Notes for “Numeriske Metoder”

Olivier Verdier

2012-02-29

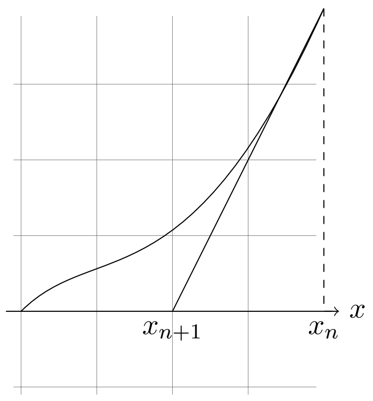
1 Cautionary Introduction

This is the set of notes for the course Numeriske Metoder. It is by no means a set of lecture notes in the standard sense, but rather the notes as a student could have taken them as I gave the course.

Week 1

2 Newton’s Method

Newton’s method may be represented by a graph.



One obtain from the graph the formula

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

Let us examine how to use this formula to compute $\sqrt{2}$.

By definition, $\sqrt{2}$ is a root of the function $f(x) := x^2 - 2$. For this function we obtain the Newton algorithm as

$$x_{n+1} = x_n - \frac{x^2 - 2}{2x}.$$

To test that method with a computer we need some basic notions of programming.

2.1 Programming Notions

2.1.1 Variables

The first essential notion is that of ***variable***. A variable may be used to store some *values*. Here are some examples:

```
a = 2.
b = 4.
c = 2*b
```

2.1.2 Assignment vs Equality

Note that the equal sign is *not at all* the same as in mathematics! It represents *assignment* instead of equality.

For instance, suppose that the variable x is assigned and contains the value 2. The following *equation* in mathematics

$$x = 2x$$

has the solution $x = 0$, but the *assignment*

```
x = 2*x
```

puts the value 4 in the variable `x`.

2.2 Computing $\sqrt{2}$

In order to compute the outcome of Newton's method, we may use different variables for each steps

```
x_0 = 1.
x_1 = x_0 - (x_0**2 - 2)/(2*x_0)
x_2 = x_1 - (x_1**2 - 2)/(2*x_1)
x_3 = x_2 - (x_2**2 - 2)/(2*x_2)
```

But this is very cumbersome! A simpler method is to reassign the same variable `x` with the new result at each step:

```
x = x - (x**2 - 2)/(2*x)
```

This way we may simply call the instruction above several times and obtain the result.

Some questions arise from this experiment:

- We chose a starting point $x_0 = 10$ and $x_0 = 10^{-7}$. How does the result depend on the starting point?

- Is it guaranteed in general that the algorithm will converge?
- If the algorithm converges, does it converge to the root we are looking for?
- How efficient is it? How much computational effort is required?

2.3 Newton's method as a fixed point interation

In order to answer all these questions, we notice that Newton's method may be written in the form

$$x_{n+1} = G(x_n), \tag{1}$$

where

$$G(x) := x - \frac{f(x)}{f'(x)}.$$

This suggest to study all the algorithms of this form. It turns out that such a study is possible.

3 Fixed Point Iterations

Algorithms of the form (1) are called ***fixed point iterations***. The reason is that if G is continuous, and if x_n converges to a value x_∞, then we have

$$G(x_\infty) = x_\infty.$$

A point x such that $G(x) = x$ is called a ***fixed point*** of G.

3.1 Graphical Interpretation of Fixed Point Iterations

Graphical interpretation, and Mathematica experiment with the logistic function.

3.2 Analysis of the Fixed Point Method

The intuition is as follows. Suppose that the function G happens to be affine:

$$G(x) = ax + b.$$

Assuming that $a \neq 1$, the fixed point $\overline{x}$ is

$$\overline{x} := \frac{b}{a-1}.$$

The *error*

$$e_n := x_n - \overline{x}$$

is such that

$$e_{n+1} = x_{n+1} - \overline{x} = a(x_n - \overline{x}) = ae_n.$$

So we conclude that the fixed point iteration converges if and only if $|a| \leq 1$.

3.3 Proof

Suppose that $\overline{x}$ is a fixed point of G and that G is differentiable at that point. Assume further that

$$|G'(\overline{x})| < 1.$$

We thus have

$$\lim_{y \to x} \frac{G(y) - G(\overline{x})}{y - \overline{x}} = G'(\overline{x}).$$

So for any η there is an ε such that for $|y - \overline{x}| \leq \varepsilon$:

$$\left| \frac{G(y) - G(\overline{x})}{y - \overline{x}} \right| \leq |G'(\overline{x})| + \eta.$$

For the error analysis we obtain

$$e_{n+1} = x_{n+1} - \overline{x} = G(x_n) - G(\overline{x}) = \frac{G(x_n) - G(\overline{x})}{x_n - \overline{x}} e_n.$$

Assume now that $|e_n| \leq \varepsilon$, and that η is chosen small enough, so that $|G'(\overline{x})| + \eta < 1$. Then we have

$$|e_{n+1}| \leq (|G'(\overline{x}) + \eta)|e_n|.$$

In particular we obtain that $|e_{n+1}| \leq \varepsilon$. If this is fulfilled in the first stage, i.e. if

$$|e_0| \leq \varepsilon, \tag{2}$$

then we have shown that for all subsequent step n, we have $e_n \leq \varepsilon$. But we have more, in fact, we have

$$e_n \leq (|G'(\overline{x})| + \eta)^n e_0.$$

We conclude that if $|G(\overline{x})| < 1$, and if $|e_0| \leq \varepsilon$, then the fixed point iteration converges.

Moreover, we see that the bigger the value $|G'(\overline{x})|$, the fastest the fixed point iteration converges.

Remember that we assumed that $|e_0|$ was sufficiently small. This is a *crucial* assumption. It means that the "initial guess" x_0 must be close enough to the actual solution $\overline{x}$.

3.4 Newton's Iteration as a Fixed Point Method

What about Newton's iteration? Newton's iteration is just a special case of a fixed point iteration for a given function G, so which conditions are fulfilled in that case?

For the Newton iteration, we have

$$G(x) = x - \frac{f(x)}{f'(x)}.$$

We first notice that

$$G(\overline{x}) = \overline{x} \iff f(\overline{x}) = 0,$$

that is, the roots of f are the fixed points of G, assuming that $f'(\overline{x}) \neq 0$. So, if the Newton algorithm converges, it must converge to a root of f.

Now the derivative of G is

$$G'(x) = \frac{f''(x)f(x)}{(f'(x))^2}$$

so we have

$$G'(\overline{x}) = 0.$$

Newton's algorithm thus corresponds to the *best* possible case of fixed point iteration.

Week 2

3.5 Checking that an algorithm works

Three steps

1. The algorithm should produce a value at each step
2. Provided that the previous step is fulfilled, the sequence of the values thus produced should *converge* to some value x_∞
3. Assuming the previous steps, the value x_∞ should be a solution of the problem

For instance, for the bisection algorithm, the first item is fulfilled as long as $f(a)f(b) \leq 0$, the second item is always true, and the third is true if f is continuous.

3.6 Improvements of Newton's method

In general, the derivative of the function f one is looking a root of, is *not* available! One generally replaces that derivative with an approximate value:

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

for a sufficiently small value of h.

The other improvement is not to recompute that derivative at each steps, and to use the same derivative for many(or even all) of the steps.

The last improvement is the *trust region*: you don't trust Newton's algorithm if it sends you too far away

To sum up:

1. Approximate derivatives are used instead of symbolic ones
2. *Simplified Newton*: The same value of the exact or approximate derivative is used for several, or all, the steps
3. *Trust Region*: the value $|x_{n+1} - x_n|$ is required to be bounded

3.7 Newton in several dimensions

Suppose that the problem at hand is instead

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

How can Newton be used for that?

Let us rewrite Newton in one dimension first:

$$f'(x_n)(x_{n+1} - x_n) = -f(x_n)$$

Now if we define

$$F(X) = (f(x,y), g(x,y)) \qquad \text{with } X = (x,y)$$

we may write Newton's algorithm as

$$F'(X_n)(X_{n+1} - x_n) = -F(X_n)$$

Notice how this is now a *linear* problem. We have thus transformed a difficult nonlinear problem into a series of linear problems.

3.8 Newton 2D Example

The problem is to find the points (x,y) on the plane, such that

$$y = x^3 \qquad \text{and } \|(x,y)\| = 1$$

We may reformulate this problem as:

Solve the following equation system

$$\begin{aligned} y &= x^3 \\ x^2 + y^2 &= 1 \end{aligned}$$

We define the function F by

$$F(x,y) := (y - x^3, x^2 + y^2 - 1)$$

The Jacobian is

$$F'(x,y) = \begin{bmatrix} -3x^2 & 1 \\ 2x & 2y \end{bmatrix}$$

We may thus formulate Newton's method as finding a solution in the unknowns $\Delta x, \Delta y$ of the following linear equation:

$$\begin{aligned} -3x_n^2 \Delta x + \Delta y &= x_n^3 - y_n \\ 2x_n \Delta x + 2y_n \Delta y &= -x_n^2 - y_n^2 + 1 \end{aligned}$$

and then compute the next values of x, y by

$$\begin{aligned} x_{n+1} &= x_n + \Delta x \\ y_{n+1} &= y_n + \Delta y \end{aligned}$$

3.9 Newton Fractals

They are obtained by looking at problems as

$$P(z) = 0$$

where P is a complex polynomial. For instance

$$P(z) = (z-1)(z+i)$$

The focus is not on finding the roots but on *which initial condition leads to which root*

Newton's algorithm is obtained by observing that P as a function of a complex variable may be regarded as a function from $\mathbf{R}^2$ to $\mathbf{R}^2$ instead.

3.10 Loops in Programming

It is very innefficient to repeat a line of code manually. Instead, one may ask a computer to repeat a given line of code. To compute square root of two, one would use

```
for i in range(10):
  x = x - (x**2 - 2)/(2*x)
```

For the moment, use a fixed, big, number of iterations. Do not forget to initialise the value of `x` before you run the loop.

3.11 Practical Newton in 2D

Just like in **??**, one can use Newton's algorithm in several dimensions even with functions for which we do not know the exact Jacobian. The partial derivative $\frac{\partial f}{\partial x}$ would be approximated by

$$\frac{\partial f}{\partial x}(x_n, y_n) \approx \frac{f(x_n + h, y_n) - f(x_n, y_n)}{h}$$

Note that we now need three evaluations of F per iterations.

In N dimensions, we would need $N+1$ evaluations of F per iterations, without mentioning that the "cost" of computing F will probably be at least proportional to N. This is our first encounter with the "curse of dimensionality".

4 The Floating Point System

4.1 Example with three decimal digits

Let us assume that we can encode real numbers only with three decimal digits, i.e., three numbers d_0, d_1, d_2 going from 0 to 9. We would by convention interpret $d_0 d_1 d_2$ as

$$\left(d_0 + \frac{d_1}{10}\right) \times 10^{d_2}.$$

We denote the three decimal digits $d_0 d_1 d_3$ by $\mathtt{d}_0\,\mathtt{d}_1|\mathtt{d}_2$.

Let us work out some examples.

$$\texttt{1 0|0} \to 1.0 \times 10^0 = 1.0$$
$$\texttt{1 1|0} \to 1.1 \times 10^0 = 1.1$$
$$\texttt{0 1|2} \to 0.1 \times 10^2 = 10$$

We first notice that two numbers may be represented in the same way! For instance both `0 1|1` and `1 0|0` correspond to the real number 1.0. We henceforth *forbid* the digit zero in the first slot, i.e., we assume that

$$d_0 \neq 0.$$

Notice how, by doing that, *we cannot represent the zero anymore!*

Next, we would like to encode also number smaller than one. We do that by *interpreting* the third digit differently. This amounts to allow the first digit to span the integers -5 to $+4$.

So we have for instance

$$\texttt{2 3|-2} \to 2.3 \times 10^{-2} = 0.023$$
$$\texttt{6 4|3} \to 6.4 \times 10^3 = 6400$$

Subnormal numbers:

In order to represent the zero, we *sacrifice* the exponent -5, and assume that for that value, we interpret it as $\frac{d_1}{10} \times 10^{-5}$ instead. So we have for instance

$$\texttt{2 3|-5} \to 0.3 \times 10^{-5} = 0.000003$$

Notice how we *disregard the first digit altogether*, when the exponent takes the special value of -5. Now we may represent the zero as

$$\texttt{1 0|-5} \to 0 \times 10^{-5} = 0$$

Note that now several triples of digits are associated to the same number, but we will see that this does not happen in base two!

Week 3

4.2 Linear Fixed Point Iterations

Consider the fixed point iteration

$$x_{n+1} = Ax_n + b$$

The question we ask ourselves are

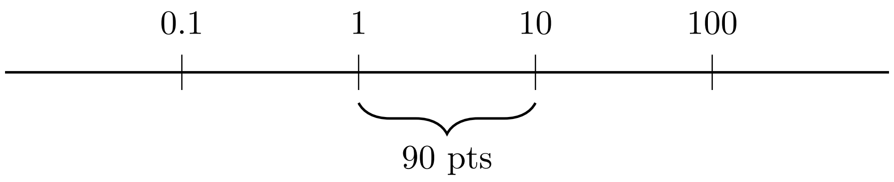


Figure 1: The floating point system in a nutshell; note that his is a *toy example* to understand the true floating point system; between every ticks, there are 90 *equidistributed* points; note that there are as many points between 1 and 10, as there are between 10 and 100; notice also the "discontinuity" in density: for instance at 10, the number below is 9.9, but the next number is 11; finally, notice that the *machine epsilon*, i.e., the distance from one to the smallest representable number greater than one, is equal to $(10-1)/90 = 0.1$.

- Does a fixed point exist?
- If it exists, does the iteration converge?
- Does convergence depend on the starting value?
- Does it depend on b?

The first calculation to do is a general one, supposing that a fixed point is $\overline{x}$

$$\begin{aligned} e_{n+1} &= x_{n+1} - \overline{x} = (Ax_n + b) - (A\overline{x} + b) \\ &= A(x_n - \overline{x}) = Ae_n \end{aligned}$$

So we already see that convergence does not depend on b!

4.2.1 Showing that $|\lambda| < 1$ is in fact necessary

4.3 Repartition of Floating Points Numbers

In the example with three decimal digits, we had the following distribution of points:

4.4 Loss of significant digits

When doing a computation such as

```
x = sqrt(2)
y = (1e6 + x) - 1e6
```

How many digits are lost, between `x` and `y`?

5 Polynomial Interpolation

5.1 Problem Statement

Given ***interpolation points*** $x_0, \dots, x_n$, and interpolation values $y_0, \dots, y_n$, the ***interpolation problem*** is to find a polynomial P such that

$$P(x_k) = y_k \qquad k = 1, \dots, n.$$

5.2 Linearity

We first show that this problem is of a linear nature, i.e., that it can be brought down to a problem of the form

$$Ax = b$$

where A is a known matrix, b is a known vector, and x somehow encodes the polynomials.

We do that by writing the polynomial in the standard Taylor basis

$$P(x) = a_0 + a_1 x + \cdots + a_n x^n.$$

Now the interpolation conditions are

$$a_0 + a_1 x_k + a_2 x_k^2 + \cdots + a_n x_k^n = y_k \qquad k = 1, \dots, n.$$

As a result, we may write the problem using a ***van der Monde matrix*** V as

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & & \vdots & \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}$$

and the interpolation problem may be formulated as finding the coefficients $a_0, a_1, \dots, a_n$ such that

$$V \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

Note that

- The matrix V depends only on the points x_k
- the right hand side depends only on the interpolation values y_k

5.3 Lagrange Polynomials

We could try and show that the matrix V is invertible in some circumstances, but it turns out to be simpler to show directly the existence of an interpolation polynomial for any given data $y_0, \dots, y_n$.

Since it is a linear problem, we may start up with looking at the particular interpolation problem when $y_0 = 1$ and $y_k = 0$ for $k = 1, \dots, n$.

We see that the following polynomial *almost* fulfills the condition:

$$\overline{P}(x) := (x - x_1)(x - x_2) \cdots (x - x_n).$$

This polynomial interpolates the points $x_1, \dots, x_n$ at the value 0 as demanded, but we see that in general $\overline{P}(x_0) \neq 1$. The solution is simply to *normalise* the value at x_0. This gives

$$P(x) = \frac{\overline{P}(x)}{\overline{P}(x_0)}.$$

It is easy to check that P now fulfills all the interpolation conditions.

Notice that it is *crucial* that $\overline{P}(x_0)$ is not zero. This happens if and only if x_0 is distinct from all the other points x_k for $k = 1, \dots, n$.

Let us write the polynomial P explicitly, and give it a new name: it is called the ***Lagrange polynomial*** ℓ_0:

$$\ell_0(x) = \frac{(x - x_1)}{(x_0 - x_1)} \frac{(x - x_2)}{(x_0 - x_2)} \cdots \frac{(x - x_n)}{(x_0 - x_n)}$$

We may similarly construct ℓ_1 which is such that $\ell_1(x_0) = 0$, $\ell_1(x_1) = 1$, ..., $\ell_1(x_n) = 0$. The other Lagrange polynomials ℓ_k are constructed in the same manner.

5.4 Solution of the Interpolation Problem

Using the Lagrange polynomials like basis functions, we are now able to solve the interpolation problem for *any* interpolation value y_k. Indeed, the polynomial

$$P(x) = y_0 \ell_0(x) + y_1 \ell_1(x) + \cdots + y_n \ell_n(x)$$

is readily checked to be such that $P(x_k) = y_k$ for $k = 0, \dots, n$.

5.5 The Interpolation Theorem

We know that the interpolation problem is a linear problem, and we know that for distinct interpolation points x_k, we may solve the interpolation problem for any interpolation values y_k. It means that the result is necessarily unique (standard result of linear algebra).

Theorem 5.1. *For distinct interpolation points x_k for $k = 0, \dots, n$, and for any interpolation values y_k for $k = 0, \dots, n$, there is a unique polynomial P of degree n such that $P(x_k) = y_k$.*

5.6 Aitken-Neville Formula

Suppose we want to interpolate between the $n+1$ points $x_0, \ldots, x_n$, and suppose that we can already interpolate between n given points. Suppose that P_n interpolates all the points except x_n, i.e., $x_0, \ldots, x_{n-1}$ and that P_0 interpolates all the points except x_0, i.e. $x_1, \ldots, x_n$. Notice how P_0 and P_n interpolate between n points, and we assume that we can do that.

The ***Aitken-Neville Formula*** is

$$P(x) = \frac{x - x_n}{x_0 - x_n} P_n(x) + \frac{x - x_0}{x_n - x_0} P_0(x).$$

If we define

$$\theta(x) = \frac{x - x_0}{x_n - x_0},$$

then notice that the formula may be rewritten

$$P(x) = (1 - \theta(x)) P_n(x) + \theta(x) P_0(x).$$

The crucial result is that P now interpolates between all the points! The result is clear for the "middle" points $x_1, \ldots, x_{n-1}$ since in that case we obtain

$$P(x_k) = (1 - \theta(x_k)) y_k + \theta(x_k) y_k = y_k$$

For the point x_0 we have $\theta(x_0) = 0$ (and thus $1 - \theta(x_0) = 1$), so

$$P(x_0) = P_n(x_0) = y_0.$$

A similar calculation leads to

$$P(x_n) = P_0(x_n) = y_n,$$

so P indeed interpolates all the points!

Week 4

5.7 The Aitken-Neville Algorithm

There is a simple way to use the Aitken-Neville formula. It can be used to compute the value of an interpolating polynomial at a point *without knowing that polynomial at all.* Suppose that we want to interpolate between the points $x_0, \ldots, x_4$. Suppose that P_0 interpolates x_0, and P_1 interpolates x_1. Then we compute $\theta(x) = \frac{x - x_0}{x_1 - x_0}$ for a *given* value of x, and we compute

$$P_{01}(x) = (1 - \theta) P_0(x) + \theta P_1(x)$$

We compute similarly P_{12}, P_{23} and P_{34}. Now we may proceed and compute $\theta = \frac{x - x_0}{x_2 - x_0}$ and

$$P_{012} = (1 - \theta) P_{01} + \theta P_{12}$$

5.8 Examples

Some interactive examples using Mathematica.

Lagrange function with 50 points.

Runge's phenomenon.

5.9 Interpolation Challenge

Suppose that you are asked to compute $p(\overline{x})$ for a given $\overline{x}$, where p interpolates the points x_0, x_1,. . . ,x_n.

You do *not* know the interpolating points. The only thing you have is a polynomial P_0 which interpolates all the points except x_0, and P_n, which intepolates all the points except x_n. No, not even that! You only know the *value* of P_0 and P_n at the point $\overline{x}$, so you only know the two *numbers*:

$$P_0(\overline{x}) \qquad P_n(\overline{x}).$$

You do not even know how many interpolation points there are! That is, you do not know n.

Can you at least know x_0 and x_n? No! Suppose you just know that $\overline{x}$ is in the middle between x_0 and x_n. So you know, for instance, that

$$P_0(\overline{x}) = 3, \qquad P_n(\overline{x}) = 5, \qquad \overline{x} = \frac{x_0 + x_n}{2}.$$

How could you ever compute $P(\overline{x})$?

Amazingly, it is possible!

The result is in fact

$$P(\overline{x}) = 4.$$

Indeed, using the Neville-Aitken formula, we obtain

$$\theta = \frac{\overline{x} - x_0}{x_n - x_0} = \frac{1}{2},$$

and

$$P(\overline{x}) = (1 - \theta)P_n(\overline{x}) + \theta P_0(\overline{x}).$$

So we obtain

$$P(\overline{x}) = \frac{3 + 5}{2} = 4.$$

5.10 Graph

See subsection 5.10 to see the relation between P_0, P_n and P appearing in the Aitken-Neville formula.

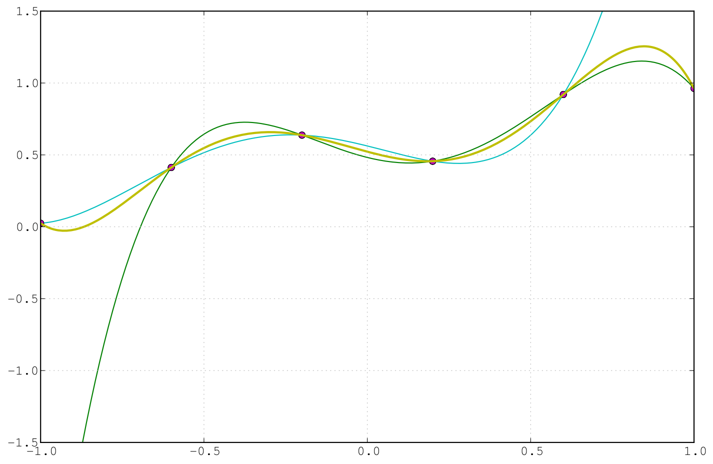


Figure 2: The bold curve, the polynomial interpolating all the points, is in between the two other polynomials P_0 and P_n

5.11 Interpolation Example

Suppose that we want to compute the value of P at the point 1, when we know that P interpolates the values

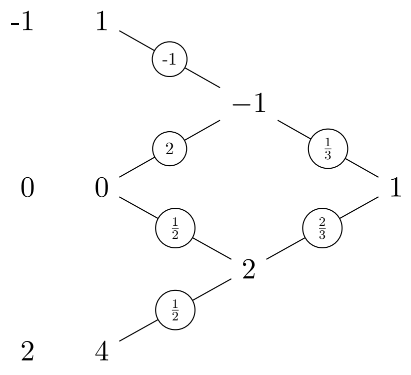
x	-1	0	2
y	1	0	4

It is done on purpose so that we know the final answer. The interpolation polynomial is indeed

$$P(x) = x^2$$

so the final answer is $P(1) = 1$, but let us pretend that we do not know the answer.

The Neville table is the following.

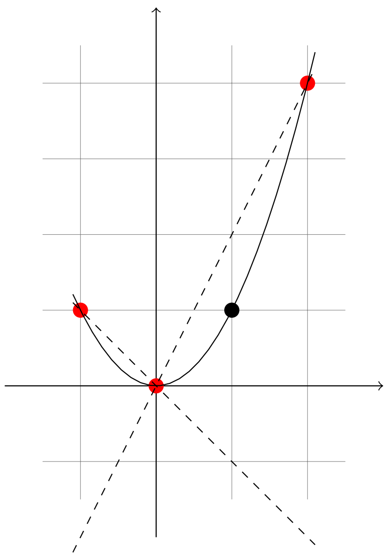


For the first value of the weight θ, we compute

$$\theta = \frac{1-(-1)}{0-(-1)} = 2.$$

Similarly, for the second value of θ, we obtain

$$\theta = \frac{1-0}{2-0} = \frac{1}{2}.$$



6 Newton's Divided Differences

We know how to compute the interpolation polynomial at a given *point* $\overline{x}$. The goal is now to compute the polynomial itself. The usual notation for polynomials is

$$P(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_n x^n.$$

However, this is *not* what we will use, although the method we present may also be adapted to that case. We will instead develop the polynomial on the following basis

$$P(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots a_n(x-x_0)\cdots(x-x_n).$$

6.1 Notation

Suppose that P interpolates the points $x_0, \dots, x_n$. The constant polynomial a_0 then clearly interpolates the first point x_0 since

$$P(x_0) = a_0 = f(x_0).$$

Now the polynomial consisting of the two first terms

$$P_1(x) := a_0 + a_1(x - x_0)$$

does interpolate the first two points x_0, x_1, because

$$P_1(x_0) = P(x_0) \qquad P_1(x_1) = P(x_1).$$

Finally, we obtain that the coefficient a_0 depends only on $x_0, f(x_0)$, the coefficient a_1 depends only on $x_0, f(x_0), x_1, f(x_1)$, and so on. This motivates the notation

$$a_0 = f[x_0], \qquad a_1 = f[x_0, x_1], \qquad \dots \qquad a_n = f[x_0, \dots, x_n].$$

Note how the number $f[x_0, \dots, x_k]$ depends on the values $x_0, \dots, x_k$ and $f(x_0), \dots, f(x_k)$.

Note also that the numbers $f[x_1]$ or $f[x_1, x_2]$ also make sense. They are given by the first coefficients of the polynomial interpolating the points x_1, x_2.

6.2 Formula

Note how the first coefficent $f[x_0]$ is easy to compute, since

$$f[x_0] = x_0.$$

Now the second coefficient is such that

$$f[x_0] + f[x_0, x_1](x_1 - x_0) = f(x_1),$$

so we obtain that

$$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0},$$

where we used that

$$f[x_1] = f(x_1).$$

The formula that we are going to prove later is that

$$f[x_0, \dots, x_k] = \frac{f[x_1, \dots, x_k] - f[x_0, \dots, x_{k-1}]}{x_k - x_0}.$$

6.3 Example

Let us compute the divided difference table for the following interpolation points:

x	-2	$-\frac{3}{2}$	$\frac{1}{2}$	1
y	2	$\frac{3}{2}$	$\frac{1}{2}$	1

$$\begin{tikzcd}[column sep=small, row sep=small]
-2 & 2 \arrow[dr, no head] & & & \\
 & & -1 \arrow[dr, no head] & & \\
-\frac{3}{2} & \frac{3}{2} \arrow[ur, no head] \arrow[dr, no head] & & \frac{1}{5} \arrow[dr, no head] & \\
 & & -\frac{1}{2} \arrow[ur, no head] \arrow[dr, no head] & & \frac{2}{15} \\
\frac{1}{2} & \frac{1}{2} \arrow[ur, no head] \arrow[dr, no head] & & \frac{3}{5} \arrow[ur, no head] & \\
 & & 1 \arrow[ur, no head] & & \\
1 & 1 \arrow[ur, no head] & & &
\end{tikzcd}$$

6.4 Proof of the Formula

The proof of Newton's formula closely follows [**?**].

The following polynomial, by definition, interpolates the points $x_0, \dots, x_{k-1}$.

$$P = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \dots, x_{k-1}](x - x_0)\cdots(x - x_{k-2})$$

Similarly, the polynomial Q defined by

$$Q = f[x_1] + f[x_1, x_2](x - x_1) + \cdots + f[x_1, \dots, x_k](x - x_0)\cdots(x - x_{k-1})$$

interpolates at the points $x_1, \dots, x_k$.

Using Neville's formula, the polynomial interpolating all the points $x_0, \dots, x_k$ must be

$$R = \frac{x - x_0}{x_k - x_0} Q + \frac{x - x_k}{x_0 - x_k} P$$

Now if we collect the highest order terms on both sides of the equality sign we obtain

$$f[x_0, \dots, x_k] = \frac{1}{x_k - x_0} f[x_1, \dots, x_k] + \frac{1}{x_0 - x_k} f[x_0, \dots, x_{k-1}],$$

which may be rewritten as

$$f[x_0, \dots, x_k] = \frac{f[x_1, \dots, x_k] - f[x_0, \dots, x_{k-1}]}{x_k - x_0}.$$

Week 5

6.5 Interpolation is a Linear Problem

Given the interpolation points x_k, the problem of interpolating the values y_k on the corresponding points is a *linear* problem. This enormously simplifies the interpolation problem.

What *linearity* means is the following. Suppose that a polynomial P is interpolating the points $(x_0, y_0), \ldots, (x_n, y_n)$. Then it is straightforward to verify that the new polynomial $2P$ will interpolate the points $(x_0, 2y_0), \ldots, (x_n, 2y_n)$. This is an example of linearity.

More generally, if another poynomial $\overline{P}$ interpolates the points $(x_0, \overline{y_0}), \ldots, (x_n, \overline{y_n})$, then $P + 2\overline{P}$ will interpolate $(x_0, y_0 + 2\overline{y_0}), \ldots, (x_n, y_n + 2\overline{y_n})$.

6.6 Programming: integer division

In Python, you should be cautious when dividing two numbers. If you use vanilla Python, then the following code has the following surprising behaviour:

```
>>> 1/2
0
```

That is because Python considers that you are trying to perform an integer division.

There are two remedies to that state of affairs. One is to use the *float notation* for *all the non-integer numbers* you are using:

```
>>> 1./2.
0.5
```

Another is to start *all* your Python files with the following code:

```
from __future__ import division
```

This will activate the more sensible division mode, where `1/2` is evaluated to `0.5`.

6.7 Programming: Functions

7 Interpolation Error

7.1 Error Formula

The formula is

$$f(x) - p(x) = (x - x_0) \cdots (x - x_n) \frac{f^{(n+1)}}{(n+1)!}.$$

Let us denote by p the polynomial which interpolates f at the points $x_0, \ldots, x_n$. Now for a fixed point x, we may interpolate the function f at those points plus the point x. This polynomial $\overline{p}$ is

$$\overline{p}(x) = p(x) + (x - x_0) \cdots (x - x_n) f[x_0, \ldots, x_n, x].$$

Now, since by definition $\overline{p}(x) = f(x)$, we have

$$f(x) = p(x) + (x - x_0) \cdots (x - x_n) f[x_0, \ldots, x_n, x].$$

The *interpolation error* is by definition the quantity $E(x) = |p(x) - f(x)|$, so we obtain

$$E(x) = |(x - x_0) \cdots (x - x_n)||f[x_0, \ldots, x_n, x]|.$$

The next step is to estimate $|f[x_0, \ldots, x_n, x]|$.

To ease the notations, we assume that $x_0 \leq x_k \leq x_n$ for $1 \leq k \leq n-1$, that is, we assume that x_0 and x_n are the smallest and biggest interpolation points respectively.

Now consider the function δ defined by

$$\delta(x) := p(x) - f(x).$$

The function δ is such that $\delta(x_k) = 0$ for $0 \leq k \leq n$, that is, δ has $n+1$ distinct zeros. As a result, δ' has n zeros, between every point x_k. We repeat the reasoning, and obtain that δ'' has $n-1$ zeros. Going on like this, we conclude that $\delta^{(n)}$ has one zero in $[x_0, x_n]$ which we denote by ξ:

$$\delta^{(n)}(\xi) = 0.$$

On the other hand, p is a polynomial of degree n, so its n-th derivative is a constant, namely

$$p^{(n)} = n! f[x_0, \ldots, x_n].$$

This shows that there exists a ξ in the interval $[x_0, x_n]$ such that

$$f[x_0, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

We now use that with $n+2$ points $x_0, \ldots, x_n, x$, and obtain that for every $x \in [x_0, x_n]$, there exists $\xi_x \in [x_0, x_n]$ *which depends on x*[1], such that

$$f[x_0, \ldots, x_n, x] = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}.$$

The quantity above is certainly smaller than

$$\sup_{\xi \in [x_0, x_n]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!},$$

so we conclude that the interpolation error E is bounded by

$$E(x) \leq |(x - x_0) \cdots (x - x_n)| \sup_{\xi \in [x_0, x_n]} \frac{|f^{(n+1)}(\xi)|}{(n+1)!} \qquad x \in [x_0, x_n].$$

Note how this formula has two completely different parts: one depends on the *interpolation points*, and one depends on *the function to interpolate.*

[1]and on all the points $x_0, \ldots, x_n$.

7.2 Chebyshev Points

The Chebyshev function is defined by

$$T_n(x) = \cos(n\theta) \qquad x = \cos(\theta) \qquad n = 0, 1, \dots \tag{3}$$

Note that *nothing* can indicate that T_n are in fact polynomials!

This is apparent once one has proved the formula

$$T_{n+1} = 2xT_n - T_{n-1}. \tag{4}$$

This is done using elementary trigonometric identities (which can be recovered using complex analysis).

Notice how the function T_{n+1} depends on the *two* previous functions T_n and T_{n-1}.

Now it is clear that

$$T_0(x) = 1,$$

because for $n = 0$ we have $\cos(n\theta) = 1$ no matter what.

For $n = 1$ we have $T_1(x) = \cos(\theta)$, and $x = \cos(\theta)$, we have

$$T_1(x) = x.$$

Using the recurrence relation (4), and the fact that T_0 and T_1 are polynomials, we have shown that the functions T_n are in fact *polynomials*!

7.3 Highest order coefficient

One can prove by induction using (4) that the highest order coefficient of T_n is 2^{n-1}. As a result, T_n may be written as

$$T_n = 2^{n-1}(x - x_0) \cdots (x - x_{n-1}), \tag{5}$$

where $x_0, \dots, x_{n-1}$ are the n roots of T_n.

Those roots are easy to compute using the original formula (3) for the definition of Chebyshev polynomials. Indeed, suppose that

$$\cos(n\theta) = 0,$$

this is fulfilled for

$$\theta = \frac{2k+1}{n}\frac{\pi}{2} \qquad k = 0, \dots, n-1.$$

Now we use that $x = \cos(\theta)$, and we obtain that

$$x_k = \cos\Big(\frac{2k+1}{n}\frac{\pi}{2}\Big) \qquad k = 0, \dots, n-1.$$

Now since, by definition, the polynomials T_n are bounded by -1 and 1, i.e.,

$$-1 \leq T_n(x) \leq 1 \qquad \text{for} \qquad -1 \leq x \leq 1,$$

and using (5), we obtain that, for the Chebyshev points $x_0, \dots, x_{n-1}$, we have

$$|(x - x_0) \cdots (x - x_{n-1})| \leq \frac{1}{2^{n-1}} \qquad \text{for} \qquad -1 \leq x \leq 1.$$

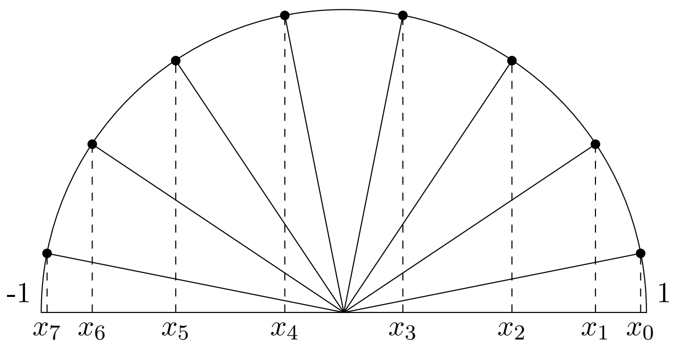


Figure 3: Illustration of the Chebyshev points for $n = 8$.

7.4 Adaption to an arbitrary interval

This all works fine for the interval $[-1, 1]$, but what if the interval is an arbitrary interval $[a, b]$? In that case, we simply rescale the points into the form

$$x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos\Big(\frac{2k+1}{n}\frac{\pi}{2}\Big).$$

7.5 Programming: Lists

Week 6

7.6 The Numerical Analysis Dilemma

The interpolation error formula subsection 7.1 may clearly be separated in *two parts.* The first part is the product

$$|(x - x_0) \cdots (x - x_n)|,$$

where the points x_k are the *interpolation points* chosen to interpolate the given function f. This part is only depending on the interpolation points, which is *our responsibility.*

The second part is

$$\frac{|f^{(n+1)}(\xi)|}{(n+1)!}.$$

This part depends on the function and is thus *given.*

One of the goals of numerical analysis is to separate the error due to the *data* (here, the function), and the error due to the *algorithm* (here, the interpolation point). The general idea is that if the data is “bad” (for instance, here, if the function is highly oscillatory and thus has a very high n-th derivative), there is nothing much we can do. However, if the data happens to be “good”, then one has to make sure that one produces an accordingly good result. We will see a similar dilemma in **??**.

7.7 Chebyshev Points are Optimal

What is even more surprising is that the Chebyshev points are the best possible choice! The idea is that every polynomial T_n oscillates from -1 to 1. To be precise, and using the definition of T_n, we see that T_n goes n times between -1 and 1. Now, suppose that for an alternative choice of roots $\overline{x_0}, \dots, \overline{x_n}$, we had

$$|(x - \overline{x}_0) \cdots (x - \overline{x}_{n-1})| \leq \frac{1}{2^{n-1}} \qquad \text{for} \qquad -1 \leq x \leq 1.$$

Let use define the corresponding polynomial

$$P(x) := 2^{n-1}(x - \overline{x}_0) \cdots (x - \overline{x}_{n-1}).$$

We see that P and T_n have the same coefficient for the highest order term, namely 2^{n-1}, so $P - T_n$ has degree $n-1$. Now using the oscillating property of T_n, we conclude that there are n distinct points ξ_k such that $P(\xi_k) = T_n(\xi_k)$. The polynomial $P - T_n$ is thus zero on n distinct points and has degree $n-1$, so it must be zero. We conclude that the roots of T_n are the best possible choice for the minimization of $(x - x_0) \cdots (x - x_n)$ on the interval $[-1, 1]$.

7.8 Implementing Neville's Algorithm

Things to know for the exam

- Fixed point theorem
- Newton's method to find root of equations: how to use it in various cases, what are its limits
- Fundamental theorem of interpolation: there is a unique polynomial of degree k which interpolates through $k+1$ points
- Lagrange polynomials
- Neville formula: build an approximation polynomial from two other interpolation polynomials
- Neville algorithm to compute the value of an interpolation polynomial without know the polynomial
- Newton's divided differences
- Interpolation Error formula: what it means, how to use it, what are its limits.
- Interpret a log-log graph
- Use Taylor's formula to show the order of a given approximation formula
- Richardson's extrapolation algorithm

- Gauss Elimination
- Cost of Gauss Elimination
- Gauss Elimination is equivalent to an LU decomposition