Chapter 1
Error analysis and linear equations

Numerical Methods for Visual Computing
WS 17/18

Michael Moeller
Visual Scene Analysis
Department of Computer Science
and Electrical Engineering
University of Siegen

Error analysis

Conceptual approach in numerics in general

Conceptual steps to solve a problem with numerical methods:

1. **Problem formulation**: What is the problem at hand?
2. **Modeling**: How can we express the problem mathematically?
3. **Discretization**: While many (physical) processes are described in a continuous world the computer needs to find a finite/discrete representation.
4. **Algorithm**: How can we solve the discretized problem?
5. **Implementation**: Code and run the algorithm.

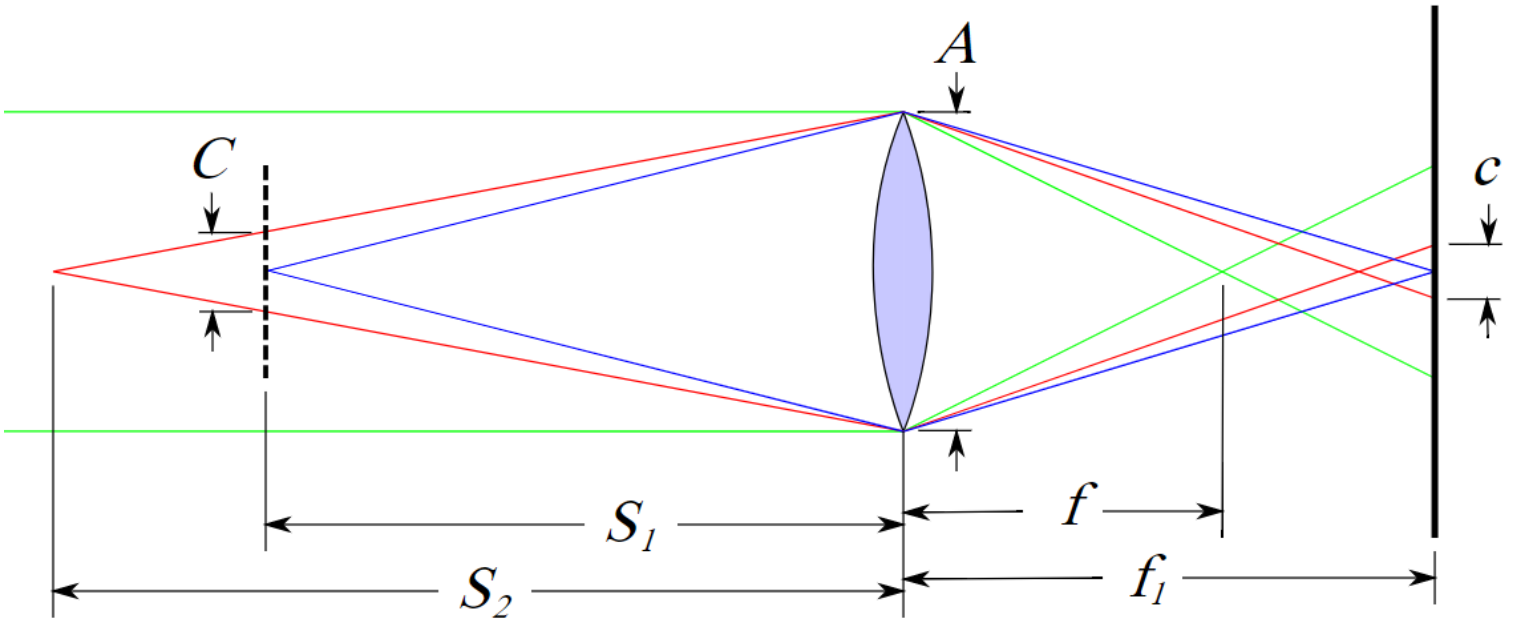
Each of these steps can lead to errors!

Example

1. Problem formulation: Simulate an out-of-focus blur given a sharp image!

Example

2. Modeling



From Wikipedia, https://en.wikipedia.org/wiki/Circle_of_confusion

- Physics: The camera settings tell me something about the *circle of confusion*.
- Modeling: The light-intensity distribution of an out-of-focus blur is Gaussian with a standard deviation which is a constant times the radius of the circle of confusion.

Example

2. Modelling: The light intensity $f(x)$ at a position x of the out of focus blur can be computed from the light intensity u of the sharp image via

$$f(x) = \frac{1}{2\pi\sigma^2} \int \exp\left(\frac{(x-x')^2}{2\sigma^2}\right) u(x')\, dx'$$

3. Discretization: We cannot evaluate $u(x')$ at every point in space - we only have the values at discrete pixels! Moreover, the Gaussian kernel has infinite support!

$\rightarrow$ We approximate the integral by a sum and the Gaussian by a discrete truncated Gaussian!

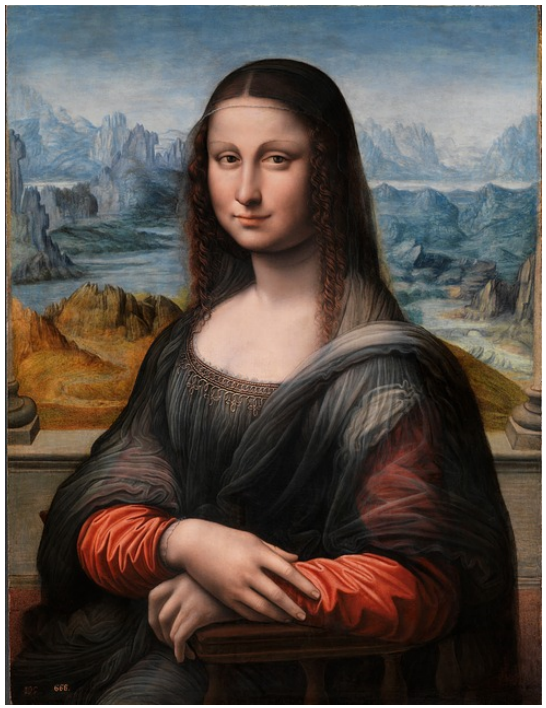
Example

4. Algorithm: At every pixel (i, j) we will compute

$$f_{i,j} = \sum_{l=-h/2}^{h/2} \sum_{k=-w/2}^{w/2} k_{l,k} u_{i-l,j-k},$$

where k denotes our truncated Gaussian kernel of size $(2h+1, 2w+1)$, and u is the sharp image.

5. Implementation: Code the above in your favorite programming language and save the resulting image.



Where do errors occur?

- **Modeling errors**: The assumption required a perfect thin lens, a perfectly planar object (perpendicular to the optical axis), Gaussian intensity distribution independent of the position, ...

- **Discretization error**: Even if all modeling assumptions were perfectly true, we introduce an error by approximating them discretely.

In this lecture we will ignore the two types of errors above and focus on two errors directly related to the algorithm:

- **Measurement error amplification**: Every device for measuring data has some kind of uncertainty, and therefore measurements always contain noise. Does this noise amplify during our algorithmic computation?
- **Rounding errors**: The computer only has a finite precision in representing numbers. Thus, any computation introduces rounding errors.

Relative Errors

When talking about errors when approximating some x by some $\tilde{x}$ the **absolute error** $\|x - \tilde{x}\|$ often does not have much expressiveness.

$\|x - \tilde{x}\| \leq 10^{-6}$ might seem good, but if $\|x\|$ itself is in the same order of magnitude, the approximation $\tilde{x}$ might be completely useless.

A better measure for the error is often the **relative error**

$$\frac{\|x - \tilde{x}\|}{\|x\|}.$$

Next we will discuss how errors arising from the finite precision of a computer behave.

Machine representation

The typical representation of a number in a computer is

$$\underbrace{(-1)^s}_{\text{sign}} \cdot \underbrace{0.c_1 c_2 \ldots c_p}_{\text{mantissa in a certain basis}} \cdot \underbrace{b^{\pm q}}_{\text{basis } b\text{, exponent } q} \qquad (1)$$

where the exponent is q is bounded due to a limited number of bits for its representation. To obtain a unique representation, we require $c_1 \neq 0$ if the represented number is nonzero.

A few 'tricks' exists for the basis $b = 2$, see IEEE standard 754.

For a given format, the number

$$\epsilon = \frac{b^{1-p}}{2}$$

is called **machine precision**.

Rounding errors

Let us denote the set of all numbers that admit a representation in the form of (1) by M. A function rd that maps any real x number to a number $\mathrm{rd}(x) \in M$ such that

$$|\mathrm{rd}(x) - x| = \min_{y \in M} |y - x|,$$

is called a **rounding function**.

Relative error of the rounding function

If we ignore bounds on the exponent q, it holds for $x \neq 0$ that

$$\left| \frac{\mathrm{rd}(x) - x}{x} \right| \leq \epsilon.$$

Proof in the exercises.

Rounding errors

Does this mean that rounding is never a problem?

Human format

As an illustrative example, let us consider: $b = 10$, $p = 3$, i.e. the set

$$M = \{\pm 0.c_1 c_2 c_3 \cdot 10^{\pm q} \mid c_1, c_2, c_3, q \in \{0, 1, \ldots 9\}, c_1 \neq 0\},$$

which is convenient for humans. A computer uses a basis $b = 2$ and predefined numbers of bits, e.g. for single or double precision, but the effects we'll discuss remain exactly the same.

Unfortunately, computations with numbers from M do not need to yield numbers in M again!

Example: $1 \in M$, $1 \cdot 10^{-3} \in M$, but

$$1 + 1 \cdot 10^{-3} = 1.001 \notin M.$$

Machine operations

We need to round after every machine operation, e.g. for $x_1, x_2 \in M$ we define

$$x_1 \oplus x_2 = \mathrm{rd}(x_1 + x_2) \in M,$$

and thus $1 \oplus 1 \cdot 10^{-3} = 1$.

This can make even elementary computations unstable i, e.g. in an effect called **catastrophic cancellation**:

$$(1 \cdot 10^{-3} \oplus 1) \oplus (-1) = 1 \oplus (-1) = 0$$

In the first case we obtain a relative error of 1! In particular, the addition is not associative anymore!

$\rightarrow$ *Show Matlab code example!*

Error amplification

For far, we have only considered rounding errors from the finite machine precision. However even without rounding errors, we should investigate what a numerical method does to an initial (measurement) error.

Consider a function $f : \mathbb{R} \to \mathbb{R}$ to investigate the following question: If $x \in \mathbb{R}$ and $\tilde{x} \in \mathbb{R}$ have a certain relative/absolute error, i.e.

$$|x - \tilde{x}| \leq \delta_{abs} \quad \text{or} \quad \frac{|x - \tilde{x}|}{|x|} \leq \delta_{rel} \text{ for } x \neq 0$$

what is the relative/absolute error in the output of f, i.e.

$$|f(x) - f(\tilde{x})| \quad \text{or} \quad \left| \frac{f(x) - f(\tilde{x})}{f(x)} \right| \text{ for } f(x) \neq 0?$$

In short: **How does the error in $\tilde{x}$ amplify by applying f?**

Analyzing the error amplification

Assume $f : \mathbb{R} \to \mathbb{R}$ is smooth. The first order Taylor expansion yields

$$f(\tilde{x}) = f(x) + f'(x) \cdot (\tilde{x} - x) + \mathcal{O}(|x - \tilde{x}|^2).$$

For $f(x) \neq 0$, this means

$$\begin{aligned} \frac{f(\tilde{x}) - f(x)}{f(x)} &= \frac{f'(x)}{f(x)} \cdot (\tilde{x} - x) + \mathcal{O}(|x - \tilde{x}|^2) \\ &= \frac{f'(x)\, x}{f(x)} \cdot \frac{\tilde{x} - x}{x} + \mathcal{O}(|x - \tilde{x}|^2) \end{aligned}$$

assuming $x \neq 0$.

Error amplification

If we ignore higher order terms, the relative error $\frac{\tilde{x}-x}{x}$ is amplified by the a factor $\frac{f'(x)\, x}{f(x)}$.

Analyzing the error amplification

There exists a multidimensional version of our computation: Let $f : \mathbb{R}^n \to \mathbb{R}$ be smooth, $f(x) \neq 0$ and $x_i \neq 0$ for all $i \in \{1, \ldots, n\}$.

Error amplification

If we ignore higher order terms, the relative error $\frac{\tilde{x}_i - x_i}{x_i}$ of each component amplifies the relative error for the result by $M_i = \left| \frac{x_i}{f(x)} \frac{\partial f}{\partial x_i}(x) \right|$, i.e.

$$\left| \frac{f(\tilde{x}) - f(x)}{f(x)} \right| \leq \sum_{i=1}^{n} M_i \cdot \left| \frac{\tilde{x}_i - x_i}{x_i} \right|.$$

Example: $f(x, y) = x + y$ leads to $M_x = \left| \frac{x}{x+y} \right|$. This is exactly the catastrophic cancellation we have seen before!

Condition and Stability

Condition

A problem is called **well-conditioned** if small changes in the input data lead to small changes in the output. If small changes in the input data lead to large changes in the output, it is called **ill-conditioned**.

Stability

A numerical algorithm is called **stable**, if the error caused by the algorithm is in the same order of magnitude as the unavoidable/analytic error given by the condition of the problem.

Example: The implementation $(x \oplus 1) \oplus (-1)$ is an unstable algorithm for the well-conditioned function $f(x) = x$.

Examples

Remark 1: Unstable algorithms are not always as easy to spot as in our simple example. For instance, a straight-forward implementation of the pq-formula for solving quadratic equations is unstable!

Remark 2: Telling if a problem is well-conditioned or not is not always easy, consider for instance

$$f(x) = \sqrt{1 + x^2} - 1$$

for small x. The condition number is bounded by 2.

Punshline

If your solution does not do what you expected it to do (and you have carefully checked your code), analyze the condition of the problem as well as the stability of your implementation!

Linear Equations

Linear equations

As a next step, we will discuss and analyze the solution to linear equations

$$Ax = b \tag{2}$$

for $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^n$.

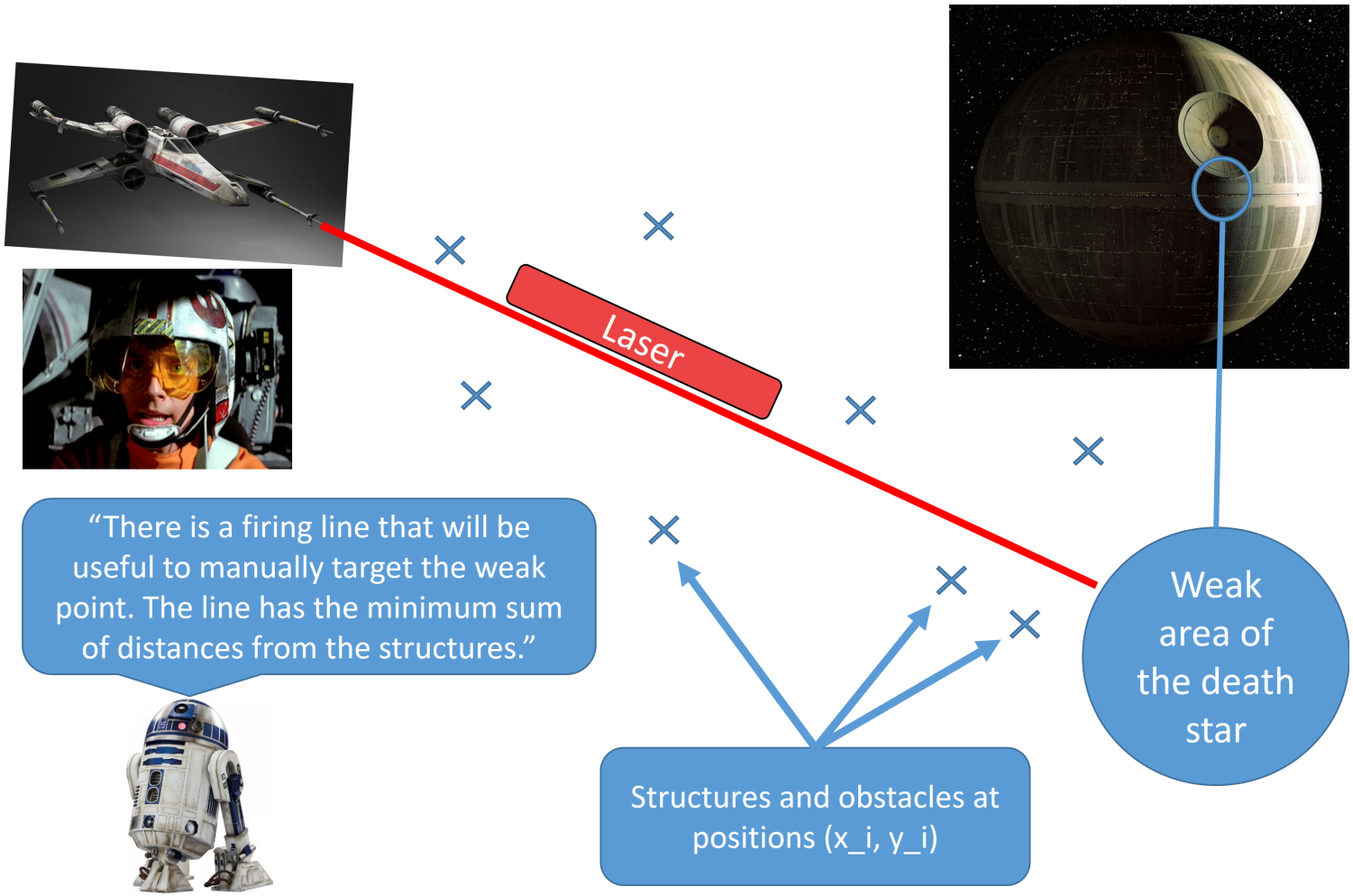
We will

- discuss ways to solve (2) exactly,
- discuss ways to solve (2) iteratively,
- analyze the condition of the function $A^{-1}b$,
- talk about the stability of the algorithms.

Before we do this, let us discuss the basics of linear regression to have a motivation for the more abstract problem (2).

Star Wars: The Linear Regression Strikes Back

Example from
`http://rfunctions.blogspot.de/2013/10/star-wars-linear-regression-strikes-back.html`,
images from `www.jedipedia.net`.



Least-Squares Solutions

Problem: Given (a_i, b_i), $i = 1, \ldots, n$, how can we find a line

$$b(x) = x_1 a + x_2$$

that minimizes

$$\sum_{i=1}^{n} (b(a_i) - b_i)^2 \ ?$$

Note that

$$\sum_{i=1}^{n} (b(a_i) - b_i)^2 = \sum_{i=1}^{n} (x_1 a_i + x_2 - b_i)^2 = \left\| A \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - b \right\|_2^2$$

where

$$A = \begin{pmatrix} a_1 & 1 \\ . & . \\ . & . \\ a_n & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} b_1 \\ . \\ . \\ b_n \end{pmatrix}$$

Least-Squares Solutions

How can we minimize

$$\|Ax - b\|_2^2 \tag{3}$$

for $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^n$ with respect to $x \in \mathbb{R}^m$?

Gaussian Normal Equation

An element $x \in \mathbb{R}^m$ minimizes (3) if and only if

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

holds. In this case we call x a **least-squares-solution** of $Ax = b$.

Remark: The set of least-squares-solutions is never empty.

Some linear algebra

Recall that

$$\ker(A) = \{x \in \mathbb{R}^m \mid Au = 0\}$$

is the **kernel** of A, and

$$\mathrm{im}(A) = \{Ax \in \mathbb{R}^n \mid x \in \mathbb{R}^m\}$$

is the **image** of A.

Remember that the inner product

$$\langle x, z \rangle = \sum_{i=1}^{m} x_i z_i$$

has the property that

$$\langle y, Ax \rangle = \langle A^T y, x \rangle$$

holds for all x, y.

Remember that we call x and z orthogonal, iff

$$\langle x, y \rangle = 0.$$

Some more linear algebra

Remember that for some subspace $V \subset \mathbb{R}^m$ we define

$$V^\perp = \{x \in \mathbb{R}^m \mid \langle x, z \rangle = 0 \quad \forall z \in V\}.$$

Remember that

$$\mathbb{R}^m = V \oplus V^\perp$$

for any subspace $V \subset \mathbb{R}^m$.

Images and kernels and transposed matrices

For any $A \in \mathbb{R}^{n \times m}$ it holds that

1. $\ker(A^T) = \mathrm{im}(A)^\perp$,
2. $\ker(A) = \ker(A^T A)$.

Now we can prove the statement about the Gaussian normal equation.

What if the solution is not unique?

We have seen that the equation

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

does not determine x uniquely if $\ker(A) \neq \{0\}$.

Let $z \in \ker(A)^\perp$ such that $A^T A z = A^T b$. Then every x of the form

$$x = z + v, \qquad v \in \ker(A)$$

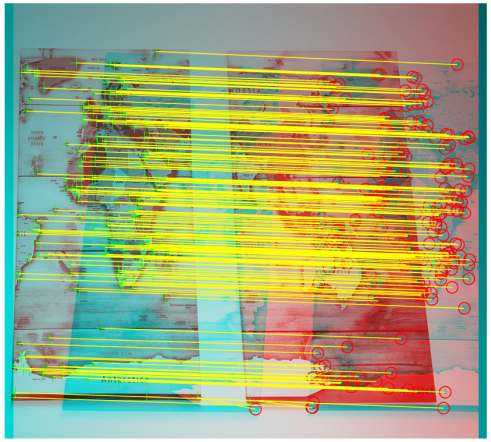
is a least-squares solution, too.

A common strategy in this case is to pick the **minimal-norm-solution**, i.e. choose $x = z$ in the above notation.

You can do so in Matlab using the command *pinv*. In general, you have to look for the terms *pseudo-inverse* or *Moore-Penrose inverse*.

Back to Least-Squares Solutions - another example

Image stitching using affine transformations.



→ Discuss formulation on the board.

Solving a linear system

The Gaussian normal equation requires us to solve

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

In other situations we might even have less structure and need to solve

$$Ax = b$$

for a general matrix $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^n$.

What is a reasonable strategy to do so?

How would you solve

$$\begin{pmatrix} 3 & 2 & 1 \\ 6 & 5 & -4 \\ -3 & 1 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 12 \\ -3 \end{pmatrix} ?$$

Gaussian elimination

Gaussian elimination with $A \in \mathbb{R}^{n \times n}$.

Set $A^{(1)} = A$, $b^{(1)} = b$.

- For $i = 1$ to $n - 1$, if $a_{i,i}^{(i)} \neq 0$
  - for $j = i + 1$ to n

    subtract $l_{ji} = \frac{a_{ji}^{(i)}}{a_{ii}^{(i)}}$ times the i-th row of $(A^{(i)}, b^{(i)})$

    from the j-th row to obtain $(A^{(i+1)}, b^{(i+1)})$.

If the algorithm can be carried out to the end, then $R := A^{(n)}$ is an upper triangular matrix, such that $Rx = b^{(n)}$ can be solved easily.

Gaussian elimination

Gaussian eliminitation with $A \in \mathbb{R}^{n \times n}$

- For $i = 1$ to $n - 1$ iteratively construct a new linear equation $A^{(i+1)}x = b^{(i+1)}$ starting from $A^{(1)} = A$, $b^{(1)} = b$, where
  - the first i rows of $A^{(i)}$ and $A^{(i+1)}$ coincide,
  - for $j = i + 1$ to $j = n$, compute

$$l_{ji} = \frac{a_{ji}^{(i)}}{a_{ii}^{(i)}}, \qquad \text{if } a_{ji}^{(i)} \neq 0$$

$$a_{jk}^{(i+1)} = a_{jk}^{(i)} - l_{ji} \cdot a_{ik}^{(i)} \quad \forall k \in \{i+1, \ldots, n\}$$

$$b_j^{(i+1)} = b_j^{(i)} - l_{ji} b_i^{(i)}$$

  - Set $a_{jk}^{(i+1)} = 0$ for all indices at which $k < i$ and $j > k$.
- Determine the solution via

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

Complexity of Gaussian elimination

How many computations have to be done to solve a linear system?

If we count one addition and one multiplication as one operation, we find

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(2 + \sum_{k=i+1}^{n} 1 \right) = \frac{1}{6}(2n^3 + 3n^2 - 5n) \in \frac{n^3}{3} + \mathcal{O}(n^2),$$

where $\mathcal{O}$ denotes the **Laudau-symbol**.

$g(n) \in \mathcal{O}(n^2)$ means that there is a constant C such that $g(n) \leq C\, n^2$ for large enough n.

Remarks on Gaussian elimination

- Gaussian elimination required $a_{ii}^{(i)} \neq 0$. If this is not the case one needs to do a permutation with a row k at which $a_{ki}^{(i)} \neq 0$. If A is invertible, such a row has to exist!
- While a full stability analysis is not easy, we can already see in

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

that possible (catastrophic) cancelations are amplified by $\frac{1}{a_{ii}^{(n)}}$. It therefore is a good idea to permute the rows of $A^{(i)}$ in every iteration in such a way that

$$|a_{ii}^{(i)}| = \max_{k \geq i} |a_{ki}^{(i)}|.$$

This is called *pivoting* and makes the Gaussian elimination stable for most practical purposes.

Gaussian elimination and LR-decompositions

The Gaussian elimination yields a upper triangular matrix $R = A^{(n)}$.

Following the algorithm closely (and ignoring pivoting for a moment), one finds that $A^{(i+1)} = L_i A^{(i)}$ for lower triangular matrices L_i with

$$L_i = \begin{pmatrix} 1 & 0 & . & . & . & . & 0 \\ 0 & 1 & 0 & . & . & . & . \\ . & 0 & . & . & . & . & . \\ . & . & . & 1 & . & . & . \\ . & . & . & -l_{i+1,i} & 1 & . & . \\ . & . & . & . & . & . & . \\ 0 & . & . & -l_{n,i} & . & 0 & 1 \end{pmatrix},$$

Since products and also the inverse of lower triangular matrices are lower triangular again, we find that A can be written as

$$A = LR$$

for a lower left matrix L and an upper right matrix R. This is interesting for several applications!

Other decompositions

In general, one requires pivoting to obtain such a representation: For every $A \in \mathbb{R}^{n \times n}$ there exists a permutation matrix P such that

$$PA = LR.$$

We call the above the **LR-decomposition** of PA.

Other important decompositions are the **Cholesky-decomposition**: If A is symmetric positive definite,

$$A = LL^T$$

for a lower triangular matrix L.

Yet another important matrix representation is the **QR-decomposition**: Every matrix $A \in \mathbb{R}^{n \times n}$ can be written as

$$A = QR,$$

where R is an upper triangular matrix and Q is orthogonal, i.e. $QQ^T = Q^TQ = I$.

We will not detail these decompositions further.

How much can go wrong?

For $0 \neq x \in \mathbb{R}^n$ and a noisy version $x^\delta \in \mathbb{R}^n$ with

$$\frac{\|x - x^\delta\|_2}{\|x\|_2} = \delta,$$

how large can

$$\frac{\|Ax - Ax^\delta\|_2}{\|Ax\|_2}$$

become?

We define $\|A\| = \sup_{\|z\|_2 \leq 1} \|Az\|_2$ to be the norm of A. If A is invertible, then

$$\frac{\|Ax - Ax^\delta\|_2}{\|Ax\|_2} \leq \underbrace{\|A\| \cdot \|A^{-1}\|}_{=:\kappa(A)} \cdot \delta.$$

The quantity $\kappa(A)$ is the **condition number of A**. If $\kappa(A)$ is large, we call A **ill-conditioned**.

Condition number and eigenvalues

Condition number and eigenvalues

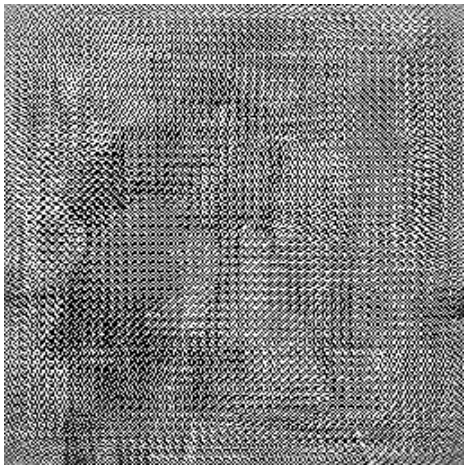
Let A be symmetric and invertible. Then $\|A\| = \max_i |\lambda_i(A)|$, where $\lambda_i(A)$ are the eigenvalues of A.

We will discuss eigenvalues in more detail in a later chapter, but this already gives an indication of the importance of eigenvalues even for analysis purposes.

Remeber the deblurring example

Original u

$f = Au + \text{noise}$



$\hat{u} = (A^T A)^{-1} A^T f$

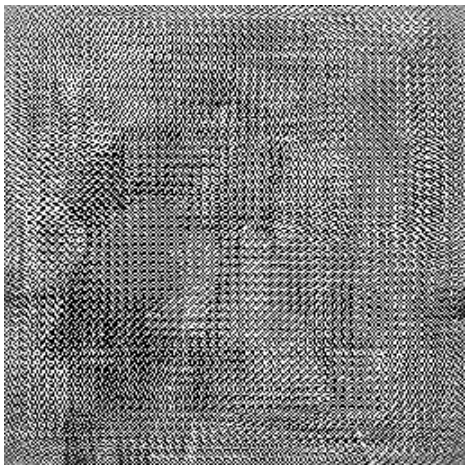
Condition of $A^T A$: $1.1 \cdot 10^9$

Practical solutions of linear systems

In practice one rarely solves $Au = f$ exactly using Gaussian elimination

- Costly, and hence impractical for large systems!
- Sparse matrices may have a rather dense LR decomposition.
- One rarely needs an exact solution.
- Bad condition numbers may even make an exact solution undesirable.

$f = Au + \text{noise}$



$\hat{u} = (A^T A)^{-1} A^T f$

25 CG iterations

Iterative solutions to linear equations

Let $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, and consider solving

$$Ax = b.$$

Idea: If A was diagonal, a linear equation is easy to solve.

Let D be the diagonal of A, i.e. $D_{ii} = A_{ii}$, and $D_{ij} = 0$ for $i \neq j$. How about using

$$Ax \approx Dx^{k+1} + (A - D)x^k$$

iteratively?

Jacobi Method

The iteration

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

is called the **Jacobi Method** for solving $Ax = b$.

Jacobi Method

Does the Jacobi-Method always work?

Certainly not! It does not even always make sense!

Convergence of the Jacobi method

If D is invertible and $\|D^{-1}(A-D)\| < 1$, then the Jacobi method converges.

We will discuss the idea of the proof on the board. The general convergence statement for these kinds of iterations is **Banach's fixed-point theorem**, but will be skipped here for the sake of simplicity.

An easy-to-check sufficient conditions for $\|D^{-1}(A-D)\| < 1$

- **strict diagonal dominance** of A, i.e.

$$|A_{ii}| > \sum_{j \neq i} |A_{ij}|.$$

- A as well as $2D - A$ are symmetric positive definite.

Gauss-Seidel method

Once we understood the idea of the Jacobi method

$$x^{k+1} = D^{-1}(b - (A - D)x^k),$$

it is very easy to come up with other iterative solution methods.

Since linear equations with lower triangular matrices are easy to solve one could try to use

$$x^{k+1} = L^{-1}(b - (A - L)x^k), \qquad \text{(GS)}$$

for L being the lower left part of A, i.e.

$$L_{ij} = \begin{cases} A_{i,j} & \text{if } j \leq i \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm (GS) is called the **Gauß-Seidel method**.

Gauß-Seidel method

Convergence of the Gauß-Seidel method

If D is invertible and $\|L^{-1}(A-L)\| < 1$, then the Jacobi method converges.

Sufficient conditions for the convergence of the Gauß-Seidel method are

- that A is symmetric positive definite,
- that A is strictly diagonally dominant.

An attractive property for an iterative solver is that it can be implemented **matrix-free**, which means one can solve linear systems where A is not given as a matrix, but rather as functions!

Example: Radon Transform (in Matlab)

Jacobi and Gauß-Seidel methods are not matrix free.

Iterative solutions to linear equations

In full analogy to the Gauß-Seidel and Jacobi methods, use

$$x^{k+1} = B^{-1}(b - (A - B)x^k),$$

for some matrix B that is easy to invert, does not use the matrix form of A, and for which

$$\|B^{-1}(A - B)\| < 1.$$

Simple choice: **Richardson iteration** $B = \frac{1}{\sigma} \cdot I$, in which case

$$x^{k+1} = x^k + \sigma(b - Ax^k).$$

To guarantee convergence, one needs

$$\|\sigma A - I\| < 1.$$

If A is positive definite, this can be ensured, e.g. by choosing

$$\sigma = \frac{2}{\lambda_{\max}(A) + \lambda_{\min}(A)}.$$

Richardson iteration and gradient descent

Interesting special case: Consider the least-squares problem

$$\min_u \|Cu - f\|^2.$$

The Gaussian normal equation is

$$C^T Cu = C^T f$$

The Richardson iteration takes the form

$$x^{k+1} = x^k - \sigma(C^T Cx^k - C^T f).$$

As we will learn later, this is the **gradient descent** algorithm for minimizing $\|Cu - f\|^2$.

This algorithm is, however, not very efficient for linear systems. We will therefore discuss one last iterative algorithm, the **conjugate gradient (CG)** method.

Conjugate Gradient

One of the most widely used and most efficient methods: **Conjugate Gradient (CG)** method for systems

$$Ax = b$$

with a symmetric positive definite $A \in \mathbb{R}^{n \times n}$.

Why **conjugate**?

Conjugate directions

We call two vectors u and v conjugate with respect to A, if

$$\langle u, Av \rangle = 0.$$

Suppose that $\{p_1, \cdots, p_n\}$ are mutually conjugate. Exercises: Then the p_i are linearly independent!

Conjugate Gradient

If the set $\{p_0, \cdots, p_{n-1}\} \subset \mathbb{R}^n$ has n linearly independent vectors, it forms a basis!

This means we can express the solution $\hat{x}$ to

$$Ax = b$$

as

$$\hat{x} = \sum_{i=0}^{n-1} \alpha_i p_i$$

for some coefficients α_i.

Therefore

$$b = A\hat{x} = \sum_{i=0}^{n-1} \alpha_i A p_i$$

and by multiplying with p_j^T from the left

$$\langle p_j, b \rangle = \sum_{i=0}^{n-1} \alpha_i \langle p_j, A p_i \rangle = \alpha_j \langle p_j, A p_j \rangle.$$

Conjugate Gradient

We therefore find

$$\alpha_j = \frac{\langle p_j, b\rangle}{\langle p_j, Ap_j\rangle}.$$

We now know the true solution has the form

$$\hat{x} = \sum_{i=0}^{n-1} \frac{\langle p_i, b\rangle}{\langle p_i, Ap_i\rangle} p_i$$

for any mutually conjugate set $\{p_0, \cdots, p_{n-1}\}$.

Conjugate gradient algorithm: A clever way to iteratively construct p_i's and

$$x_{k+1} = \sum_{i=0}^{k} \frac{\langle p_i, b\rangle}{\langle p_i, Ap_i\rangle} p_i$$

Conjugate Gradient

Idea: Define the residuum

$$r_k = Ax_k - b$$

(assuming $x_0 = 0$), and choose the next direction p such that it can represent the current residuum,

$$p_k \in \text{span}\{p_0, \ldots, p_{k-1}, r_k\}.$$

To ensure that p_k is indeed mutually conjugate to the p_j, $j < k$, one finds

$$p_k = -r_k + \beta_k p_{k-1},$$

for a suitable $\beta_k \in \mathbb{R}$.

Very efficient: The computation of the new p_k merely requires the current residuum and the previous p_{k-1}!

Conjugate Gradient

Algorithm: Set $x = 0$, $r = b$, $p_0 = -r_0$, choose an accuracy ϵ, and do for $k = 0, \ldots, N_{\max}$

- If $\|r_k\| \leq \epsilon$, stop
- Set $\alpha_k = \frac{\|r_k\|^2}{\langle p_k, Ap_k \rangle}$
- Set $x_{k+1} = x_k + \alpha_k p_k$
- Set $r_{k+1} = r_k + \alpha_k A p_k$
- Set $\beta_{k+1} = \frac{\|r_{k+1}\|^2}{\|r_k\|^2}$
- Set $p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$

See Chapter 5.1 in Nocedal, Wright, *Numerical Optimization* for further details.