1 Introduction and examples

1.1 Basic Information

The course will provide an introduction to numerical methods. It covers

- Taylor series
- Base representation
- Linear and nonlinear systems of equations
- Numerical differentiation and integration
- Interpolation,approximation and least-squares method
- Introduction to ordinary differential equations

Numerical methods bridge the gap between mathematics and its practical applications in various branches of science and engineering. In many situations one needs to know:
How to approximately compute?
How to estimate?
How to model?

Literature: Primary reading D. Kincaid and W. Cheney *Numerical Analysis: Mathematics of Scientific Computing*, Any Brooks/Cole Publishing Company, 1991
G. Dalquist and A. Björck *Numerical Methods*, Dover, 2003
Further reading: W. Boehm and H. Prautzsch: *Numerical Methods* AK Peters, 1992

Lecture notes are intended to help to refresh the material discussed during classes. They neither fully cover the classes nor substitute for systematic reading. I apologize for possible typos and will be indebted if they are reported back to me.

1.2 Introduction

Examples of numerical procedures:

- **A. Iterations** We need to accurately estimate $x = \sqrt{c}$, i.e. solve $x^2 = c$. We may rewrite it as
$$x = F(x), \quad F(x) = \frac{1}{2}\left(x + \frac{c}{x}\right)$$
and then use an iterative procedure $x^{(n)} = F(x^{(n-1)})$, beginning from some approximate $x^{(0)}$, $n = 0 \ldots$. It will converge fast. However, if we have tried $F(x) = c/x$, the procedure would diverge!
- **B. Interpolation** If we need to numerically compute $\int_a^b f(x)dx$, we subdivide the interval (a, x) into a partition $x_i = a + hi$, $i = 0, 1, \ldots, M$, with $h = (b - a)/M$. We can estimate the integral as $\sum_{n=1}^{M} hf(x_{n-1}$. However, we can do much more accurately if we think about interpolating linearly or quadratically on partition intervals (with additional function estimates).
- **C. Integration** Given $dy/dx = f(x)$ for $x \in [a, b]$, $y(a) = y_0$ we can try to proceed in the simplest way, introducing partitioning as in **B** and writing $y(x_i) - y(x_{i-1} = hf(x_{i-1})$. However, this may prove unstable and diverge, or it may prove to be not accurate enough.

Numerical methods are the algorithms to solve mathematical problems or equations when algebraic or analytical approaches are hard or impossible to use (solution method is unknown or inefficient).

Our aim is to study simple but efficient numerical methods and understand how they work, what are their limitations, what are the errors and applicability bounds.
Algorithmic examples:

- Horner's rule: To formally compute $p(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ for some $x = c$ we need 6 multiplications and 3 additions. Rewriting

$$p(x) = ((a_3x + a_2)x + a_1)x + a_0$$

one needs only 3 multiplications and 3 additions. This is an example of recursive computations.

- Compute

$$y_n = \int_0^1 \frac{x^n}{x+10}dx \quad \text{for} \quad n = 0, 1, \ldots 5.$$

We can note that $y_n + 10y_{n-1} = 1/n$ (prove) and proceed recursively as $y_n = 1/n - 10y_{n-1}$ based on $y_0 = \ln(x+10)|_0^1 \approx 0.0953$. If we do further computation with this estimate, the result is 0.0470 ($n = 1$), 0.0300 ($n = 2$), 0.0333 ($n = 3$), -0.0830 ($n = 4$), which is obviously incorrect! What happens here is the propagation of error in y_0. Explain why does it happen.

1.3 Taylor series

On an elementary level many function estimates are possible through using the Taylor series.
Given

- A function $f : \mathbf{R} \to \mathbf{R}$ that is difficult to compute
- f and some derivatives of f at c are known
- Question:
  Can we use this information to estimate (approximate) $f(x)$ at some x?
  if yes, how accurately can we do it?

Consider $f(x) = \cos(x)$. We need to compute $f(x)$ for $x = 0.1$. Take $c = 0$. We know that $\cos(0) = 1$ and we know that $(\cos(c))' = -\sin(c)$ and that $(\cos(c))'' = -(\sin(c))' = -\cos(c)$, so all derivatives at $c = 0$ are given, they are either 0 (odd) or ± 1 even. So how to get $\cos(0.1)$?

Taylor series

Let $f : \mathbf{R} \to \mathbf{R}$ be a function with existing derivatives $f^{(k)}(c)$ for $k = 0, 1, 2 \ldots$
Then the Taylor series of f at c is

$$f(x) \approx f(c) + f'(c)(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \cdots = \sum_{k=0}^{\infty} \frac{f^{(k)}}{k!}(x-c)^k.$$

The Taylor series can be used to compute the cosine in the example above. One should know how many terms to take and whether the series converges at all.

- Taylor series for the cosine $f(x) = \cos(x)$ at $c = 0$:
- $\cos(x) = f(0) + f'(0) \cdot x + f''(0) \cdot x^2/2 + \ldots = 1 - x^2/2 + \ldots$
- Taylor series for the exponential function $f(x) = e^x$ at $c = 0$:
- $f^{(k)}(x) = e^x$, hence $f^{(k)}(0) = 1$
- $e^x = f(0) + f'(0) \cdot x + f''(0) \cdot x^2/2 + \ldots = 1 + x + x^2/2 + \ldots$

The Taylor series consists of an infinite number of additive terms. What happens if we truncate it (leave a finite number of terms)? An error will be created.

Comparing the plot of $y = e^x$ or $y = \cos(x)$ with the plots of its truncated Taylor series we may develop an idea of how accurate we are. For the cosine function, by comparing the truncations 1, $1 - x^2/2$,

$1 - x^2/2 + x^4/24$ or $1 - x^2/2 + x^4/24 - x^6/720$ and so on one readily sees that the error is reduced if x is small. Even for $|x| > 1$, despite the cosine varies between -1 and 1, the agreement becomes better if more terms (and higher powers of x) are added. However, the procedure can be highly suboptimal in this case (how to improve it?) A general idea that can be derived from these examples is that approximation is improved if we take more terms.

For polynomial functions the Taylor expansion is finite: take $f(x) = 3x^2 + 10x + 5$ at $c = 2$ to get

$$\begin{aligned} f(x) = 3x^2 + 10x + 5 \quad &\to f(2) = 27, \\ f'(x) = 6x + 10 \quad &\to f'(2) = 22, \\ f''(x) = 6, \quad f^{(k)} &= 0, k > 2. \end{aligned}$$

Hence $f(x) = 27 + 22(x-2) + 3(x-2)^2$. Here the Taylor series is just re-arrangement of the terms and no error is created.

In a general case there are truncation errors. We may use Taylor's theorem to estimate errors.

1.4 Taylor's theorem

Let $f \in C^{n+1}[a, b]$, i.e. f is $n+1$ times differentiable over the interval $[a, b]$ and has continuous derivatives. Then for $x, c \in [a, b]$

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(c)}{k!}(x-c)^k + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-c)^{n+1},$$

where $\xi \in (\min(c, x), \max(c, x))$. The sum represents the truncated Taylor series, and the last term is the error term (the remainder).

For $n = 0$ Taylor's theorem is the same as the statement that there exists ξ such that

$$f(b) = f(a) + (b-a)f'(\xi)$$

for some $\xi \in (a, b)$. This is the mean-value theorem. Geometrical interpretation: a continuous differentiable function $y = f(x)$ varies from $f(a)$ to $f(b)$ over the interval $[a, b]$. If the function is linear, the statement is trivial. If it is not, there is always a point ξ where the tangent is parallel to the linear function $y = f(a) + (f(b) - f(a))(x-a)/(b-a)$ (secant). Moreover, $f'(\xi) = (f(b) - f(a))/(b-a)$ is the estimate of the derivative within the interval (a, b).

DEFINITION: One says that a Taylor series of function f *represents the function* f **iff** the error term converges to 0 for $n \to \infty$.

- Consider $f(x) = e^x$ developed at $c = 0$.
- Taylor's theorem:

$$e^x = \sum_{k=0}^{n} \frac{x^k}{k!} + \frac{e^{\xi(x)}}{(n+1)!}x^{n+1}$$

- Suppose $|x| \le s$. Then $|\xi(x)| \le s$ and $e^{\xi(x)} \le e^s$. Hence

$$\lim_{n\to\infty} \left| \frac{e^{\xi(x)}}{(n+1)!}x^{n+1} \right| \le \lim_{n\to\infty} \frac{e^s}{(n+1)!}s^{n+1} = 0,$$

so that

$$\lim_{n\to\infty} \sum_{k=0}^{n} \frac{x^k}{k!} = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

Since the series converges for any x, it can be used to estimate e^x, and the question is only which n provides the accuracy needed.

However, there are complications

- Consider $f(x) = \ln(1+x)$ at point $c = 0$
- $f'(x) = (1+x)^{-1}$,
- $f''(x) = -(1+x)^{-2}$,
- $f^{(k)} = (-1)^{k-1}(k-1)!(1+x)^{-k}$

As a consequence we have

$$\ln(1+x) = \sum_{k=1}^{n} (-1)^{k-1} \frac{1}{k} x^k + E_n,$$

where the error term

$$E_n = (-1)^n \frac{1}{n+1} (1+\xi)^{-(n+1)} x^{n+1}.$$

We see that

$$\lim_{n\to\infty} E_n = \lim_{n\to\infty} \frac{(-1)^n}{n+1} \left(\frac{x}{\xi+1}\right)^{n+1} = 0$$

if $0 < \xi < x \leq 1$. The Taylor series does not represent the function everywhere, for the error term tends to zero only in the vicinity of $x = 0$. For example, we cannot use the Taylor expansion for $x > 1$.

Question: what will happen if x is negative? The most straightforward way of exploring convergence is to apply the *ratio test* by computing $R = |a_{n+1}/a_n|$, where a_n is the shortcut for the nth term of the Taylor series. The theorem (without proof) states that if

$$L = \lim_{n\to 0} R < 1,$$

the series converges absolutely. It diverges if $L > 1$ and test is inconclusive if $L = 1$. For $f(x) = \ln(1+x)$ the Taylor series converges if $-1 < x \leq 1$.

Conclusion: One has to explore the *range of convergence* before applying the Taylor expansion.

Practical notes. Return to Example 1 (cos(0.1)).
Its actual value: $\cos(0.1) = 0.995004165278...$
Taylor expansion at $c = 0$: $\cos x = 1 - x^2/2! + x^4/4! - \cdots$.

n	estimate	error
0,1	1	$< 0.01/2!$
2,3	0.995	$< 0.0001/4!$
4,5	0.99500416	$0.000001/6!$
...	...	...

Good approximation is obtained quickly in this example. (Is it helped by the fact that the terms of series are with alternating signs?) In general case we need to know: (i) the conditions the series is converging and (ii) how fast does it converge(or how many terms we have to take to get a good approximation). *Importantly*, the convergence rate can sometimes be improved.

For example, we need to compute $\ln(2)$. If we use the Taylor series for $\ln(1+x)$ at $c = 0$ and evaluate it at $x = 1$: $\ln 2 = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 - 1/8 + \cdots$, we get, keeping 8 terms, $\ln 2 \approx 0.63452$. But we can use the Taylor series for $\ln((1+x)/(1-x))$. First, since $\ln((1+x)/(1-x)) = \ln(1+x) - \ln(1-x)$, we can easily develop the Taylor series. Second the argument becomes 2 for $x = 1/3$, which warrants that the series converges much faster. Indeed,

$$\ln 2 = 2\left(\frac{1}{3} + \frac{1}{3^3 \cdot 3} + \frac{1}{3^5 \cdot 5} + \frac{1}{3^7 \cdot 7} + \cdots\right),$$

and 4 terms of expansion provide $\ln(2) \approx 0.69313$, to be compared with the exact value of 0.69315. Question: how to approximately compute $\ln(17)$? (Use $x = 8/9$. The series will still converge.) Remark: the convergence rate is improved the closer x is to c.

The Taylor's theorem can be rewritten in an equivalent form:

Let $f \in C^{n+1}[a,b]$. Then, if x, $x+h \in [a,b]$,

$$f(x+h) = \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!} h^k + \frac{f^{(n+1)}(\xi(h))}{(n+1)!} h^{n+1},$$

where ξ belongs to the open interval between x and $x+h$: $|x-\xi| < h$. This form is just the change of notation. It is more convenient to introduce terminology.

- If $h \to 0$ the error term converges to 0 at least as h^{n+1} if the $(n+1)$ derivative is bounded in the interval $[x, x+h]$. We say that the error term is $\mathcal{O}(h^{n+1})$ in this case.
- Using $\mathcal{O}$ notation we mean that there exists such C that $|E_n| \leq C|h|^{n+1}$.

In this notation it should be clear that an estimate of $f(x+h)$ can be made sufficiently accurate by selecting h sufficiently small. Taylor's theorem does not impose limitations on h and gives an idea of the error term for any h in the interval where f is sufficiently differentiable. Provided we know the bound on $f^{(n+1)}(\xi(h))$, we can also think of admissible h ensuring that the error is less than some error bound e.

An example from a different area shows that we cannot generally rely on the estimate of the first dropped term in expansion. Consider

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

The thousandth term in this expansion $\leq 10^{-6}$. Does it imply that we already have 5 correct decimals? No, because $\sum_{n=1001}^{\infty} \frac{1}{n(n+1)} = 1/1001$. It is important to have a correct error estimate for the dropped terms.

2 Number representations

2.1 On errors

- **A.** Errors in input data, partly because of round-off
- **B.** Round-off errors during computations. If two numbers with s significant digits are multiplied, the result will be $2s-1$ or $2s$ significant digits, they will be rounded-off.
- **C.** Truncation errors (Discretization has finite accuracy)

Let $\tilde{a}$ be an approximation of a.
$\tilde{a}-a$ is its absolute error, $(\tilde{a}-a)/a$ is its relative error. *Error bound* is the magnitude of admissible errors.

If the magnitude of errors in $\tilde{a}$ does not exceed $\frac{1}{2}\cdot 10^{-t}$, we say that there are t correct decimals. Digits that occupy positions greater than or equal to 10^{-t}, with leading zeros removed, are called significant digits.

0.001234 with error $0.000004 = 0.4\cdot 10^{-5}$ has 5 correct and 3 significant digits.
0.001234 with error $0.000006 = 0.06\cdot 10^{-4}$ has 4 correct and 2 significant digits.

Chopping at t decimals — the error 10^{-t}
Rounding: if the contribution to the right of t is less than $\frac{1}{2}\cdot 10^{-t}$, the digit at t is left as it is, the rest is truncated. If it is larger, the t digit is incremented by one. The case of equality depends on agreement, in most cases it is incremented. The error is between $\pm\frac{1}{2}\cdot 10^{-t}$.

Theorem (proof is elementary) In addition and subtraction the bound for absolute error are added.

In division or multiplication the bounds on relative errors are added.

Term cancellation: Subtraction when the difference in terms is significantly smaller in amplitude that the terms proper results in very poor relative accuracy.

Solve $x^2-56x+1=0$ to get $x_1 = 28-\sqrt{783} \approx 28-27.982 = 0.018 \pm \frac{1}{2}\cdot 10^{-3}$. Here despite 5 significant digits in the computations of square root we get only 2 significant digits in the answer.

Error propagation If $y(x)$ is a smooth (differentiable) function, $|y'(x)|$ can be interpreted as sensitivity of $y(x)$ to errors in x. The *error propagation theorem* is the generalization of this observation to functions of several variables:

$$|\Delta y| \leq \sum_i \left|\frac{\partial y}{\partial x_i}\right| |\Delta x_i|,$$

where $\Delta y = \tilde{y} - y^{(0)}$ and $\Delta x = \tilde{x}_i - x_i^{(0)}$. Clearly it has an asymptotic sense – the errors should be small enough.

2.2 Base representations

Let $b \in \mathbb{N}\setminus\{1\}$.
Every number $x \in \mathbb{N}_0$ can be written as a unique expansion with respect to the base b as

$$x = a_0 b^0 + a_1 b^1 + \ldots + a_n b^n = \sum_{i=0}^{n} a_i b^i,$$

where $a_i \in \mathbb{N}_0, \quad a_i < b$.

Base 10:

- $825346 = 6 + 40 + 300 + 5000 + 20000 + 800000 =$
  $6 \times 10^0 + 4 \times 10^1 + 3 \times 10^2 + 5 \times 10^3 + 2 \times 10^4 + 8 \times 10^5$.
- We write $a_n \cdots a_0 = \sum_{k=0}^{n} a_k 10^k$
- How to represent fractions? $0.521 = 5 \times 10^{-1} + 2 \times 10^{-2} + 1 \times 10^{-3}$
- Real numbers: $a_n \cdots a_0.b_1 \cdots = \sum_{k=0}^{n} a_k 10^k + \sum_{k=1}^{\infty} b_k 10^{-k}$
- For irrational numbers an infinite number of coefficients is required. However, it is also so for some rational numbers, such as 1/3 or 3/17. So an infinite representation does not necessarily imply that the number is irrational.

A number with simple base representation with respect to one base may have a complicated expression with respect to another base. Consider 0.1_{10}. It is

$$(0.1)_{10} = (0.0001100110011\ldots)_2$$

Question: How to get this?
Computer systems are using

- base 2 (binary)
- base 8 (octal)
- base 16 (hexadecimal)
- The question is how do we get from one base representation to the other one? We are in most cases interested in converting between bases 2,8, 16 and 10.

Conversion $b \to 10$

- By definition $(a_n a_{n-1} \cdots a_0)_b = a_n b^n + a_{n-1} b^{n-1} \cdots + a_0$
- Just perform computations $(42)_8 = 4 \times 8^1 + 2 \times 8^0 = 34$
- Conversions between 2 and 8: Three consecutive bits represent one octal digit. For example,

$$(551.624)_8 = (101\,101\,001.110\,010\,100)_2$$

- Conversions between 2 and 16: similar, but four consecutive bits are involved to represent one hexadecimal digit

Conversion $10 \to b$ is only a bit more complicated. There are two approaches: The algorithm by Euclid (330-275 BC)
The algorithm based on Horner's scheme (1786-1827)

- The Euclid algorithm
- Input: $(x)_{10}$
- Output $(x)_b$
- 1. Determine the smallest n such that $x < b^{n+1}$
  2. For $i = n : -1 : 0$, compute
  $$a_i := x \mathrm{div} b^i \quad \text{integer division}$$
  $$x := x \mathrm{mod} b_i \quad \text{modulo operation}$$
  3. Return $a_n a_{n-1} \cdots a_0$

Example: Conversion $10 \to 8:\ (34)_{10}$

1. $8^1 < 34 < 8^2 \to n = 1$
2. Iteration:
   - $a_1 := 34 \text{div} 8^1 = 4$
   - $x := 34 \text{mod} 8^1 = 2$
   - $a_0 := 2 \text{div} 8^0 = 2$
   - $x := 2 \text{mod} 8^0 = 0$
3. Output $(42)_8$

The algorithm can be extended to real numbers, but one will need a stopping criterion. However, its first step (determining n) is inefficient for a computer implementation.

Horner's scheme relies on the nested form of polynomial representation:

$$(a_n a_{n-1} \cdot a_0) = a_n b^n + a_{n-1} b^{n-1} + \ldots + a_2 b^2 + a_1 b^1 + a_0 b^0 =$$

$$((\ldots((a_n)b + a_{n-1})b \ldots + a_2)b + a_1)b + a_0$$

The algorithm:

- Input: $(x)_{10}$
- Output $(x)_b$

1. $i := 0$
2. While $x \neq 0$, compute
   - $a_i := x \text{mod} b$
   - $x := x \text{div} b$
   - $i := i + 1$
3. Return $a_n a_{n-1} \cdot a_0$

Convert $10 \to 8 : (34)_{10}$

- $a_0 := 34 \text{mod} 8 = 2$
- $x := 34 \text{div} 8 = 4$
- $a_1 := 4 \text{mod} 8 = 4$
- $x := 4 \text{div} 8 = 0$
- Stop and output $(42)_8$

This algorithms is directly applicable to real numbers, but once again one should provide a criterion to stop if the representation is infinite. It does not need the estimate of n. It does not involve division by large numbers, which is important for the accuracy of computations.

2.3 Computer representation

On computers only a finite number of digits (or bits) is used to represent the numbers. Hence infinite representations are truncated. Truncation introduces a *truncation (roundoff) error*.

- A normalized floating point representation with respect to the base b stores a number x as

$$x = 0.d_1 d_2 \cdots d_k \cdot b^n,$$

  where $d_i \in \{0, \ldots, b-1\}$ for $i = 1, \ldots k$ and $d_1 \neq 0$.

- $d_i \in \{0, \ldots, b-1\}$ is called the *mantissa* with precision k
- n is called the *exponent*
- The constraint $d_1 \neq 0$ gives a normalization, which makes the representation unique

Examples

- $b = 10$: $33.213 \to 0.33213 \times 10^2$
- $b = 2$: On computers

$$x = \pm 0.b_1 b_2 \ldots \times 2^n$$

The first bit b_1 is always 1. In principle it needs not to be stored, but it is needed if the sign bit is not stored. Since one typically operates with a known base, the base needs not to be stored.

Assume we use 4 bytes=32 bits to represent a number. We need 1 bit to represent the sign of mantissa; 1 bit to represent the sign of exponent.

The rest is commonly split as 23 bits for mantissa;
7 bits for exponent (integer).

In reality this implies that we have 24 bits for the mantissa (why?). Since 7 bits allow us to represent the largest integer 127, and since $2^{127} \approx 10^{38}$, this gives the range of representable numbers: $10^{-38} < |x| < 10^{38}$. Since $2^{-24} \approx 10^{-7}$ it means that we will be able to represent only seven significant digits. The numbers that have more than 7 digits in their representation will be therefore approximated. This example corresponds to what is commonly called *single precision*. One frequently needs much better representation, say *double precision* where 8 bytes are used for number representation.

Los of significant bit – addition

- Is $(a+b)+c = a+(b+c)$ is always true on a computer? Consider a mantissa with 7 significant digits and take $b = 10$. Take four numbers $a = 0.00000033$, $b = a$, $c = 0.00000034$ and $d = 1.000000$. If we compute $a + b + c + d$ as they appear, the result is 1.000001. If we reverse the order, we get 1.000000. Indeed, we have

$$0.3300000 \times 10^{-6} + 0.3300000 \times 10^{-6} + 0.3400000 \times 10^{-6} + 0.1000000 \times 10^{1}$$

  If the first three numbers are added first, the exponent will become -5, and they will be taken it into account. If the order is reversed, their contribution will be lost. A more dramatic example is to consider (assuming 7 significant digits) $(\sum_1^{10^7} 1) + 10^7$ and $10^7 + 1 + \ldots + 1$, where 1 appear 10^7 times. The results differs twice.

- We are losing significant bits in summation
- The problem is most expressed when summing numbers with different size $\to$ Observe the order when summing!

Loss of significant bits – subtraction

- Estimate $x - \sin x$ for small x. Take $x = 1/15$.

- $x = 0.6666666667 \times 10^{-1}$
- $\sin x = 0.6661729492 \times 10^{-1}$
- Their difference is $x = \sin x = 0.0004937175 \times 10^{-1} = 0.4937175000 \times 10^{-4}$
- Three significant digits are lost. Subtraction of similar values should be avoided.

But how? In this case simply, for one can use the Taylor series expansion. Indeed,

$$x - \sin x = \frac{x^3}{3!} - \frac{x^5}{5!} + \frac{x^7}{7!} - \dots$$

Only three terms of this series is sufficient to estimate the difference with 9 correct significant digits.

THEOREM Let x and y be normalized floating point numbers such that $x > y > 0$ and let $b = 2$. If $\exists$ $p, q \in \mathbb{N}_0$ such that

$$2^{-p} \leq 1 - y/x \leq 2^{-q}$$

then at most p and at least q significant bits will be lost during subtraction.

Proof: multiply with x to see that there will be between p and q leading zeros

3 Linear equation systems

3.1 Gaussian elimination

Linear equation system occur very frequently in practice, at least in numerical analysis. Imagine that we have a vector with coordinates $\mathbf{x} = (x_1, x_2, x_3)$ with respect to some basis, but need to find its coordinates in a new basis given by three vectors $\mathbf{a} = (a_1, a_2, a_3)$, $\mathbf{b} = (b_1, b_2, b_3)$ and $\mathbf{c} = (c_1, c_2, c_3)$ in the old basis. We seek the coefficients α, β and γ such that $\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$, which leads to a system of three linear equations.

$$\begin{aligned}\alpha a_1 + \beta b_1 + \gamma c_1 &= x_1\\ \alpha a_2 + \beta b_2 + \gamma c_2 &= x_2\\ \alpha a_3 + \beta b_3 + \gamma c_3 &= x_3.\end{aligned}$$

α, β, γ are obtained from solution of the system above. Many numerical algorithms of solving partial differential equations result in solving linear equation systems.

The general form of linear equation system is

$$\begin{aligned}a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1\\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2\\ \vdots \quad + \quad \vdots \quad + \quad \ddots \quad + \quad \vdots \quad &= \vdots\\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m\end{aligned}$$

Here the x_i are unknows. Since all they have degree 1, the system is linear in x_i. A compact notation is

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,\ 1 \le i \le m.$$

Even more compact writing involves matrices, $\mathrm{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{x} = (x_1, \ldots, x_n)^T$, $\mathbf{b} = (b_1, \ldots, b_n)^T$ are the column vectors of unknowns and the right hand side respectively, and A is the matrix containing a_{ij} with i and j the row and column indices respectively.

The question is how to solve for $\mathbf{x} = (x_1, \ldots, x_n)^T$? A special case is when $n = m$, i. e. the system is square. In this case the number of equations equals the number of unknowns, and we may hope to get a solution provided some conditions are satisfied. If $m \neq n$, we have situations when there are either too many or insufficient number of equations. We begin from the case $m = n$.

- Idea: Express first x_1 from the first equation, and substitute to the remaining $n-1$ equations. Then express x_2 from the resulting second equation, substitute into the remaining n_2 equations, and so on. As a result, we transform our equation system to the upper triangular one. The procedure just described is *forward elimination*.
- Calculate the unknowns in the reverse order using *backward substitution*.

The procedure came to be known as the Gaussian elimination. Let's try it.

Example

Compact notation

$$\left.\begin{array}{rrrrr} 6x_1 & -2x_2 & +2x_3 & +4x_4 & = 16\\ 12x_1 & -8x_2 & +6x_3 & +10x_4 & = 26\\ 3x_1 & -13x_2 & +9x_3 & +3x_4 & = -19\\ -6x_1 & +4x_2 & +1x_3 & -18x_4 & = -34 \end{array}\right\} \Rightarrow$$

$$\left(\begin{array}{rrrr|r} 6 & -2 & 2 & 4 & 16\\ 12 & -8 & 6 & 10 & 26\\ 3 & -13 & 9 & 3 & -19\\ -6 & 4 & 1 & -18 & -34 \end{array}\right)$$

Take the first row as a pivot row and the first coefficient as a pivot element. Do: (row 2)=(row 2)-2*(row 1), (row 3)=(row 3)-(row 1)/2 and (row 4)=(row 4)+(row 1). The result is:

$$\left(\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & -12 & 8 & 1 & -27 \\ 0 & 2 & 3 & -14 & -18 \end{array}\right)$$

Now take row 2 as the pivot row, and element -4 in this row as the pivot element. Do (row 3)=(row 3)-3(row 2) and (row 4)=(row 4)+(row 2)/2:

$$\left(\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & 0 & 2 & -5 & -9 \\ 0 & 0 & 4 & -13 & -21 \end{array}\right)$$

Do (row 4)=(row 4)-(row 3)*2:

$$\left(\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & 0 & 2 & -5 & -9 \\ 0 & 0 & 0 & -3 & -3 \end{array}\right)$$

The system (its matrix) becomes upper triangular (the entries below the diagonal are all zeros).

The backward substitution: The last equation implies that

$$-3x_4 = -3, \Rightarrow x_4 = 1.$$

The second last equation

$$2x_3 - 5x_4 = -9 \Rightarrow 2x_3 = -4 \Rightarrow x_3 = -2.$$

Continuing we get $(x_1, x_2, x_3, x_4)^T = (3, 1, -2, 1)^T$. Notice that we were taking diagonal elements as pivots.

In a general case,

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Forward elimination

- for $k = 1 : n - 1$ (go over all pivot rows – the last row is not needed)
- for $i = k + 1 : n$ (go over all rows below the pivot one)
- for $j = k : n$ (go over all columns from the pivot)
- $a_{ij} := a_{ij} - (a_{ik}/a_{kk})a_{kj}$ (a_{kk} is the pivot element. The matrix entries are updated)
- $b_i := b_i - (a_{ik}/a_{kk})b_k$ (update the rhs)

Backward substitution

- $x_n := b_n/a_{nn}$ (Compute the last unknown)
- for $i = n - 1 : -1 : 1$ (Return back row by row)
- rhs $:= b_i$
- for $j = i + 1 : n$ (visit all columns right to the pivot element)
- rhs := rhs $- a_{ij}x_j$ (all x_j here are already computed)

- $x_i := \text{rhs}/a_{ii}$ (Compute x_i)

Does it always work? There is an obvious limitation that all pivots should be non-zeros. (Clearly A should be non-singular. Is it sufficient? No!) Although there is a theorem stating when the elimination is possible, it is cumbersome to test. Two special cases when the Gaussian elimination will work: (i) A is diagonally dominant, (ii) It is symmetric and positive definite.

3.2 Gaussian elimination with scaled partial pivoting

Examples:
Consider

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_1 + x_2 + 2x_3 &= 2 \\ x_1 + 2x_2 + 2x_3 &= 1 \end{aligned}$$

The system is nonsingular and has the solution $x_1 = 1, x_2 = -1$ and $x_3 = 1$. If we express $x_1 = 1 - x_2 - x_3$ from the first equation and insert into the remaining ones, the result will be

$$\begin{aligned} x_3 &= 1 \\ x_2 + x_3 &= 0. \end{aligned}$$

The pivot is zero. However, we may exchange (permute) equations 2 and 3 to continue the elimination. So permutations are necessary!

There is a more significant problem: Even if pivots are non-zeros, difficulties may come from the finite precision of numerical operations on computers. To see consider a system

$$\begin{aligned} \epsilon x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}$$

If $\epsilon \to 0$, $x_1, x_2 \to 1$.

If we formally do Gaussian elimination, we take $x_1 = (1 - x_2)/\epsilon$. Eliminating it from the second equation gives

$$x_2(1 - 1/\epsilon) = 2 - 1/\epsilon, \Rightarrow x_2 = \frac{2 - 1/\epsilon}{1 - 1/\epsilon}$$

In backsubstitution we compute x_2 first. If ϵ is sufficiently small, then given finite precision of number representation $x_2 = 1$ and hence $x_1 = 0$.

Clearly the problem comes from the pivot be too small. Since the order of equations does not matter, we can switch the order of equations:

$$\begin{aligned} x_1 + x_2 &= 2 \\ \epsilon x_1 + x_2 &= 1 \end{aligned}$$

In this case Gaussian elimination leads to

$$x_2 = \frac{1 - 2\epsilon}{1 - \epsilon}, \; x_1 = 2 - x_2.$$

This gives a correct answer. Why the difference? We compute first x_2, and because of the finite precision, it is computed with error. In the procedure of backward substitution to compute x_1, we multiply x_2 with $1/\epsilon$ in the first case, hence error amplification!

What matters in reality is the relative size of the pivoting element with respect to the other elements in the row. In the example that fails to provide an exact solution we may multiply the first equation with $1/\epsilon$. The pivot element will be 1, but nothing will change since the error will be amplified in the same way.

Clearly the pivot elements should be *the largest* to avoid error amplification on backward substitution. This gives the idea: Do elimination in the order such that the relative pivot element magnitude is the largest. The relative pivot element magnitude is the ratio of the pivot element the the largest row element.

We can do *partial* (i.e. row) pivoting, but we can also do full, i.e., row and column, pivoting. The first is sufficient if augmented with scaling.

Algorithm

- Initialize a permutation vector $\mathbf{l} = (1, 2, \ldots, n)$.
- Compute $\mathbf{s}$ such that $s_i = \max_j |a_{ij}|$
- Forward elimination
  - for $k = 1 : n - 1$ (go over all (permuted) pivot rows)
  - for $i = k : n$ (go over all rows below the (permuted) pivot)
  - compute relative pivot element $|a_{l_i k}/s_{l_i}|$
  - find row j with the largest relative pivot element
  - switch l_j and l_k in the permutation vector $\mathbf{l}$
  - perform forward elimination in row l_k
- Perform backward substitution using the order of $\mathbf{l}$

Take a system

$$\left(\begin{array}{cccc|c} 3 & -13 & 9 & 3 & -19 \\ -6 & 4 & 1 & -18 & -34 \\ 6 & -2 & 2 & 4 & 16 \\ 12 & -8 & 6 & 10 & 26 \end{array}\right)$$

Initialize the permutation vector: $\mathbf{l} = (1, 2, 3, 4)$
The vector of maximum magnitudes: $\mathbf{s} = (13, 18, 6, 12)$ First iteration: $k = 1$

$$\left(\left|\frac{a_{l_i k}}{s_{l_i}}\right|\right) = (3/13, 6/18, 6/6, 12/12)$$

Take $j = 3$ and exchange l_k and l_j: $\mathbf{l} = (3, 2, 1, 4)$. Perform the first forward elimination. The row 3 is left as it is, (row 1)=(row 1)-(row 3)/2, (row 2)=(row 2)+(row 3), (row 4)=(row 4)-2*(row 3). The result is

$$\left(\begin{array}{cccc|c} 0 & -12 & 8 & 1 & -27 \\ -0 & 2 & 3 & -14 & -18 \\ 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & 6 \end{array}\right)$$

The second iteration k=2

$$\left(\left|\frac{a_{l_i k}}{s_{l_i}}\right|\right)\Bigg|_{i=2:4} = (2/18, 12/13, 4/12)$$

The maximum is found for $i = 3$, so $j = 3$ and we exchange l_j and l_k to get $\mathbf{l} = (3, 1, 2, 4)$. Now we perform forward elimination for the pivot row $l_2 = 1$: Replace (row 2)=(row 2)+(row 1)/6 and (row 4)=(row 4)+(row 1)/3:

$$\left(\begin{array}{cccc|c} 0 & -12 & 8 & 1 & -27 \\ -0 & 0 & 13/3 & -83/6 & -45/2 \\ 6 & -2 & 2 & 4 & 16 \\ 0 & 0 & 14/3 & 7/3 & -3 \end{array}\right)$$

The third iteration $k = 3$. The relative pivot elements

$$\left(\left|\frac{a_{l_i k}}{s_{l_i}}\right|\right)\Bigg|_{i=3:4} = (13/54, 14/36)$$

We exchange l_4 and l_3 to get $\mathbf{l} = (3, 1, 4, 2)$. The rest of elimination is straightforward. Backward substitution is performed in the reverse order for rows 2, 4, 1 and 3 (i.e., the x_4 is expressed from row 2, and so on).

Gaussian elimination with scaled pivoting works if a unique solution exists, i.e. if the matrix is nonsingular. It is singular if some of rows are linear combinations of other rows.

- A nonsingular matrix is referred to as *regular*. Such a matrix has full rank and has an inverse matrix.
- A square matrix is non-singular iff its determinant is non-zero.
- If matrix is singular, the Gaussian elimination procedure (with or without scaled partial pivoting) at some stage will lead to division by zero because of occurring zero pivots. (Why it is so?) It will therefore fail.
- In practice one need not check if matrix is singular. If it is, a floating exception error will be reported on executing the algorithm.

3.3 Time complexity

This is a synonym to counting operation needed to execute the algorithm. It is easier to count for Gaussian elimination without pivoting. In this case we have three embedded cycles, with the length proportional to n. This means that we need Cn^3 operations, where C is some factor smaller than one because internal cycles are shorter than n. We say that the Gaussian elimination procedure has time complexity $\mathcal{O}(n^3)$. To get the factor C more accurately, we have to note that for $k = 1$ we do a cycle over $n-1$ rows doing n multiplications. For $k = 2$ it becomes $n-2$ and $n-1$ and so on. Altogether this gives $\sum_{i=2}^{n}(i-1)i \approx n^3/3$.

It is easy to see that backward substitution has time complexity proportional to n^2, so it does not influence our estimate.

It is also easy to see that even is scaled partial pivoting is involved, the operation count will increase, but will behave approximately in the same way.

We conclude that the algorithm has complexity $f(n) = \mathcal{O}(n^3)$, i. e, there exist constants c and C such that

$$c|n^3| \leq |f(n)| \leq C|n^3|.$$

Traditionally only operations of multiplication and division are counted. On present-day hardware they can be nearly same fast as the operations of addition and subtraction, and the operations of exchange with memory are not least important.

This mean that there is difference between the time complexity estimated by us and real performance.

The cubic behavior n^3 implies that the number of operations is growing very fast with the size of the system. Are there situations when we can solve equation systems faster? The answer is yes, but for special cases.

3.4 Banded systems

An equation system is called banded if $a_{ij} = 0$ for $|i-j| \leq k < n$ In this case non-zero entries in the system matrix are arranged around its diagonal, as in the matrix below

$$\begin{pmatrix} \times & \times & 0 & \cdots & & \cdots & 0 \\ \times & \times & \times & \ddots & \ddots & \ddots & \vdots \\ 0 & \times & \times & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \times & \times & \times \\ \vdots & \ddots & \ddots & \ddots & 0 & \times & \times \end{pmatrix}$$

It corresponds to the case $k = 1$. Such a banded system is called three-diagonal:

$$\begin{pmatrix} d_1 & c_1 & & & 0 \\ a_1 & d_2 & c_2 & & \\ & a_2 & \ddots & \ddots & \\ & & \ddots & d_{n-1} & c_{n-1} \\ & & & a_{n-1} & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}.$$

The Gaussian elimination algorithm is in this case much simpler. Note also that the number of operation in the elimination phase will be proportional to k, and not n for each row.

The algorithm:

- Forward elimination:
  - for $i = 2 : n$,
  - a_i will be eliminated, we do not need to compute them; all c_i will not change; we only need to update d_i and the right hand side vector components b_i.
  - $d_i := d_i - (a_{i-1}/d_{i-1})c_{i-1}$,
  - $b_i := b_i - (a_{i-1}/d_{i-1})b_{i-1}$,
- Backward substitution:
  - $x_n := b_n/d_n$;
  - for $i = n- : -1 : 1, x_i := (b_i - c_i x_{i+1})/d_i$

The time complexity becomes $\mathcal{O}(n)$ (why?). The method can be generalized to other k (just do Gaussian elimination, but only the operation needed), and the time complexity will remain $\mathcal{O}(n)$.

A caveat: Same as for full version of the Gaussian elimination, here problems may emerge if some d_i are small. Luckily there is a large class of problems where matrices are diagonally dominant, and the pure Gaussian elimination will work for them.

3.5 Errors and ill-conditioned systems

One can always assess the error by inserting the solution found $\tilde{\mathbf{x}}$ into the equation system $\mathrm{A}\mathbf{x} = \mathbf{b}$:

$$\mathbf{r} = \mathrm{A}\tilde{\mathbf{x}} - \mathbf{b}.$$

$\mathbf{r}$ is the *residual vector*. We expect that if $\|\mathbf{r}\|$ is very small compared with $\|\mathbf{b}\|$, our solution is accurate. The norm can be defined in several ways depending on the problem in hand. If all b_i are similar, one can use an L_2 or L_1 norms. Errors occur because of finite accuracy of computations.

An example which we will also encounter later is the Vandermonde matrix

$$\mathrm{A} = \begin{pmatrix} 1 & 2 & 4 & 8 & \cdots & 2^{n-1} \\ 1 & 3 & 9 & 27 & \cdots & 3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n+1 & (n+1)^2 & (n+1)^3 & \cdots & (n+1)^{n-1} \end{pmatrix}$$

Let us construct an equation system with the solution which we know. Take $(x_1, \ldots, x_n)^T = (1, \ldots, 1)^T$ and compute the right hand side vector. Using the expression fora geometric series one finds $b_i = ((i+1)^n - 1)/i$. Now, if we are willing to solve the resulting system $\mathrm{A}\mathbf{x} = \mathbf{b}$, the result's accuracy will depend on its size (n) and the numerical precision used to represent floating numbers. If single precision is used (7 significant digits), one will get truncations in representing all significant digits for the entries in the matrix that are larger than 10^7. For $n = 8$ the entries in matrix and b_i are still in range, but they are out for $n \leq 9$. This will necessarily lead to errors starting from $n = 9$.

We call matrices that may lead to large $\mathbf{r}$ ill conditioned (the precise definition will be given in Numerical Methods II). We conclude that the Vandermonde system is ill conditioned. And we only talked of truncations in representing matrix entries, the loss of significant digits in operations needed to compute the solution will be even more severe. For any given precision in number representation one can construct examples when all matrix entries are of similar magnitude, but the solution is very inaccurate because of term cancelation in computations.

4 LU decomposition, Cholesky decomposition

4.1 LU decomposition

Performing Gaussian elimination we transformed the system matrix to the upper triangular form. Imagine a situation when we have to solve the equation systems several times with the same matrix, but different right hand sides $\mathrm{A}\mathbf{x} = \mathbf{b}$, $\mathrm{A}\mathbf{x} = \mathbf{b}_1$, and so on to $\mathrm{A}\mathbf{x} = \mathbf{b}_N$.

Instead of applying the Gaussian elimination procedure $N + 1$ times we may first find this upper triangular matrix, and also create the matrix that corresponds to the operations on the right hand side. We then will apply these matrices to solve the systems. We shall see that this leads to the decomposition $\mathrm{A} = \mathrm{LU}$, where L and U are the lower and upper triangular matrices.

Let us return to the procedure and the example of $\mathrm{A}\mathbf{x} = \mathbf{b}$ we were using:

$$\begin{pmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 16 \\ 26 \\ -19 \\ -34 \end{pmatrix}$$

In the first elimination step we took the first row as a pivot row, and replaced the other rows as: (row 2)=(row 2)-2*(row 1), (row 3)=(row 3)-(row 1)/2 and (row 4)=(row 4)+2*(row 1). This can be succinctly written as the multiplication of the original system with matrix M_1

$$\mathrm{M}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -1/2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix},$$

to have

$$\mathrm{M}_1\mathrm{A}\mathbf{x} = \mathrm{M}_1\mathbf{b},$$

or

$$\begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ -27 \\ -18 \end{pmatrix}$$

The next elimination step was in taking the second row as a pivot one, and replacing the third and forth with (row 3)=(row 3)-3*(row 2), (row 4)=(row 4)+(row 2)/2. This can be written as a multiplication with

$$\mathrm{M}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & 1/2 & 0 & 1 \end{pmatrix},$$

to have

$$\mathrm{M}_2\mathrm{M}_1\mathrm{A}\mathbf{x} = \mathrm{M}_2\mathrm{M}_1\mathbf{b},$$

or

$$\begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ -9 \\ -21 \end{pmatrix}$$

Then, the final step is equivalent to the multiplication with

$$\mathrm{M}_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -2 & 1 \end{pmatrix},$$

to have

$$\mathrm{M}_3\mathrm{M}_2\mathrm{M}_1\mathrm{A}\mathbf{x} = \mathrm{M}_3\mathrm{M}_2\mathrm{M}_1\mathbf{b},$$

or

$$\begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ -9 \\ -3 \end{pmatrix}$$

Thus, in the end we have that

$$\mathrm{U} := \mathrm{M}_3\mathrm{M}_2\mathrm{M}_1\mathrm{A}$$

is an upper triangular matrix. We got it as the result of forward elimination. Note the terminology: we call a matrix upper triangular if the entries below the diagonal are zeros.

Now, multiplying U with inverse matrices, we get

$$\mathrm{A} = \mathrm{M}_1^{-1}\mathrm{M}_2^{-1}\mathrm{M}_3^{-1}\mathrm{U} = (\mathrm{M}_3\mathrm{M}_2\mathrm{M}_1)^{-1}\mathrm{U}.$$

The matrix

$$\mathrm{L} := \mathrm{M}_1^{-1}\mathrm{M}_2^{-1}\mathrm{M}_3^{-1}$$

is a lower triangular one. This statement relies on the simple structure of M_i whereby the inverse matrices are the same matrices, with nonzero values below the diagonal replaced with their additive inverse:

$$\mathrm{M}_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1/2 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix},$$

$$\mathrm{M}_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & -1/2 & 0 & 1 \end{pmatrix},$$

$$\mathrm{M}_3^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{pmatrix},$$

or

$$\mathrm{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1/2 & 3 & 1 & 0 \\ -1 & -1/2 & 2 & 1 \end{pmatrix}.$$

So the product of all three matrices is obtained by putting ones at the diagonal and adding the entries below the diagonal componentwise! It is easy to see that this result is general and will be valid for general square matrices. This implies that L is computed as a by-product of the Gaussian elimination: Its entries are the negative of the multipliers appearing in the elimination procedure.

THEOREM

Let A be a non-singular square matrix. Then, if Gaussian elimination is possible, there exist a decomposition

$$\mathrm{A} = \mathrm{LU},$$

with L being a lower triangular matrix and U being an upper triangular matrix. Their entries are obtained by the Gaussian elimination procedure.

The procedure of Gaussian elimination will come to stop if we meet zeros at the main diagonal in the process of elimination. The theorem below (without proof) gives conditions when LU decomposition is indeed possible.

THEOREM
If all n leading principal minors of $n \times n$ matrix A are nonsingular, A has an LU-decomposition.

If it is not the case, the existence of an LU decomposition is not guaranteed for nonsingular A. However, since the procedure of scaled partial Gaussian elimination is possible, it will exist for the permutted matrix.

Return to our motivation. We are willing to solve $\mathrm{A}\mathbf{x} = \mathbf{b}$ for multiple right hand side vectors $\mathbf{b}$.

- Note that $\mathrm{A}\mathbf{x} = \mathbf{b}$ is equivalent to $\mathrm{LU}\mathbf{x} = \mathbf{b}$.
- Solve $\mathrm{L}\mathbf{y} = \mathbf{b}$
- Solve $\mathrm{U}\mathbf{x} = \mathbf{y}$

Since we deal with triangular matrices, both steps are done by performing substitutions. It is in forward direction in the first step and in the backward in the second. Clearly this procedure can be generalized to the case when the LU decomposition is done for the permutted matrix.

Time complexity: Computing LU decomposition to solve a single equation $\mathrm{A}\mathbf{x} = \mathbf{b}$ has the same time complexity as the original Gaussian elimination.

However, if one needs to solve this system many times, the LU decomposition will be known from the first solution, and the procedure above can be used for the rest. The time complexity of any of two step is the same as that of back substitution, i.e. $\mathcal{O}(n^2)$ (n rows and less than n multiplication in each of them).

Note that we used direct Gaussian elimination, which is problematic when pivot elements are relatively small. However, the LU decomposition procedure can be done with scaled partial pivoting if needed.

4.2 Cholesky decomposition

For symmetric, positive-definite matrices, in addition to LU, other decompositions can be found. We will introduce the Cholesky decomposition (André Louis Cholesky (1885 – 1918))

Let $\mathrm{A}\mathbf{x} = \mathbf{b}$ be a linear equation system with a symmetric positive-definite matrix A.

- A symmetric, positive-definite A is non-singular and hence we can perform its LU decomposition.
- A symmetric $\mathbf{A}$ can be decomposed as
$$\mathrm{A} = \mathrm{LDL}^T,$$
where D is a diagonal matrix and L is a lower triangular matrix with ones at its diagonal.
- Since A is positive definite, D can be decomposed as
$$\mathrm{D} = \mathrm{D}^{1/2}\mathrm{D}^{1/2}.$$
As a result,
$$\mathrm{A} = \mathrm{LD}^{1/2}\mathrm{D}^{1/2}\mathrm{L}^T = \mathrm{L}_1\mathrm{L}_1^T,$$
where $\mathrm{L}_1 = \mathrm{LD}^{1/2}$ is a lower triangular matrix with positive entries on its diagonal.

How to compute L_1?

- We write a general expression
$$\mathrm{L}_1 = \begin{pmatrix} l_{11} & \cdots & 0 \\ \vdots & \ddots & \\ l_{n1} & \cdots & l_{nn} \end{pmatrix}$$
- We should have
$$\mathrm{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & \cdots & 0 \\ \vdots & \ddots & \\ l_{n1} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & \cdots & l_{n1} \\ & \ddots & \vdots \\ 0 & & l_{nn} \end{pmatrix}$$

- Hence
$$a_{ij} = \sum_{k=1}^{j} l_{ik} l_{jk}, \quad i \geq j.$$
- If $i = j$, $a_{ii} = \sum_{k=1}^{i} l_{ik}^2 \Rightarrow l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$.
- Now let $i > j$, and express l_{ij}: $l_{ij} = (1/l_{jj})(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk})$.
- If $i < j$ $l_{ij} = 0$.

Now, it should be clear that we can first compute l_{11} and when it is known, all l_{i1}. Then l_{22} will be computed, and so on. The procedure will be possible if the expression under square root will be positive. It will be so for a symmetric positive definite matrix, but we need not check for it: If the procedure works, the matrix L_1 exists, hence the matrix A is positive definite. If it does not work, then L_1 does not exist and A is not positive-definite.

The algorithm:

- Do $i = 1 : n$,
- – Do $j = 1 : i - 1$, here $i > j$,
  – $y = a(i, j)$
  – Do $k = 1 : j - 1$, $y = y - l(i, k) * l(j, k)$,
  – $l(i, j) = y/l(j, j)$
- – $y = a(i, i)$ (here $i = j$
  – do $k = 1 : i - 1$, $y = y - l(i, k) * l(i, k)$
  – if $y \leq 0$ exit (no solution exists), else $l(i, i) = \sqrt{y}$.

Although it looks as if we try to compute off-diagonal l_{ij} first, it is not so, for if $i = 1$, the j cycle is empty.

Consider an example. Take
$$\mathrm{A} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{pmatrix} = \begin{pmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} \\ & l_{22} & l_{32} \\ & & l_{33} \end{pmatrix}$$
We have first $l_{11} = \sqrt{a_{11}} = 1$. The other column entries follow as $l_{21} = a_{21}/l_{11} = 1$ and $l_{31} = a_{31}/l_{11} = 1$. Next, $l_{22} = \sqrt{a_{22} - l_{21}^2} = 1$, followed by $l_{32} = (1/l_{22})(a_{32} - l_{31} l_{21}) = 3 - 1 = 2$. Finally $l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2} = 1$. The result is
$$\mathrm{L}_1 = \begin{pmatrix} 1 & & \\ 1 & 1 & \\ 1 & 2 & 1 \end{pmatrix}.$$
It is easy to verify that the original matrix is recovered. What would happen if $a_{33} = 4$?

Similar to the case of LU decomposition we may use the Cholesky decomposition to reduce the burden of computations given the linear equation system $\mathrm{A}\mathbf{x} = \mathbf{b}$ with a symmetric positive-definite system matrix and the need to solve it for multiple right hand sides.

- Given the Cholesky decomposition, $\mathrm{A}\mathbf{x} = \mathbf{b}$ becomes $\mathrm{L}_1 \mathrm{L}_1^T \mathbf{x} = \mathbf{b}$.
- We solve $\mathrm{L}_1 \mathbf{y} = \mathbf{b}$
- We then solve $\mathrm{L}_1^T \mathbf{x} = \mathbf{y}$

Same as in the case of LU decomposition, here we deal with triangular matrices, and solution is straightforward.

The time complexity of Cholesky decomposition is still $\mathcal{O}(n^3)$, however, approximately twice less operations are needed than for the LU decomposition.

5 Nonlinear equations, Bisection method, Newton's method, Secant method

Surface gravity waves $\eta = \eta_0 e^{-i\omega t + ikx}$ propagating over the layer of water with uniform unperturbed thickness H satisfy the dispersion equation $\omega^2 = gk\tanh(kH)$. Here ω is the frequency and $k = 2\pi/\lambda$ the wave number, with λ the wave length. Let perturb the water surface at some $\omega = \Omega$, and ask what is the wave length of the excited wave. We consider positive k and write

$$f(kH) = \frac{\Omega^2 H}{gkH} - \tanh(kH),$$

and finding k is equivalent to solving $f(kH) = 0$ We can try to look at this ploblem graphically, by plotting $y = \Omega^2 H/(gkH)$ and $y = \tanh(kH)$, to see whether they intersect. Take, for example, $\Omega = 1$ s^{-1}, $g = 10$ $\mathrm{m/s}^2$ and $H = 10$ m, which leads to $f(a) = 1/a - \tanh(a)$.

We need to solve a nonlinear equation $f(a) = 0$. The problem is referred to as *locating roots of nonlinear equations*. Any solution a of $f(a) = 0$ is called a *root* of f. We will learn how to solve such and other equations.

Sometimes we have a method how to solve.

- Quadratic equations

$$f(x) = 12x^2 - 5x - 2 = 0 \Rightarrow f(x) = (4x+1)(3x-2) = 0 \Rightarrow x_1 = -1/4,\ x_2 = 2/3.$$

- Some trigonometric equations

$$f(x) = \cos 5x - \cos 3x = 0 \Rightarrow f(x) = -2\sin 4x \sin x = 0,$$

  The roots are $x = \pi n/4, n \in \mathbb{Z}$.

However, there are much more cases when we do not know how to solve, for example

$$f_1(x) = e^x - 5x + 1 = 0,$$

$$f_2 = \log(3 + x^4) + 3\sin x = 0.$$

If there is no obvious way to solve, we need to resort to numerical methods.

- We need a 'robust' root localization method (working for any non-linear equation)
- We do not know a priori how many roots there are. We consider methods that find one root per search.

5.1 Bisection Method

- Let $f \in C^0[a,b]$ ($f(x)$ is continuous on $[a,b]$).
- Let $f(a) \cdot f(b) < 0$. (The function changes sign on $[a,b]$).
- By continuity, $\exists r \in (a,b) : f(r) = 0$.

The following procedure can be proposed:

- The interval $[a,b]$ is bisected into $[a,c]$ and $[c,b]$ with $a < c < b$. There are three possibilities
- $f(c) = 0 \Rightarrow r = c$
- $f(a)f(c) < 0 \Rightarrow r \in (a,c)$ and we continue with this interval.
- $f(a)f(c) > 0 \Rightarrow r \in (c,b)$ and we continue with this interval.

Give graphical example.

- Note that $f(a)f(c) > 0$ does not imply that there are no roots in $[a, c]$, we switch to $[c, b]$ because we know that there is a root there.
- Bisection procedure will find one root in $[a, b]$, but not all roots.

Algorithm

- Input $f(x) \in C^0[a, b] : f(a)f(b) < 0$.
- Find a root in $[a, b]$.
- bisection(a,b)
  - $c := (a + b)/2$;
  - if $c - a < \varepsilon$, return c (ε is tolerance. The algorithm stops if the error is smaller than tolerance).
  - if $f(c) = 0$, return c
  - else if $f(a)f(c) < 0$, bisection(a,c); else bisection(c,b)

Example

- $f(x) = x^5 + 3x + 1$ on $[-1, 1]$.
- $f(x) \in C^0[-1, 1]$, $f(-1)f(1) = (-3)5 = -15 < 0$.
- $a = -1$, $b = 1$, $c = (a + b)/2 = 0$, $f(-1)f(0) = (-3)1 = -3 < 0$. Take $[a, c]$. Reinitialize $b := c = 0$
- $a = -1$, $b = 0$, $c = (a + b)/2 = -1/2$, $f(-1)f(-1/2) = (-3)(-9/16) > 0$. Take $[c, b]$. Reinitialize $a := c = -1/2$
- $a = -1/2$, $b = 0$, $c = (a + b)/2 = -1/4$, $f(-1/2)f(-1/4) = (-9/16)(63/256) < 0$. Take $[a, c]$. Reinitialize $b := c = -1/4$
- $a = -1/2$, $b = -1/4$, $c = (a + b)/2 = -3/8$, $f(-1/2)f(-3/8) > 0$. Take $[c, b]$. Reinitialize $a := c = -3/8$
- $\cdots$ The root is very close to -1/3=-3/9,and we are already close to it, and will be very close after two more iterations.

Convergence
Draw a plot.

- After first iteration $|r - c_1| < (b - a)/2$. Every next iteration will add a factor $1/2$. Hence, after n iterations $|r - c_n| < (b - a)/2^n$
- When the recursive bisection algorithm is applied to $f(x) \in C^0[a, b]$ such that $f(a)f(b) < 0$ on the interval $[a, b]$, after n steps the root r will be approximated with error $E < (b - a)/2^n$.
- Given the maximum error (tolerance) ε we must ensure that $E < \varepsilon$, hence we need $n > \log((b - a)/\varepsilon))/\log 2$

For example, for $\varepsilon = 2^{-20} \approx 10^{-6}$ and $b - a = 1$, we need $n > 20$ iterations.

In fact we proved that the bisection method always converges, and it convergence rate is independent of continuous function f. What will happen if $f(x)$ is discontinuous?

5.2 Newton's Method

Can it be done more efficiently?

Using the bisection algorithm we only required that $f(x) \in C^0[a,b]$.

- Consider $f \in C^1[a,b]$, i.e., $f(x)$ is differentiable.
- At any $x_0 \in (a,b)$ the derivative gives the slope of tangent, so that the tangent line passing through $(x_0, f(x_0))$ is
$$t(x) = f(x_0) + f'(x_0)(x - x_0).$$
- In the vicinity of x_0 $t(x)$ provides a good approximation to $f(x)$.

This geometrical consideration can also be supported analytically. Indeed, the Taylor expansion in small vicinity of x_0
$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \mathcal{O}((x - x_0)^2)$$
implies that the tangent line is a linear approximation to the function $f(x)$.

The idea is to use this approximation and find the root of $t(x)$

$$t(x_1) = f(x_0) + f'(x_0)(x_1 - x_0) = 0 \Rightarrow x_1 = x_0 - f(x_0)/f'(x_0).$$

- The root x_1 of $t(x)$ provides a better approximation of root r if the starting point x_0 was close enough to r.
- Since x_1 is closer to r than x_0, it can be selected as a starting point for a new iteration.
- It can be repeated as many times as necessary, leading to an iterative sequence
$$x_{n+1} = x_n - f(x_n)/f'(x_n).$$
- This gives Newton's method (or Newton–Raphson iteration).

Take our previous example $f(x) = x^5 + 3x + 1 = 0$.

- Take $x_0 = 0$ and compute $f(0) = 1$ and $f'(0) = 5x^4 + 3|_{x=0} = 3$.
- Find $x_1 = x_0 - f(x_0)/f'(x_0) = 0 - 1/3 = -1/3$.
- Compute $f(-1/3) = -1/243$ and $f'(-1/3) = 5/81 + 3 = 248/81$
- Find $x_2 = x_1 - f(x_1)/f'(x_1) = -1/3 + 81/(243 \cdot 248) = -1/3 + 1/744$

Here we did only 2 iterations and already got that the second iteration only adds a small correction. If we continue, we will find

- $x_2 = -0.331989247311828$, $f(x_2) = -6.664064304828798e-07$
- $x_3 = -0.331989029584515$, $f(x_3) = -1.731947918415244e-14$
- The value of $f(x)$ is the measure of error, and it drops quadratically with each new iteration.

A caveat here is the selection of the starting point. If it is not taken carefully, the algorithm may diverge. The initial point needs to be taken sufficiently close to the root, within an interval where the curvature of the function does not change its sign.

Examples:

- Take $f(x) = x/(1 + x^2)$. It has a root $r = 0$. If we take $|x_0| > 1$, x_n will diverge, as can be seen geometrically.
- If we take $|x_0| = 1$, we find $f'(x_0) = 0$, so the algorithm fails from the very beginning.
- There might be other traps such as oscillations of x_n in more complicated cases.

5.3 Secant Method

The drawback of Newton's method is that $f'(x)$ is needed. If $f(x)$ is given analytically, we can supply an analytic expression for it. However, frequently $f(x)$ is given by a complicated routine or algorithm.

We need then an estimate for the derivative.

Since

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h},$$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h},$$

if h is sufficiently small. Set $x := x_n, h := x_{n-1} - x_n$ to get

$$f'(x_n) \approx \frac{f(x_{n-1}) - f(x_n)}{x_{n-1} - x_n}.$$

Then the approximate version of Newton's algorithm becomes

$$x_{n+1} = x_n - \frac{x_{n-1} - x_n}{f(x_{n-1}) - f(x_n)} f(x_n), \quad n = 1, 2, 3 \ldots$$

This is the secant method. Its geometrical interpretation: instead of tangent line $t(x)$ through $(x_n, f(x_n))$ we draw its secant approximation passing through $(x_n, f(x_n))$ and $(x_{n-1}, f(x_{n-1}))$.

In our last example we might take $x_0 = 0$ and $x_1 = -1$. This gives

$$x_2 = x_1 - f(x_1)(x_0 - x_1)/(f(x_0) - f(x_1)) = -1 - (-3)1/4 = -0.25$$

$$x_3 = x_2 - f(x_2)(x_1 - x_2)/(f(x_1) - f(x_2)) \approx -0.307484220018034$$

$$x_4 = x_3 - f(x_3)(x_2 - x_3)/(f(x_2) - f(x_3)) \approx -0.332163533117736$$

The convergence is slower than with Newton's method, after three iterations we have only 2 correct significant digits. Note, however, that the initial phase of the method depends on the initial interval.

5.4 Convergence

We have seen that the bisection method converges always if $f(x)$ is continuous and changes sign on $[a, b]$. The solution accuracy improves one bit per iteration, hence the convergence is linear.

Newton's method:

THEOREM

- Let $f \in C^2[a, b]$, its root $r \in (a, b)$ and $f'(r) \neq 0$
- Then $\exists\, \delta > 0$: if $|r - x_0| \leq \delta$ then
  - a) $|r - x_n| \leq \delta\ \forall n$
  - b) $\lim_{n\to\infty} x_n = r$ and
  - c) $|r - x_{n+1}| \leq c(\delta)|r - x_n|^2$ for some constant $c(\delta)$.

If $f(x)$ is smooth enough and initial point is taken sufficiently close to the root r, the method is stable, convergent, and its convergence is quadratic.

The proof is by induction.

For $n = 0$ a) $|r - x_0| \leq \delta$ is the condition.

c)$|r - x_1| \leq c(\delta)|r - x_0|^2$ can be always achieved if x_1 is finite, and it will be so if δ is sufficiently small.

Assume that all is true for n and prove that it is then true for $n + 1$. We need to show that $|r - x_{n+1}| \leq c(\delta)|r - x_n|^2$ for constnt $c(\delta)$.

$$r - x_{n+1} = r - x_n + \frac{f(x_n)}{f'(x_n)} = \frac{(r - x_n)f'(x_n) + f(x_n)}{f'(x_n)}.$$

Taylor's theorem states: $\exists \zeta_n$ between r and x_n such that

$$0 = f(r) = f(x_n) + (r - x_n)f'(x_n) + (1/2)(r - x_n)^2 f''(\zeta_n)$$

Hence

$$|r - x_{n+1}| \le c(\delta)|r - x_n|^2,$$

where

$$c(\delta) = (1/2)\frac{\max f''(x)}{\min f'(x)}\bigg|_{|x-r|<\delta}$$

Here we rely on $|r - x_n| < \delta$ and $|r - \zeta_n| < \delta$.

Now a) follows from this estimate:

$$|r - x_{n+1}| \le c(\delta)|r - x_n|^2 \le \rho|r - x_n|, \quad \rho = |r - x_n|c(\delta) \le \delta c(\delta)$$

By selecting δ sufficiently small ρ can be made less than one, which complete the proof of a). The proof of b) follows by writing

$$|r - x_{n+1}| \le \rho|r - x_n| \le \dots \rho^n|r - x_0| \to 0 \,\text{for}\, n \to \infty.$$

The convergence rate for the secant method:

$$|r - x_{n+1}| \le c|r - x_n|^\alpha$$

where $\alpha = (1 + \sqrt{5})/2 \approx 1.6$ It is better than linear, but worse than quadratic of Newton's method.

5.5 System of nonlinear equations

Newton's method can be generalized to find roots of systems of nonlinear equations. We consider the case of two equations, and the rest can be done similarly.

Let $f(x, y)$ and $g(x, y)$ be C^1 on $S:\ a_x \le x \le b_x, a_y \le y \le b_y$ such that $f(r_x, r_y) = 0$ and $g(r_x, r_y) = 0$ at some $(r_x, r_y) \in S$.

We are willing to find a root of the system of equations

$$\begin{cases} f(r_x, r_y) = 0, \\ g(r_x, r_y) = 0. \end{cases}$$

For $(x_0, y_0) \in S$, which is sufficiently close to the root, we can write

$$f(x, y) \approx f(x_0, y_0) + \partial f/\partial x|_{(x_0,y_0)}(x - x_0) + \partial f/\partial y|_{(x_0,y_0)}(y - y_0) = R_f(x, y),$$

$$g(x, y) \approx g(x_0, y_0) + \partial g/\partial x|_{(x_0,y_0)}(x - x_0) + \partial g/\partial y|_{(x_0,y_0)}(y - y_0) = R_g(x, y).$$

Here the rhs define tangent planes $z(x, y) = R_f$, $z(x, y) = R_g$, and their intersection with $z = 0$ will define a point (x_1, y_1) which should be closer to the root (r_x, r_y) than the original point $x_0, y_0)$: We find

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} - \mathrm{J}^{-1}_{(x_0,y_0)} \begin{pmatrix} f(x_0, y_0) \\ g(x_0, y_0) \end{pmatrix}$$

Where J is the Jacobian matrix

$$\mathrm{J} = \begin{pmatrix} \partial f/\partial x & \partial f/\partial y \\ \partial g/\partial x & \partial g/\partial y \end{pmatrix}$$

Clearly, the matrix should be nonsingular for the procedure were possible. Because we require the continuity of derivatives, we must reguire that the Jacobian is nonsingular at the root. Then it will be nonsingular in some vicinity of the root.

The iterations can be continued, and the general formula will be obtained by replacing the index 0 with i and 1 with $i + 1$.

6 Polynomial interpolation. Lagrange interpolation

Doing measurements, one gets a set of values of the measured quantity for the set of parameters. For example, we may measure the density or viscosity of water for a set of temperatures. Now we are willing to learn how to estimate the density at some intermediate temperature.

More complicated tasks may occur if we carry out measurements over some domain, but a finite set of locations. For example, we are interested in assessing the temperature distribution in a lake or a sea, but can afford measurements only at a number of location. How to estimate the temperature at a location that was not occupied?

To answer these questions we do interpolation.

Some examples.

Nearest interpolation. Take the value at the closest location.

Linear interpolation: Let the measured quantity v be v_1 at x_1 and v_2 at $x_2 > x_1$. For $x =\in [x_1, x_2]$ we take

$$v(x) = \frac{x_2 - x}{x_2 - x_1} v_1 + \frac{x - x_1}{x_1 - x_0} v_2.$$

This is the equation of line drawn through points (x_1, v_1) and (x_2, v_2).

Explain graphically.

We may see that sometimes our data points deviate from the linear line. This hints to the need of higher-order interpolation.

Terminology:

- We will call *points* the set of data values $\mathbf{p}_0, \ldots, \mathbf{p}_n \in \mathbb{R}^d$
- We will call *knots* the ordered set of variables $u_0, \ldots, u_n$: $u_0 < u_1 < \ldots < u_n$, related to the data.
- A function (curve) $\mathbf{p}(u)$ such that $\mathbf{p}(u_i) = \mathbf{p}_i$, $i = 0, \ldots, n$ is called *interpolating* function (curve).

Note that terminology is a bit ambiguous, for points are sometimes used in place of knots. Points==data points in our case.

Select a set of polynomial functions $\phi_0(u), \ldots \phi_n(u)$.

Write

$$\begin{pmatrix} \phi_0(u_0) & \ldots & \phi_n(u_0) \\ \vdots & & \vdots \\ \phi_0(u_n) & \ldots & \phi_n(u_n) \end{pmatrix} \begin{pmatrix} \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} = \begin{pmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_n \end{pmatrix}.$$

Here the coefficient $\mathbf{x}_0, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ are caled *nodes*. They are the weight we multiply the polynomial functions to get interpolation. In the matrix form

$$\Phi \mathbf{X} = \mathbf{P}$$

Here the matrix Φ is called the *collocation matrix*. It is $n+1$ by $n+1$ in size. Clearly the matrix should be non-singular in order the equation system be solvable.

- Φ is non singular if ϕ_i form a basis in the space of polynomials of degree $\leq n$.
- The result is

$$\mathbf{p}(u) = \sum_{i=0}^{n} \phi_n(u) \mathbf{x}_i.$$

  $\mathbf{p}(u)$ *interpolates* the points $\mathbf{p}_i$ at knots u_i

An example: Take monomials $\phi_i(u) = u^i$, $i = 0, \ldots, n$. They obviously form a basis since any polynomial of degree up to n. This case will lead to the Vandermonde matrix we already met

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 2^2 & \ldots & 2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & n+1 & (n+1)^2 & \ldots & (n+1)^n \end{pmatrix} \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} = \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_n \end{pmatrix}$$

Recall possible difficulties with solving this system if n is large (badly conditioned matrix).

For $n = 2$ and $d = 1$: $\begin{array}{c|c|c|c} u_i & 1 & 2 & 3 \\ \hline p_i & 2 & 6 & 12 \end{array}$

The equation system to determine the nodes x_i

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \\ 12 \end{pmatrix}$$

The solution is $x_1 = 0$, $x_2 = 1$, $x_3 = 1$. The interpolating function $p(u) = 0 \cdot u^0 + 1 \cdot u + 1 \cdot u^2 = u + u^2$.

6.1 Lagrange interpolation

If n is large, solving the linear equation system above can be difficult. Is it possible to propose a better basis?

- The idea is due to Lagrange (1736-1813). It consists in defining a special basis so that each ϕ_i satisfies the property $\phi_i(u_j) = \delta_{ij}$.
- The matrix Φ becomes an identity matrix.
- No solution is needed, for $\mathbf{x}_i = \mathbf{p}_i$ (nodes=points).

Thus, given $n + 1$ points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ and the related knots $u_0, \ldots, u_n$ the interpolating function is written as

$$\mathbf{p}(u) = \sum_{i=0}^{n} \mathbf{p}_i L_i^n(u),$$

where L_i^n are called Lagrange polynomials. Given the set of $n+1$ knots they are uniquely defined by the requirement $L_i^n(u_j) = \delta_{ij}$. Here δ_{ij} is the Kronecker delta, equal to one if both indices coincide, and zero otherwise.

Assume we have just u_0 and u_1, so $n = 1$. In this case obviously $L_0^1 = (u - u_1)/(u_0 - u_1)$ and $L_1^1 = (u_0 - u)/(u_0 - u_1)$. This gives an idea of how to construct the Lagrange polynomials.

A multiplier $(u - u_i)$ included into the polynomial would ensure that it becomes zero at $u = u_i$.

For $n = 2$ we have u_0, u_1 and u_2. Take linear polynomials $l_1 = (u-u_0)$, $l_1 = (u-u_1)$ and $l_2 = (u-u_2)$. To construct L_0^2 we write

$$L_0^2 = l_1 l_2 / (l_1(u_0) l_2(u_0)),$$

and similarly for the rest. Note that including l_1 and l_2 we ensure that L_0^2 is zero at u_1 and u_2, and the denominator ensures normalization to unity at $u = u_0$.

Draw examples. The general expression is

$$L_i^n = \frac{\prod_{j=0, j\neq i}^{n} (u - u_j)}{\prod_{j=0, j\neq i}^{n} (u_i - u_j)}.$$

Consider the interpolation problem $\begin{array}{c|c|c|c} u_i & 0 & 1 & 2 \\ \hline p_i & 1 & 2 & 4 \end{array}$ First write explicitly the Lagrange polynomials for $n = 2$.

$$L_0^2(u) = \frac{(u - u_1)(u - u_2)}{(u_0 - u_1)(u_0 - u_2)} = (u - 1)(u - 2)/2 = (u^2 - 3u + 2)/2;$$

$$L_1^2(u) = \frac{(u - u_0)(u - u_2)}{(u_1 - u_0)(u_1 - u_2)} = -u(u - 2) = -u^2 + 2u;$$

$$L_2^2(u) = \frac{(u - u_0)(u - u_1)}{(u_2 - u_0)(u_2 - u_1)} = u(u - 1)/2 = u^2/2 - u/2;$$

The interpolating function is

$$p(u) = L_0^2(u) + 2L_1^2(u) + 4L_2^2(u).$$

Some properties:

Partition of unity:

$$\sum_{i=0}^{n} L_i^n(u) \equiv 1$$

A recursive computation: If $i \leq k-1$,

$$L_i^k(u) = L_i^{k-1}(u)\alpha_{ik}, \quad \alpha_{ik} = \frac{u - u_k}{u_i - u_k}.$$

If $i = k$, we use partition of unity:

$$L_k^k(u) = 1 - \sum_{i=0}^{k-1} L_i^k(u) = 1 - \sum_{i=0}^{k-1} L_i^{k-1}(u)\alpha_{ik} = \sum_{i=0}^{k-1} L_i^{k-1}(u)(1 - \alpha_{ik}).$$

7 Newton interpolation. Bézier curves

7.1 Newton interpolation

Although the Lagrange polynomials are convenient, we have to recompute all them if the number of knots n is changed. Can we propose an approach where adding a knot only requires incremental computation?

Newton polynomials

- Given: An ordered set of knots $u_0 < u_1 \ldots < u_n$.
- Construct the polynomials P_i, $i = 0, \ldots, n$ so that P_i is of degree i with roots at knots $u_0, \ldots, u_{i-1}$.
- Then $P_i(u) = \prod_{j=0}^{i-1}(u - u_j)$ if $i \neq 0$, and $P_0(u) = 1$.
- Polynomials $P_0(u), \ldots, P_n(u)$ form a basis for the space of polynomials up to degree n.

Newton interpolation:

- Let $\mathbf{p}[u_0 \ldots u_i]$ be the leading coefficient of the polynomial of degree i interpolating points $\mathbf{p}_0, \ldots, \mathbf{p}_i$ at knots $u_0, \ldots, u_i$.
- The function $\mathbf{p}(u)$ that interpolates all the points $\mathbf{p}_0, \ldots, \mathbf{p}_n$ at all knots $u_0, \ldots, u_n$ is

$$\mathbf{p}(u) = \sum_{i=0}^{n} \mathbf{p}[u_0 \ldots u_i] P_i(u).$$

- Hence $\mathbf{p}[u_0 \ldots u_i]$ are the nodes $\mathbf{x}_i$.

Compare Newton and Lagrange interpolation.

The Newton polynomials are simpler to compute and Newton interpolation can be extended from n to $n+1$ points by only computing one additional polynomial and respective node. In contrast, we have to recompute all the polynomials in Lagrange interpolation.

However, Newton interpolation requires computing the nodes $\mathbf{x}_i = \mathbf{p}[u_0 \ldots u_i]$, whereas this is trivial for Lagrange interpolation where $\mathbf{x}_i = \mathbf{p}_i$.

Example: Given

u_i	0	1	2
p_i	1	2	4

- 1-point interpolation

$$p(u) = p[u_0]P_0(u) = p[u_0] \cdot 1 \Rightarrow p[u_0] = p(u_0) = p_0 = 1.$$

The interpolating polynomial is $p(u) = 1 \cdot P_0(u) = 1$;

- 2-point interpolation

$$p(u) = p[u_0]P_0(u) + p[u_0u_1]P_1(u) = 1 + p[u_0u_1](u - 0) \Rightarrow p(u_1) = 1 + p[u_0u_1]u_1 = p_1.$$

Hence $p[u_0u_1] = p_1 - 1 = 1$ and the interpolating polynomial becomes $p(u) = 1 \cdot P_0(u) + 1 \cdot P_1(u) = 1 + u$;

- 3-point interpolation

$$p(u) = p[u_0]P_0(u) + p[u_0u_1]P_1(u) + p[u_0u_1u_2]P_2(u) = 1 + 1 \cdot (u - 0) + p[u_0u_1u_2](u - 0)(u - 1).$$

Equate

$$p(u_2) = 1 + 2 + p[u_0u_1u_2] \cdot 2 \cdot 1 = p_2 = 4, \Rightarrow p[u_0u_1u_2] = 1/2;$$

The interpolating polynomial is $p(u) = 1 \cdot P_0(u) + 1 \cdot P_1(u) + (1/2)P_2(u) = 1 + u + u(u-1)/2$.

- Obviously, the resulting polynomial is the same as in expansion in Lagrange polynomials. There is only a unique polynomial of second degree interpolating three points (of nth degree passing through $n+1$ points).
- The collocation matrix for Newton polynomials is lower triangular, so the nodes of interpolation can be obtained by substitution, as we have seen in the example.
- Without proof, the solution for nodes is

$$\mathbf{p}[u_0 \dots u_i] = \sum_{k=0}^{i} \frac{\mathbf{p}_k}{\prod_{j=0, j\neq k}^{i} (u_k - u_j)}.$$

Since the collocation matrix is lower triangular, the time complexity of Newton interpolation is $\mathcal{O}(n^2)$.

7.2 Divided differences

For the polynomial interpolation in the Newton form we used special notation for the nodes (coefficients of interpolation). Let we are given a set of points $\mathbf{p}_i$ and knots u_i, $i = 0, \dots, n$. We require the knots to be distinct (but not necessarily ordered).

The Newton polynomials for the given set of knots are

$$\begin{aligned} P_0(u) &= 1, \\ P_1(u) &= u - u_0, \\ P_2(u) &= (u - u_0)(u - u_1), \dots \\ P_n(u) &= \prod_{i=0}^{n-1} (u - u_i). \end{aligned}$$

The interpolating polynomial is then

$$p(u) = \sum_{i=0}^{n} c_i P_i(u).$$

The collocation matrix is in this case lower triangular. We can proceed without writing it. Indeed,

$$p(u_0) = c_0 \cdot 1 = p_0 \Rightarrow c_0 = p_0;$$

$$p(u_1) = c_0 \cdot 1 + c_1(u_1 - u_0) = p_1 \Rightarrow c_1 = (p_1 - p_0)/(u_1 - u_0);$$

and so on. We see that c_0 depends on p_0, c_1 depends on p_0 and p_1. We therefore use the notation $c_i = p[u_0 \dots u_i]$. The quantities $p[u_0 \dots u_n]$ is called *divided difference* of order n of $p(u)$ at knots $u_0, \dots, u_n$. It represents the coefficient appearing with u^n in the interpolating polynomial $p(u)$. These quantities are also the coefficients appearing with the Newton polynomials in the interplating polynomial $p(u)$. Obviously, if we interpolate only $i+1$ first point, $p[u_0 \dots u_i]$ is the coefficient with the highest monomial u^i.

Since the interpolating polynomial is unique, $p[u_0 \dots u_n]$ does not depend on permutation of points we are interpolating. Indeed, the set of requirements $p(u_i) = \mathbf{p}_i$ defines a unique $p(u)$. Obviously it does not depend on the order we are writing the conditions $p(u_i) = \mathbf{p}_i$.

The expression for $p[u_0 u_1]$ above can be also written as

$$\mathbf{p}[u_0 u_1] = \frac{\mathbf{p}_1 - \mathbf{p}_0}{u_1 - u_0} = \frac{\mathbf{p}[u_1] - \mathbf{p}[u_0]}{u_1 - u_0}.$$

It turns out that a similar recursion formula is valid for any order.

THEOREM
Divided differences satisfy

$$\mathbf{p}[u_0 \dots u_n] = \frac{\mathbf{p}[u_1 \dots u_n] - \mathbf{p}[u_0 \dots u_{n-1}]}{u_n - u_0}.$$

Proof:
Let q be a polynomial that interpolates $\mathbf{p}_1, \ldots, \mathbf{p}_n$ at $u_1, \ldots u_n$, and let q_{n-1} be a polynomial that interpolates $\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}$ at $u_0, \ldots u_{n-1}$. Then, the polynomial

$$q_n(u) = q(u) - \frac{u - u_n}{u_0 - u_n}(q(u) - q_{n-1}(u))$$

interpolates $\mathbf{p}_0, \ldots, \mathbf{p}_n$ at $u_0, \ldots u_n$. Indeed, we need only to check that $q_n(u_0) = \mathbf{p}_0$ and that $q_n(u_n) = \mathbf{p}_n$, which is indeed so. Since the divided differences are the coefficients with the highest order monomials, we have that $\mathbf{p}[u_0 \ldots u_n]$ on the left equals to the coefficient on the right hand side, which is the statement of the theorem.

7.3 Errors

Given the points and knots, we get a unique polynomial interpolating all them. To estimate the occurring errors one takes a known function $f(u)$ and samples it at the given knots u_i to obtain the corresponding points p_i. One then compares the result $p(u)$ with $f(u)$. At knots $f(u) = p(u)$, but it is not necessarily so at other u.

Interpolation error theorem

If $p(u)$ is a polynomial of degree $\leq n$ that interpolates $f(u) \in C^{n+1}[a, b]$ at $n + 1$ distinct knots $u_0, u_1, \ldots, u_n \in [a, b]$, then for any $u \in [a, b]$ there exists $\xi \in [a, b]$ such that

$$f(u) - p(u) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (u - u_i)$$

Proof:
Let

$$w(u) = \prod_{i=0}^{n} (u - u_i).$$

For $u = u_i$ we have $w(u_i) = 0$ and $f(u_i) - p(u_i) = 0$. In this case the theorem assertion is obviously true. For $u \neq u_i$ $w(u) \neq 0$. Let

$$c = (f(u) - p(u))/w(u).$$

For a fixed u, c=const. Consider now a function

$$\phi(x) = f(x) - p(x) - cw(x).$$

Obviously $\phi(u) = 0$. Moreover, $\phi(u_i) = 0$ for all knots u_i, hence $\phi(x)$ has $n + 2$ roots. It follows then that its $n + 1$ derivative has 1 root. Let ξ be that root, then

$$0 = \phi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - p^{(n+1)}(\xi) - cw^{(n+1)}(\xi),$$

which proves the claim because $p^{(n+1)}(\xi) = 0$ and $w^{(n+1)}(\xi) = (n + 1)!$.

Consider $f(x) = \sin x$ on $x \in [0, 1]$. We use a polynomial of degree nine (10 knots) to interpolate it on this interval. Since $|f^{(n+1)}| \leq 1$ and $\prod_{i=0}^{9} |x - x_i| \leq 1$, we have

$$|\sin x - p(x)| \leq \frac{1}{10!}.$$

To get a better estimate we need to know the knots.

Since the magnitude of the product term depends on the knots, and a question arises whether the knots can be selected in an optimized way. The answer is yes, and involves the Chebyshev polynomials,

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \, (n \geq 1).$$

Their closed form is $T_n = \cos(n \cos^{-1} x)$, $n \geq 0$. They are considered on the interval $[-1, 1]$.

A theorem (without proof) states that

$$\max_{|x|\leq 1}\left|\prod_{i=0}^{n}(x-x_i)\right| \geq 2^{-n},$$

and the minimum is attained when $\prod_{i=0}^{n}(x-x_i) = 2^{-n}T_{n+1}$. In this case x_i are the roots of the polynomial, i.e.,

$$x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right) \quad (0 \leq i \leq n).$$

The interpolation error becomes

$$|f(x)-p(x)| \leq \frac{1}{2^n(n+1)!}\max_{|\xi|\leq 1}|f^{(n+1)}(\xi)|.$$

For a continuous function on an interval $[a,b]$ one expects to have a uniform convergence of interpolating polynomials as their degree is increased. In fact, if n is increased, polynomial interpolation $p(u)$ may exhibit large-amplitude oscillations, and may not converge to the function f. A famous example dating back to 1901 considers $f(u) = 1/(1+u^2)$ on the interval $[-5,5]$ for equidistant knots. There are large oscillations closer to ends of this interval, and $\|f(u)-p_n(u)\|_\infty$ is in fact unbounded.

For any set of knots there are continuous functions such that interpolating polynomials fail to converge to them. But for any continuous function there exist a system of knots such that $\|f(u)-p_n(u)\|_\infty$ tends to zero with n.

7.4 Bézier Curves

Polynomial bases are convenient to use. We were demanding previously that points are exactly interpolated. Now, we will explore another possibility: Instead of requiring that points are interpolated, we seek an approach where the interpolating curve would converge to the interpolated function as the number of knots (samples) increases.

Binomial expansion

$$1 = 1^n = (u+(1-u))^n = \sum_{i=0}^{n}\binom{n}{i}u^i(1-u)^{n-i}.$$

Each term in the sum is a polynomial of degree n.

- Degree 0:
$$B_0^0(u) = 1$$
- Degree 1:
$$B_0^1(u) = 1-u, \quad B_1^1(u) = u$$
- Degree 2:
$$B_0^2(u) = (1-u)^2, \quad B_1^2(u) = 2u(1-u), \quad B_2^2(u) = u^2$$

B_i^n are the Bernstein polynomials. They are linearly independent, hence form a basis. Properties:

- Symmetry
$$B_i^n(u) = B_{n-1}^n(1-u)$$
- Roots:
$$B_i^n(0) = B_{n-i}^n(1) = \begin{cases}1 & \text{if } i=0\\ 0 & \text{otherwise}\end{cases}$$
- Partition of unity
$$\sum_{i=0}^{n}B_i^n(u) = 1$$

- Positiveness:

$$B_i^n(u) > 0,\ u \in (0,1).$$

Recursive definition

$$B_0^0 = 1$$

$$B_i^{n+1}(u) = uB_{i-1}^n(u) + (1-u)B_i^n(u),$$

with the agreement $B_{-1}^n(u) = B_{n+1}^n(u) \equiv 0$.

Definition:

- The polynomial function

$$\mathbf{b}(u) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(u),\ u \in [0,1]$$

  is called the *Bézier representation* of degree n.
  The nodes $\mathbf{b}_i$ are called the *Bézier points*. The Bézier points form a Bézier polygon.

- Since $B_i^n(u)$ form a basis, each polynomial of degree up to n has a unique Bézier representation. P. Bézier (1910-1999)

- Although defined on the interval [0,1], it can be generalized to arbitrary interval $[a,b]$ by the variable change

$$t = \frac{u-a}{b-a},\ t \in [0,1],\ u \in [a,b].$$

Any Bézier representation $\mathbf{b}(u)$ on the interval $[a,b]$ interpolates the endpoints: $\mathbf{b}(a) = \mathbf{b}_0$, $\mathbf{b}(b) = \mathbf{b}_n$. This does not necessarily hold for other points.

Examples

If we have two points $\mathbf{b}_0$ and $\mathbf{b}_1$, the Bézier curve is a line connecting them,

$$\mathbf{b}(u) = (1-u)\mathbf{b}_0 + u\mathbf{b}_1.$$

In this case we only have end points.

If we have three points $\mathbf{b}_0$, $\mathbf{b}_1$ and $\mathbf{b}_2$, we deal with quadratic Bézier curves

$$\mathbf{b}(u) = (1-u)^2\mathbf{b}_0 + 2u(1-u)\mathbf{b}_1 + u^2\mathbf{b}_2.$$

For cubic case we write

$$\mathbf{b}(u) = (1-u)^3\mathbf{b}_0 + 3u(1-u)^2\mathbf{b}_1 + 3u^2(1-u)\mathbf{b}_2 + u^3\mathbf{b}_3.$$

Convergence theorem:

- Let $f(u)$ be a continuous function, $f(u) \in C^0[0,1]$.

- Let $b(u)$ be the Bézier representation (curve) with nodes $b_i = f(i/n)$, i.e., nodes represent an equidistant sampling of $f(u)$ on [0,1],

$$b(u) = \sum_{i=0}^{n} f(i/n)B_i^n(u)$$

- Then

$$\lim_{n\to\infty} \sup_{u\in[0,1]} |f(u) - b(u)| = 0$$

We take it now without proof. Points (u_i, b_i) form a Bézier polygon. The Bézier curve $b(u)$ lies inside the convex hull around the Bézier points. The function $f(u)$ passes through the points, but the Bézier curve passes only through the endpoints.

8 Piecewise interpolation. Spline interpolation

We observed with Lagrange and Newton's polynomials that if they fulfil many constraints (interpolate many points), they tend to oscillate between the points. A complementary approach would be to use lower-order polynomials, each defined only on a part of knots. The challenge is to join these interpolating polynomials so that the interpolating curve is of desired smoothness.

8.1 Splines

Piecewise polynomials of lower degree may have less oscillations than high-order interpolations.

Definition

A function $\mathbf{s}(u)$ with domain $[a, b]$ is calles a spline of degree k (or of order $k + 1$) if

- $\mathbf{s}(u) \in C^{k-1}[a, b]$ and
- there are knots $a = u_0 < u_1 < \ldots < u_n = b$ such that $\mathbf{s}(u)$ is a polynomial of degree k on each subinterval $[u_i, u_{i+1}]$ for $i = 0, \ldots, n - 1$.

Spline of degree 0 are piecewise linear constants.

$$s(u) = \begin{cases} s_0(u) = c_0 & u \in [u_0, u_1) \\ s_1(u) = c_1 & u \in [u_1, u_2) \\ \vdots & \vdots \\ s_{n-1}(u) = c_{n-1} & u \in [u_{n-1}, u_n) \end{cases}$$

We select interval so that there is no ambiguity, but other variants are possible.

Splines of degree 1 are piecewise linear functions. We can write them as

$$s(u) = \begin{cases} s_0(u) = a_0u + b_0 & u \in [u_0, u_1) \\ s_1(u) = a_1u + b_1 & u \in [u_1, u_2) \\ \vdots & \vdots \\ s_{n-1}(u) = a_{n-1}u + b_{n-1} & u \in [u_{n-1}, u_n) \end{cases}$$

In this case the representation is continuous. If the coefficients a_i and b_i are determined, the interpolation consists first in finding an interval $[u_i, u_{i+1})$ that contains u, and then in using the expression for $s(u)$ within this interval. Sometimes for convenience the expression s_0 is extended to the entire inerval $(-\infty, u_1)$, and the expression for s_{n-1} is extended to $[u_{n-1}, \infty)$.

- for $i = 1, 2, \ldots, n - 1$,
- if $u < u_i$ then $s(u) = s_{i-1}$, output $s(u)$, exit, end if
- end, $s(u) = s_{n-1}(u)$, output $s(u)$

Cubic splines are used most frequently in practice. We are given the set of knots $u_0, \ldots, u_n$ and points $p_0, \ldots, p_n$. We seek the interpolation such that $s_i(u)$ is a cubic polynomial on $[u_i, u_{i+1}]$. Polynomials $s_{i-1}(u)$ and $s_i(u)$ interpolate the same point u_i at their boundary and hence are continuous. The derivatives $s'(u)$ and $s''(u)$ should be continuous too.

There are n intervals, and 4 unknown coefficients in each. On each interval there are 2 interpolation conditions $s_i(u_i) = p_i$ and $s_i(u_{i+1}) = p_{i+1}$. The continuity of $s'(u)$ and $s''(u)$ give another $2(n - 1)$ conditions. 2 remaining conditions are set separately.

Let z_i be as yet unknown values of second derivatives at u_i. Then

$$s_i''(u) = \frac{z_i}{h_i}(u_{i+1} - u) + \frac{z_{i+1}}{h_i}(u - u_i), \quad h_i = u_{i+1} - u_i.$$

or

$$s_i(u) = \frac{z_i}{6h_i}(u_{i+1} - u)^3 + \frac{z_{i+1}}{6h_i}(u - u_i)^3 + C(u - u_i) + D(u_{i+1} - u).$$

Since it has to pass through the points at the ends of its interval,

$$C = \left(\frac{p_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right), \quad D = \left(\frac{p_i}{h_i} - \frac{z_i h_i}{6}\right).$$

It remains to determine z_i, which is done from the continuity of first derivatives $s_i'(u_i) = s_{i-1}'(u_i)$ at u_i, which gives

$$h_{i-1}z_{i-1} + (h_i + h_{i-1})z_i + h_i z_{i+1} = \frac{6}{h_i}(p_{i+1} - p_i) - \frac{6}{h_{i-1}}(p_i - p_{i-1}).$$

This is a three-diagonal system, however it is still underdetermined, because it cannot be used for $i = 0$ and $i = n$. One takes commonly $z_0 = z_n = 0$, which leads to a *natural* spline. The system is diagonally dominant and can be solved by Gaussian elimination.

After z_i are found, the cubic polynomials s_i can be determined. Given u, we first find the interval it belongs to. By convention, s_0 will be used on $(-\infty, u_1)$, and s_n on (u_{n-1}, ∞). The interval is found by testing $u - u_i$ and looking where the sign is changed.

To minimize the computational work, the expression for $s_i(u)$ is rearranged as

$$s_i(u) = p_i + (u - u_i)[C_i + (u - u_i)[B_i + (u - u_i)A_i]],$$

with

$$A_i = (z_{i+1} - z_i)/(6h_i),\ B_i = z_i/2,\ C_i = -h_i z_{i+1}/6 - h_i z_i/3 + (p_{i+1} - p_i)/h_i.$$

Here $u - u_i$ should be known after the interval search.

THEOREM

Let $f \in C^2[a, b]$ and $a = u_0 < u_1 < \ldots < u_n = b$. If $s(u)$ is the natural spline interpolating f at knots u_i, then

$$\int_a^b [s''(x)]^2 dx \le \int_a^b [f''(x)]^2 dx.$$

The proof consists in defining $g = f - s$ and proving that $\int_a^b s''g''dx \ge 0$. Natural spline is the smoothest possible interpolating function.

As an example we consider $(u_0, u_1, u_1) = (1, 0, 1)$:

$$s(u) = au^3 + bu^2 + cu + d, \quad u \in [-1, 0],$$

$$s(u) = eu^3 + fu^2 + gu + h, \quad u \in [0, 1].$$

Interpolation conditions:

$$s(-1) = 1 \Rightarrow -a + b - c + d = 1$$

$$s(0) = 2 \Rightarrow d = h = 2$$

$$s(1) = -1 \Rightarrow e + f + g + h = -1$$

The first derivative:

$$s'(u) = 3au^2 + 2bu + c, \quad u \in [-1, 0];$$

$$s'(u) = 3eu^2 + 2fu + g, \quad u \in [0, 1].$$

Its continuity at point $u_1 = 0$ gives $c = g$.

The second derivative:

$$s''(u) = 6au + 2b,\ u \in [-1, 0],\ s''(u) = 6eu + 2f,\ u \in [0, 1].$$

Its continuity gives $b = f$. We still have insufficient number of equations. The two remaining ones are suplied by the natural spline conditions:

$$s''(-1) = 0 \Rightarrow -6a + 2b = 0; \quad s''(1) = 0 \Rightarrow -6e + 2f = 0.$$

Solving the resulting system of 8 linear equations one gets

$$s(u) = \begin{cases} -u^3 - 3u^2 - u + 2,\ u \in [-1, 0] \\ u^3 - 3u^2 - u + 2,\ u \in [0, 1] \end{cases}$$

Tension splines

In some situations we want to have more control over interpolated functions. In tension splines we seek $s(u) : s(u) \in C^2[u_0, u_n]$, $s(u_i) = p_i\, (0 \le i \le n)$ and $s^{(4)}(u) - \tau^2 s''(u) = 0$ on each interval (u_i, u_{i+1}).

If $\tau = 0$, we return to cubic polynomials. If τ is large, we get linear functions except for the vicinities of points $u = u_i$ where $s(u)$ has to change.

We write first

$$\begin{aligned} s_i^{(4)}(u) - \tau^2 s_i''(u) = 0, \\ s(u_i) = p_i,\ s(u_{i+1}) = p_{i+1}, \\ s''(u_i) = z_i,\ s''(u_{i+1}) = z_{i+1}, \end{aligned}$$

where z_i are as yet unknown.

The solution is (verify)

$$\begin{aligned} s_i(u) = (z_i \sinh(\tau(u_{i+1} - u)) + z_{i+1} \sinh(\tau(u - u_i))/(\tau^2 \sinh(\tau h_i)) \\ +(p_i - z_i/\tau^2)(u_{i+1} - u)/h_i + (p_{i+1} - z_{i+1}/\tau^2(u - u_i)/h_i. \end{aligned}$$

Same as with natural splines we get a three-diagonal system of linear equations demanding that first derivatives are continuous through the knots u_i

$$\alpha_{i-1} z_{i-1} + (\beta_{i-1} + \beta_i) z_i + \alpha_i z_{i+1} = \gamma_i - \gamma_{i-1},\ (1 \le i \le n-1)$$

where

$$\begin{aligned} \alpha_i = 1/h_i - \tau/\sinh(\tau h_i), \\ \beta_i = \tau \cosh(\tau h_i)/\sinh(\tau h_i) - 1/h_i, \\ \gamma_i = \tau^2 (p_{i+1} - p_i)/h_i. \end{aligned}$$

Same as with natural spline, the system is completed by $z_0 = z_n = 0$. One computes the coefficients and solves the system for z_i through Gaussian elimination. Then the formula for $s_i(u)$ can be used once the interval (u_i, u_{i+1}) containing u is defined.

8.2 Higher-degree natural splines

Natural splines can be introduced for odd degrees, so one writes their degree as $2m+1$, and cubic splines become the case $m = 1$. They are polynomials of degree $\le 2m+1$. For convenience, introduce truncated power function $u_+^n = u^n \theta(u)$, where θ is the Heaviside function. The truncated power function is C^{n-1} continuous. For the intervals $(-\infty, u_0)$ and (u_n, ∞) it will be required that we get a polynomial of at most degree m. This is a generalization of what we did previously for natural cubic splines.

THEOREM

A natural spline has the representation

$$s(u) = \sum_{i=0}^{m} a_i u^i + \sum_{i=0}^{n} b_i (u - u_i)_+^{2m+1},$$

with $\sum_{i=0}^{n} b_i u_i^j = 0$ for $0 \le j \le m$. Here in the interval $(-\infty, u_0)$ we get a polynomial of degree m, to be denoted s_0 (note the difference in indexing compared to previous consideration). We have

$$s_0^{(j)}(u_0) = s_1^{(j)}(u_0) (0 \le j \le 2m).$$

We can write (Taylor's theorem)

$$s_1(u) = \sum_{j=0}^{2m+1} \frac{1}{j!} s_1^{(j)}(u_0)(x - x_0)^j =$$

$$\sum_{j=0}^{2m} \frac{1}{j!} s_1^{(j)}(u_0)(x - x_0)^j + b_0(u - u_0)^{2m+1} = s_0(u) + b_0(x - u_0)^{2m+1},$$

or

$$s(u) = s_0(u) + b_0(u - u_0)_+^{2m+1} (-\infty, u_1).$$

Note that s_0 is of degree $\leq m$, so $s^{(j)}(u_0) = 0$ for $m \leq j \leq 2m$. Continuing this procedure further, we get the representation in hand. The condition on b_i follows from the requirement that we get a polynomial of degree most of m in the interval (u_n, ∞).

Now, we seek a_j, b_j by requiring that

$$s(u_i) = \sum_{j=0}^{m} a_j u_i^j + \sum_{j=0}^{n} b_j (u_i - u_j)_+^{2m+1} = p_i (i = 0, \ldots, n),$$

and that

$$\sum_{i=0}^{n} b_i u_i^j = 0 (0 \leq j \leq m).$$

There are $m + n + 2$ equations and the same number of unknowns. It can be proven that the system is nonsingular.

9 Least square interpolation

Sometimes we may guess that a function measured in experiment should be smooth. Our measurements, however accurate, contain errors. By drawing an interpolating curve through experimental points we will introduce oscillations, keeping these errors.

In this case we might be willing to draw a smooth approximating curve that stays close to data. If errors are known, the curve should ideally stay within the error bars of the data points.

Examples:
The drag force is known to be proportional to wind speed squared in some range of wind velocities. Our measurements may deviate due to inaccuracy of our instruments and the fact that flow is turbulent and fluctuates.

The deformation of the spring is known to be a linear functions of the applied force. We measure force y_i needed to produce the displacement x_i and seek the best linear fit $y = ax + b$ that we can draw through the 'cloud' of points.

9.1 Approximation error and minimization

Let $y = ax + b$ be the line that we fit to the couples (x_i, y_i), $i = 0, \ldots, n$ of knots (x_i) and points (y_i). Because of errors, generally $y_i \neq ax_i + b$ and we have an error

$$e_i = |ax_i + b - y_i|$$

at each point. The total error can be estimated in different norms. For example, one can take an L_1 norm

$$E_1 = \sum_{i=0}^{n} e_i.$$

We are willing to use a differentiable function, so we select an L_2 norm

$$E_2 = \phi(a, b) = \sum_{i=0}^{n} (ax_i + b - y_i)^2.$$

Minimization of this error is called the least squares fit (approach, method). The values of a and b minimizing the error $\phi(a, b)$ can be found by setting the derivatives to zero: $\partial_a \phi(a, b) = 0$, $\partial_b \phi(a, b) = 0$.

$$\frac{\partial \phi(a,b)}{\partial a} = 2 \sum_{i=0}^{n} x_i (ax_i + b - y_i) = 0,$$

$$\Rightarrow a \sum_{i=0}^{n} x_i^2 + b \sum_{i=0}^{n} x_i = \sum_{i=0}^{n} x_i y_i;$$

$$\frac{\partial \phi(a,b)}{\partial b} = 2 \sum_{i=0}^{n} (ax_i + b - y_i) = 0,$$

$$\Rightarrow a \sum_{i=0}^{n} x_i + b(n+1) = \sum_{i=0}^{n} y_i;$$

We have two equations for two unknown. Solving the equation system we find

$$a = (1/d)[(n+1) \sum_{i=0}^{n} x_i y_i - (\sum_{i=0}^{n} x_i)(\sum_{i=0}^{n} y_i)],$$

$$a = (1/d)[(\sum_{i=0}^{n} x_i^2)(\sum_{i=0}^{n} y_i) - (\sum_{i=0}^{n} x_i)(\sum_{i=0}^{n} x_i y_i)],$$

where d is the determinant of system matrix $d = (n+1)(\sum_{i=0}^{n} x_i^2) - (\sum_{i=0}^{n} x_i)^2$.

- The approach is not limited to linear functions.
- We can assume that the function is given by an expansion in some basis, i. e.

$$y = \sum_{i=0}^{m} c_i g_i(x),$$

where $g_i(x)$ are some known functions.

- The error function is

$$\phi(c_0, \ldots, c_m) = \sum_{k=0}^{n} \left(\sum_{i=0}^{m} c_i g_i(x_k) - y_k \right)^2$$

where we assume that $n \geq m$ (Question: Why we need this?)

- Partial derivatives $\partial\phi/\partial c_j = 0$ lead to $m+1$ equations which can be solved to determine coefficients. These equations are of the form

$$\sum_{k=0}^{n} 2 \left(\sum_{i=0}^{m} c_i g_i(x_k) - y_k \right) g_j(x_k) = 0,$$

- They lead to the set of equations

$$\sum_{k=0}^{n} \left(\sum_{i=0}^{m} g_i(x_k) g_j(x_k) \right) c_i = \sum_{k=0}^{n} y_k g_j(x_k)$$

for $j = 0, \ldots, m$.

- They are called normal equations. They are linear in unknowns c_i. We have exactly $m+1$ equations for $m+1$ unknowns.

Example

- Let $y = a \ln x + b \cos x$
- The error function is $\phi(a, b) = \sum_{k=0}^{n} (a \ln x_k + b \cos x_k - y_k)^2$
- Equating to zero $\partial\phi(a, b)/\partial a = 0$ and $\partial\phi(a, b)/\partial b = 0$ we get the normal equations

$$\left(\sum_{k=0}^{n} (\ln x_k)^2 \right) a + \left(\sum_{k=0}^{n} \ln x_k \cos x_k \right) b = \sum_{k=0}^{n} y_k \ln x_k,$$

$$\left(\sum_{k=0}^{n} \ln x_k \cos x_k \right) a + \left(\sum_{k=0}^{n} (\cos x_k)^2 \right) b = \sum_{k=0}^{n} y_k \cos x_k,$$

This is a linear system with two unknowns a and b.

It can be noted that the system can be rewritten as

$$\begin{pmatrix} \ln x_0 & \ldots & \ln x_n \\ \cos x_0 & \ldots & \cos x_n \end{pmatrix} \begin{pmatrix} \ln x_0 & \cos x_0 \\ \vdots & \vdots \\ \ln x_n & \cos x_n \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \ln x_0 & \ldots & \ln x_n \\ \cos x_0 & \ldots & \cos x_n \end{pmatrix} \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix}$$

or

$$\mathrm{G}^T \mathrm{G} \begin{pmatrix} a \\ b \end{pmatrix} = \mathrm{G}^T \begin{pmatrix} y_0 \\ \vdots \\ y_n \end{pmatrix}$$

In more general case the normal equations can be rewritten as

$$\mathrm{G}^T\mathrm{G}\mathbf{c} = \mathrm{G}^T\mathbf{y},$$

where $\mathbf{c} = (c_0, \ldots, c_m)^T$ is the vector of unknowns and $\mathbf{y} = (y_0, \ldots, y_n)^T$ are the values to be fit. The matrix G in this case contains the values of basis functions evaluated at knots

$$\mathrm{G} = \begin{pmatrix} g_0(x_0) & \ldots & g_m(x_0) \\ \vdots & & \vdots \\ g_0(x_n) & \ldots & g_m(x_n) \end{pmatrix}$$

In some cases we know the functions and need to determine the unknown coefficients. However, frequently the functions are unknown. We take then any orthogonal basis, i. e. the basis satisfying the condition

$$< g_i g_j >= \sum_{k=0}^{n} g_i(x_k) g_j(x_k) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Because of orthogonality the answer for the coefficients becomes

$$c_i = \sum_{k=0}^{n} y_k g_i(x_k).$$

Orthogonal bases can be obtained using the Gram–Schmidt ortogonalization procedure. In practice one frequently uses polynomial bases (as the simplest possibility) or sine and cosine functions leading to the truncated Fourier series.

An example of polynomial basis:
Let us take monomials $g_0(x) = 1, g_1(x) = x$ and $g_2(x) = x^2$. Let our data be $y = (3, 1, -1, 1, 3)$ at $x = (-1, 0, 1, 2, 3)$. We write normal equations

$$\mathrm{G}^T\mathrm{G}\mathbf{c} = \mathrm{G}^T\mathbf{y},$$

with

$$\mathrm{G} = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 3 \\ 1 \\ -1 \\ 1 \\ 3 \end{pmatrix}$$

Performing computations we get

$$\begin{pmatrix} 5 & 5 & 15 \\ 5 & 15 & 35 \\ 15 & 35 & 99 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 7 \\ 33 \end{pmatrix}.$$

The solution can be found through Gaussian elimination $(c_0, c_1, c_2) = (19/35, -12/7, 6/7)$. Draw it.

We can compute the error $\phi(c_0, c_1, c_2)$ for the found values of c_i: It is $32/35$. There is no other quadratic function that fits the data better than the function found.

Note that since the function $y = c_0 + c_1 x + c_2 x^2$ provides the best fit to the data, we can also say that the values of c_i found provide the best approximate solution to the equation system

$$\begin{aligned} c_0 - 1c_1 + 1c_2 &= 3 \\ c_0 + 0c_1 + 0c_2 &= 1 \\ c_0 + 1c_1 + 1c_2 &= -1 \\ c_0 + 2c_1 + 4c_2 &= 1 \\ c_0 + 3c_1 + 9c_2 &= 3 \end{aligned}$$

This is an overdetermined system. It does not have a solution in a strict sense. But we were able to find its solution in a least square sense (but actually looking into another problem).

Return to systems of linear equations.
Consider

$$\sum_{i=0}^{m} a_{ki}x_i = b_k,$$

for $k = 0, \ldots, n$, $n > m$. We may use the previous short notation $\mathrm{A}\mathbf{x} = \mathbf{b}$. We call the difference $r_k = \sum_{i=0}^{m} a_{ki}x_i - b_k$ the kth residual. If $\mathbf{x}$ is the solution to the system, all r_k must vanish. If it is impossible, the system is overdetermined. In this case, to proceed, we minimize the error function

$$\phi(\mathbf{x}) = \sum_{k=0}^{n} \left(\sum_{i=0}^{m} a_{ki}x_i - b_k \right)^2 = \sum_{k=0}^{n} r_k^2.$$

This will lead to the normalized equations

$$\sum_{i=0}^{m} (\sum_{k=0}^{n} a_{kj}a_{ki})x_i = \sum_{k=0}^{n} a_{kj}b_k$$

for $j = 0, \ldots, m$, which is obviously

$$\mathrm{A}^T\mathrm{A}\mathbf{x} = \mathrm{A}^T\mathbf{b}.$$

The matrix $\mathrm{A}^T\mathrm{A}$ is symmetric and positive definite. It will have a full rank if the rank of A is m. The above example of quadratic fit can be reconsidered now as an example of solving the respective overdetermined system of linear equations.

9.2 Other polynomial basis functions and polynomial regression

One can use any convenient basis functions. A good basis is the basis with functions that differ a lot. Monomials are not the best choice.

Chebyshev polynomials over $[-1, 1]$:

$$\begin{aligned} T_0 &= 1 \\ T_1 &= x, \\ T_j(x) &= 2xT_{j-1}(x) - T_{j-2}(x),\ j \geq 2. \end{aligned}$$

$T_2(x) = 2x^2 - 1$, $T_3(x) = 4x^3 - 3x$, $T_4(x) = 8x^4 - 8x^2 + 1$.

The increase in the degree of polynomials allows for a better fit, but may emphasize oscillations. How to determine an optimal degree of polynomials if it is not known?

Polynomial regression
Let $P_N = \sum_{i=0}^{N} a_i x^i$ be a polynomial that represents data (measurements) without noise. The measured values y are contaminated with noise ε so that $y = p_N(x) + \varepsilon$.

The variance

$$\sigma_N^2 = \frac{1}{m-N} \sum_{i=0}^{m} (y_i - p_N(x_i))^2.$$

From statistics we know that $\sigma_0^2 > \sigma_1^2 > \ldots > \sigma_N^2 = \sigma_{N+1}^2 = \ldots \sigma_{m-1}^2$. The strategy is then to increase the degree of polynomials N untill we reach $\sigma_N^2 \approx \sigma_{N+1}^2$. The algorithm:
Given: $m + 1$ measurements
for $N = 0 : m$
Determine a polynomial fit using the least-squares method.
Compute σ_N^2. If is close to σ_{N-1}^2 return p_{N-1}.

Summary:
If measurements are subject to noise, approximation is preferrable. If the functional behavior of data is

known, we do a best fit to the known function, determining its optimal parameters. The least square metod will lead to a system of linear equations on the parameters whatever the function.

If function is unknown, some polynomial basis can be used. The order of polynomials can be determined by polynomial regression.

The least squares method can be also used to solve overdetermined systems of equations.

10 Difference schemes. Richardson interpolation

Derivatives of functions occur everywhere, and we have already used them, beginning from the first lecture. In numerical computations we are bound to approximate them. We will learn now how.

10.1 Forward, backward and central differencing

- From calculus we know that
$$f'(x) = \lim_{h\to\infty} \frac{f(x+h)-f(x)}{h}.$$
So the numerical approximation, called forward differencing, is
$$f'(x) \approx \frac{f(x+h)-f(x)}{h}.$$
How large is the error?

- Use Taylor's theorem to write (f is C^2 continuous)
$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(\xi), \quad \xi \in (x, x+h).$$
Thus,
$$f'(x) = \frac{f(x+h)-f(x)}{h} - \frac{1}{2}hf''(\xi).$$

- The last term is the error term, so the error is $\mathcal{O}(h)$. Can it be made smaller?

Central differencing

- Use two Taylor series (f is at least C^3 continuous)
$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + \ldots,$$
$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{3!}h^3 f'''(x) + \ldots,$$

- Subtract the second one from the first to get
$$f'(x) = \frac{f(x+h)-f(x-h)}{2h} - \frac{1}{6}h^2 f'''(x) + \ldots.$$

- We can use Taylor's theorem to see that the error is
$$\frac{1}{2h}(\frac{1}{6}h^3 f'''(\xi_1) + \frac{1}{6}h^3 f'''(\xi_2)), \quad \xi_1 \in (x, x+h), \quad \xi_2 \in (x-h, x).$$
The error is $\mathcal{O}(h^2)$.

Can we do even better? Can a general procedure be proposed that allows us to achieve any accuracy needed?

Remark. Instead of forward differencing one can use backward differencing
$$f'(x) \approx \frac{f(x)-f(x-h)}{h}.$$
Its error is $\mathcal{O}(h)$, same as for forward differencing.

10.2 Richardson Extrapolation

Doing central differencing we get

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + a_2 h^2 + a_4 h^4 + a_6 h^6 + \dots.$$

Denoting $\phi(h) = (f(x+h) - f(x-h))/(2h)$, we write

$$\phi(h) = f'(x) - a_2 h^2 - a_4 h^4 - a_6 h^6 - \dots$$

The idea is to halve the step h:

$$\phi(h/2) = f'(x) - a_2 h^2/4 - a_4 h^4/16 - a_6 h^6/64 - \dots$$

Then

$$\phi(h) - 4\phi(h/2) = -3f'(x) - \frac{3}{4} a_4 h^4 - \frac{15}{16} a_6 h^6 - \dots.$$

Or

$$f'(x) = -\frac{1}{3}\phi(h) + \frac{4}{3}\phi(h/2) - \frac{1}{4} a_4 h^4 - \frac{5}{16} a_6 h^6 - \dots.$$

The first two terms on the rhs give us a new estimate, the rest is the error. The new estimate is

$$f'(x) \approx \frac{f(x-h) - f(x+h) + 8f(x+h/2) - 8f(x-h/2)}{6h}.$$

Its error is $\mathcal{O}(h^4)$.

Remark: The procedure is known as the *Richardson extrapolation*. We can apply this process, halving the step each time, to obtain expessions with errors decaying as higher degrees of h as $h \to 0$. This is accompanied by the increase in the cost of computations.

Example

Compute the derivative of $f(x) = x^3$ at $x = 1$ with $h = 0.1$.

- Analytically $f'(x) = 3x^2$, $f'(1) = 3$.
- Forward differences $f'(1) = (1.331 - 1)/0.1 = 3.31$;
- Central differences $f'(1) = (1.331 - 0.729)/0.2 = 3.01$;
- The Richardson extrapolation $f'(1) = 3$. (Why the answer is exact?)

10.3 Higher-order derivatives

We can apply similar techniques to compute higher-order derivatives. Taylor series can be used to find approximations, and the Richardson extrapolation can be used to improve the error terms.

Example

Write two expansions as previously

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + \dots,$$

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{3!}h^3 f'''(x) + \dots,$$

but now add them to get

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{2}{4!}h^4 f^{(4)}(x) + \dots,$$

so that the estimate for the second derivative becomes

$$f''(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2}.$$

The error term for $f \in C^4$ is

$$\frac{h^2}{24}(f^{(4)}(\xi_1) + f^{(4)}(\xi_2)), \quad \xi_1 \in (x, x+h), \quad \xi_2 \in (x-h, x).$$

Summary:

- Derivatives of a function f at a point x can be computed by estimating the values of function at locations close to x.
- Taylor series can be used to find estimates and their error terms.
- Central differencing is more accurate than forward or backward differencing.
- Richardson extrapolation can be used to reduce errors.

What will happen if we know our function $f(x)$ at a set of discrete knots x_i that are not equidistant? How to compute derivatives? An idea is to use polynomial interpolation. For example, Lagrange polynomials can be constructed and then analytically differentiated to get an answer.

For example, if we use x_i and its nearest neighbors x_{i-1} and x_{i+1}, we can interpolate them with a quadratic polynomial, getting a linear representation for the derivative in the vicinity of x_i. The error will be $\mathcal{O}(h^2)$ for h sufficiently small.

11 Quadrature rules

The fundamental theorem of calculus:
If $f(x) \in C^0[a,b]$ and F is the antiderivative of f, then

$$\int_a^b f(x)dx = F(b) - F(a).$$

Example

- For $f(x) = x^2$ the antiderivative $F(x) = x^3/3 + C$, $C \in \mathbb{R}$.

$$\int_a^b f(x)dx = b^3/3 - a^3/3,$$

  and we can do computations analytically.

- For $f(x) = e^{x^2}$ the antiderivative is formally written as $F(x) = \int_0^x e^{t^2} dt$, but it does not help us to compute $\int_a^b e^{x^2} dx$.

Analytical computations of integrals is not always possible, one has to resort to numerical solutions. The question is how to make them accurate.

Let $f(x) \geq 0$ for $x \in [a,b]$.
In this case $\int_a^b f(x)dx$ is the area under $f(x)$.
If $f(x)$ takes negative values and is bounded, we may consider

$$\int_a^b (f(x) - \inf_{x\in[a,b]} f(x))dx + (b-a) \inf_{x\in[a,b]} f(x).$$

This means that we can deal with non-negative functions, the result will be generalizible to other functions.
We would like to develop methods to estimate the area under $f(x)$.

Let P be a partition of the interval $[a,b]$

$$P : (a = x_0 < x_1 < \ldots < x_n = b).$$

The partition divides the interval $[a,b]$ into n subintervals $[x_i, x_{i+1}]$.

- Let m_i be the *greatest lower bound* of $f(x)$ on the ith subinterval,

$$m_i = \inf\{f(x),\, x_i \leq x \leq x_{i+1}\}.$$

- Let M_i be the *least upper bound* of $f(x)$ on the ith subinterval,

$$M_i = \sup\{f(x),\, x_i \leq x \leq x_{i+1}\}.$$

The lower sum with respect to partition P is given by

$$L(f;P) = \sum_{i=0}^{n-1} m_i(x_{i+1} - x_i).$$

The upper sum with respect to partition P is

$$U(f;P) = \sum_{i=0}^{n-1} M_i(x_{i+1} - x_i).$$

The lower and upper sums can be used as estimates for the definite integral. Furthermore,

$$L(f;P) \le \int_a^b f(x)dx \le U(f;P).$$

This is true for any partition P.

THEOREM

For all paartitions P we consider the least upper bound and the greatest lower bound. $f(x)$ is Riemann intergable *iff*

$$\sup_P L(f;P) = \inf_P U(f;P).$$

Example

Consider the Dirichlet function

$$d(x) = \begin{cases} 0, \, x \in \mathbb{Q} \\ 1, \, x \notin \mathbb{Q} \end{cases}$$

For any partition P, we find $L(d;P) = 0$ and $U(d;P) = b - a$. Hence

$$0 = \sup_P L(d:P) \neq \inf_P U(d;P) = b - a.$$

This function is not Riemanm-integrable (it has too many discontinuities).

Consider $f(x) = e^{-x^2}$. This function is continuous.

THEOREM A continuous function is Riemann-integrable.

- We are willing to compute $\int_0^1 e^{-x^2}\, dx$.
- Use equidistant partitioning $x_i = ih$, $h = 1/n$, $i = 0, \ldots, n$.
- The lower sum
$$L(f;P) = \sum_{i=0}^{n-1} hf(x_{i+1}) = \sum_{i=0}^{n-1} he^{-x_{i+1}^2}.$$
- The upper sum
$$U(f;P) = \sum_{i=0}^{n-1} hf(x_i) = \sum_{i=0}^{n-1} he^{-x_i^2}.$$
- Error analysis
$$U(f,P) - L(f;P) = -hf(x_n) + hf(x_0) = h(1 - 1/e).$$

The error we obtain using the lower or the upper estimate for the definite integral is $\mathcal{O}(h)$.

Can we do better?

The idea is to estimate the area as a sum of trapezoids instead of rectangles. Let P be a partitioning of $[a, b]$. For each subinterval we write

$$\int_{x_i}^{x_{i+1}} f(x)\, dx \approx (x_{i+1} - x_i)\frac{f(x_i) + f(x_{i+1})}{2}.$$

The first multiplier on the rhs is the base of the trapezoid, and the second one is its averaged height. This gives the trapezoidal rule

$$\int_a^b f(x)\, dx \approx T(f;P) = (1/2)\sum_{i=0}^{n-1}(x_{i+1} - x_i)(f(x_i) + f(x_{i+1})).$$

We can reduce the amount of computations if the partitioning P is equidistant:

$$x = a + ih,\ h = (b-a)/n,\ i = 0, \ldots, n.$$

Then

$$T(f,P) = (h/2)\sum_{i=0}^{n-1}(f(x_i) + f(x_{i+1})) = h\sum_{i=1}^{n-1} f(x_i) + (h/2)(f(x_0) + f(x_n)).$$

Example:
Compute

$$\int_0^1 \frac{\sin x}{x}\, dx$$

using an equidistant partitioning with $n = 5$.
We compute the function values:

x_i	0	0.2	0.4	0.6	0.8	1
$f(x_i)$	1	0.99335	0.97355	0.94107	0.89670	084147

Performing operations, we find $T(f;P) = 0.94508$ to be compared with the accurate value of 0.9460830704...
Our error is relatively small, considering that we took only $n = 5$ intervals.

Obviously when partitioning is fine enough and f is continuous

$$T(f;P) = (1/2)(L(f;P) + U(f;P)),$$

so that the trapezoid rule produces the result between the lower and upper sum. What is about the error?

THEOREM OF PRECISION

- Let $f(x) \in C^2[a,b]$ and P be an equidistant partition of $[a,b]$.
- Then the error is given by

$$\left|\int_a^b f(x)\, dx - T(f;P)\right| \le \left|\frac{1}{12}(b-a)h^2 f''(\xi)\right|$$

  with $\xi \in a, b$.

Indeed, on each subinterval using the trapezoid rule implies that we use linear interpolation. For a linear interpolating polynomial $p(x)$ we have

$$f(x) - p(x) = \frac{1}{2}f''(\xi_i)(x - x_i)(x - x_{i+1}),$$

with $\xi_i \in (x_i, x_{i+1})$. Yet ξ_i here depends on x, so we take a value which corresponds to the maximum of $f''(\xi_i(x))$ on $x \in [x_i, x_{i-1}]$ and write $|f(x) - p(x)| \le \frac{1}{2}|f''(\xi_i)|(x - x_i)(x - x_{i+1})$. Integrating this over the subinterval we have the theorem claim for each subinterval. Then using the continuity of second derivative we can prove that there exist a ξ such that $\sum_{i=0}^{n-1} hf''(\xi_i) = (b-a)f''(\xi)$, which completes the proof.

How many subintervals n do we need to approximate

$$\int_0^1 \frac{\sin x}{x}\, dx$$

with an error that is $\le 5 \cdot 10^{-6}$?

Estimate the second derivative:

$$\frac{\sin x}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \ldots$$

$$\Rightarrow \left(\frac{\sin x}{x}\right)'' = -\frac{2}{3!} + \frac{3\cdot 4\cdot x^2}{5!} - \frac{5\cdot 6\cdot x^4}{7!} + \dots$$

Hence, for $\xi \in (0,1)$, the absolute value of derivative is less than

$$|f(\xi)''| < \frac{2}{3!} + \frac{3\cdot 4}{5!} + \frac{5\cdot 6}{7!} + \ldots < \frac{1}{2}$$

Using the theorem of precision we conclude that the error is

$$|f''(\xi)(b-a)h^2/12| < h^2/24.$$

Setting $h^2/24 < 5\cdot 10^{-6}$ we estimate $h \le \sqrt{1.2}\cdot 10^{-2}$. Hence we need $n = (b-a)/h \approx 91.3$.

Recursive trapezoid rule. Let us use an equidistant partitioning, in which case

$$T(f,P) = h\sum_{i=1}^{n-1} f(a+ih) + (h/2)(f(a)+f(b))$$

with $h = (b-a)/n$. Let take $n = 2^m$.

$$T_m(f,P) = h\sum_{i=1}^{2^m-1} f(a+ih) + (h/2)(f(a)+f(b)) = h\sum_{i=0}^{2^m-1} f(a+ih) + C.$$

$$T_{m-1}(f,P) = 2h\sum_{i=1}^{2^{m-1}-1} f(a+2ih) + 2C.$$

Take

$$T_m(f;P) - (1/2)T_{m-1}(f;P) = h\sum_{i=1}^{2^{m-1}} f(a+(2i-1)h)$$

To move from $T_{m-1}(f;P)$ to $T_m(f;P)$ we need not do all function estimates, part can be reused.

We iteratively compute $T_m(f;P)$ for $m = 0,1,2,\ldots$:

- $$T_0(f;P) = (1/2)(b-a)(f(a)+f(b))$$
- $$T_m(f;P) = (1/2)T_{m-1}(f;P) + h\sum_{i=1}^{2^{m-1}} f(a+(2i-1)h)$$

  for $m \ge 1$ with $h = (b-a)/2^m$.
- Stop the recursion if we see that the estimate does not improve significantly.

Significant improvement can be based on the idea similar to that of Richarson extrapolation. This leads to the Romberg algorithm we return to in part II.

11.1 Gaussian Quadrature

For a partitioning P most of numerical integration schemes provide the rule

$$\int_a^b f(x)\,dx = A_0 f(x_0) + A_1 f(x_1) + \ldots + A_n f(x_n),$$

$a \le x_0 < x_1 < \ldots < x_n \le b$. We already know that weights A_i are important (all the difference between the upper, lower and trapezoid sums were due to the weights). How can we find the best weights if the knots x_i are given?

- Approximate $f(x)$ by a polynomial $p(x)$ of degree n that interpolates points $f(x_0), \dots, f(x_n)$ at knots $x_0, \dots, x_n$.
- We can use the Lagrangian interpolation, in which case

$$p(x) = \sum_{i=0}^{n} f(x_i) \cdot L_i(x), \quad L_i(x) \prod_{j=0, i \neq j}^{n} \frac{x - x_j}{x_i - x_j}.$$

- If situations $p(x) \approx f(x)$ on $x \in [a, b]$, we write

$$\int_a^b f(x)\, dx \approx \int_a^b p(x)\, dx = \sum_{i=0}^{n} f(x_i) \int_a^b L_i(x) dx = \sum_{i=0}^{n} A_i f(x_i).$$

The weights A_i only depend on the knot sequence, but do not depend on the function $f(x)$ we are willing to interpolate! Hence if the knot sequence is given, the weights A_i can be precomputed and stored, making computations elementary. Note that this presents a generalization of the trapezoid rule where linear polynomials were used on each subinterval.

Example:

Let $[a, b] = [-2, 2]$ and $(x_0, x_1, x_2) = (-1, 0, 1)$. (Note that the ends of the interval are not in the knot sequence.) The polynomials:

$$L_0 = \left(\frac{x - 0}{-1 - 0}\right)\left(\frac{x - 1}{-1 - 1}\right) = \frac{1}{2} x(x - 1);$$

$$L_1 = -(x - 1)(x + 1); \quad L_2 = \frac{1}{2} x(x + 1).$$

Performing integration,

$$A_0 = \int_{-2}^{2} L_0(x)\, dx = 8/3; \quad A_1 = \int_{-2}^{2} L_1(x)\, dx = -4/3; \quad A_2 = \int_{-2}^{2} L_2(x)\, dx = 8/3.$$

Thus for any function on the interval $[-2, 2]$ we obtain the estimate

$$\int_{-2}^{2} f(x)\, dx = \frac{8}{3} f(-1) - \frac{4}{3} f(0) + \frac{8}{3} f(1).$$

Using this result, we estimate

$$f(x) = 1 \Rightarrow \int_{-2}^{2} f(x)\, dx = \frac{8}{3} - \frac{4}{3} + \frac{8}{3} = 4;$$

$$f(x) = x \Rightarrow \int_{-2}^{2} f(x)\, dx = -\frac{8}{3} - 0 + \frac{8}{3} = 0;$$

If $f(x)$ is a polynomial of degree $\leq n$, we have $f(x) = p(x)$ and the interpolation is exact if we are using $n + 1$ knots. What will happen if $f(x)$ is a polynomial of degree $> n$?

The Simpson scheme:

The trapezoid rule was obtained by using linear interpolation on subintervals. If one uses quadratic interpolation, one gets the Simpson scheme:

$$\int_{x_i}^{x_{i+1}} f(x)\, dx = \frac{1}{6}(x_{i+1} - x_i)(f(x_i) + 4f(x_i/2 + x_{i+1}/2) + f(x_{i+1})).$$

If we compare it to the combination of trapezoid rules for two subintervals, we see that there is larger weight with the central value for the Simpson scheme. If the Simpson scheme is applied to the entire interval $[a, b]$, we can group the terms contributing to neighboring subintervals, with the final result

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left\{ f(a) + f(b) + 4\sum_{i=1}^{n/2} f(a+(2i-1)h) + 2\sum_{i=1}^{n/2-1} f(a+2ih) \right\} - \frac{b-a}{180}h^4 f^{(4)}(\xi).$$

We see that the accuracy is $\mathcal{O}(h^4)$. Here $h = (b-a)/n$. Recursive procedure and the procedure that reminds Richardson extrapolation are possible here too.

Thus far we have assumed that equidistant knots $x_0, \ldots, x_n$ are given. However, other than simplicity of resulting expressions, there is no reason why we need to select equidistant knots.

Gauss (1777-1855) found that one may essentially improve accuracy of integration if the knots are selected in a special way. It is the subject of Gauss quadrature theorem. Quadrature is the rule used to approximate a definite interval. The rules we used thus far can be also referred to as quadrature. However, most frequently we use the term to stress that we deal with specially arranged knots.

Gaussian quadrature theorem

- Let $q(x)$ be a polynomial of degree $n+1$ such that

  $$\int_a^b q(x)x^k dx = 0$$

  for $k = 0, \ldots, n$
- Let $x_0, \ldots, x_n$ be the roots of polynomial q.
- Then

  $$\int_a^b f(x)dx = \sum_{i=0}^{n} A_i f(x_i), \quad A_i = \int_a^b L_i(x)dx$$

  for all polynomials $f(x)$ of degree $\leq 2n+1$.

Proof:
Let $f(x)$ be a polynomial of degree $\leq 2n+1$. In this case we can write

$$f(x) = s(x)\cdot q(x) + r(x)$$

where $s(x)$ and $r(x)$ are polynomials of degree $\leq n$.

$$\int_a^b f(x)dx = \int_a^b s(x)q(x)dx + \int_a^b r(x)dx = 0 + \int_a^b r(x)dx.$$

The second equality on the rhs is by virtue of theorem condition. Thus

$$\int_a^b f(x)dx = \int_a^b r(x)dx = \sum_{i=0}^{n} A_i r(x_i) = \sum_{i=0}^{n} A_i (f(x_i) - s(x_i)q(x_i)) = \sum_{i=0}^{n} A_i f(x_i).$$

The last equality is because x_i is the root of $q(x)$.

Note that the polynomial q is not defined uniquely because it has $n+2$ coefficients, and we have $n+1$ equations.

- Following the Gaussian quadrature theorem we fix the $n+1$ knots $x_0, \ldots, x_n$. They are referred as Gaussian nodes (Gaussian quadrature points, etc.)
- The knots uniquely define the polynomial $p(x)$ that is assumed to approximate $f(x)$. We use the Lagrangian polynomials to determine the weights $A_0, \ldots, A_n$. We thus define $2(n+1)$ variables for $n+1$ knot.

- The procedure is known as *Gaussian quadrature*.

Both the knots and A_i are independent of function $f(x)$ we are going to interpolate. They can be precomputed given the degree n of the interpolating polynomial and the interval $[a, b]$.

Example:
Use Gaussian quadrature to determine the value of

$$\int_{-1}^{1} f(x)dx$$

using three knots. Hence $n + 1 = 3$.
The degree of $q(x)$ is 3, and the degree of $p(x)$ is 2. Hence we write

$$q(x) = c_0 + c_1x + c_2x^2 + c_3x^3.$$

We have three conditions:

$$\int_{-1}^{1} q(x)dx = 0, \quad \int_{-1}^{1} xq(x)dx = 0, \quad \int_{-1}^{1} x^2q(x)dx = 0.$$

If we select $c_0 = 0$ and $c_2 = 0$, we have only one constraint

$$\int_{-1}^{1} xq(x)dx = 0 \Rightarrow \frac{2}{3}c_1 + \frac{2}{5}c_3 = 0,$$

giving a polynomial

$$q(x) = -3x + 5x^3.$$

Its roots are 0 and $\pm\sqrt{3/5}$. These are the knots we were looking for. Our quadrature rule becomes

$$\int_{a}^{b} f(x)dx = A_0f(-\sqrt{3/5}) + A_1f(0) + A_2f(\sqrt{3/5}).$$

At this stage we could write Lagrangian polynomials for x_0, x_1, x_2 and do calculations to determine A_i. A simple approach is obtained is we recall that the quadrature should be exact for at least any polynomial $f(x)$ of degree ≤ 2. We take therefore $f(x) = 1$, $f(x) = x$ and $f(x) = x^2$.

$$\begin{aligned} 2 &= \int_{-1}^{1} 1 \cdot dx = A_0 + A_1 + A_2; \\ 0 &= \int_{-1}^{1} dx = -\sqrt{3/5}A_0 + \sqrt{3/5}A_2; \\ 2/3 &= \int_{-1}^{1} x^2dx = (3/5)A_0 + (3/5)A_2. \end{aligned}$$

Solving it gives us $A_0 = A_2 = 5/9$ and $A_1 = 8/9$. As the result, the quadrature rule becomes

$$\int_{a}^{b} f(x)dx = \frac{5}{9}f(-\sqrt{3/5}) + \frac{8}{9}f(0) + \frac{5}{9}f(\sqrt{3/5}).$$

This estimate is exact for all polynomials of degree ≤ 5. We can test it taking, for example, $f(x) = x^4$:

$$\int_{-1}^{1} x^4dx = 2/5 = \frac{5}{9}\left(\frac{3}{5}\right) + 0 + \frac{5}{9}\left(\frac{3}{5}\right).$$

The interval $[a, b]$ can be transformed to $[-1, 1]$ through the coordinate transform

$$t = \frac{2x - (a + b)}{b - a},$$

which gives

$$x = (a+b)/2 + (b-a)t/2.$$

For $t \in [-1,1]$ $x \in [a,b]$. For this reason it suffice to consider the interval $[-1,1]$. The polynomials q, normalized by the condition $q(1) = 1$ are known as the *Legendre polynomials*. They are

$$\begin{aligned} q_0 &= 1; \\ q_1 &= x; \\ q_2 &= (3/2)x^2 - 1/2; \\ q(3) &= (5/2)x^3 - 3x/2; \\ q_n(x) &= \frac{2n-1}{n} x q_{n-1}(x) - \frac{n-1}{n} q_{n-2}(x). \end{aligned}$$