarse-grid error to the fine grid by $\mathbf{e}^h = I_{2h}^h \mathbf{e}^{2h}$ and correct the fine-grid approximation by $\mathbf{v}^h \leftarrow \mathbf{v}^h + \mathbf{e}^h$.
- Relax ν_2 times on $A^h \mathbf{u}^h = \mathbf{f}^h$ with initial guess \mathbf{v}^h .

A^{ph}	Number of rows	Number of nonzeros	Density (% full)	Average entries per row
A^h	4096	20224	0.001	4.9
A^{2h}	2048	17922	0.004	8.8
A^{4h}	542	4798	0.016	8.9
A^{8h}	145	1241	0.059	8.6
A^{16h}	38	316	0.219	8.3
A^{32h}	12	90	0.625	7.5
A^{64h}	5	23	0.920	4.6

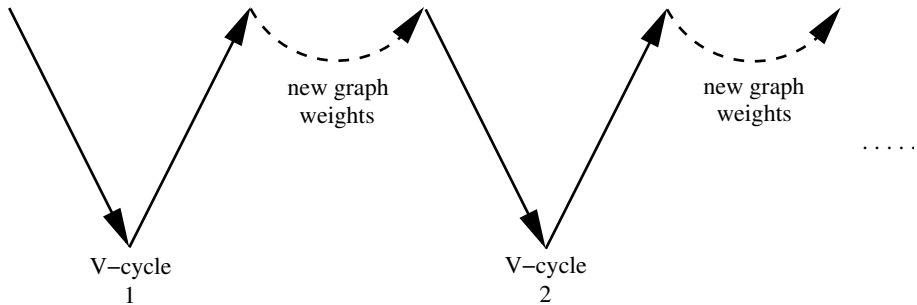
Table 8.2: *Properties of A^{ph} , $p = 2^k$, $k = 0, 1, \dots, 6$, for AMG applied to a two-dimensional problem.*

ADDITIONAL SOLUTIONS AT THE COARSEST LEVEL



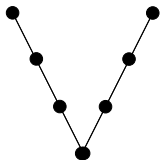
How to choose a set of solutions at the coarsest level ?

MULTIPLE V-CYCLES

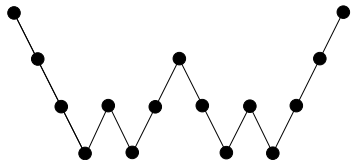


► details

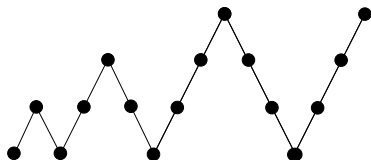
SCHEDULING SCALES



V-cycle

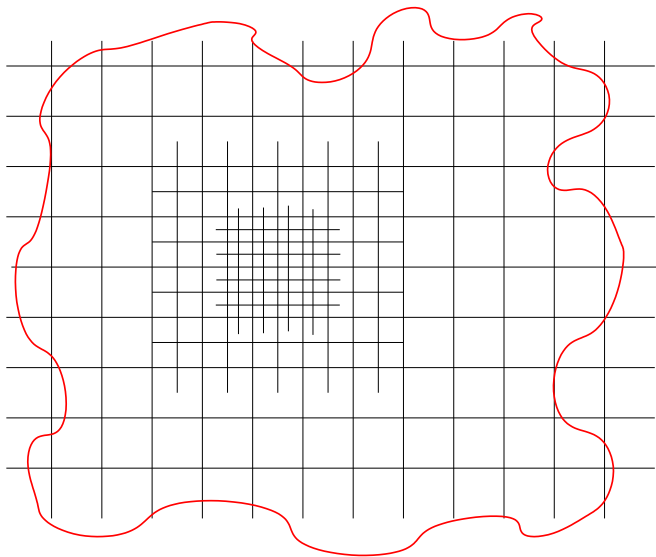


W-cycle



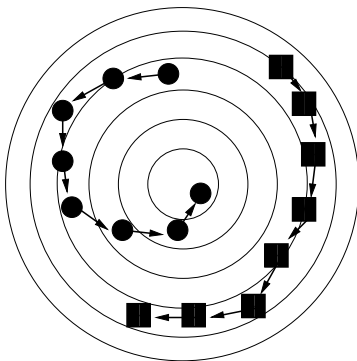
FMG-scheme

ADAPTIVITY

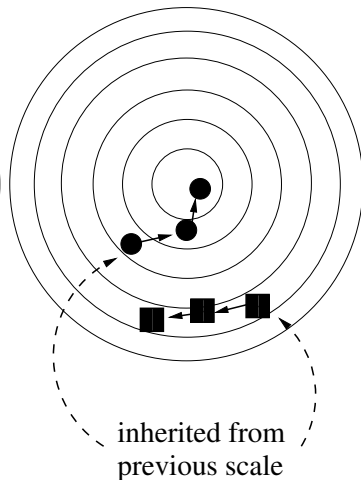


STOCHASTICITY

Regular scheme



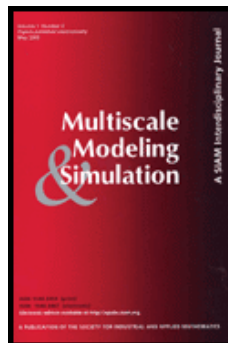
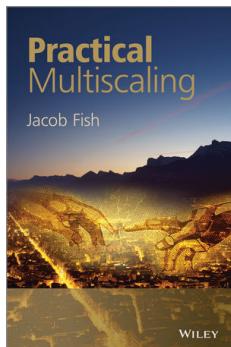
Multiscale scheme



Subsection 2

Examples of Multiscaling

LITERATURE

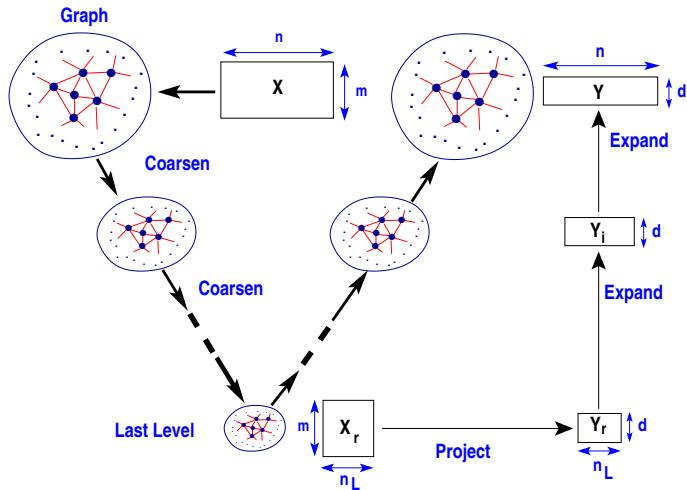


DIMENSIONALITY REDUCTION PROBLEM

Problem 17.1.

Given a set of high dimensional data represented by vectors x_1, \dots, x_n in R^m , the task is to represent these with low dimensional vectors $y_1, \dots, y_n \in R^d$ with $d \ll m$, such that nearby points remain nearby, and distant points remain distant.

MULTISCALE DIMENSIONALITY REDUCTION



Fang, Sakellari, Saad, "Multilevel Nonlinear Dimensionality Reduction for Manifold Learning", 2009

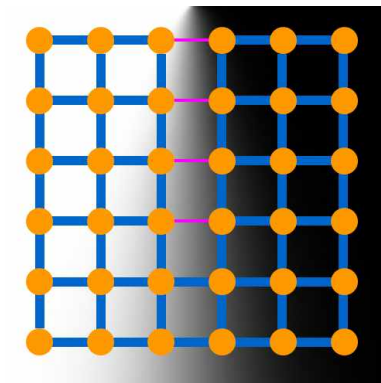
MULTISCALE IMAGE SEGMENTATION

Sharon, Galun, Sharon, Basri and Brandt, "Hierarchy and adaptivity in segmenting visual scenes", *Nature*, 2006



THE PIXEL GRAPH

Low contrast - strong coupling
High contrast - weak coupling

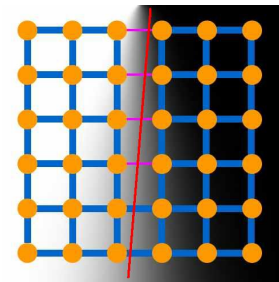


THE PIXEL GRAPH

Segmentation \equiv Low-energy cut

$$\text{minimize } \Gamma(u) = \frac{\sum_{i>j} w_{ij}(u_i - u_j)^2}{\sum_{i>j} w_{ij}u_iu_j}$$

Any boolean u that yields a low-energy $\Gamma(u)$ corresponds to a salient segment



THE PIXEL GRAPH

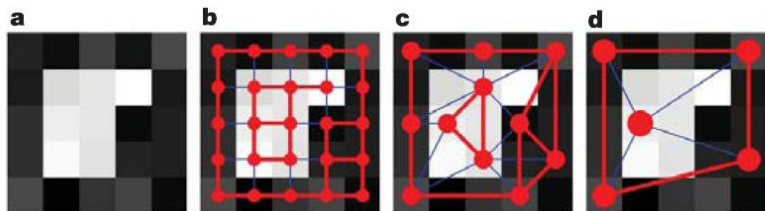


Figure 2 | The multiscale normalized cut graph approach. **a**, A simple image. **b**, Pixels of the image are nodes, represented by filled circles; strong coupling is represented by thick red lines, and weak coupling by thin blue lines. **c**, Adaptive coarsening. Each pixel in **b** is strongly coupled to one of the chosen seeds shown here (thus, pixels strongly coupled to a given seed form an aggregate). Couplings between the seeds are shown. **d**, An additional coarsening level. In this case, this is the level at which the salient segment is detected.

IMAGE SEGMENTATION



Spectral normalized cuts



Multilevel normalized cuts

MULTISCALE CLUSTERING

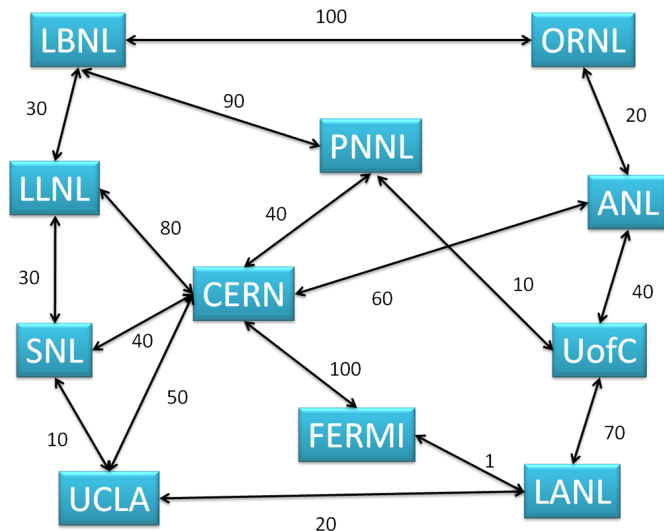
Problem 17.2.

Given a data set, clustering algorithms seek a partition of the data to coherent groups, in a sense that data points in the same group share similar properties. Many approaches try to solve the clustering problem by optimizing a global cost function, expressed in terms of the local similarities between data points.

(*)

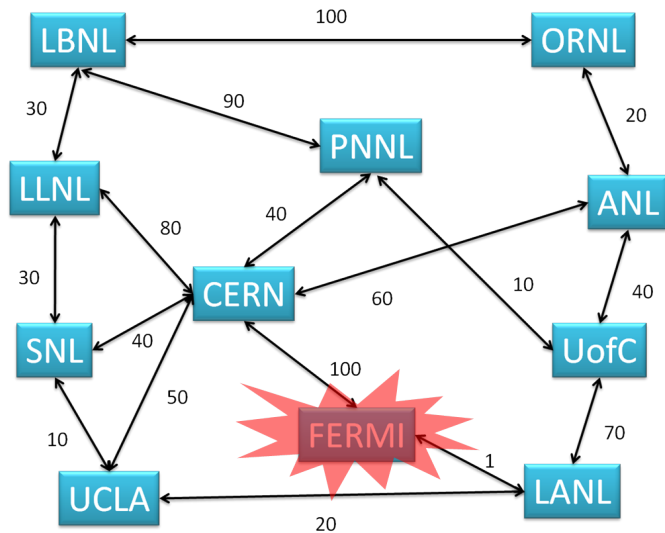
- ▶ Dhillon, Guan, Kulis, "A Fast Kernel-based Multilevel Algorithm for Graph Clustering"
- ▶ Goldschmidt, Galun, Sharon, Basri, Brandt, "Fast Multilevel Clustering"
- ▶ Karypis, Han, Kumar, "Multilevel Refinement for Hierarchical Clustering"
- ▶ Oliveira, Seok, "A Multilevel Approach for Document Clustering"
- ▶ Kushnir, Galun, Brandt, "Fast multiscale clustering and manifold identification" (*)

RESPONSE TO EPIDEMICS, CYBER ATTACKS



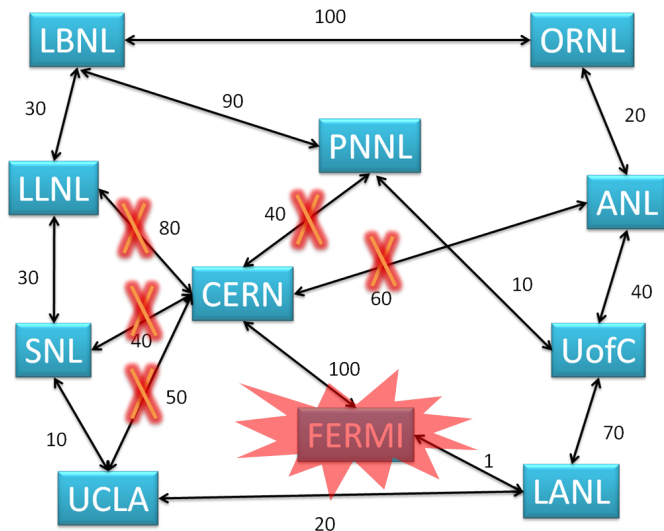
Open Science Grid: collaboration network example

RESPONSE TO EPIDEMICS, CYBER ATTACKS



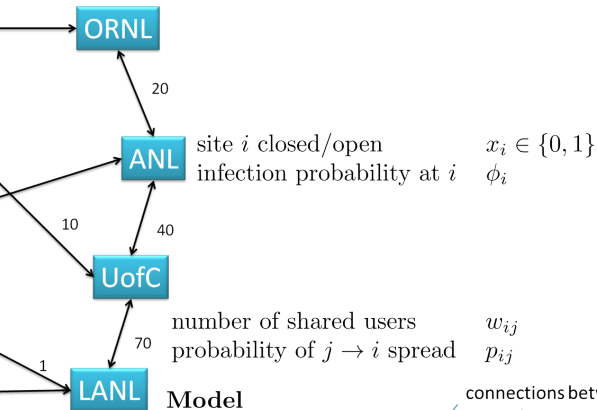
Open Science Grid: collaboration network example

RESPONSE TO EPIDEMICS, CYBER ATTACKS



Open Science Grid: collaboration network example

RESPONSE TO EPIDEMICS, CYBER ATTACKS



Model

maximize x

subject to

infection at node i is less than some constant

connections between open sites

$$\sum_{ij \in E} w_{ij} x_i x_j$$

$$x_i - \prod_{j \in N(i)} (1 - p_{ij} \phi_j x_j) \leq t_i \quad \forall i \in V$$

$$x \in \{0, 1\}^n$$

function MSSolve(G)

if G is small **then**

$S_f \leftarrow$ solve the problem
exactly

else

order infected nodes

find coarse variables

$G_c \leftarrow$ create coarse graph

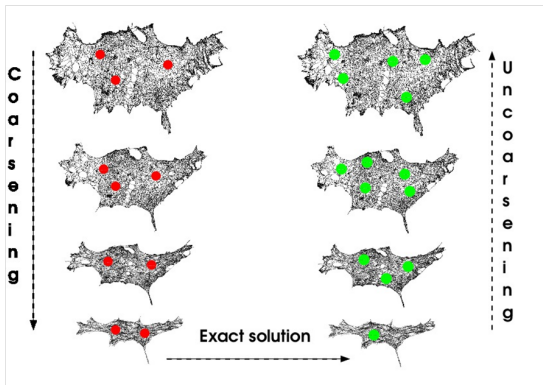
$S_c \leftarrow$ MSSolve(G_c)

$S_f \leftarrow$ Interpolate(S_c)

$S_f \leftarrow$ LocalRefinement(S_f)

end if

return S_f



Coarsening

sum-of-degrees matrix



adjacency matrix

Jacobi over-relaxation $H = (1 - \omega)I + \omega D^{-1}W$

Algebraic distance is a strength of connection $\varrho_{ij}^{(k)} = \left(\sum_{r=1}^R |\chi_i^{(k,r)} - \chi_j^{(k,r)}|^p \right)^{1/p}$, where $\chi^{(k,r)} = H^k \chi^{(0,r)}$

Used as sparsification for Galerkin and in detection of coarse variables

Ron, S, Brandt "Relaxation-based coarsening and multiscale graph organization"



Coarse model

New linear term

maximize X

$$\sum_{ij \in E_c} W_{ij} X_i X_j + \sum_{i \in V_c} A_i X_i$$

Links between accumulated nodes

subject to

$$X_i - \prod_{j \in N(i)} (1 - P_{ij} \Phi_j X_j) \leq T_i \quad \forall i \in V_c$$

$$X \in \{0, 1\}^n$$

$P_{ij}, W_{ij}, \Phi_i, X_i, T_i \leftarrow$ AMG coarsening,
Galerkin reinforced by algebraic distance

$$\text{maximize}_x \quad \sum_{i,j \in S} w_{ij} x_i x_j + \sum_{i \in S, j \notin S} w_{ij} x_i \tilde{x}_j + \sum_{i \in S} a_i x_i$$

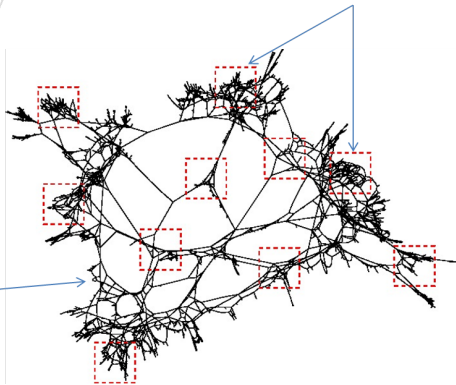
$$\text{subject to} \quad x_i - k_i \prod_{\substack{j \in N(i) \\ j \in S}} (1 - p_{ij} \phi_{j,t-1} x_j) \leq b_i \quad \forall i \in V$$

$$x_i \in \{0, 1\} \quad \forall i \in V$$

$$k_i = \prod_{j \in N(i), j \notin S} (1 - p_{ij} \phi_{j,t-1} \tilde{x}_j)$$

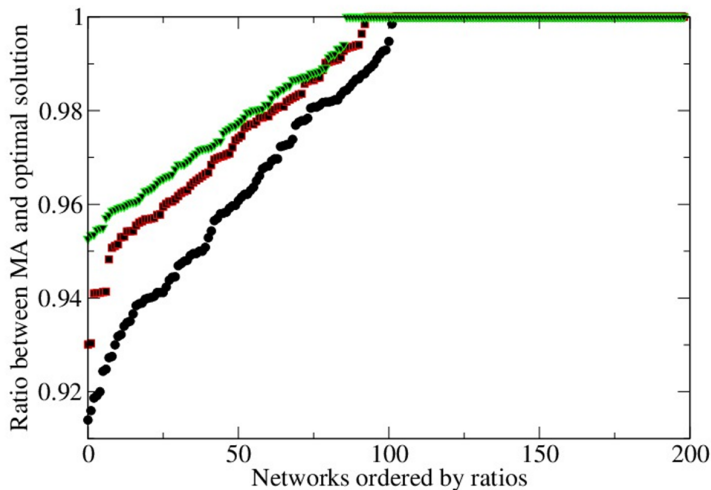
Boundary conditions

Local refinement

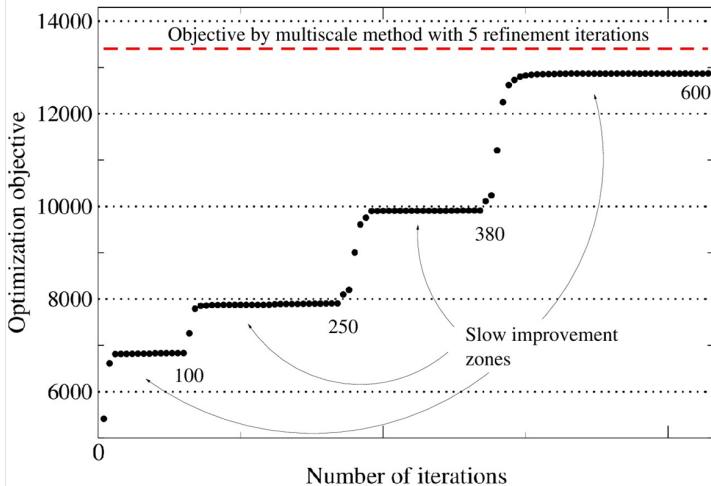


Small random graphs, < 80 nodes, < 400 edges

Erdos-Renyi, Barabasi-Albert, and R-MAT models

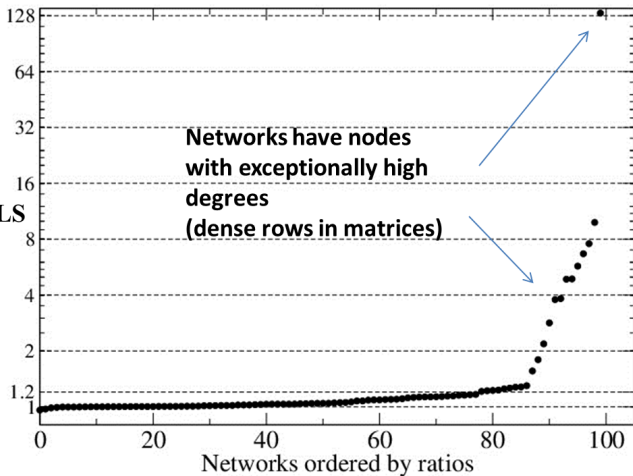


Iterated Local Search vs Multiscale HIV spread model



Large-scale networks

Ratios
Between
MA and ILS



Network Generation, A Practical Approach

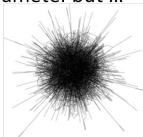
Theoretical questions

- What processes form a network?
- How to predict its future structure?
- Why should network have property X?

Practical question

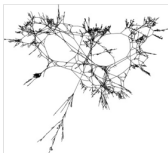
- **Will my algorithm/heuristic work on networks created by similar processes?**

This artificial network has similar degrees, some eigs, diameter but ...



Is it really similar to the original network?

Artificial network



Artificial network



Artificial network



Artificial network



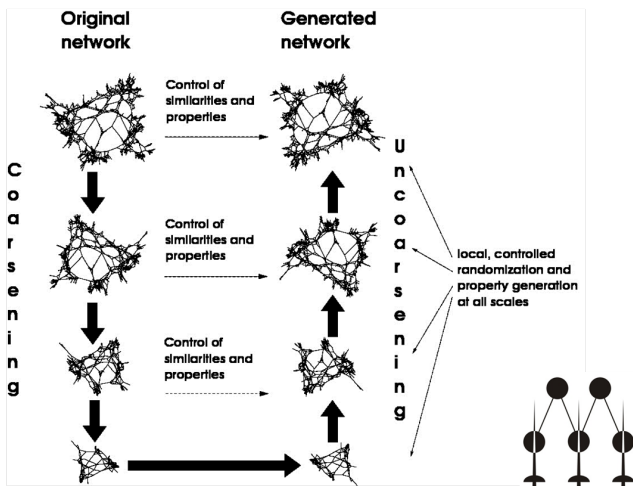
Original network



Properties taken into account by most of the existing network generators
 degree distribution, clustering coefficient, some eigenvalues, diameter, etc.

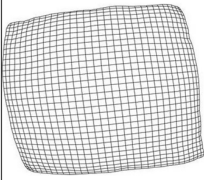
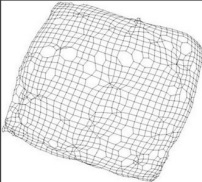
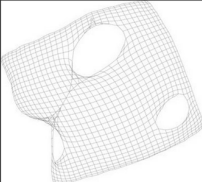
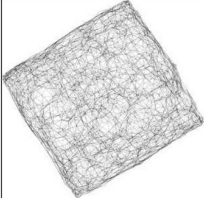
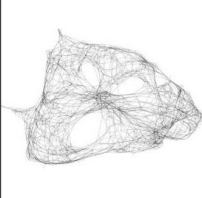
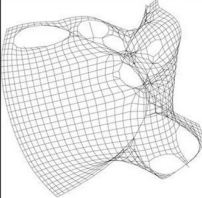
They are different at different resolutions; no similarity at coarse resolutions; usual
 operations (randomization, replication) take us away from the realistic structure

MULTI- SCALE ENTROPIC NETWORK GENERATOR




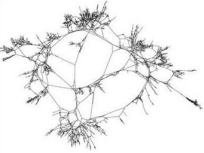

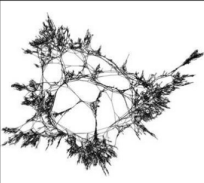
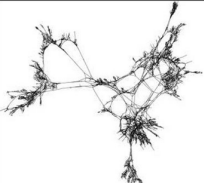
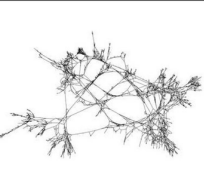
Toy Example: Mesh 33x33 by



Original graph: mesh 33x33	Generation with local changes	Generation with small number of global changes
		
Number of generated nodes is 3 times bigger	Global changes and number of generated nodes is 3 times bigger	Generation with small number of global changes
		

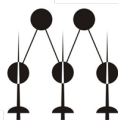
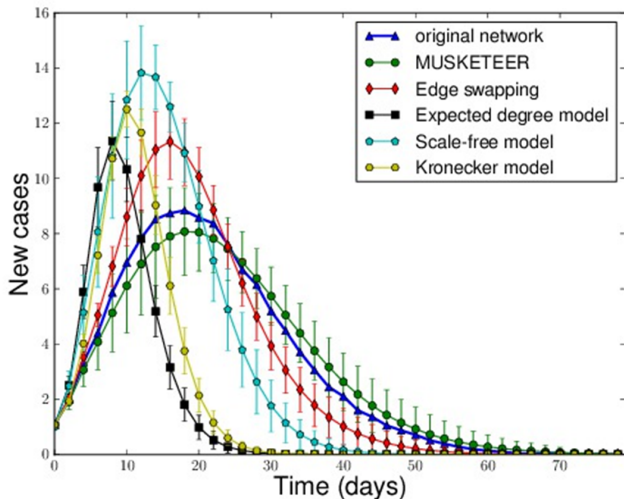
Example: Power Grid by



Original graph: US western states power grid, Watts, Strogatz, Nature, 1998	Generation with local changes	Generation with small number of global changes
		
Number of generated nodes is 3 times bigger	Global changes and number of generated nodes is twice bigger	Generation with small number of global changes
		

SEIR cascade on Colorado Springs Network

susceptible \rightarrow exposed \rightarrow recovered \rightarrow susceptible



Homework 23 (RT).

- ▶ Read handout about AMG
- ▶ Solve Ex 11

Homework 23 (RT).

- ▶ Read handout about AMG
- ▶ Solve Ex 11

Homework 24 (I).

- ▶ Download and install PyAMG <http://pyamg.org/>
- ▶ Generate and solve Poisson matrix A of size 500×500 with random b
- ▶ Print out both A , and b
- ▶ Upload both A and b to Matlab and solve $Ax = b$
- ▶ Compare running times

Subsection 3

Nonlinear Multigrid

NONLINEAR MULTIGRID

(see “Multigrid methods for nonlinear problems: an overview” by Van E. Henson)

Reminder: Classical multigrid begins with a two-grid process. First, iterative relaxation is applied, whose effect is to smooth the error. Then a coarse-grid correction is applied, in which the smooth error is determined on a coarser grid. This error is interpolated to the fine grid and used to correct the fine-grid approximation. Applying this method recursively to solve the coarse-grid problem leads to multigrid.

- ▶ The coarse-grid correction works because the residual equation is linear. But this is not the case for nonlinear problems.
- ▶ Nonlinear problems necessarily must be solved using iterative methods, and for this reason it is natural to expect that multigrid ideas should be effective on these problems.

- ▶ GMG, AMG: We solve

$$Au = f \Rightarrow r = f - Av \Rightarrow Ae = r \Rightarrow r^{2h} = I^{2h}r^h \Rightarrow A^{2h}e^{2h} = r^{2h}$$

- ▶ Nonlinear MG: consider a system of *nonlinear* equations, $A(u) = f$, where $u, f \in \mathbb{R}^n$
- ▶ The error is given by $e = u - v$ while now $r = f - A(v)$ is the residual. Subtracting the original equation from the residual, we obtain

$$A(u) - A(v) = r \tag{11}$$

- ▶ A is nonlinear, in general $A(e) \neq r$, implying that for the nonlinear problem we can not determine the error by solving $Ae = r$ on the coarse grid, as in GMG/AMG.

- ▶ GMG, AMG: We solve $Au = f \Rightarrow r = f - Av \Rightarrow Ae = r \Rightarrow r^{2h} = I^{2h}r^h \Rightarrow A^{2h}e^{2h} = r^{2h}$
- ▶ Nonlinear MG: consider a system of *nonlinear* equations, $A(u) = f$, where $u, f \in \mathbb{R}^n$
- ▶ The error is given by $e = u - v$ while now $r = f - A(v)$ is the residual. Subtracting the original equation from the residual, we obtain

$$A(u) - A(v) = r \quad (11)$$

- ▶ A is nonlinear, in general $A(e) \neq r$, implying that for the nonlinear problem we can not determine the error by solving $Ae = r$ on the coarse grid, as in GMG/AMG.

There are two basic approaches to using multigrid in (11).

- ▶ use multigrid as the linear solver in a standard linearization, such as in Newtons method.
- ▶ use Full Approximation Scheme (FAS), that is to apply multigrid methodology directly to the original equation $A(u) = f$ and to base the coarse-grid correction on the nonlinear residual equation.

NEWTON'S MULTIGRID

Suppose we solve the scalar equation $F(x) = 0$. By expanding F in a Taylor series around x_0 we obtain

$$F(x_0 + s) = F(x_0) + sF'(x_0) + \frac{s^2}{2}F''(a)$$

for some $a \in (x_0, x_0 + s)$. The Newton iteration is

$$x_j \leftarrow x_j - \frac{F(x_j)}{F'(x_j)}.$$

We use (11) as a basis for multigrid by applying NM to the system

$$A(x) = \begin{pmatrix} A_1(x_1, x_2, \dots, x_n) \\ \dots \\ A_n(x_1, x_2, \dots, x_n) \end{pmatrix} = \begin{pmatrix} 0 \\ \dots \\ 0 \end{pmatrix}.$$

Let $J(v) = \left(\frac{\partial A_i}{\partial x_j} \right)_{ij}$ be its Jacobian, and if $u = v + e$ is the exact solution, the Taylor series for the system is

$$A(v + e) = A(v) + J(v)e + \text{high order terms}$$

Neglecting HOT and subtracting $A(v)$ from both sides we have

$$J(v)e = r,$$

that is an approximation to the nonlinear system (11).

Let $J(v) = \left(\frac{\partial A_i}{\partial x_j}\right)_{ij}$ be its Jacobian, and if $u = v + e$ is the exact solution, the Taylor series for the system is

$$A(v + e) = A(v) + J(v)e + \text{high order terms}$$

Neglecting HOT and subtracting $A(v)$ from both sides we have

$$J(v)e = r,$$

that is an approximation to the nonlinear system (11). It can be solved for e and the current approximation of v can be updated by $v \leftarrow v + e$. Iteration step is a form of Newton's method

$$v_j \leftarrow v_j + J^{-1}(v_j)(f_j - A(v_j)).$$

By using multigrid to solve this linear system at each step we obtain a combination of Newton's method for the outer iteration and multigrid for the (linear) inner iteration.

NONLINEAR MULTIGRID

$$A(u)=f$$

$$e(\text{rror}) = u-v$$

$$r(\text{esidual}) = f - A(v)$$

A is linear \longrightarrow

$$Ae = r$$

Algebraic Multigrid
Geometric Multigrid

A is nonlinear

$A(e) \neq r$ $\xrightarrow{\text{Newton Multigrid}}$

$$A(v+e)=A(v)+J(v)e+\text{high order terms}$$

solve $J(v)e=r$
 $v_j = v_j + J^{-1}(v_j)(f_j - A(v_j))$

Full Approximation Scheme

FULL APPROXIMATION SCHEME

While Newton-MG is often an extremely effective method, it does not use multigrid ideas to treat the nonlinearity directly. To do this, we return to the residual equation (11) and use it to determine a coarse-grid correction. Suppose we have found an approximation, v^h , to the original fine-grid problem

$$A^h(u^h) = f^h.$$

Then the coarse-grid version of 11 is

$$A^{2h}(v^{2h} + e^{2h}) - A^{2h}(v^{2h}) = r^{2h}.$$

FULL APPROXIMATION SCHEME

- ▶ Restrict the fine-grid approximation $r^{2h} = I_h^{2h} r^h = I_h^{2h}(f^h - A^h(v^h))$
and $v^{2h} = I_h^{2h} v^h$

$$A^{2h}(\underbrace{I_h^{2h} v^h + e^{2h}}_{u^{2h}}) = \underbrace{A^{2h}(I_h^{2h} v^h) + I_h^{2h}(f^h - A^h(v^h))}_{f^{2h}}$$

- ▶ Solve the coarse-grid problem $A^{2h}(u^{2h}) = A^{2h}(v^{2h}) + r^{2h}$
- ▶ Compute coarse-grid approximation to $e^{2h} = u^{2h} - v^{2h}$
- ▶ Interpolate the error approximation $v^h \leftarrow v^h + I_{2h}^h e^{2h}$

FULL APPROXIMATION SCHEME

- ▶ Restrict the fine-grid approximation $r^{2h} = I_h^{2h} r^h = I_h^{2h}(f^h - A^h(v^h))$
and $v^{2h} = I_h^{2h} v^h$

$$A^{2h}(\underbrace{I_h^{2h} v^h + e^{2h}}_{u^{2h}}) = \underbrace{A^{2h}(I_h^{2h} v^h) + I_h^{2h}(f^h - A^h(v^h))}_{f^{2h}}$$

- ▶ Solve the coarse-grid problem $A^{2h}(u^{2h}) = A^{2h}(v^{2h}) + r^{2h}$
- ▶ Compute coarse-grid approximation to $e^{2h} = u^{2h} - v^{2h}$
- ▶ Interpolate the error approximation $v^h \leftarrow v^h + I_{2h}^h e^{2h}$

Rewrite

$$A^{2h}(u^{2h}) = f^{2h} + \tau_h^{2h}, \text{ where } \tau_h^{2h} = A^{2h}(I_h^{2h} v^h) - I_h^{2h} A^h(v^h)$$

FULL APPROXIMATION SCHEME

- ▶ Restrict the fine-grid approximation $r^{2h} = I_h^{2h} r^h = I_h^{2h}(f^h - A^h(v^h))$
and $v^{2h} = I_h^{2h} v^h$

$$A^{2h}(\underbrace{I_h^{2h} v^h + e^{2h}}_{u^{2h}}) = \underbrace{A^{2h}(I_h^{2h} v^h) + I_h^{2h}(f^h - A^h(v^h))}_{f^{2h}}$$

- ▶ Solve the coarse-grid problem $A^{2h}(u^{2h}) = A^{2h}(v^{2h}) + r^{2h}$
- ▶ Compute coarse-grid approximation to $e^{2h} = u^{2h} - v^{2h}$
- ▶ Interpolate the error approximation $v^h \leftarrow v^h + I_{2h}^h e^{2h}$

Rewrite

$$A^{2h}(u^{2h}) = f^{2h} + \tau_h^{2h}, \text{ where } \tau_h^{2h} = A^{2h}(I_h^{2h} v^h) - I_h^{2h} A^h(v^h)$$

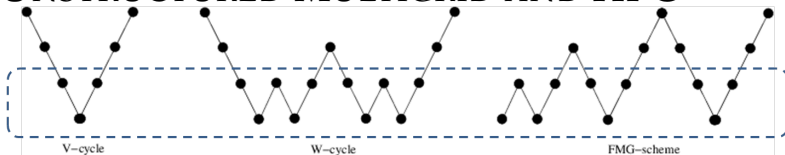
FAS

Full Approximation Scheme (FAS)

- Restrict the fine-grid approximation and its residual: $\mathbf{r}^{2h} = I_h^{2h}(\mathbf{f}^h - A^h(\mathbf{v}^h))$ and $\mathbf{v}^{2h} = I_h^{2h}\mathbf{v}^h$.
- Solve the coarse-grid problem $A^{2h}(\mathbf{u}^{2h}) = A^{2h}(\mathbf{v}^{2h}) + \mathbf{r}^{2h}$.
- Compute the coarse-grid approximation to the error: $\mathbf{e}^{2h} = \mathbf{u}^{2h} - \mathbf{v}^{2h}$.
- Interpolate the error approximation to the fine grid and correct the current fine-grid approximation: $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\mathbf{e}^{2h}$.

It is easy to see that if A is a linear operator, then FAS reduces directly to the linear coarse-grid correction scheme. Hence, FAS is in fact a generalization of the coarse-grid correction to nonlinear problems. It is also useful to observe that \mathbf{v}^h is a fixed point of the FAS iteration if and only if it is an exact solution of the fine-grid problem.

UNSTRUCTURED MULTIGRID AND HPC



Multigrid for unstructured systems does not scale well on distributed memory HPC! **Reason: coarse scales density.**

Possible solutions:

1. Reduced interpolation order (price – no full proofs yet)
2. Combinatorial sparsification (price – hidden constants and nearly linear time)
3. Combination of 1 and 2
Example: Lean AMG [LB12]
4. New HPC with faster communication for relatively small but dense scales

