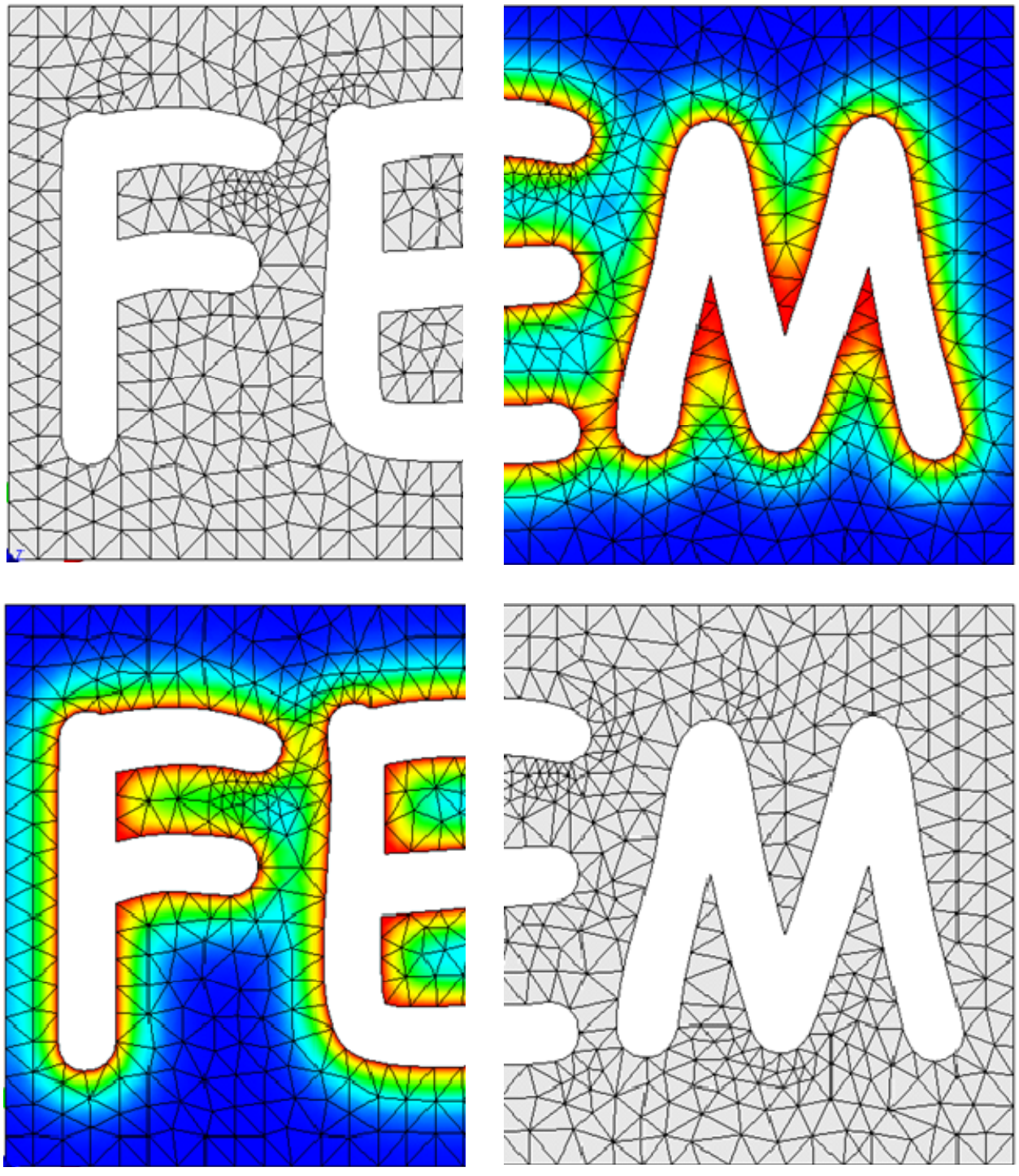
A Tiny Tale of some Atoms in Scientific Computing

Bertil Nilsson
Halmstad University
2018-12-01



1. Computer arithmetic and error analysis

1.1 Computer representation of Integers

Since we will be using the computer throughout this course, we have to point out some properties of computer arithmetic. We are distinguishing arithmetic carried out on a computer from the "theoretical" arithmetic we learn about in school. The fundamental issue that arises when using a computer stems from the physical limitation in memory. A computer must store numbers on a physical device which cannot be "infinite". Hence, a computer can only represent a finite number of numbers. Every computer language has a finite limit on the numbers it can represent. It is quite common for a computer language to have integer and long integer types of variables, where an integer variable is an integer in the range of {−32 768, −32 767, …, 32 767}, which are the numbers that take two bytes of storage, and a long integer variable is an integer in the range {−2 147 483 648, −2 147 483 647, …, 2 147 483 647}, which are the integers requiring four bytes of storage. Here, a "byte" of memory consists of 8 "bit-cells", each capable of storing either a zero or a one. This can have some serious consequences. In particular, we cannot check whether some fact is true for all integers using a computer to test each case.

1.2 Decimal expansion of Rational numbers

The most useful way to represent a rational number is in the form of a decimal expansion, such as $\frac{1}{2} = 0.5$, $\frac{5}{2} = 2.5$, and $\frac{5}{4} = 1.25$. In general, a *finite decimal expansion*, a (long) float or floating point variable, is a number of the form

$$\pm\, p_m\, p_{m-1} \cdots p_2\, p_1\, p_0.q_1\, q_2 \cdots q_n,$$

where the digits p_m, p_{m-1}, …, p_0, q_0, …, q_n, are each equal to one of the natural numbers {0, 1, … , 9} while m and n are natural numbers. The decimal expansion is a shorthand notation for the number

$$\pm\, p_m 10^m + p_{m-1}10^{m-1} + \cdots + p_1 10^1 + p_0 10^0 + q_1 10^{-1} + \cdots + q_{n-1}10^{-(n-1)} + q_n 10^{-n}$$

for example

$$432.576 = 4\cdot 10^3 + 3\cdot 10^2 + 2\cdot 10^0 + 5\cdot 10^{-1} + 7\cdot 10^{-2} + 6\cdot 10^{-3}.$$

The same story goes for irrational numbers, like π, e and $\sqrt{2}$. There is always an algorithm to make an approximate decimal expansion of them. The difference is that a rational number has periodic decimal expansion while irrational number don't.

1.3 Computer representation of Rational numbers

The decimal expansion $\pm\, p_m\, p_{m-1} \cdots p_2\, p_1\, p_0.q_1\, q_2 \cdots q_n$ uses the base 10 and consequently each of the digits p_i and q_j may take on one of the 10 values 0, 1, 2, …, 9. Of course, it is possible to use bases other than ten. For example, the Babylonians used the base sixty and thus their digits range between 0 and 59. The computer operates with the base 2 and the two digits 0 and 1. A base 2 number has the form

$$\pm\, p_m 2^m + p_{m-1}2^{m-1} + \cdots + p_1 2^1 + p_0 2^0 + q_1 2^{-1} + \cdots + q_{n-1}2^{-(n-1)} + q_n 2^{-n}$$

which we again may write in short hand

$$\pm\, p_m\, p_{m-1} \cdots p_2\, p_1\, p_0.q_1\, q_2 \cdots q_n$$

where again n and m are natural numbers, and now each p_i and q_j take the value 0 or 1. For example, in the base two

$$11.101 = 1\cdot 2^2 + 1\cdot 2^0 + 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} = 4 + 1 + 0.5 + 0.125 = 5.625.$$

In the floating point arithmetic of a computer using the standard 32 bits, numbers are represented in the form

$$\pm\, r\, 2^N,$$

where $1 \leqslant r \leq 2$ is the mantissa and the exponent N is an integer. Out of the 32 bits, 23 bits are used to store the mantissa, 8 bits are used to store the exponent and finally one bit is used to store the sign. Since $2^{10} \approx 10^{-3}$ this gives 6 to 7 decimal digits for the mantissa while the exponent N may range from −126 to 127, implying that the absolute value of numbers stored on the computer may range from approximately 10^{-40} to 10^{40}. Numbers outside these ranges cannot be stored by a computer using 32 bits. Some languages permit the use of double precision variables using 64 bits for storage with 11 bits used to store the exponent, giving a range of $-1022 \leqslant N \leqslant 1023$, 52 bits used to store the mantissa, giving about 15 decimal places.

We point out that the finite storage capability of a computer has two effects when storing rational numbers. The first effect is similar to the effect of finite storage on integers, namely only rational numbers within a finite range can be stored. The second effect is more subtle but actually has more serious consequences. This is the fact that numbers are stored only up to a specified number of digits. Any rational number that requires more than the finite number of digits in its decimal expansions, which included all rational numbers with infinite periodic expansions for example, are therefore stored on a computer with an error. So for example $\frac{2}{11}$ is stored as 0.1818181 or 0.1818182 depending on whether the computer truncate or rounds.

Rounding is a little bit elaborate, correct rounding is based on three rules. First determine what your rounding digit is and look to the right side of it; 1. If that digit is less than 5, simply drop the digits to the right of it, 2. If that digit is grater than 5 add one to the rounding digit and drop all digits to the right of it, 3. If the digit(s) to the right is 5(0…), add one to the rounding digit if it's odd, and drop the digit(s) to the right of it. We say that we have n correct decimals if the error is less than 0.5×10^{-n}, and that we have n significant digits if there are n correct rounded digits, counted from the left omitting leading zeros.

A typical truncation error comes from modelling with Taylor expansion (Brook Taylor, 1685-1731, English mathematician), of $f(x+h)$ about x, i.e.

$$f(x+h) = f(x) + h\,f'(x) + \frac{h^2}{2}f''(\xi), \quad \xi \in (x,\, x+h) \tag{1.1}$$

For such an expansion to be valid, we assume that $f(x)$ has two continuous derivatives. After rearranging we obtain a formula for differentiation

$$f'(x) = \frac{f(x+h)-f(x)}{h} - \frac{h}{2}f''(\xi), \quad \xi \in (x,\, x+h) \tag{1.2}$$

The first term is the numerical counterpart to differentiation $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ and the second term on the right-hand side of (1.2) is the error term. Since the approximation to the derivative can be thought of as being obtained by truncating this term from the exact formula (1.2), this error is called the truncation error. The small parameter h denotes the distance between the two points x and $x+h$. As this distance tends to zero, i.e., $h\downarrow0$, the two points approach each other and we expect the approximation to improve. This is indeed the case if the truncation error goes to zero, which in turn is the case if $f''(\xi)$ is well defined and bounded in the interval $(x,\, x+h)$. The “speed” at which the truncation error goes to zero as $h\downarrow0$ is called the rate of convergence. When the truncation error is $O(h)$, we say that the method is a first order method, and we refer to a method of order p if the truncation error is $O(h^p)$. The higher order p the better.

There are practical considerations when it comes to using floating point arithmetic in a computer. The typical question is how to choose the size of h. A large value for h producing a distant support for the secant is clearly not good, and a too ambitious small one, makes (1.2) run into a $\frac{0}{0}$ disaster due to rounding error in subtraction and cancellation.

A choice for h which is small without producing a large rounding error is $x\sqrt{\varepsilon}$ where the machine epsilon ε is typically of the order 10^{-15} for a double precision (64-bit) variable. It's essentially all about some technicalities in the representation of floating point numbers in binary form. Even if x is a machine-representable number we must realize that $x+h$ will almost certainly not be. This means that $(x+h)-x$ is not equal h. In order to find ε for the computer at hand we choose a machine-representable form $\varepsilon = 2^{-n}$, and find the largest n for which $1+\varepsilon > 1$ still holds in floating point operations.

But this is not the end of the story. Introduction of an error in the 7'th or 15'th digit would not be so serious except for the fact that round-off errors accumulate when arithmetic operations are performed. In other words, if we add two numbers with a small error, the result may have a larger error being the sum of the individual errors (unless the errors have opposite sign or even cancel). Cancellation errors appear for instance when subtracting numbers of almost equal size giving numerical zero. So working with finite decimal representations with different kind of errors may have consequences. The technicalities of computer arithmetic operations are standardized in IEEE.

1.4 How can we define error in practical computations?

In its simplest form, is the error the difference between the computed x and the exact one x^*. Hence, we can write

$$\text{error} = x - x^*$$

Since we are usually interested in the magnitude or absolute value of the error we can also define

$$\text{absolute error} = |\,x - x^*\,|$$

In practical calculations, it is important to obtain an upper bound on the error ε, such that

$$|\,x - x^*\,| < \varepsilon$$

Clearly, we would like ε to be small! In practice we are often more interested in so called relative error rather than absolute error and we define

$$\text{relative error} = \frac{|x-x^*|}{|x|}$$

In order to avoid division by zero, a common fix in practical computations is to make a mix of absolute error then $|x|$ is small, and relative error when $|x|$ is big

$$\text{relative error} = \frac{|x-x^*|}{1+|x|}$$

Hence, a given ε may be a good or a bad relative error depending on $|x|$. We will address the issue of errors in the chapters that follows.

1.5 Problems

1.1 Consider the recipe for the machine epsilon ε given in the text. Write a small program to determine ε and the corresponding n for your computer. $\{$Depending on hardware ... $\varepsilon = 1.42109\times10^{-14}$, $n = 46\}$

1.2 Determine the largest h in $x \in [0, h]$ for which we can assure four and six correct decimals for the approximations

a) $\sin(x) \approx x$, $\{h_4 < 0.0669,\ h_6 < 0.0144\}$

b) $\cos(x) \approx 1 - \frac{x^2}{2}$, $\{h_4 < 0.186,\ h_6 < 0.0588\}$

c) $\frac{1}{1-x^2} \approx 1 + x^2$, $\{h_4 < 0.0839,\ h_6 < 0.0266\}$

1.3 We want to illustrate the concept of local linearization by replacing $y(x) = \sqrt{x}$ with a straight line in the neighborhood of $x = 1$.

a) Let $h > 0$ be a small number and determine the straight line $\ell(x) = kx + m$ between the points $\left(1 - h, \sqrt{1-h}\right)$ and $(1, 1)$. Let the difference $\delta(x) = |y(x) - \ell(x)|$, $x \in [1 - h, 1]$ and find algebraically the extreme point x^* for maximum. Collect your numerical calculations of $(x^*, \delta(x^*))$ in a table for some $h = 10^{-k}$, $k = 1, 2, 3, 4$.

$$\left\{\begin{array}{lll} h & x^* & \delta^* \\ \hline 0.1 & \{x \to 0.949342\} & 0.000337844 \\ 0.01 & \{x \to 0.994994\} & 3.14861\times10^{-6} \\ 0.001 & \{x \to 0.9995\} & 3.12734\times10^{-8} \\ 0.0001 & \{x \to 0.99995\} & 3.12409\times10^{-10} \end{array}\right\}$$

b) Verify your findings in **a)** by Taylor expansion that $\delta(x^*) \approx ch^p$ and identify c and p. $\left\{c = \frac{1}{32} \approx 0.03125,\ p = 2\right\}$

2. Interpolation

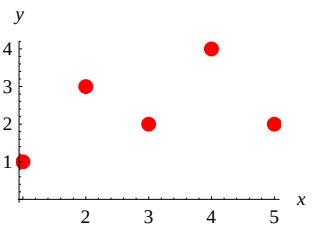
In engineering and science, one often has a number of data points (x_i, y_i), $i = 0, \ldots, n$, obtained by sampling or experimentation, which represent the values of an unknown function f for a limited number of values of the independent variable,

$$\begin{pmatrix} x_0 & y_0 \\ x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}.$$

The points $x_0 < x_1 < \ldots < x_n$ are called the interpolation points. The property of "passing through these points" is referred to as interpolating the data. It is often required to interpolate (i.e. estimate) the value of that "function" for an intermediate value of the independent variable $y = f(x)$. We say that we interpolate if $x \in [x_0, x_n]$, and extrapolate otherwise. Beware of the latter. The situation is similar in a multidimensional setting.

```
data = {{1, 2, 3, 4, 5}, {1, 3, 2, 4, 2}}^T;

gd = ListPlot[data, PlotStyle → {Red, PointSize[Large]}, AxesLabel → {x, y}]
```



The interpolation problem is to construct a function $P(x)$ that passes through these points, i.e., to find a function $P(x)$ such that the interpolation requirements $P(x_i) = y_i$, $i = 0, \ldots, n$, are satisfied. Such a function is called an interpolant. In this context is the interpolation points also called support points, depending how $P(x)$ is furnished. Due to the interpolation requirements we can always form P as a linear combination of a set of basis functions

$$P(x) = \sum_{i=0}^{n} c_i \varphi_i(x) \tag{2.1}$$

From this point of view is P a function in a function space that is spanned by the basis functions. There are many ways of choosing the basis functions. The most common is the monom x^i resulting in an interpolating polynomial, but also harmonic functions, $\sin(x)$ and $\cos(x)$, are of interest, especially when the data represents periodic behavior. It's also common to use a set of orthogonal basis functions, for instance the Legendre polynomials or Chebyshev polynomials.

There are many different interpolation methods, some of which are described below. Some of the things we need to take into account when choosing an appropriate algorithm are: How accurate is the method? How expensive is it? Is the interpolant smooth enough to represent velocity and acceleration? How many data points are needed? One may even ask if interpolation makes sense? For example election to the parliament, when results are known every fourth year.

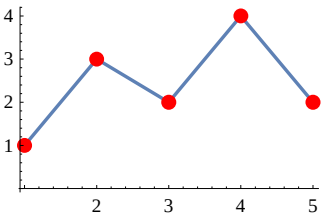
2.1 Piecewise linear interpolation

The simplest interpolation method is to use a linear function between the data points. In this sense is this an interpolation with a polynomial of order one with smallest possible support, the two nearest points. When not all support points are included we talk about local interpolation. The basis functions in this case are the so called hat-functions.

$$\varphi_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}}, & x_{i-1} < x < x_i \\ \frac{x_{i+1} - x}{x_{i+1} - x_i}, & x_i < x < x_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Clearly by construction $\varphi_i(x_j) = \delta_{ij}$, where $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$ is the Kronecker delta function. The end point hats are of natural reasons called half-hats and are defined accordingly.

```
Show[ListPlot[data, Joined → True], gd]
```



Find for example the interpolated value y for $x = 3.9$.

```
Interpolation[data, InterpolationOrder → 1][3.9]
```

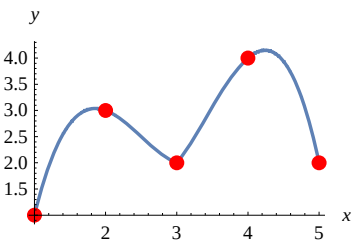
3.8

We can already by inspection see that the interpolant is continuous, C^0, but the derivatives are poorly accomplished.

2.2 Higher order polynomials

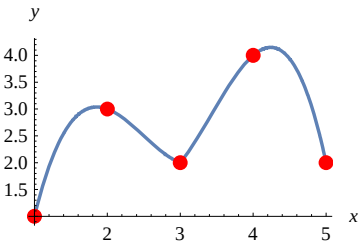
It is straight forward to increase the order of the interpolating polynomial $p_n(x)$. For order n we need support of $n + 1$ data points. Also in this case is the closest data points the natural selection. A very popular one is the local cubic interpolation, i.e. $p_3(x)$, with a moving window containing the closest four data points. This is the standard selection for interpolation i *Mathematica*.

```
Show[Plot[Interpolation[data][x], {x, 1, 5}, AxesLabel → {x, y}], gd]
```



Another popular variant of the local cubic interpolation, $p_3(x)$, is the Hermite polynomial (Charles Hermite, 1822-1901, French mathematician). This, like the linear interpolation, is a local interpolation between two data points. As $p_3(x)$ requires four conditions to be unique we also prescribe the derivative at the data points. This can be done by numerical differentiation of the data or via suitable constraints inherited from the engineering problem at hand.

```
Show[Plot[Interpolation[data, Method → "Hermite"][x], {x, 1, 5}, AxesLabel → {x, y}], gd]
```



Finally we come to a polynomial of highest order possible, due to the number of data points. This is usually called global interpolation. A direct approach in this case is to solve a linear system of equations for the unknown coefficients c_i in the interpolating polynomial $p_n(x) = \sum_{i=0}^{n} c_i x^i = c_0 + c_1 x + \ldots + c_n x^n$, with the monoms x^k as basis functions. The unknown coefficients c_i are derived by applying $p_n(x)$ to each data point.

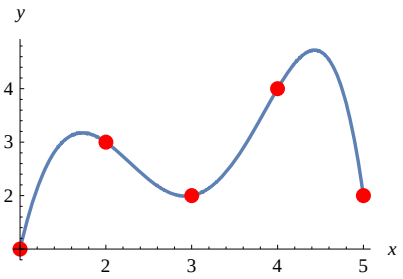
$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ & & \vdots & & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{2.3}$$

For large n the coefficient matrix, the Vandermonde matrix (Alexandre-Théophile Vandermonde, 1735-1796, French mathematician), will become ill-conditioned as the determinant numerically approaches zero. In *Mathematica* this is readily done.

```
p = Fit[data, {1, x, x^2, x^4, x^5}, x]
```

$-0.0754902\, x^5 + 0.590686\, x^4 - 9.47304\, x^2 + 23.899\, x - 13.9412$

```
Show[Plot[p, {x, 1, 5}, AxesLabel → {x, y}], gd]
```



The global interpolation polynomial can be more cleverly defined in order to avoid the problem of ill-condition. The first one is the Newton interpolation polynomial (Isaac Newton, 1642-1727, English scientist), where we can easily identify the basis functions.

$$N_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots \tag{2.4}$$

The advantage here is that you can easily add new data points without the need to completely reformulate the interpolant. It's also straight forward to find the constants c_i, by applying the data points one at a time.

$$\begin{cases} y_0 = N_n(x_0) = c_0 \\ y_1 = N_n(x_1) = c_0 + c_1(x_1 - x_0) \\ y_2 = N_n(x_2) = c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1) \\ \quad \vdots \end{cases}$$

This linear system of equations is easily solved by a Gaussian backward substitution step. The other one is called the Lagrange interpolation polynomial (Joseph-Louis Lagrange, 1736-1813, French mathematician) of order n through the data points

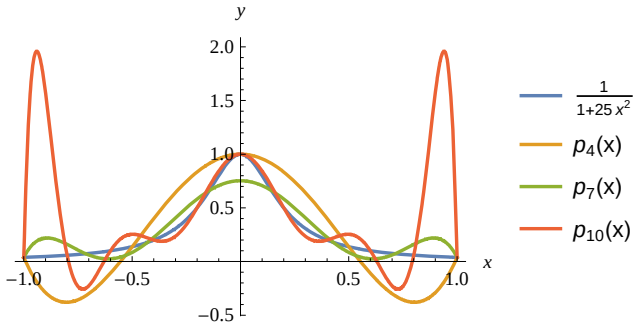
$$L_n(x) = \sum_{k=0}^{n} y_k l_k^n(x) \tag{2.5}$$

where the basis functions $l_k^n(x)$ are given by

$$l_k^n(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}, \quad k = 0, 1, \ldots, n \tag{2.6}$$

By this construction the interpolation polynomial satisfies the interpolation conditions $L_n(x_i) = y_i$, and the basis functions $l_k^n(x_i) = \delta_{ik}$, where δ_{ik} is the Kronecker delta (Leopold Kronecker, 1823-1891, German mathematician). It's now straightforward to find a derivative of desired order, provided that n, the number of interpolation points, is sufficiently large.

A severe problem that comes for free with interpolation using high order global polynomials is the so called Runge's phenomena (Carl Runge, 1856-1927, German mathematician) resulting in high oscillations close to the end points of the interval. Here is an example of $\frac{1}{1+25x^2}$ interpolated at equally spaced data points on $[-1, 1]$ with three polynomials $p_n(x)$ of order n. So, don't cheat yourself with the easy answer "The higher order polynomial used the better interpolation!"



2.3 Splines

Due to Runge's phenomena and in almost every practical situation is interpolation taken to be a local affair, especially with irregular spaced data in higher dimension. A solution is to combine Hermite third degree polynomial with a scheme to calculate the derivatives at the data points. One such method is the Spline interpolation polynomial $s(x)$. The idea is to mimic a long ruler, *ri*, used by ship hull designers. This ruler acts as an ordinary beam and when forced to pass the data points it deforms in such a way that the bending energy $\int_{x_0}^{x_n}\left(\frac{d^2s}{dx^2}\right)^2 dx$ is minimized. The result is a global interpolation with local polynomial support with continuous second order derivatives that not lacks the Runges phenomena. We define $s(x)$ such that

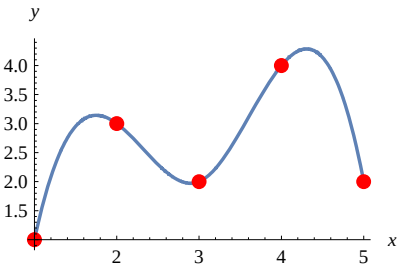
1. $s(x_i) = y_i$
2. $s(x)$ is a piecewise polynomial of degree three over each segment $[x_{j-1}, x_j]$.
3. $s(x)$ is globally C^2. That is $s(x_i^-) = s(x_i^+)$, $s'(x_i^-) = s'(x_i^+)$ and $s''(x_i^-) = s''(x_i^+)$, where +/- refer to one-sided limits.

Let us count the degrees of freedom. A cubic polynomial has four coefficients. There are $n + 1$ points, hence n segments, for a total of $4n$ numbers to be specified. The interpolating conditions $s(x_j) = y_j$ specify two degrees of freedom per polynomial: one value at the left endpoint x_{j-1}, and one value at the right endpoint x_j. That is $2n$ conditions. Continuity of $s(x)$ follows automatically. The continuity conditions on s', s'' are imposed only at the interior grid points x_i for $i = 1, \ldots, n - 1$, so this gives rise to $2(n - 1)$ additional conditions, for a total of $4n - 2$ equations. There is a mismatch between the number of unknowns, $4n$, and the number of conditions $4n - 2$. Two more constraints are needed to completely specify a cubic spline interpolant. The most widespread choices for fixing these two degrees of freedom are:

• Natural spline: $s''(x_0) = s''(x_n) = 0$. This is the classic version that mimics the displacement of the *ri*-beam, a condition of vanishing second derivative corresponds to moment free end points. If nothing else is stated this is understood to be "the spline".
• Clamped spline: $s'(x_0) = q_0$, $s'(x_n) = q_n$. Here q_0 and q_n are specified derivatives that depend on the particular application.
• Periodic spline: assuming that $s(x_0) = s(x_n)$, then we also impose $s'(x_0) = s'(x_n)$, $s''(x_0) = s''(x_n)$.

A combination of the first two (natural&clamped) are equally used in practice. Here comes the natural spline passing our data points.

```
Show[Plot[Interpolation[data, Method → "Spline"][x], {x, 1, 5}, AxesLabel → {x, y}], gd]
```



2.4 Bezier splines

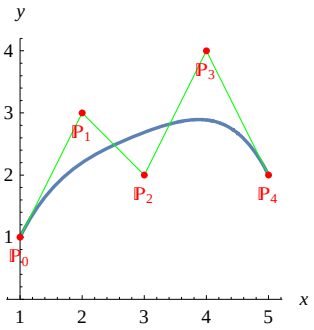
A clever reformulation of the spline concept was done by Pierre Bezier (1910-1999, French mathematician) in the beginning of the 1960's when he was a mathematician at the Renault car manufacturer. It is now a days the standard when it comes to describe complicated sculptured geometry in computer graphics, CAD and even font types. It is heavily used in car, airplane and interactive game applications. The concept is further developed in the last 30 years, especially the NURBS (Non Uniform BSplines) which is a special spline function of rational form. But still people are talking about Bezier-patches that cover the body surface of a car.

The Bezier concept is given in parameter form, in order to ease the work in a 2D or 3D setting. The shape of the curve is geared by what is called control points, and the curve is passing through the first and last control point only. The tangent line there coincides with the line segment to the neighboring control points. This is the reason for its popularity, the control points have a direct geometrical meaning. Just drag the control points to change the shape and to ensure continuity to the next curve. Another nice property of the curve is that it is fully contained in the convex hull of the control points. The polyline connecting the control points is often called the control polygon, a little bit odd naming. Finally the form of the Bezier curve is

$$\mathbb{B}(u) = \sum_{i=0}^{n} \mathbb{P}_i b_{i,n}(u) \tag{2.7}$$

where the basis functions $b_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}$, $u \in [0, 1]$ are the Bernstein basis functions, $\binom{n}{i} = \frac{n!}{k!\,(n-k)!}$ the binomial coefficient, and $\mathbb{B}$, $\mathbb{P}_i$ are vectors in the space under consideration. Here is the Bezier curve of forth order in 2D generated by our data points as control points. If the data points had been 3D a curve in the 3D space had been generated.

```
ParametricPlot[BezierFunction[data][u], {u, 0, 1},
 PlotRange → {Automatic, {0, 4.2}}, Axes → True, AxesLabel → {x, y},
 Epilog → {Green, Line[data], Red, PointSize[0.025], Point[data],
   Table[Text[ℙi-1, data[[i]] + {0, -0.3}], {i, Length[data]}]}]
```

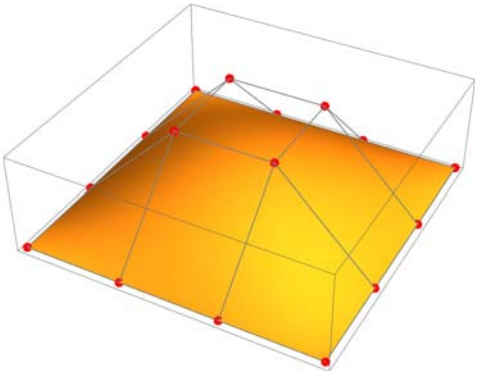


A popular extension is weighted Bezier curve, which is a rational version where a weight ω_i can be added to each point $\mathbb{P}_i$ making them more or less important in order to get the curve better aligned to a desired form. This is also known as "spline under tension" according to its ability to mimic a rubber band.

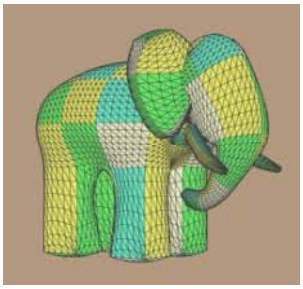
$$\mathbb{B}(u) = \frac{\sum_{i=0}^{n} \omega_i \mathbb{P}_i b_{i,n}(u)}{\sum_{i=0}^{n} \omega_i b_{i,n}(u)}, \quad \omega_i \in (-\infty, +\infty) \tag{2.8}$$

Needless to say, this form does not obey the convex hull property, unless $\omega_i > 0$, $\forall i$. If $\omega_i \to +\infty$, $\forall i$, the curve will coincide with the control polygon, and if some $\omega_i < 0$ the curve will typically leave the convex hull rather quickly. Finally, the tale is similar for a Bezier surface patch, $\mathbb{B}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbb{P}_{i,j} B_i^n(u) B_j^m(v)$. Here is a typical one.

```
pts = {{{0, 0, 0}, {0, 1, 0}, {0, 2, 0}, {0, 3, 0}}, {{1, 0, 0}, {1, 1, 1}, {1, 2, 1}, {1, 3, 0}},
   {{2, 0, 0}, {2, 1, 1}, {2, 2, 1}, {2, 3, 0}}, {{3, 0, 0}, {3, 1, 0}, {3, 2, 0}, {3, 3, 0}}};
Show[Graphics3D[{PointSize[Medium], Red, Map[Point, pts]}],
 Graphics3D[{Gray, Line[pts], Line[Transpose[pts]]}],
 ParametricPlot3D[BezierFunction[pts][u, v], {u, 0, 1}, {v, 0, 1}, Mesh → None]]
```



A typical body paved with Bezier surface patches.



2.5 Problems

2.1 Plot the three hat-functions (2.2) according to the interior points in our **`data`**.

2.2 Estimate y at $x = 4.2$ in **`data`** using linear interpolation. Use both **`Interpolation`** and calculations by hand with *Mathematica* as a pocket calculator of course.

2.3 Fit a full interpolation polynomial $p_4(x)$ to **`data`** using the Vandermonde formulation (2.3). Compare with the result given by **`Fit`**. Estimate $y(4.2)$ and $y'(4.2)$. Compare $y(4.2)$ with **Problem 2.2**.

2.4 Focus on the last three points i **`data`**. **a)** Find the interpolation polynomial using Newtons form (2.4), **b)** Lagrange form (2.5). They should of course be the same (as if we have used (2.3)!), **c)** Plot one of them, **d)** Derive and plot the corresponding Lagrange basis functions (2.6), **e)** Find the extreme point and value.

2.5 London Royal Rowing Club is of various reasons interested in finding a mathematical description of the northern shore of river Thames. Given is a table of (x, y)-coordinates of the northern entrance to eight bridges

Battersea Br	(7.4, 4.2)	Waterloo Br	(24.0, 6.0)
Chelsea Br	(12.8, 2.4)	Blackfriars	(27.1, 4.7)
Vauxhall Br	(17.6, 0.6)	London Br	(30.2, 1.8)
Westminster	(21.5, 4.0)	Tower Br	(32.5, −0.6)

River Thames is at least C^1, and the part between Battersea Br and Chelsea Br should be a straight line. Determine a spline curve between Chelsea Br and Tower Br with suitable end point conditions. Plot the curve between the eight bridges!

2.6 Estimate the length of river Thames from Battersea Br and Tower Br.

2.7 The Bezier curve generated by the control points (0, 0), (−2, 3), (3, 2) and (−1, 1) intersects itself at one point (x_c, y_c). Find it using a minimization formulation.

2.8 Wrap the Plot of the Bezier curve, including control points and polygon, according to (2.7) and our **`data`**, in **`Manipulate`** with handles on the control points. Play with the shape of the curve by dragging the handles.

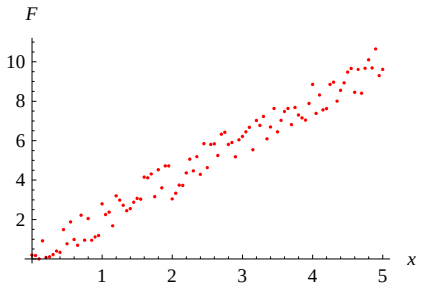
2.9 Plot the Bezier under tension curve, including polygon and control points, according to (2.8) and our **`data`**. Let $\omega = \{1, 1, 1, \omega_3, 1\}$ and wrap it in **`Manipulate`** with control $\omega_3 \in [-0.5, 5]$ and starting value equal to 1.

3. Method of least squares

3.1 Introduction

Experiments generally have errors or uncertainty in measuring their outcome. Errors can be human, but it's more likely due to inherent limitations in the equipment being used to make the measurements. We often want to represent the experimental data by some functional expression. Interpolation is often unsatisfactory because it fails to account for the error or uncertainty in the data.

For example, we have perhaps learned in school that the relationship between the force and the deflection for an ordinary spiral spring is $F = kx$, where F is the force, k the spring constant, and x the deflection from the unloaded state. We want to determine k experimentally in the laboratory by measuring x for given F. Here is the result for some stretching of a typical spring.



Unfortunately, it is extremely unlikely that the data we observe will, in this case, align to our linear relationship. There are at least two reasons for this; experimental error and the linear model is just a model, the real world behavior is much more complicated. Any how by inspection it seems to be fairly linear and we want to find the best k from our hard labor. The Least squares method (LSM) can help us, and the tale goes like this.

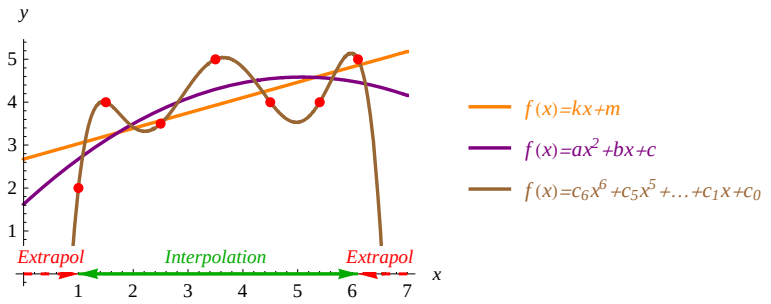
3.2 The least squares method (LSM)

We have some data to which we want to fit a function "as good as possible". Let this function be furnished $y = f(x, \mathbb{p})$, where x is the independent variable, y the dependent, and $\mathbb{p} = (p_1, p_2, \ldots)^T$ a vector of parameters that should be determined by the method of least squares. There can, of course, be more than one independent variable, for instance $T = f(\mathbb{x}, \mathbb{p}) = f((x, y, z, t)^T, \mathbb{p})$ as a model for the temperature in a room over time. We usually denote f the model and $\mathbb{p}$ for the model parameters. For example

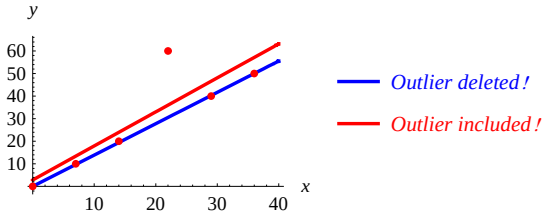
$$\begin{array}{lll} y = kx + m & \Rightarrow & \mathbb{p} = (k, m)^T \\ y = ae^{bx} & \Rightarrow & \mathbb{p} = (a, b)^T \end{array} \quad \left| \quad \begin{array}{lll} y = c_0 + c_1 x + c_2 x^2 & \Rightarrow & \mathbb{p} = (c_0, c_1, c_2)^T \\ y = a\cos(\omega t) + b\sin(\omega t) & \Rightarrow & \mathbb{p} = (a, b, \omega)^T \end{array}$$

The typical scenario is that we must furnish the model f with the intrinsic functions free of choice, (LSM) then helps us to determine the parameters $\mathbb{p}$. That is, a spectra of good and bad fit according to our setup of the model. You need to know that! Beware! In practice, hopefully, we know a mathematical background and the measurements are undertaken to determine $\mathbb{p}$ in order to use f in a simulation model.

It is of course important to select functions in the model that reflects the behavior of the system we study in a qualitative way. Note in the figure below that a complicated model passing all data points, for instance the interpolation polynomial of order n to the $n + 1$ data points, is probably not a very wise decision. Maybe the physics is better mirrored by the two "less nervous" polynomials of lower order. We may also perhaps need to take the derivatives into account, if for example f is responsible for manage the gas pedal giving smooth riding comfort in an autonomous car? The word interpolation is in this context used for deriving $f(x)$ when x is inside the range covered by the data points, and extrapolation for outside. Beware the latter! The model is usually not a good fellow here! Runge's phenomena!

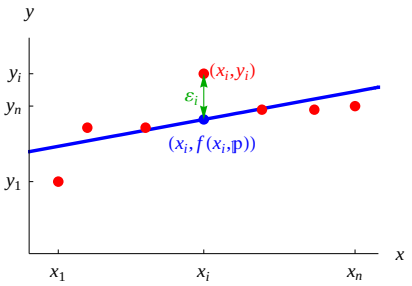


If possible, inspect the final result in a graph like the one above. The eye is namely very good on "looks well" and to see if there are any outliers, large errors at some points. Included, they may have a severe impact on the quality of the model.



Now, The least squares method relies on well known analysis, minimization of a quadratic error function. And of course therefor its naming. See the figure below from left to right. Suppose a collection of data points is given. Form at every data point (x_i, y_i) the error ε_i between the model $f(x_i, \mathbb{p})$ and the measured value y_i, that is $\varepsilon_i = f(x_i, \mathbb{p}) - y_i$. Now it's equally bad if ε_i is positive or negative, this needs to be cured. We can take the absolute value of ε_i, but this one is not differentiable at zero, so the natural choice is the smooth quadratic function, ε_i^2. Finally add them together.

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}$$



$$S = \sum_{i=1}^{n} \varepsilon_i^2 = \varepsilon_1^2 + \varepsilon_2^2 + \ldots + \varepsilon_n^2 = \sum_{i=1}^{n} (f(x_i, \mathbb{p}) - y_i)^2 \tag{3.1}$$

A sum of quadratic terms is always non-negative, that is $S \geqslant 0$, and in the ideal case $S = 0$ the model passes all the data points. As all of the data points are applied the sum will be a function of the model parameters only, $S(\mathbb{p})$. Now the method suggests to minimize it

$$\min_{\mathbb{p}} S(\mathbb{p}) \tag{3.2}$$

We say that the model f is fitted to the data in the sense of the least squares method. From analysis we know that seeking minima to a function is equal to seeking the extreme point where the first order partial derivatives according to $\mathbb{p}$ vanishes. In a general setting $\mathbb{p}$ is usually vector valued and we have to solve the system of equations

$$\min_{\mathbb{p}} S(\mathbb{p}) \quad \Leftrightarrow \quad \frac{\partial S}{\partial \mathbb{p}} = \mathbb{0} \quad \Leftrightarrow \quad \begin{cases} \frac{\partial S}{\partial p_1} = 0 \\ \frac{\partial S}{\partial p_2} = 0 \\ \vdots \end{cases} \tag{3.3}$$

with the same number of equations as number of unknown model parameters $\mathbb{p} = (p_1, p_2, \ldots)^{\mathrm{T}}$. If the system of equations is linear we have a linear least squares method, otherwise nonlinear. A linear model can always be written as a scalar dot product of the intrinsic functions, collected as a vector, and the parameters, that is $f(x, \mathbb{p}) = (f_1, f_2, \ldots)^{\mathrm{T}} \cdot \mathbb{p} = \mathbb{f} \cdot \mathbb{p}$. Typical examples of models ending up as linear models are

$y = kx$	$\Rightarrow \quad \mathbb{f} = (x)^{\mathrm{T}},\ \mathbb{p} = (k)^{\mathrm{T}}$	$y = c_0 + c_1 x + c_2 x^2$	$\Rightarrow \quad \mathbb{f} = (1,\ x,\ x^2)^{\mathrm{T}},\ \mathbb{p} = (c_0,\ c_1,\ c_2)^{\mathrm{T}}$
$y = kx + m$	$\Rightarrow \quad \mathbb{f} = (x,\ 1)^{\mathrm{T}},\ \mathbb{p} = (k,\ m)^{\mathrm{T}}$	$y = ax + b\sin(x)$	$\Rightarrow \quad \mathbb{f} = (x,\ \sin(x))^{\mathrm{T}},\ \mathbb{p} = (a,\ b)^{\mathrm{T}}$

As S is a quadratic function it implies that S has only one local minima, which then also is global, ensures that the coefficient matrix in $\frac{\partial S}{\partial \mathbb{p}} = \mathbb{0}$ is symmetric and positive definite. Thus, a unique solution to our problem of minimization is expected. Safe! In this situation we can rely on an easy applicable function in *Mathematica* called **`Fit[data,f,x]`**. Just feed with the measurement data matrix and the vector $\mathbb{f}$. On output comes directly f in useful expanded form $\mathbb{f} \cdot \mathbb{p}$. Of course the system of equations could equally be solved by **`Solve`**.

Example 3.1: Let us take a look at the canonical example, the straight line $y = f(x, \mathbb{p}) = kx + m$, with the model parameters $\mathbb{p} = (k,\ m)^{\mathrm{T}}$. First

$$S = \sum_{i=1}^{n} (kx_i + m - y_i)^2$$

then partial derivatives, with inner derivatives in fresh memory, takes us eventually to the desired system of linear equations for the unknown parameters $\mathbb{p}$.

$$\min_{\mathbb{p}} S(\mathbb{p}) \quad \Leftrightarrow \quad \frac{\partial S}{\partial \mathbb{p}} = \mathbb{0} \quad \Leftrightarrow \quad \begin{cases} \frac{\partial S}{\partial k} = \sum_{i=1}^{n} 2(kx_i + m - y_i)\, x_i = 0 \\ \frac{\partial S}{\partial m} = \sum_{i=1}^{n} 2(kx_i + m - y_i) = 0 \end{cases} \quad \Leftrightarrow \quad \begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & n \end{pmatrix} \begin{pmatrix} k \\ m \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} y_i \end{pmatrix} \tag{3.4}$$

■

Example 3.2: Let us study a tiny setup of data points $\begin{pmatrix} 0 & 7 & 14 & 22 \\ 0 & 10 & 20 & 30 \end{pmatrix}^T$ and a straight line as model.

Solution: First a straight line passing the origin, that is $y = kx = (x)^T(k)^T = \mathbb{f} \cdot \mathbb{p}$. We find

$$S = \sum_{i=1}^4 (kx_i - y_i)^2 \Rightarrow \frac{\partial S}{\partial k} = \sum_{i=1}^4 2(kx_i - y_i)x_i = 0 \Rightarrow k\sum_{i=1}^4 x_i^2 = \sum_{i=1}^3 x_i y_i$$

where

$$\begin{cases} \sum_{i=1}^4 x_i^2 = 0^2 + 7^2 + 14^2 + 22^2 = 729 \\ \sum_{i=1}^4 x_i y_i = 0\cdot 0 + 7\cdot 10 + 14\cdot 20 + 22\cdot 30 = 1010 \end{cases} \Rightarrow k = \frac{1010}{729} = 1.38546$$

Now let *Mathematica* enter the scene to confirm our boring labor

```
f1 = Fit[{{0, 7, 14, 22}, {0, 10, 20, 30}}^T, {x}, x]
```

1.38546 *x*

Then, what happens if we release the constraint to pass the origin, giving the model $y = kx + m = (x, 1)^T(k, m)^T = \mathbb{f} \cdot \mathbb{p}$? Just plug directly into the system of equations (3.4) derived in **Example 3.1**

$$\begin{pmatrix} \sum_{i=1}^4 x_i^2 & \sum_{i=1}^4 x_i \\ \sum_{i=1}^4 x_i & n \end{pmatrix}\begin{pmatrix} k \\ m \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^4 x_i y_i \\ \sum_{i=1}^4 y_i \end{pmatrix} \Rightarrow \begin{pmatrix} 729 & 43 \\ 43 & 4 \end{pmatrix}\begin{pmatrix} v \\ m \end{pmatrix} = \begin{pmatrix} 1010 \\ 60 \end{pmatrix} \Rightarrow \mathbb{p} = \begin{pmatrix} k \\ m \end{pmatrix} = \frac{1}{1067}\begin{pmatrix} 6350 \\ 950 \end{pmatrix} = \begin{pmatrix} 1.36832 \\ 0.290534 \end{pmatrix}$$

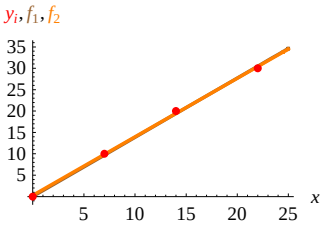
Mathematica agrees

```
f2 = Fit[{{0, 7, 14, 22}, {0, 10, 20, 30}}^T, {x, 1}, x]
```

1.36832 *x* + 0.290534

As told before, an illuminating plot of data and fitted model is strongly urged. In this case is the difference between the models small and mainly of academic art, but emphasize the statement that different models give different results.

```
Plot[Evaluate[{f1, f2}], {x, 0, 25}, PlotStyle → {Brown, Orange}, AxesLabel → {x, "yi, f1, f2"},
  Epilog → {Red, PointSize[0.03], Point[{{0, 7, 14, 22}, {0, 10, 20, 30}}^T]}]
```

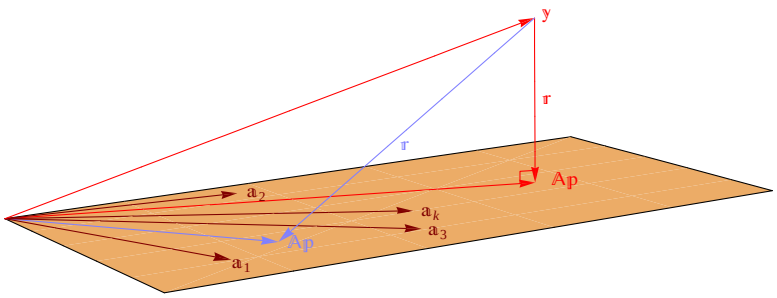


3.3 The normal equations

If we have a linear model we can land in the desired linear system of equations via the overdetermined system of equations, which is generated by applying the model to each data point one at a time. In case of $y = f(x, \mathbb{p}) = kx + m$ the result is

$$\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}\begin{pmatrix} k \\ m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \Leftrightarrow \mathbb{A}\mathbb{p} = \mathbb{y} \tag{3.5}$$

Due to the fact that there are more equations than unknowns, a solution is not to be expected in a general setting. But it is possible to find a solution in the meaning of (LSM). We start by convincing ourselves with a geometrical study.



The column vectors $\mathbb{a}_k$ in $\mathbb{A}$ spans a linear subspace and $\mathbb{A}\mathbb{p}$ is thus a vector in that space with $\mathbb{p}$ as components in the base $\mathbb{a}_k$. As $\mathbb{p}$ is not solving $\mathbb{A}\mathbb{p} = \mathbb{y}$ means obviously that $\mathbb{y}$ is not in the subspace. The residual vector $\mathbb{r} = \mathbb{A}\mathbb{p} - \mathbb{y}$ is then a vector from the end point of $\mathbb{y}$ to the end point of $\mathbb{A}\mathbb{p}$ in the subspace. The least squares method now corresponds to minimizing $|\mathbb{r}|$. Recall that shortest distance from a point to a plane goes perpendicular to the plane. Thus, minimum of $|\mathbb{r}|$ occurs when $\mathbb{r} = \mathbb{r} = \mathbb{A}\mathbb{p} - \mathbb{y}$ is perpendicular to all the basis vectors $\mathbb{a}_k$. Orthogonality implies that the scalar dot product is zero, giving

$$\begin{cases} \mathbb{a}_1 \cdot \mathbb{r} = 0 \\ \mathbb{a}_2 \cdot \mathbb{r} = 0 \\ \quad \vdots \end{cases} \Leftrightarrow \mathbb{A}^T\mathbb{r} = \mathbb{0} \Leftrightarrow \mathbb{A}^T(\mathbb{A}\mathbb{p} - \mathbb{y}) = \mathbb{0} \Leftrightarrow \mathbb{A}^T\mathbb{A}\mathbb{p} = \mathbb{A}^T\mathbb{y}$$

More formally we have $\min_{\mathbb{p}}|\mathbb{r}|^2 = \min_{\mathbb{p}}(\mathbb{r}\cdot\mathbb{r}) \Rightarrow \frac{\partial}{\partial \mathbb{p}}(\mathbb{r}^T\mathbb{r}) = \frac{\partial}{\partial \mathbb{p}}((\mathbb{A}\mathbb{p} - \mathbb{y})^T(\mathbb{A}\mathbb{p} - \mathbb{y})) = \frac{\partial}{\partial \mathbb{p}}((\mathbb{p}^T\mathbb{A}^T - \mathbb{y}^T)(\mathbb{A}\mathbb{p} - \mathbb{y})) =$

$= \frac{\partial}{\partial \mathbb{p}}(\mathbb{p}^T\mathbb{A}^T\mathbb{A}\mathbb{p} - \mathbb{p}^T\mathbb{A}^T\mathbb{y} - \mathbb{y}^T\mathbb{A}\mathbb{p} + \mathbb{y}^T\mathbb{y}) = \frac{\partial}{\partial \mathbb{p}}(\mathbb{p}^T\mathbb{A}^T\mathbb{A}\mathbb{p} - \mathbb{p}^T\mathbb{A}^T\mathbb{y} - \mathbb{p}^T\mathbb{A}^T\mathbb{y} + \mathbb{y}^T\mathbb{y}) = 2\mathbb{A}^T\mathbb{A}\mathbb{p} - 2\mathbb{A}^T\mathbb{y} = \mathbb{0} \Leftrightarrow \mathbb{A}^T\mathbb{A}\mathbb{p} = \mathbb{A}^T\mathbb{y}.$ (3.7)

We call $\mathbb{A}^T\mathbb{A}\mathbb{p} = \mathbb{A}^T\mathbb{y}$ for the normal equations to the overdetermined system $\mathbb{A}\mathbb{p} = \mathbb{y}$ above. The coefficient matrix $\mathbb{A}^T\mathbb{A}$ is by construction always square, symmetric and nonsingular. The solution to the normal equations can be written formally as $\mathbb{p} = (\mathbb{A}^T\mathbb{A})^{-1}\mathbb{A}^T\mathbb{y} = \mathbb{A}^+\mathbb{y}$, where $\mathbb{A}^+ = (\mathbb{A}^T\mathbb{A})^{-1}\mathbb{A}^T$ is called the pseudo inverse of $\mathbb{A}$. Of course is $\mathbb{p}$ not calculated in this costly way with inverses, but in a variant of standard Gaussian elimination method applied on the normal equations. In fact directly on the overdetermined system without a premultiplication with $\mathbb{A}^T$.

If $\mathbb{A}\mathbb{p} = \mathbb{y}$ is **overdetermined** then $\mathbb{A}^T\mathbb{A}\mathbb{p} = \mathbb{A}^T\mathbb{y}$ is the corresponding **normal equations** and has **a unique solution** $\mathbb{p}$ in the sense of the least squares method, that is $\frac{\partial S}{\partial \mathbb{p}} = \mathbb{0} \Leftrightarrow \mathbb{A}^T\mathbb{A}\mathbb{p} = \mathbb{A}^T\mathbb{y}$. (3.8)

Example 3.3: Repeat **Example 3.2** using the theory just scrutinized.

Solution: A quick and dirty hack…with matrix candy along the road.

```
data = {{0, 7, 14, 22}, {0, 10, 20, 30}}^T; A = {#, 1} & /@ data[[All, 1]]; ( A  A^T  A^T.A  A^T.data[[All, 2]] )
```

$$\left(\begin{pmatrix} 0 & 1 \\ 7 & 1 \\ 14 & 1 \\ 22 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 7 & 14 & 22 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 729 & 43 \\ 43 & 4 \end{pmatrix} \quad \{1010, 60\}\right)$$

```
NSolve[A^T.A.{k, m} == A^T.data[[All, 2]]]
```

$\{\{k \to 1.36832, m \to 0.290534\}\}$

Linear models can always be treated by the normal equations. But nonlinear models will directly put us into nasty terrain, and if you have found a local minima to *S* you can always question if it's a global one. In *Mathematica* there is a bunch of extremely powerful optimization functions to our help; **`FindFit`**, **`(N)Minimize`** and **`(N)Maximize`** where the last two ones claim the ability to deliver a global optima. They can also take linear and nonlinear constraints into account. The advantage with **`FindFit`** is that *S* don't need to be formulated explicitly, just feed with the data and the nonlinear model. Above this, the "old" ones **`FindMinimum`** and **`FindMaximum`** are still kicking, but perhaps a little bit left behind by the other.

Example 3.4: The relation between pressure p and volume v in an ideal gas is governed by the law $pv^n = c$, where n and c are constants to be determined.

v	4.60	7.20	10.1	15.3	20.4	30.0
p	14.2	7.59	4.74	2.66	1.78	1.04

Solution: This is a nonlinear model, but can be linearized by taking the logarithm of both sides.

$$pv^n = c \Leftrightarrow \ln(p) + n\ln(v) = \ln(c) \Leftrightarrow \ln(c) \; - n\ln(v) = \ln(p) \Leftrightarrow \begin{pmatrix} 1 & -\ln(v_1) \\ 1 & -\ln(v_2) \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \ln(c) \\ n \end{pmatrix} = \begin{pmatrix} \ln(p_1) \\ \ln(p_2) \\ \vdots \end{pmatrix} \Leftrightarrow \mathbb{A} \begin{pmatrix} \ln(c) \\ n \end{pmatrix} = \mathbb{y}$$

Form the normal equations $\mathbb{A}^T\mathbb{A}\begin{pmatrix} \ln(c) \\ n \end{pmatrix} = \mathbb{A}^T\mathbb{y}$…

```
data = {{4.60, 7.20, 10.1, 15.3, 20.4, 30.0},
        {14.2, 7.59, 4.74, 2.66, 1.78, 1.04}}ᵀ;
𝔸 = {1, -Log[#]} & /@ data[[All, 1]];
( 𝔸  𝔸ᵀ  𝔸ᵀ.𝔸  𝔸ᵀ.Log[data[[All, 2]]] )
```

$$\left(\begin{pmatrix} 1 & -1.52606 \\ 1 & -1.97408 \\ 1 & -2.31254 \\ 1 & -2.72785 \\ 1 & -3.01553 \\ 1 & -3.4012 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -1.52606 & -1.97408 & -2.31254 & -2.72785 & -3.01553 & -3.4012 \end{pmatrix} \begin{pmatrix} 6. & -14.9573 \\ -14.9573 & 39.6764 \end{pmatrix} \{7.83027, -16.1894\} \right)$$

…and solve them.

```
lncAndn = Solve[𝔸ᵀ.𝔸.{lnc, n} == 𝔸ᵀ.Log[data[[All, 2]]]] // First
```

{lnc → 4.77908, n → 1.39359}

The model

```
p = e^lnc / v^n /. lncAndn
```

$$\frac{118.995}{v^{1.39359}}$$

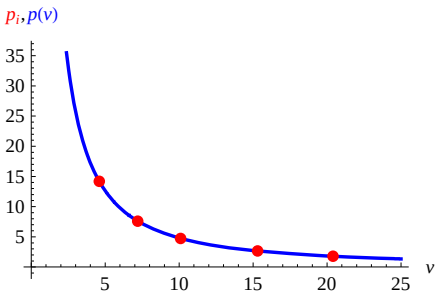
A direct attack on the nonlinear model

```
ℙ = FindFit[data, c / v^n, {c, n}, v]
```

{c → 119.337, n → 1.39505}

Plot…

```
Plot[p, {v, 0, 25}, PlotStyle → Blue, AxesLabel → {"v", "pᵢ,p(v)"},
     Epilog → {Red, PointSize[0.03], Point[data]}]
```



3.4 The continuous least squares method

So far we have treated discrete least squares method, as the number of data points has been finite. Let in some sense the number of data points go infinite, and in the limit we have a continuous least squares method.

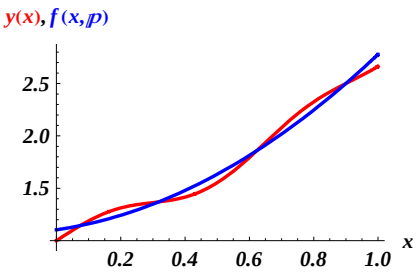
$$\min_{\mathbb{p}} S(\mathbb{p}) = \min_{\mathbb{p}} \sum_{i=1}^{n} (f(x_i, \mathbb{p}) - y_i)^2 \quad \underset{n\to\infty}{\rightarrow} \quad \min_{\mathbb{p}} \int_{\Omega} (f(x, \mathbb{p}) - y(x))^2 dx \tag{3.9}$$

As a poor mans picture, we can interpret the process as minimizing the area enclosed by the two curves, but not exactly as the integrand is squared. A better view is to recall the volume given by the washer formula when one of the curves rotates around the other, except for the π. Continuous least square is handy if we want to approximate a complicated time consuming function with an easier one, for example in a real time system. Also in this context of continuous (LSM) may linear or linear models be considered.

Example 3.5: As an illustration of continuous (LSM) we approximate $y(x) = e^x + 0.1\sin(10x)$ on $\Omega = [0, 1]$ with a simple linear model $f(x, \mathbb{p}) = ax^2 + bx + c = (x^2, x, 1)^T \cdot (a, b, c)^T = \mathbb{f} \cdot \mathbb{p}$, where $\mathbb{f} = (x^2, x, 1)^T$ are the intrinsic functions and $\mathbb{p} = (a, b, c)^T$ the model parameters of interest.

Solution: A quick hack with intermediate output cells suppressed.

```
p = {a, b, c};
f = {x^2, x, 1}.p;
S = ∫_0^1 (f - (e^x + 0.1 Sin[10 x]))^2 dx;
Plot[{e^x + 0.1 Sin[10 x], f /. Solve[D[S, #] == 0 & /@ p] // First}, {x, 0, 1},
     PlotStyle → {Red, Blue}, AxesLabel → {"x", "y(x), f(x,p)"}]
```



Finally, as for interpolation, we may also add weights to the data points in order to raise or lower their importance, if there are outliers for instance. This is weighted least squares method

$$\min_{\mathbb{p}} \sum_{i=1}^{n} \omega_i (f(x_i, \mathbb{p}) - y_i)^2 \quad \text{and} \quad \min_{\mathbb{p}} \int_{\Omega} \omega_i (f(x, \mathbb{p}) - y(x))^2 dx \tag{3.10}$$

3.5 Problems

3.1 Fit $y(x) = ax^2 + bx + c$ to

x	1	2	3.5	5.8
y	1.1	1.8	4	5.5

. **a)** Formulate the normal equations and solve them with **`Solve`**, **b)** Use **`Fit`** and compare the two results, and **c)** Plot the data points together with $y(x)$.

3.2 The same story as **Problem 3.1** but $y(x) = a + bx^2$ and the data

x	5	7.5	12	15	25
y	13.1	18.1	70.2	109	301

.

3.3 The same story as **Problem 3.1** but $y(x) = ax + bx^2$ and the data

x	2	3	5	8
y	1.9	0.1	−11	−39

.

3.4 Fit $y(x) = \frac{x}{ax+b}$ to

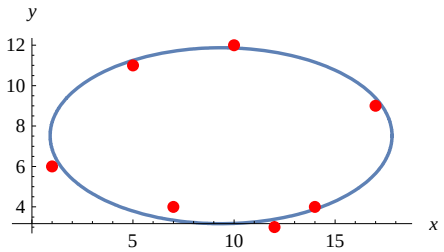
x	1	2.5	4	6	8
y	1	1.5	2	2.5	3

. **a)** Linearize the model, formulate the normal equations and solve them with **`Solve`**, **b)** Use **`FindFit`** and compare the two results, and **c)** Plot the data points together with $y(x)$.

3.5 Use **`FindFit`** to fit an ellipsis, with axes parallel to the coordinate axes, to the points

x	1	7	10	17	5	12	14
y	6	4	12	9	11	3	4

. That is to find the parameters x_0, y_0, a, and b in the model $\left(\frac{x-x_0}{a}\right)^2 + \left(\frac{y-y_0}{b}\right)^2 = 1$. This is a bit of a nasty nonlinear problem, so you probably need to give a helping hand with initial guesses of the parameters. Present your findings together with the given points in the same graph.



4. Numerical differentiation

4.1 Basic concepts

This chapter deals with numerical approximations of derivatives. The first question that springs to mind is: why do we need to approximate derivatives at all? After all, we do know how to analytically differentiate every function. Nevertheless, there are several reasons to why we still need to approximate derivatives:

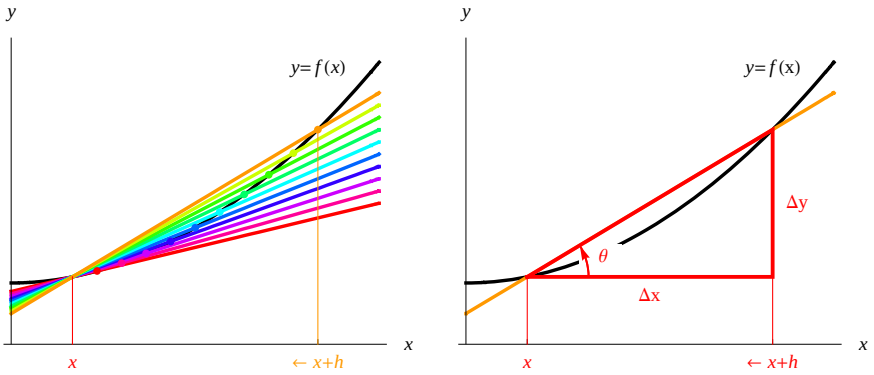
• Even if there exists an underlying function that we need to differentiate, we might know its values only at a sampled data set without knowing the function itself.

• There are some cases where it may not be obvious that an underlying function exists and all that we have is a discrete data set. We may still be interested in studying changes in the data, which are related, of course, to derivatives.

• There are occasions at which exact formulas are available but they are very complicated to the point that an computation of the exact derivative requires a lot of function evaluations, and thus becomes costly in terms of time and/or data size. It might be significantly simpler to approximate the derivative instead of computing its exact value.

• When approximating solutions to ordinary (or partial) differential equations, we typically represent the solution as a discrete approximation that is defined on a grid. Since we then have to evaluate derivatives at the grid points, we need to be able to come up with methods for approximating the derivatives at these points, and again, this will typically be done using only values that are defined at the point in question and its neighbors. The underlying function itself (which in this case is the solution of the differential equation) is unknown.

Recall the definition of the derivative, where a secant line goes down to a tangent line, which is the geometric interpretation of the derivative. As h approaches zero, the slope of the secant line approaches the slope of the tangent line. Therefore, the true derivative of f at x is the limit of the value of the difference quotient as the secant lines get closer and closer to being a tangent line.



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

$$k_T = \tan(\theta) = f'(x)$$

Thus, a simple approximation of the first derivative is

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{4.1}$$

where we assume that $h > 0$. What do we mean when we say that the expression on the right-hand-side of (4.1) is an approximation of the derivative? For linear functions (4.1) is actually an exact expression for the derivative. But for almost all other functions, (4.1) is not the exact derivative.

Let's compute the approximation error. The Taylor expansion will be the main vehicle throughout this chapter. So, write a Taylor expansion of $f(x + h)$ about x, i.e.

$$f(x+h) = f(x) + h\, f'(x) + \frac{h^2}{2} f''(\xi), \quad \xi \in (x,\, x+h) \tag{4.2}$$

For such an expansion to be valid, we assume that $f(x)$ has two continuous derivatives. Re-arranging the Taylor expansion (4.2) means that we can replace the approximation (4.1) with an exact formula of the form

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(\xi), \quad \xi \in (x,\, x+h) \tag{4.3}$$

Since this approximation of the derivative at x is based on the values of the function at x and $x + h$, the approximation (4.1) is called a forward differencing or one-sided differencing. The approximation of the derivative at x that is based on the values of the function at $x - h$ and x, i.e.

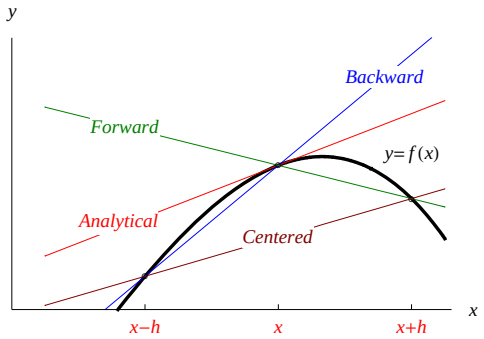
$$f'(x) \approx \frac{f(x) - f(x-h)}{h} \tag{4.4}$$

is called a backward differencing. Both (4.1) and (4.4) are, for obvious reasons, called one-sided differencing formulas.

It's possible to write more accurate formulas than (4.1) or (4.4). For example, a more accurate approximation for the first derivative that is based on the values of the function at the points $x - h$ and $x + h$ is the centered differencing formula

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{4.5}$$

Here they are, all together in the same figure. Apparently, the centered secant is "more parallel" to the analytical tangent than the forward and backward secants are.



Let's verify that (4.5) is indeed a more accurate formula than the one-sided ones in (4.1) and (4.4), which obviously are $O(h)$ according to (4.3). Taylor expansions in higher terms of the quantities on the right hand side of (4.5) gives

$$f(x+h) = f(x) + h\,f'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1), \quad \xi_1 \in (x,\, x+h)$$

$$f(x-h) = f(x) - h\,f'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi_2), \quad \xi_2 \in (x-h,\, x)$$

and the desired result of $O(h^2)$ after subtraction

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}[f'''(\xi_1) + f'''(\xi_2)]$$

In a similar way we can approximate the values of higher order derivatives. For example, using the Taylor expansions

$$f(x \pm h) = f(x) \pm h\,f'(x) + \frac{h^2}{2}f''(x) \pm \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_\pm)$$

it's easy to verify that the following is a second order approximation of the second derivative

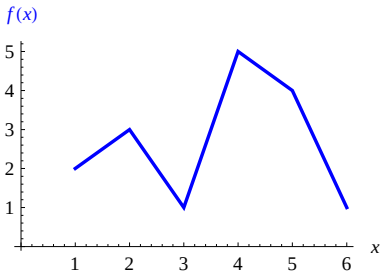
$$f''(x) \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \tag{4.6}$$

Example 4.1: Estimate $f'(4)$ and $f''(4)$, where $f(x)$ is given by the table

x	1	2	3	4	5	6
$f(x)$	2	3	1	5	4	1

Solution: First a figure of the situation

```
ListPlot[{{1,2,3,4,5,6},{2,3,1,5,4,1}}ᵀ, Joined → True, PlotStyle → Blue, AxesLabel → {"x", "f(x)"}]
```



Clearly $h = 1$. Using the formulas for forward (4.1), backward (4.4), centered (4.5), and (4.6) for second derivative above, we have

```
{f_f'[4] → (4 - 5)/1, f_b'[4] → (5 - 1)/1, f_c'[4] → (4 - 1)/(2 × 1), f''[4] → (1 - 2 × 5 + 4)/1²}
```

$$\left\{f_f'(4) \to -1,\ f_b'(4) \to 4,\ f_c'(4) \to \frac{3}{2},\ f''(4) \to -5\right\}$$

■

4.2 Differentiation via interpolation

In this section we demonstrate how to generate differentiation formulas by differentiating an interpolant. The idea is straightforward, the first stage is to construct an interpolating polynomial from the data. An approximation of the derivative at any point can then be obtained by a direct differentiation of the interpolant. We follow this recipe and assume that the data set $f(x_0)$, $f(x_1)$, …, $f(x_n)$ is given. The Lagrange interpolation polynomial of order n through these points is a linear combination of the interpolation points and a set of basis functions

$$L_n(x) = \sum_{k=0}^{n} f(x_k)\, l_k^n(x) \tag{4.7}$$

where the basis functions $l_k^n(x)$ are given by

$$l_k^n(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)},\ k = 0,\ 1,\ \ldots,\ n \tag{4.8}$$

By this construction the interpolation polynomial satisfies the interpolation conditions $L_n(x_i) = f(x_i)$, and the basis functions $l_k^n(x_i) = \delta_{ik}$, where δ_{ik} is the Kronecker delta. It's now straightforward to find a derivative of desired order, provided that n, the number of interpolation points, is sufficiently large. Note that $L_2'(x_1)$ reproduces the centered difference formula (4.5).

A more direct approach is to solve a linear system of equations for the unknown coefficients c_i in the interpolating polynomial $p_n(x) = c_0 + c_1 x + \ldots + c_n x^n$. That is

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ & & \vdots & & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} \tag{4.9}$$

For large n the coefficient matrix, the Vandermonde matrix, will become ill-conditioned as the determinant numerically approaches zero. However, interpolation is best treated as a local affair and the support for the interpolation polynomial is therefore taken at nearby points to the one under consideration. Large n are rare, typically $n = 2$ is often a decent choice giving $O(h^2)$ results. A nice feature is that local interpolation with a polynomial is not sensitive to equidistant distance h between the supporting grid points, this is in marked contrast to the difference schemes presented above.

Example 4.2: Estimate $f'(4)$ and $f''(4)$, where $f(x)$ is given by the table

x	1	2	3	4	5	6
$f(x)$	2	3	1	5	4	1

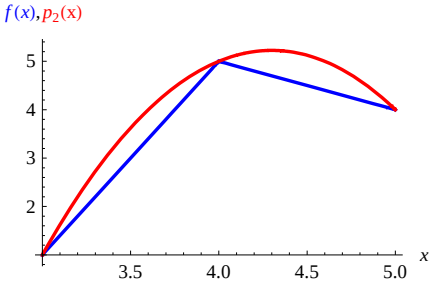
.

Solution: Second order polynomial interpolation using formula (4.9) above, centered around $x = 4$, giving the coefficients

```
{Solve[{{1, 3, 3^2}, {1, 4, 4^2}, {1, 5, 5^2}} . {c0, c1, c2} == {1, 5, 4}], p2[x_] = Fit[{{3, 4, 5}, {1, 5, 4}}^T, {1, x, x^2}, x]}
```

$$\left\{\left\{\left\{c_0 \to -41,\ c_1 \to \frac{43}{2},\ c_2 \to -\frac{5}{2}\right\}\right\},\ -2.5\,x^2 + 21.5\,x - 41.\right\}$$

```
Plot[{Interpolation[{{3, 4, 5}, {1, 5, 4}}^T, InterpolationOrder → 1][x], p2[x]}, {x, 3, 5},
  PlotStyle → {Blue, Red}, AxesLabel → {"x", "f(x), p2(x)"}]
```



Finally calculate, and compare with the findings in **Example 4.1.** We note especially that the second order polynomial interpolation is in full agreement with the centered differencing formulas (4.5) and (4.6).

```
{p2'[4], p2''[4]}
```

{1.5, −5.}

■

4.3 Extrapolation

Extrapolation can be viewed as a general procedure for improving the accuracy of approximations when the structure of the error is known. While we study it here in the context of numerical differentiation, it's by no means limited only to that, and we will come back to it later on when we study methods for numerical integration. We start with an example in which we show how to turn a second order approximation of the first derivative into a fourth order approximation of the same quantity. We already know that we can write a second order approximation of $f'(x)$ in term of the values $f(x \pm h)$, see (4.5). In order to improve this approximation we will need some more insight into the internal structure of the error. Let $D_k(h)$ be the differentiation with error term of order k, thus

```
D2[h_] := (f[x + h] - f[x - h])/(2 h)
```

and start with the Taylor expansions for h and $2h$.

```
exp = Series[{D2[h], D2[2 h]}, {h, 0, 5}]
```

$$\left\{f'(x)+\frac{1}{6}f^{(3)}(x)\,h^2+\frac{1}{120}f^{(5)}(x)\,h^4+O\left(h^6\right),\ f'(x)+\frac{2}{3}f^{(3)}(x)\,h^2+\frac{2}{15}f^{(5)}(x)\,h^4+O\left(h^6\right)\right\}$$

By inspection we recognize that the leading error term h^2 can be eliminated between these expansions

```
Solve[Normal[{D2[h], D2[2 h]} == exp], f'[x], f'''[x]] // ExpandAll
```

$$\left\{\left\{f'(x)\to\frac{1}{30}f^{(5)}(x)\,h^4+\frac{4\,\text{D2}(h)}{3}-\frac{1}{3}\text{D2}(2\,h)\right\}\right\}$$

Thus giving us a $O\left(h^4\right)$ interpolation

$$D_4(h) = \tfrac{1}{3}(4D_2(h) - D_2(2h)) + O\left(h^4\right) \tag{4.10}$$

or expanded

```
D4[h_] = 1/3 (4 D2[h] - D2[2 h]) // Simplify
```

$$\frac{f(x-2\,h)-8\,f(x-h)+8\,f(h+x)-f(2\,h+x)}{12\,h}$$

It's straight forward to continue with an $O\left(h^6\right)$ interpolation, by eliminating the leading error term h^4.

```
Solve[Normal[{D4[h], D4[2 h]} == Series[{D4[h], D4[2 h]}, {h, 0, 6}]],
      f'[x], f'''''[x]] // ExpandAll
```

$$\left\{\left\{f'(x)\to-\frac{4}{315}f^{(7)}(x)\,h^6+\frac{16\,\text{D4}(h)}{15}-\frac{1}{15}\text{D4}(2\,h)\right\}\right\}$$

Thus giving us an $O\left(h^6\right)$ interpolation

$$D_6(h) = \tfrac{1}{15}(16D_4(h) - D_4(2h)) + O\left(h^6\right) \tag{4.11}$$

or expanded

```
1/15 (16 D4[h] - D4[2 h]) // Simplify
```

$$\frac{1}{360\,h}\left(-f(x-4\,h)+40\,f(x-2\,h)-256\,f(x-h)+256\,f(h+x)-40\,f(2\,h+x)+f(4\,h+x)\right)$$

This process can continue and is usually known as Richardson extrapolation, (Lewis Richardson, 1881-1953, English mathematician). One can fairly easy derive the recurrence formula

$$D_k(h) = \tfrac{1}{2^{k-2}-1}\left(2^{k-2}D_{k-2}(h) - D_{k-2}(2h)\right) + O\left(h^k\right),\ \ k = 4,\ 6,\ 8,\ \ldots \tag{4.12}$$

Example 4.3: Estimate $f'(0.5)$, where $f(x)$ is given by the table

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$f(x)$	0.0998	0.1987	0.2955	0.3894	0.4794	0.5646	0.6442	0.7174	0.7833

Solution: Clearly $h = 0.1$. Using the formulas above, we have the forward, backward, centered (=D_2) and extrapolated D_4.

```
{(0.5646 - 0.4794)/0.1, (0.4794 - 0.3894)/0.1, (0.5646 - 0.3894)/(2 × 0.1), 1/3 (4 (0.5646 - 0.3894)/(2 × 0.1) - (0.6442 - 0.2955)/(2 × 0.2))}
```

{0.9, 0.852, 0.876, 0.877417}

Then second order polynomial interpolation

```
p2[x_] = Fit[{{0.4, 0.5, 0.6}, {0.3894, 0.4794, 0.5646}}ᵀ, {1, x, x²}, x]
```

$-0.24\,x^2 + 1.116\,x - 0.0186$

and the same result as for D_2 follows. The tabulated data is actually sin(x), so we can eventually compare all the estimates with the exact one.

```
{p2'[0.5], Cos[0.5]}
```

{0.876, 0.877583}

4.4 Problems

4.1 Consider **Example 4.1** once again, but this time for $f'(3)$ and $f''(3)$. $\{f_f'(3) \to 4,\ f_b'(3) \to -2,\ f_c'(3) \to 1,\ f''(3) \to 6\}$

4.2 Estimate $f'(1.2)$, using backward, forward, and centered with $h = 0.1$ and $h = 0.2$ where $f(x)$ is given in the table.

x	1.0	1.1	1.2	1.3	1.4
$f(x)$	1.000	1.008	1.061	1.192	1.414

Use extrapolation to obtain $D_4(h)$. $\{h = 0.1 : f_f'(1.2) \to 1.31,\ f_b'(1.2) \to 0.53,\ f_c'(1.2) \to 0.92\}$,
$\{h = 0.2 : f_f'(1.2) \to 1.765,\ f_b'(1.2) \to 0.305,\ f_c'(1.2) \to 1.035\}$, $\{D_4 \to 0.881667\}$

4.3 Consider $f(x) = xe^x$, tabulated below

x	0.96	0.97	0.98	0.99	1.00	1.01	1.02	1.03	1.04
$f(x)$	2.5072	2.5588	2.6112	2.6643	2.7183	2.7731	2.8287	2.8851	2.9424

a) Find exactly $f'(1.0)$ and $f''(1.0)$. $\{f'(1.0) \to 5.43656,\ f''(1.0) \to 8.15485\}$
b) Use the tabulated values together with (4.5), (4.6) and $h = 0.01$ to estimate $f'(1.0)$ and $f''(1.0)$. Compare with **a)**.
$\{f'(1.0) \to 5.44,\ f''(1.0) \to 8.0\}$

4.4 Consider **Example 4.2** once again, but this time for $f'(3)$ and $f''(3)$ and compare your findings with **Problem 4.1**.
$\{c_0 \to 25,\ c_1 \to -17,\ c_2 \to 3,\ p_2'(3) \to 1,\ p_2''(3) \to 6\}$

4.5 Consider $f(x) = \sin(x)$, using the following non-equidistant grid data.

x	1.4	1.5	1.7
$f(x)$	0.98545	0.99749	0.99167

Estimate $f'\left(\frac{\pi}{2}\right)$, using $p_2(x)$ interpolation. Implement (4.9) and verify with **Fit**. Compare your answer with $f'\left(\frac{\pi}{2}\right) = \cos\left(\frac{\pi}{2}\right) = 0$.
$\left\{p_2(x) \to -0.498333\,x^2 + 1.56557\,x - 0.22961,\ p_2'\left(\frac{\pi}{2}\right) \to 6.32763 \times 10^{-6}\right\}$

4.6 Calculate $f'(x)$ for $f(x) = \sin(x)$, $x = 0, 0.1, 0.2, \ldots 1.0$ and $h = 0.001$, using (1.5). Compare with $f'(x) = \cos(x)$.

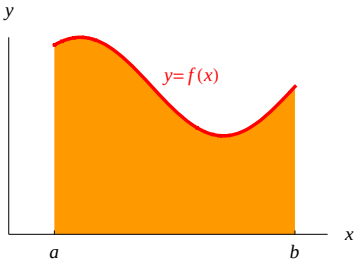
```
{{0    0.1       0.2       0.3       0.4       0.5       0.6       0.7       0.8       0.9       1.      },
 {1.   0.995004  0.980066  0.955336  0.921061  0.877582  0.825335  0.764842  0.696707  0.62161   0.540302},
 {1    0.995004  0.980067  0.955336  0.921061  0.877583  0.825336  0.764842  0.696707  0.62161   0.540302}}
```

4.7 Consider $f(x) = \sin(x)$. Estimate $f'\left(\frac{\pi}{2}\right)$, using backward, forward, and centered for $h = 10^{-k}$, $k = 1, 2, 3, \ldots$ until the computations break down on your computer. Compare with the correct one $f'\left(\frac{\pi}{2}\right) = \cos\left(\frac{\pi}{2}\right) = 0$ during the journey.

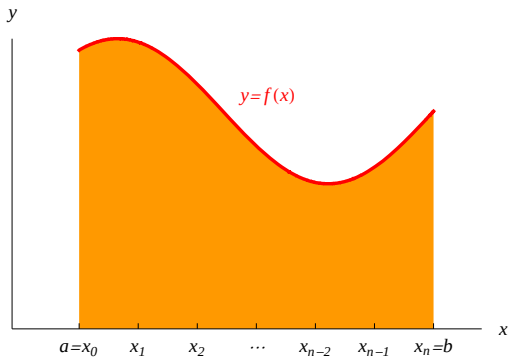
5. Numerical integration

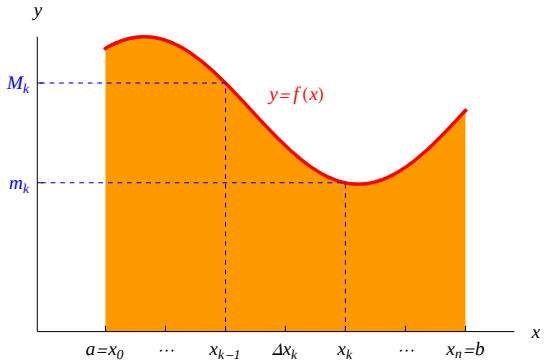
5.1 Basic concepts

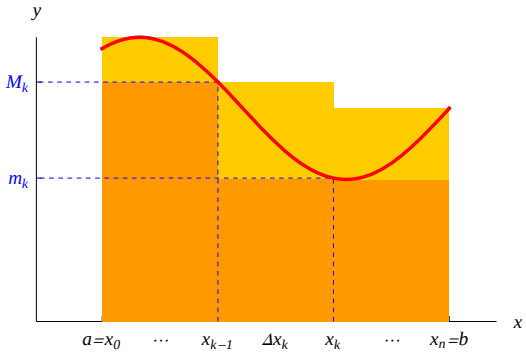
Recall that the original reason for introducing the definite integral was to measure the area A of a plane figure. Assume that this figure is enclosed by the curve $y = f(x)$, the x–axis and the lines $x = a$ and $x = b$, as indicated in the figure.



Suppose $f(x)$ is continuous in the interval J and that a and b are the end points of a compact set, $[a, b] \subset J$. The idea is now to slice $[a, b]$ into smaller pieces, a so-called partition, $a = x_0 < x_1 < \ldots < x_n = b$. In-between two subsequent points we have the interval $\Delta x_k = [x_{k-1}, x_k]$. Let $m_k = \min_{x \in \Delta x_k} f(x)$ and $M_k = \max_{x \in \Delta x_k} f(x)$ be the extreme values of f on the interval Δx_k.







It's now possible to define the lower sum L and upper sum U for A_n respresenting the area A for this partition. They are related to each other as

$$L = \sum_{k=1}^{n} L_k = \sum_{k=1}^{n} m_k |\Delta x_k| \le A_n \le \sum_{k=1}^{n} M_k |\Delta x_k| = \sum_{k=1}^{n} U_k = U$$

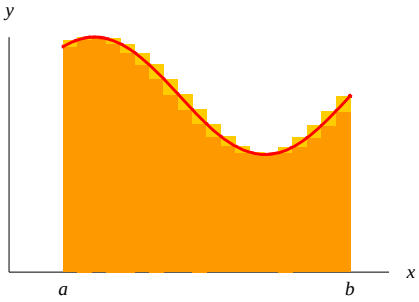
If we refine the partition in such a way that

$$\left.\begin{array}{l} n \to \infty \\ \max |\Delta x_k| \to 0 \end{array}\right\} \Rightarrow \quad |U - L| \to 0,$$

and by the limit process

$$A_n \to A = L = U$$

we say that the area A is measurable and that f is integrable according to Riemann (Bernhard Riemann, 1826 – 1866, German mathematician).



As f is continuous we have points $\xi_k \in \Delta x_k$ and deduce from the intermediate value theorem and the limit process above, that we may deduce the following limit in terms of the definite integral

$$\sum_{k=1}^{n} f(\xi_k) |\Delta x_k| \xrightarrow[\max |\Delta x_k| \to 0]{n \to \infty} I = \int_a^b f(x)\, dx$$

The sum in finite form, is called a Riemann sum. We can clearly see the heritage of the different parts, $f(\xi_k) \to f(x)$ the integrand, and $|\Delta x_k| \to dx$ the measure covering, in small pieces, the interval $[a, b]$, where a and b are called the lower and upper limit, respectively, as $n = 1, 2, \ldots$. In the light of these facts, it's not surprising that the Riemann sum is the starting point for numerical integration, that is the numeric evaluation of definite integrals.

We know that if f is such that we can find a differentiable function F whose derivative is f, then we can evaluate I by applying the familiar formula, where F is a primitive function to f.

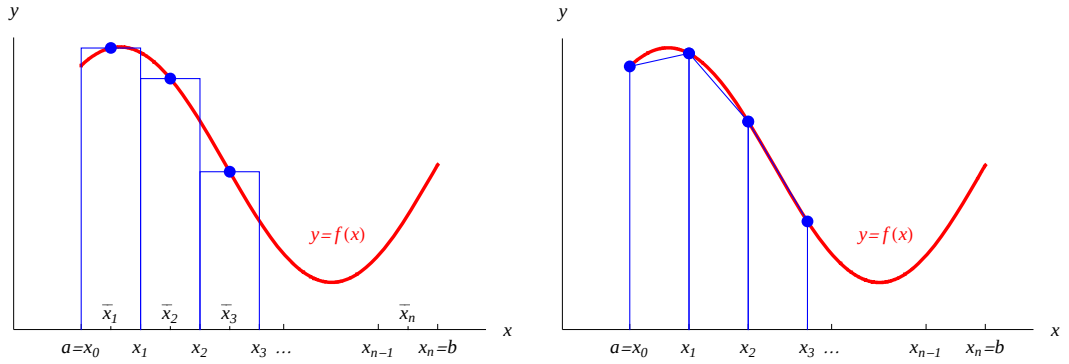
$$I = \int_a^b f(x)\,dx = F(b) - F(a)$$

Tables of integrals or a CAS (*Mathematica*, Maple, etc.) may be helpful for this purpose. However, we often have the same situation as stated in the numerical differentiation chapter. Applications often lead to integrals whose analytic evaluation would be very time consuming, difficult or even impossible (not that rare), or whose integrand is an empirical function given by recorded numeric values. Often is also the analytical value unnecessary good for the application at hand! Then we may obtain approximate numeric values of the integral by a numerical integration method. Below we will present some of the most widely used.

5.2 Rectangular rule and Trapezoidal rule

Numeric integration methods are obtained by approximating the integrand f by functions that can easily be integrated. The simplest formula, the rectangular rule or midpoint rule, is obtained if we subdivide the interval of integration $a \le x \le b$ into n subintervals of equal length $h = \frac{b-a}{n}$, giving a so called partition, $a = x_0 < x_1 < \ldots < x_n = b$, and in each subinterval approximate f by the constant $f(\bar{x}_i)$, the value of f at the midpoint $\bar{x}_i$ of the i^{th} subinterval $[x_{i-1}, x_i]$. The n rectangles in the figure to the left have areas $f(\bar{x}_1)h$, $f(\bar{x}_2)h$, …, $f(\bar{x}_n)h$, and the rectangular rule follows

$$R(h) = \int_a^b f(x)\,dx \approx h(f(\bar{x}_1) + f(\bar{x}_2) + \ldots + f(\bar{x}_n)) \tag{5.1}$$



The trapezoidal rule is generally more accurate. It's easily obtained by taking the same subdivision as before and approximate f by a piecewise linear function, of line segments with endpoints $(x_0, f(x_0))$, $(x_1, f(x_1))$, …, $(x_n, f(x_n))$ on the curve of f, see figure to the right above. Then the area under the curve of f between a and b is approximated by n trapezoids of areas $\frac{1}{2}h(f(x_0) + f(x_1))$, $\frac{1}{2}h(f(x_1) + f(x_2))$, …, $\frac{1}{2}h(f(x_{n-1}) + f(x_n))$. By taking their sum we obtain the trapezoidal rule

$$T(h) = \int_a^b f(x)\,dx \approx h\left(\tfrac{1}{2}f(x_0) + f(x_1) + f(x_2) + \ldots + f(x_{n-1}) + \tfrac{1}{2}f(x_n)\right) \tag{5.2}$$

When it comes to error analysis we simply state that the trapezoidal rule is $O(h^2)$. Thus the story for extrapolation follows the one for centered differentiation.

$$T_4(h) = \tfrac{1}{3}(4T(h) - T(2h)) \tag{5.3}$$

with the error bound

$$\varepsilon_h = \tfrac{1}{3}(T(h) - T(2h)) \tag{5.4}$$

Taking into account the effect of interval halving on h, we can reduce the amount of calculation of $T(h)$ by making use of the result obtained for $T(2h)$.

$$\begin{aligned} T(h) &= h\left(f(x_1) + f(x_2) + f(x_3) + \ldots + f(x_{n-1}) + \tfrac{1}{2}(f(x_0) + f(x_n))\right) \\ T(2h) &= 2h\left(f(x_2) + f(x_4) + f(x_6) + \ldots + f(x_{n-2}) + \tfrac{1}{2}(f(x_0) + f(x_n))\right) \end{aligned}$$

Here $f(x_0)$, $f(x_2)$, $f(x_4)$, $f(x_6)$, …, $f(x_n)$ are all calculated in the evaluation of $T(2h)$. Rearranging

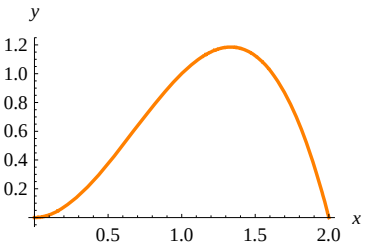
$$\begin{aligned} T(h) &= h(f(x_1) + f(x_3) + f(x_5) + \ldots + f(x_{n-1})) \\ &\quad + \tfrac{1}{2}\times 2\left(f(x_2) + f(x_4) + f(x_6) + \ldots + f(x_{n-2}) + \tfrac{1}{2}(f(x_0) + f(x_n))\right) \\ &= h(f(x_1) + f(x_3) + f(x_5) + \ldots + f(x_{n-1})) + \tfrac{1}{2}T(2h) \end{aligned}$$

Optimized computer implementation should take this time saving into consideration.

Example 5.1: Estimate $\int_a^b 2x^2 - x^3 dx$, with $a = 0$, $b = 2$ using $R(h)$ and $T(h)$ with $n = 5$, 10, and 20. Compare extrapolation for the first two, with the last one and the exact one.

Solution: First the integrand, and a plot.

```
f[x_] := 2 x^2 - x^3
Plot[f[x], {x, 0, 2}, PlotStyle → Orange, AxesLabel → {x, y}]
```



Now, $R(h)$ and $T(h)$ for the partitions requested, using formulas (5.1) and (5.2) above

```
b = 2.0
a = 0.0
n = 5
h = (b - a)/n;
Do[x_i = a + i h, {i, 0, n}];
{h Sum[f[(x_(i-1) + x_i)/2], {i, 1, n}], h/2 Sum[(f[x_(i-1)] + f[x_i]), {i, 1, n}]}
```

{1.36, 1.28}

```
b = 2.0
a = 0.0
n = 10
h = (b - a)/n;
Do[x_i = a + i h, {i, 0, n}];
{h Sum[f[(x_(i-1) + x_i)/2], {i, 1, n}], h/2 Sum[(f[x_(i-1)] + f[x_i]), {i, 1, n}]}
```

{1.34, 1.32}

```
b = 2.0
a = 0.0
n = 20
h = (b - a)/n;
Do[x_i = a + i h, {i, 0, n}];
{h Sum[f[(x_(i-1) + x_i)/2], {i, 1, n}], h/2 Sum[(f[x_(i-1)] + f[x_i]), {i, 1, n}]}
```

{1.335, 1.33}

Finally extrapolation, using formula (5.3) and the trapezoidal values for $n = 5 : T(2h)$ and $n = 10 : T(h)$. Compare.

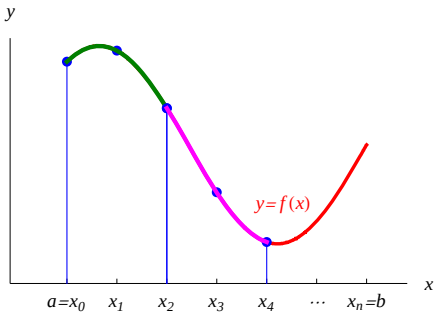
```
{1/3 (4 × 1.32 - 1.28), ∫_a^b f[x] dx}
```

{1.33333, 1.33333}

5.3 Simpson's rule

Piecewise constant approximation of f led to the rectangular rule (5.1), piecewise linear approximation to the trapezoidal rule (5.2), and piecewise quadratic approximation will lead to Simpson's rule (Thomas Simpson, 1710-1761, English mathematician), which is of great practical importance because it is sufficiently accurate for most problems, but still rather simple. To derive Simpson's rule,

we divide the interval of integration $a \le x \le b$ into an even number of equal subintervals, say, into $n = 2m$ subintervals of length $h = \frac{b-a}{2m}$, with the partition $a = x_0, x_1, \ldots, x_{2m-1}, x_{2m} = b$, see fig below. We now take the intervals two by two and approximate $f(x)$ over them by a second order interpolation polynomial $p_2(x)$ with support at the grid points, as in chapter 4.2, formula (4.9).



We exemplify with the first Simpson interval. Start by finding the coefficients in $p_2(x) = c_0 + c_1 x + c_2 x^2$ when it passes through the three points $(x_0,\ f(x_0))$, $(x_1,\ f(x_1))$ and $(x_2,\ f(x_2))$. Hand over the hard labor to *Mathematica*.

```
c012 = Solve[{{1, x0, x0^2}, {1, x0 + h, (x0 + h)^2}, {1, x0 + 2 h, (x0 + 2 h)^2}} . {c0, c1, c2} == {f[x0], f[x1], f[x2]}, {c0, c1, c2}]
```

$$\left\{\left\{c_0 \to -\frac{1}{2h^2}\left(-2 f(x_0) h^2 - 3 f(x_0) x_0 h + 4 f(x_1) x_0 h - f(x_2) x_0 h - f(x_0) x_0^2 + 2 f(x_1) x_0^2 - f(x_2) x_0^2\right),\right.\right.$$
$$\left.\left. c_1 \to -\frac{1}{2h^2}\left(3 h f(x_0) + 2 x_0 f(x_0) - 4 h f(x_1) + h f(x_2) - 4 f(x_1) x_0 + 2 f(x_2) x_0\right),\ c_2 \to -\frac{-f(x_0) + 2 f(x_1) - f(x_2)}{2h^2}\right\}\right\}$$

Giving the integral over the first Simpson interval

```
Integrate[(c0 + c1 s + c2 s^2 /. c012), {s, x0, x0 + 2 h}]
```

$$\left\{\frac{1}{3} h f(x_0) + \frac{4}{3} h f(x_1) + \frac{1}{3} h f(x_2)\right\}$$

Summing up over the whole domain under consideration, provides us with the Simpson's rule

$$S(h) = \int_a^b f(x)\,dx \approx \frac{h}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 2f(x_{2m-2}) + 4f(x_{2m-1}) + f(x_{2m})\right) \tag{5.5}$$

The relation between Simpson's and the Trapezoidal rule is, see (5.3)

$$S(h) = T_4(h)$$

The following extrapolation and error bound holds

$$S_4(h) = \frac{1}{16}(15S(h) - S(2h)) \tag{5.6}$$

$$\varepsilon_h = \frac{1}{15}(S(h) - S(2h)) \tag{5.7}$$

Example 5.2: Estimate $\int_0^2 2x^2 - x^3 dx$, using $S(h)$ with $n = 2m = 4$. Compare with the exact one.

Solution: Same integral as in **Example 5.1**, the exact value and $S(h)$ in very good agreement.

```
{Integrate[f[x], {x, 0, 2}], (2 - 0) / 4 / 3 (f[0] + 4 f[1/2] + 2 f[1] + 4 f[3/2] + f[2])}
```

$$\left\{\frac{4}{3}, \frac{4}{3}\right\}$$

■

5.4 Gaussian quadrature

Integrals of the form $\int_{-1}^{1} f(\xi)d\xi$, where f is a polynomial are very common for a large number of mathematical modeling situations. We have already seen Simpson's method using piecewise polynomial of degree two. The popularity and success of using polynomials is mostly due to the fact that they are the simplest functions we have in mathematics. To evaluate them we only need multiplication and addition. Moreover, addition, subtraction, multiplication, derivation, and integration results in a new polynomial. Finally, due to a theorem of Karl Weierstrass (1815-1897, German mathematician), any continuous function on a closed and bounded interval can by uniformly approximated on that interval by polynomials to any degree of accuracy. Thus, it's not surprising that a

nice integration scheme has been developed based on these facts. Consequently, recall the definite integral as the limit of a Riemann sum, and introduce a numerical integration scheme where we simply approximate the integral with a sum. This integration is often called quadrature in one dimension and cubature in higher dimensions. It's usually referred to as Gaussian quadrature, in recognition of the famous German mathematician Carl Friedrich Gauss (1777-1855). The recipe is simple

$$\int_{-1}^{1} g(\xi)d\xi = \sum_{i=1}^{n} w_i g(\xi_i) + R \tag{5.8}$$

and defines the Gauss quadrature of order n or n-point Gauss quadrature. The Gauss points, or sampling points, $\xi_i \in [-1, 1]$, are symmetrically located around zero, and so cleverly chosen that any polynomial of order up to $p = 2n - 1$ is integrated exactly. The w_i are the corresponding weights, and R an error term without any practical meaning. The points and weights are tabulated in almost every textbook in numerical analysis. Below is such a table with data for the first three Gauss quadrature rules. By using as low order as possible, is called optimal order.

n	p	i	ξ_i	w_i
1	1	1	0	2
2	3	1	+0.57735027	1
		2	−0.57735027	1
3	5	1	0	0.88888889
		2	+0.77459667	0.55555556
		3	−0.77459667	0.55555556

(5.9)

Gauss used Legendre polynomials to derive the table under the properties prescribed, using the fact that all Legendre polynomials are pairwise orthogonal, that is $\int_{-1}^{1} P_i(\xi)P_j(\xi)\,d\xi = 0$. It can be verified that the n Gauss points are the roots of the Legendre polynomial $P_n(\xi)$ of order n, and that the weights are given by $w_i = 2\left(P_n'(\xi_i)^2\left(1-\xi_i^2\right)\right)^{-1}$. In *Mathematica*, it is easy to derive Gauss quadrature data of any desired order. For example order $n = 3$ in table (5.9).

```
ξi = NSolve[LegendreP[3, ξ] == 0]
```

{{ξ → 0.774597}, {ξ → −0.774597}, {ξ → 0.}}

```
wi = 2 (D[LegendreP[3, ξ], ξ]^2 (1 - ξ^2))^-1 /. ξi
```

{0.555556, 0.555556, 0.888889}

The tricky part is to transform $\int_a^b f(x)dx$ to $\int_{-1}^{1} g(\xi)d\xi$ in (5.8). The variable transformation reads $x = \frac{a+b}{2} + \xi\frac{b-a}{2}$, $\xi \in [-1, 1]$, giving the Jacobian $\mathbb{J} = \frac{dx}{d\xi} = \frac{b-a}{2}$. Finally $\int_a^b f(x)dx = \int_{-1}^{1} f\left(\frac{a+b}{2} + \xi\frac{b-a}{2}\right)\mathbb{J}\,d\xi = \int_{-1}^{1} g(\xi)d\xi$. Thus $g(\xi) = f\left(\frac{a+b}{2} + \xi\frac{b-a}{2}\right)\frac{b-a}{2}$.

Example 5.3: Estimate $\int_0^2 2x^2 - x^3 dx$, using optimal Gauss quadrature. Compare with the exact one.

Solution: Same integral as before in **Example 5.1**. The integrand is of order 3, so optimal order, table (5.9), is a two-point Gauss integrating a polynomial of order $2\times 2 - 1 = 3$ exactly. First variable substitution from $x \in [0, 2]$ to the standard Gauss interval $\xi \in [-1, 1]$, $x = \frac{0+2}{2} + \xi\frac{2-0}{2} = 1 + \xi$, giving $\mathbb{J} = \frac{dx}{d\xi} = 1$. Now, the exact value and Gauss quadrature (5.8) with parameters from (5.9).

```
{∫_0^2. f[x] dx, 1 × f[1 + 0.57735027] + 1 × f[1 + (-0.57735027)]}
```

{1.33333, 1.33333}

5.5 Adaptive integration

The idea is to adapt h to the variability of $f(x)$. That is, when f varies but little, we can proceed in large h without causing a substantial error in the integral, but where f varies rapidly, we have to take a small h in order to stay everywhere close enough to the curve of f. Changing h is done systematically, usually by halving h, and automatically (not "by hand") depending on the size of the (estimated) error over a subinterval. The idea is to start with a rather large h over the whole domain. The subintervals are then halved if the corresponding error is still too large, that is, larger than a given tolerance ε (maximum admissible absolute error), or is not halved if the error is less than or equal to ε. This continues in a recursive fashion. Adaptivity is one of the basic techniques typical of modern software. It can be applied to various numerical methods in different areas.

5.6 Problems

5.1 Consider the function table

x	1	2	3	4	5	6
$f(x)$	2	3	1	5	4	1

, and estimate $\int_2^6 f(x)dx$ using the Rectangular rule with $h = 2$. Compare with **Integrate** and linear **Interpolation** applied to data. $\{R(h) \to 10\}$

5.2 Consider the function table

x	1	2	3	4	5	6
$f(x)$	2	3	1	5	4	1

, and estimate $\int_4^6 f(x)dx$ using the Trapezoidal rule with $h = 1$. Compare with **Integrate** and linear **Interpolation** applied to data. $\{T(h) \to 7\}$

5.3 Use the trapezoidal rule to evaluate $\int_{0.2}^{0.8} e^{-x^2}dx$. Write a small snippet and take the size h equal to 0.6, 0.3, 0.15, 0.075, 0.0375 in turn and use extrapolation to improve the accuracy of you answer. $\left\{\begin{pmatrix} 0.6 & 0.3 & 0.15 & 0.075 & 0.0375 \\ 0.446425 & 0.456853 & 0.459443 & 0.460089 & 0.460251 \end{pmatrix}\right\}$, $\{T_4(0.0375) \to 0.460305\}$

5.4 Use Simpson's rule with $n = 2$ and 4 for the integral in **Problem 5.3**.
$\{n \to 2,\ h \to 0.3,\ S(0.3) \to 0.460328\}$, $\{n \to 4,\ h \to 0.15,\ S(0.15) \to 0.460307\}$.

5.5 Use Gauss two- and three-point quadrature for the integral in **Problem 5.3**. $\{G(2) \to 0.460289\}$, $\{G(3) \to 0.460305\}$

5.6 Apply **NIntegrate** (as the integral is not elementary) to the integral in **Problem 5.3**.

5.7 Use the trapezoidal rule with interval halving to evaluate $\int_0^1 \ln(\cos(x))dx$ to 4 decimal digits. Compare with **NIntegrate**. $\{-0.187538\}$

5.8 When using numerical methods in general you cannot be too overcautious, and there is no exceptions for commercial software like Matlab. Use the integrator **quad**, and try to understand why Matlab without any warnings delivers a completely wrong answer to $\int_0^8 \sin(x + e^x)dx$. The correct one is about 0.3474, which is also given by *Mathematica* (of course ;-). $\{\text{Matlab} \to 0.2511\}$

5.9 The integral $\int_0^1 \frac{\sin(x)}{\sqrt{x}}dx$ is a generalized integral due to the singularity at $x = 0$. Apply a suitable substitution to cure the problem, resulting in a Fresnel type integral. Finally apply Simpsons rule with 2^k, $k = 0, 1, \ldots, 4$ intervals. Compare with **NIntegrate**. $\{S_{16} \to 0.809048\}$

5.10 Apply a suitable substitution $x = t^\alpha$ in order to eliminate the singularity in $\int_0^1 x^{-2/3}e^x dx$. Finally apply Simpsons rule with 2^k, $k = 0, 1, \ldots, 4$ intervals. Compare with **NIntegrate**. $\{S_{16} \to 0.809048\}$

5.11 The normal distribution type integral $\int_0^\infty e^{-x^2}dx$ is also generalized due an unbounded interval. Make the partition $I = \int_0^a e^{-x^2}dx + \int_a^\infty e^{-x^2}dx = I_a + I_\infty$. Use the substitution $t = \sqrt{x}$ in I_∞ and show the estimation $0 < I_\infty < \frac{1}{2a}e^{-a^2}$ for the tail. Find an $a \in \mathbb{N}$ such that $I_\infty < 10^{-7}$. Finally apply Simpsons rule with 2^k, $k = 0, 1, \ldots, 4$ intervals to I_a giving an estimation of original I with an $\varepsilon < 10^{-7}$. Compare with **NIntegrate**. $\{S_{16} \to 0.8862369\}$

5.12 Integrate by parts the generalized integral $I = \int_0^1 \ln(\tan(x))dx$ preparing it applicable for a four point Gauss quadrature. Compare with **NIntegrate**. $\{I \to -0.869182\}$

5.13 Consider the integral $I_n = \int_0^1 x^n e^{x-1}dx$, $n = 0, 1, 2, \ldots$

a) Show that $I_0 = 1 - e^{-1}$.

b) Show the recurrence formula $I_{n+1} = 1 - (n + 1)I_n$, $n = 0, 1, 2, \ldots$.

c) Calculate I_{15} using **a)** and **b)**. Compare with **Integrate**. $\{I_{15} \to 0.590338\}$

5.14 Consider the integral $I_n = \int_0^{\pi/4} \tan^n(x)dx$.

a) Show that $I_0 = \frac{\pi}{4}$ and $I_1 = \frac{1}{2}\ln(2)$.

b) Use the trigonometric identity $\cos^2(x) + \sin^2(x) = 1$ to show the recurrence formula $I_n = \frac{1}{n-1} - I_{n-2}$, $n = 2, 3, \ldots$.

c) Calculate I_6 and I_7 using **a)** and **b)**. Compare with **Integrate**. $\left\{I_6 \to \frac{13}{15} - \frac{\pi}{4},\ I_7 \to \frac{5}{12} - \frac{1}{2}\ln(2)\right\}$

5.15 Use adaptivity strategy to evaluate $\int_0^2 (x - 1)^8 dx$. Start with $n = 2$, and use $T(h)$ and $T(2h)$ together with (5.4) and a tolerance ε of your choice in each interval to recursively adapt the work. Indicate in a plot the final set of intervals used.

6. Numerical solution of equations

6.1 Basic concepts

The equation $f(x) = 0$, or system of them, is the true atom of all mathematical models. The structure of f comes in a variety of flavors and we are interested in a solution x^* to the equation. For example in the linear case $\mathbb{A}\mathbb{x} = \mathbb{b}$ for instance, a system with one million unknown is routinely solved when it comes to the well known finite element method for solving partial differential equations numerically. In the nonlinear case the idea is to use pure search methods or to linearize the system into a sequence of linear systems. In the one-dimensional case there sometimes exists simple solution formulas. A classical example that we are all familiar with is the case in which $f(x)$ is a quadratic polynomial, $f(x) = x^2 + ax + b$. It is then well-known that the solution is given by a simple formula

$$x^*_{1,2} = -\frac{a}{2} \pm \sqrt{\left(\frac{a}{2}\right)^2 - b}$$

In the case of polynomial equations the solutions are also known as roots. There are known formulas (Girolamo Cardano, 1501-1576, Italian mathematician) for finding roots of polynomials of degree 3 and 4, but these are rather complex. In more general cases, when $f(x)$ is a polynomial of degree greater than 4, formulas for the roots no longer exist (Niels Henrik Abel, 1802-1829, Norwegian mathematician). Of course, there is no reason to limit ourselves to studying polynomials, and in most cases, when $f(x)$ is an arbitrary function, there are no analytic tools for calculating the desired solution. Instead, we must use approximation methods. In fact, even in cases in which exact formulas are available (such as with polynomials of degree 2, 3 or 4) an exact formula might be too complex (require to much time to evaluate) to be used in practice. An approximate numerical method may, on the other hand, provide an accurate solution rather more quickly.

The methods that we will describe all belong to the category of iterative methods. Such methods will typically start with an initial guess x_0 in the neighborhood of the solution and then generate a sequence x_0, x_1, … of hopefully even better approximations, and eventually terminating in x^* to a desired tolerance. There are several questions to consider, for instance

- How many solutions are there? Plot! Which one of them is of interest for the application at hand?
- Real and/or complex solution?
- Is the iterative method approaching the solution, and how fast? This is called convergence otherwise divergence.
- Desired tolerance in solution? A common termination criterion of the iterative process is $|x_{k+1} - x_k| < \varepsilon$.
- Speed of the iterative method? May sometimes be critical.

There is a large number of different iterative methods, some of which are highly specialized to solve certain types of problems. The methods are mainly divided into derivative free or not. Methods that make use of the derivative of f usually gain more information along the road and the convergence is therefore in general faster. On the other hand, if derivatives of f may not be available, or are to expensive to evaluate, derivative free methods need to be used. In fact the derivative free methods are often the simplest, and most robust ones. We present some of the most widely used.

6.2 The Bisection method

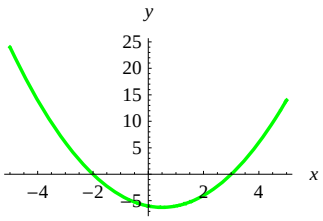
This is one of the most intuitive derivative free methods one can easily come up with, it is called the bisection method, and is applicable to a function of one variable. We have a function f which we assume is continuous on the interval $[a, b]$. We also assume that it has opposite signs at both edges of the interval, i.e., $f(a)f(b) < 0$. We then know, from the intermediate theorem, that $f(x) = 0$ has at least one solution in $[a, b]$. Of course there may be more than one solution in the interval, but the bisection method is only going to converge to one of the them. If approaching the wrong one, simple narrow $[a, b]$, covering the solution desired, and start again. As usual, a nice plot of the scenario is never wrong!

The story goes like this. The first step is to calculate the midpoint $c = \frac{1}{2}(a + b)$ of the interval. This generates two subintervals, $[a, c]$ and $[c, b]$, of equal length. We want to keep the subinterval that is guaranteed to contain a solution. Of course, in the rare event where $f(c) = 0$ we are done. Otherwise, we check if $f(a)f(c) < 0$. If yes, we keep the left subinterval $[a, c]$, otherwise we keep the right subinterval $[c, b]$. This procedure repeats until the stopping criterion $|f(c)| < \delta$ or $b - a < \varepsilon$ is satisfied, due to some predefined small parameters δ, $\varepsilon > 0$. Then we choose $x^* = c$.

Example 6.1: Find the largest root to $x^2 - x - 6 = 0$ using the Bisection method. Compare with the exact one.

Solution: First the function, and a plot

```
f[x_] := x^2 - x - 6
Plot[f[x], {x, -5, 5}, PlotStyle → Green, AxesLabel → {"x", "y"}]
```



The exact roots using *Mathematica*.

```
Solve[f[x] == 0]
```

$\{\{x \to -2\}, \{x \to 3\}\}$

Go for $x^* = 3$, using Bisection method. Let $\varepsilon = 0.1$, and visually from the graph a starting interval [1, 4] ought to work. There is one solution only, so we expect a true success while iterating.

```
a = 1; b = 4;
While[b - a > 0.1,
   c = 0.5 (a + b);
   Print[{a, c, b}];
   If[f[a] f[c] < 0, b = c, a = c]
  ]
```

{1, 2.5, 4}

{2.5, 3.25, 4}

{2.5, 2.875, 3.25}

{2.875, 3.0625, 3.25}

{2.875, 2.96875, 3.0625}

We would now like to understand if the bisection method always converges to a solution. We would also like to figure out how close we are to a solution after iterating the algorithm several times. Let $[a_k, b_k]$ be the interval after k iterations, and note that

$$a_0 \le a_1 \le a_2 \le \ldots \le b_0, \text{ and } b_0 \ge b_1 \ge b_2 \ge \ldots \ge a_0.$$

We also know that every iteration shrinks the length of the interval by half, i.e.

$$b_k - a_k = \tfrac{1}{2}(b_{k-1} - a_{k-1})$$

which means that

$$b_k - a_k = 2^{-k}(b_0 - a_0).$$

The sequences $\{a_k\}$ and $\{b_k\}$ are monotone and bounded, and hence converge. Also

$$\lim_{k\to\infty}(b_k - a_k) = \lim_{k\to\infty}\left(2^{-k}(b_0 - a_0)\right) = 0,$$

so that both sequences converge to the same value. We denote that value by x^*, i.e.

$$\lim_{k\to\infty} a_k = \lim_{k\to\infty} b_k = x^*$$

Since $f(a_k)f(b_k) \le 0$, we know that $f(x^*)^2 \le 0$, which means that $f(x^*) = 0$, i.e., x^* is a solution.

6.3 Fixed point iteration

This method relies upon that we in some way or another can algebraically transform $f(x) = 0$ into the form $x = g(x)$. Choose a starting point x_0 and compute $x_1 = g(x_0)$, $x_2 = g(x_1)$, and in general

$$x_{k+1} = g(x_k), \quad k = 0, 1, \ldots \tag{6.1}$$

A solution x^* of (6.1) is called a fixed point of g, motivating the name of the method, also known as Picard iteration (Emile Picard, 1856-1941, French mathematician). This means that x^* also is a solution to $f(x) = 0$. We may get several different forms of (6.1) from $f(x) = 0$. The behavior of corresponding iterative sequences $x_0, x_1, x_2, \ldots$ may differ, in particular, with respect to their speed of convergence. Indeed, some of them may not converge at all.

If the natural convergence test $|x_{k+1} - x_k| < \varepsilon$ is satisfied for some predefined small parameter $\varepsilon > 0$, then we can choose $x^* = x_{k+1}$. The machinery relies on the following theorems of Brouwer and Banach, ranging from existence to uniqueness.

Brouwer fixed point theorem. Assume that $g(x)$ is continuous on the closed interval $[a, b]$. Assume that the interval $[a, b]$ is mapped to itself by $g(x)$, that is, for any $x \in [a, b]$, $g(x) \in [a, b]$. Then there exists a point $x^* \in [a, b]$ such that $x^* = g(x^*)$. The point x^* is a fixed point of $g(x)$. (Jan Brouwer, 1881-1966, Dutch mathematician.) ■

Proof. Let $u(x) = x - g(x)$. Since $g(a) \in [a, b]$ and also $g(b) \in [a, b]$, we know that $u(a) = a - g(a) \leqslant 0$ and $u(b) = b - g(b) \geqslant 0$. Since $g(x)$ is continuous in $[a, b]$, so is $u(x)$, and hence according to the intermediate value theorem, there must exist a point $x^* \in [a, b]$ at which $u(x^*) = 0$. At this point $x^* = g(x^*)$. ■

An iteration process defined by (6.1) is called convergent for a value x_0 if the corresponding sequence $x_0, x_1, x_2, \ldots$ is convergent. A sufficient condition for convergence is given in the following theorem, which has various practical applications.

Banach fixed point theorem. Let x^* be a solution of $x = g(x)$ and suppose that g has a continuous first derivative in some interval I containing x^*. Then if $|g'(x)| \leq L < 1$ in I, the iteration process defined by (6.1) converges for any x_0 in I, and the limit of the sequence $\{x_k\}$ is x^*. (Stefan Banach, 1892-1945, Polish mathematician.) ■

Proof. By the mean value theorem there is a t between x, $x^* \in I$, such that

$$g(x) - g(x^*) = g'(t)\,(x - x^*).$$

Since $x^* = g(x^*)$ and $x_1 = g(x_0)$, $x_2 = g(x_1)$, … we obtain from this and the condition on $|g'(x)|$ in the theorem

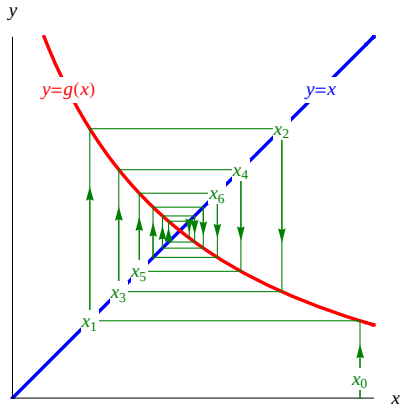
$$|x_k - x^*| = |g(x_{k-1}) - g(x^*)| = |g'(t)|\,|x_{k-1} - x^*| \leq L\,|x_{k-1} - x^*|.$$

Applying this inequality k times, for k, $k-1$, …, 1 gives

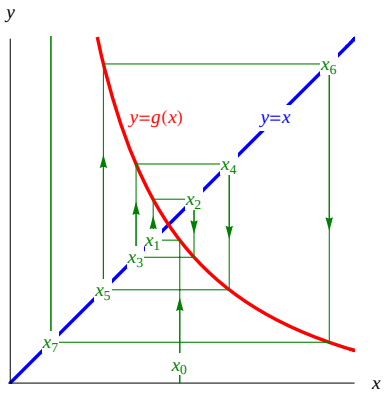
$$|x_k - x^*| \leq L\,|x_{k-1} - x^*| \leq L^2\,|x_{k-2} - x^*| \leq \ldots \leq L^k\,|x_0 - x^*|.$$

Since $L < 1$, we have $L^k \to 0$, hence $|x_k - x^*| \to 0$. ■

A function g satisfying the condition in the theorem is called a contraction because $|g(x) - g(y)| \leq L\,|x - y|$, $x, y \in I$, where $L < 1$. Furthermore, L gives information on the speed of convergence, smaller L implies faster convergence, i.e. less number of iterations. Note, $|g(x) - g(y)| \leq L\,|x - y|$ is called the Lipschitz condition (Rudolf Lipschitz, 1832-1903, German mathematician) where L is the Lipschitz constant, and offer an alternative and more demanding definition of continuity and derivation that is *constructive*, in comparison with the "usual" Cauchy *existence* ε-δ definitions we happened to learn in a standard analysis course. This theorem will guarantee convergence to a fixed point, x^*, of $g(x)$ if we start the iterations sufficiently close to that point x^*. Starting "far" from x^* may or may not lead to a convergence to x^*. Also, since we consider only a neighborhood of the fixed point x^*, we can no longer guarantee the uniqueness of the fixed point. Finally, we illustrate the fixed point iteration process



Convergence, $|g'(x^*)| < 1$



Divergence, $|g'(x^*)| > 1$

Example 6.2: Find the largest root to $x^2 - x - 6 = 0$ using fixed point iteration. Compare with the exact one.

Solution: The roots are $x_1 = -2$ and $x_2 = 3$. See **Example 6.1**. Rewrite the equation into the form $x = g(x) = \sqrt{x+6}$. Clearly $g'(3) = \frac{1}{2\sqrt{3+6}} < 1$, and the conditions are satisfied for convergence, if we start sufficiently close to 3.

```
g[x_] := √(x + 6) ; NestList[g, 3.25, 8]
```

{3.25, 3.04138, 3.00689, 3.00115, 3.00019, 3.00003, 3.00001, 3., 3.}

■

6.4 Newton's method

Newton's method, also known as the Newton-Raphson's method (Joseph Raphson, 1648-1715, English mathematician), is a relatively simple, practical, and widely-used solution finding method. The derivative of f is required. This extra information is paid back in speed. It's easy to see that while in some cases the method rapidly converges to a root of the function, in some other cases it may fail to converge at all. This is one reason why it is important not only to understand the construction of the method, but also to understand its limitations.

As always, we assume that $f(x) = 0$ has at least one (real) solution, and denote it by x^*. We start with an initial guess for the location of the solution, say x_0. We then let $l(x)$ be the tangent line to $f(x)$ at x_0, i.e.,

$$l(x) - f(x_0) = f'(x_0)(x - x_0).$$

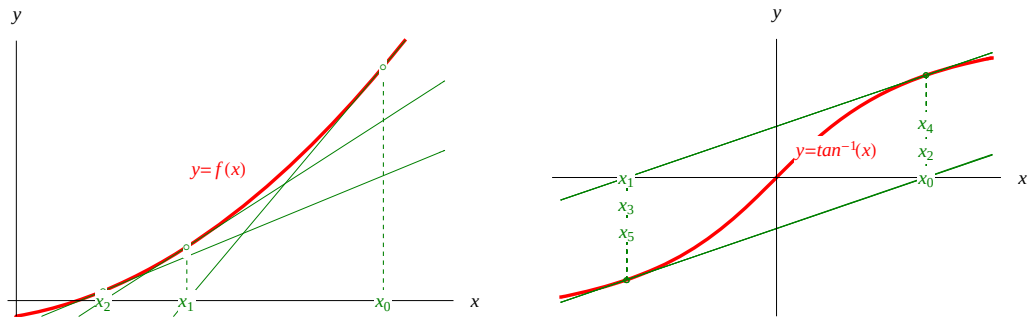
The intersection of $l(x)$ with the x-axis serves as the next estimate of the solution. We denote this point by x_1 and write

$$0 - f(x_0) = f'(x_0)(x_1 - x_0) \Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{6.2}$$

In general, the Newton's method for finding a solution is now given by iterating (6.2) repeatedly, i.e.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{6.3}$$

The stopping criterion is usually the same as for the previous methods, if $|f(x_{k+1})| < \delta$ or $|x_{k+1} - x_k| < \varepsilon$ is satisfied with some predefined small parameters δ, $\varepsilon > 0$. Then we choose $x^* = x_{k+1}$. Two sample iterations of the method are shown in the left figure below. Starting from a point x_0, we find the next approximation x_1, from which we find x_2 and so on. In this case, we do converge to the solution x^* of $f(x) = 0$. It is easy to see that Newton's method does not always converge. We demonstrate such a case in the right figure below. Here we consider the function $f(x) = \tan^{-1}(x)$ and show what happens if we start with a point which is a fixed point. In this case, $x_0 \approx 1.39175$ is such a point. The iteration is stuck to jumping between two values given by x_0 and x_1.



Example 6.3: Find the largest root to $x^2 - x - 6 = 0$ using Newton's method. Compare with the exact one.

Solution: The roots are $x_1 = -2$ and $x_2 = 3$. See previous examples. Use (6.3) with $x_0 = 2.5$.

```
f[x_] := x^2 - x - 6; Newtons[x_] := x - f[x]/f'[x]; NestList[Newtons, 2.5, 5]
```

{2.5, 3.0625, 3.00076, 3., 3., 3.}

We will not dwell on the error analysis, but mention that if $f'(x^*) \neq 0$, and $f''(x^*) \neq 0$, then the convergence is quadratic (if it converges ;-), that is about the error in each iteration $|x_{k+1} - x^*| \approx |x_k - x^*|^2$. Generally, if we replace 2 by α, we say that the convergence is of order α.

Finally we expand Newton's method into a multidimensional setting. Let $f : \mathbb{R}^n \to \mathbb{R}^n$, and we are now interested in the solution vector x^* of the system of nonlinear equations

$$f(x) = 0 \quad \Leftrightarrow \quad \begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases} \tag{6.4}$$

Fortunately the algebra stays virtually unchanged for all n. Compared with the one-dimensional case above, instead of a single number $f'(x) = \frac{df}{dx}$, the derivative becomes a matrix, the Jacobian matrix $J(x)$. It contains the first order partial derivatives $J_{ij} = \frac{\partial f_i}{\partial x_j}$ of the n functions f_i with respect to the n variables x_j. The multiplication $J(x^{(k)})(x^{(k+1)} - x^{(k)})$, where k is the iteration counter, still gives the first order correction to f in equation (6.4). Therefore, rearrange the one-dimensional formula for Newton iteration (6.3),

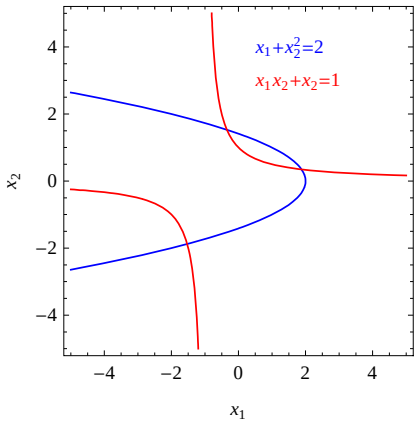
$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \Leftrightarrow f'(x_k)(x_{k+1} - x_k) = -f(x_k)$, and plug in, we finally ends up with Newton's method in a multidimensional setting. Iteratively solve the system of linear equations

$$J\left(x^{(k)}\right)\left(x^{(k+1)} - x^{(k)}\right) = -f\left(x^{(k)}\right) \tag{6.5}$$

Needless to say, this system is of course solved using Gaussian elimination method rather than matrix inversion. Finally, in case of an overdetermined system of nonlinear equations we follow the concept of normal equations we learned in the chapter of least squares, that is (6.5) modified to

$$J\left(x^{(k)}\right)^T J\left(x^{(k)}\right)\left(x^{(k+1)} - x^{(k)}\right) = -J\left(x^{(k)}\right)^T f\left(x^{(k)}\right) \tag{6.6}$$

Example 6.4: Study the system of nonlinear equations $\begin{cases} x_1 + x_2^2 = 2 \\ x_1 x_2 + x_2 = 1 \end{cases}$.



Solution: We have the following solutions

```
NSolve[{x1 + x2^2 == 2, x1 x2 + x2 == 1}]
```

$\{\{x_1 \to -0.347296,\ x_2 \to 1.53209\},\ \{x_1 \to 1.87939,\ x_2 \to 0.347296\},\ \{x_1 \to -1.53209,\ x_2 \to -1.87939\}\}$

Seek the first one, starting at the point $(0,\ 2)$. First the equations f, then the Jacobian matrix J, finally some Newton iterations.

```
ff = {x1 + x2^2 - 2, x1 x2 + x2 - 1}
```

$\{x_2^2 + x_1 - 2,\ x_1 x_2 + x_2 - 1\}$

```
JJ = D[ff, {{x1, x2}}]
```

$$\begin{pmatrix} 1 & 2x_2 \\ x_2 & x_1 + 1 \end{pmatrix}$$

```
x1 = 0.0; x2 = 2.0;
Do[Print[{x1, x2} = {x1, x2} - Inverse[JJ].ff], {3}]
```

{−0.285714, 1.57143}

{−0.345756, 1.53209}

{−0.347296, 1.53209}

```
FindRoot[ff == 0, {x1, 0}, {x2, 2}]
```

$\{x_1 \to -0.347296,\ x_2 \to 1.53209\}$

6.5 The Secant method

We recall that Newton's solution finding method is given by equation (6.3). We now assume that we do not know whether the function $f(x)$ is differentiable at x_k, and thus can not use Newton's method directly. Instead, we can replace the derivative $f'(x_k)$ that appears in Newton's method by a difference approximation. A particular choice of such an approximation is the one-sided difference scheme $f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$, leading to the secant method.

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}. \tag{6.7}$$

A geometric interpretation of the secant method indicates that we in fact are using the secant line instead of the tangent line in our strive for the next approximation. We therefore note that the secant method requires two initial points, and uses the last two points on f when forming the next secant in the iterative process. While this is an extra requirement compared with the Newton's method, we note that in the secant method there is no need to evaluate any derivatives. In addition, if implemented properly, every stage requires only one new function evaluation. Knowing this, it's straight forward to replace the analytical derivatives in the Jacobian with the approximate ones, and derive a secant method to address multidimensional situations.

As the iteration goes on, the secant line will more and more mimic the tangent line and the secant method the Newton method. Due to this fact the convergence for the secant method is super-linear, meaning better than linear but less than quadratic, as Newton is.

6.6 Problems

6.1 Consider the equation $3x = 1$. Perform three iterations with the Bisection method, starting with [0, 1].
$\{\{a, b\} \to \{0, 1\}$, after 3 iterations $\{a, c, b\} \to \{0.25, 0.3125, 0.375\}, \{x^* \to 0.333333\}\}$

6.2 Find the solution of $f(x) = x^3 + x - 1 = 0$. Plot, and sort out the number of solutions. Then find a suitable start interval for the Bisection method. Compare with **`(N)Solve`** for solving a pure polynomial equation.
$\{\{a, b\} \to \{0, 1\}$, after 9 iterations $\{a, c, b\} \to \{0.681641, 0.682617, 0.683594\}, \{x^* \to 0.682328\}\}$

6.3 Rewrite the equation in **Problem 6.2** in such a form that the fixed point iteration converges. Why? $\left\{x_0 \to 0.5,\ x_{k+1} = \sqrt[3]{1 - x_k}\right\}$

6.4 Newton's method for the equation in **Problem 6.2**, starting at the point $x_0 = 1$. $\{x_3 \to 0.68234\}$

6.5 Consider the equation $x\sin(x) - 4\cos(x) + e^x = 0$, and use Newton's method to determine the solution near $x = -4$. Compare with **`FindRoot`** for solving a system of nonlinear equations. $\{x^* \to -3.93824\}$

6.6 We want to determine the real root of the equation $x^3 + 2x - 11 = 0$. Which of the following fix point iteration methods can be used? Investigate speed of convergence. $\{b, c, d$; b slowest and d fastest. Why ?$\}$

a) $x_{k+1} = \frac{11 - x_k^3}{2}$ **b)** $x_{k+1} = \frac{11 + 8x_k - x_k^3}{10}$ **c)** $x_{k+1} = \sqrt[3]{11 - 2x_k}$ **d)** $x_{k+1} = \frac{11 + 11x_k - x_k^3}{13}$

6.7 Apply Newton's method (6.5), and **Example 6.4**, to solve the system of linear equations $\begin{cases} 2x - 3y = 1 \\ 4x + 7y = 5 \end{cases}$. Compare with **`(N)Solve`**.

6.8 Mimic **Example 6.4** with $\begin{cases} x_2^3 - 10x_1x_2^2 = 2 \\ x_1^2x_2 + x_2^2 = 1 \end{cases}$. Draw a picture and find all real solutions. Compare with **`NSolve`** and **`FindRoot`**.

6.9 When a computer calculates the square root function of a given number a, it actually solves the equation $x^2 - a = 0$ using the fixed point iteration formula $x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$ derived from Newton's method. Show this result, and explain why x_{k+1} is a better approximation to $\sqrt{2}$ than both x_k and $\frac{a}{x_k}$. Finally mimic your computer to find $\sqrt{2}$ to six decimal places.

6.10 Newton's(!) heat/cooling model, a differential equation, typically presents a solution with exponential behavior. Find the time t^* when the two models $T_1(t) = 100\left(1 - e^{-0.2t}\right)$, and $T_2(t) = 40\,e^{-0.01t}$ reach the same temperature. $T_1(t^*)$ is also of interest! Method? Newton of course ;-). Compare with **`FindRoot`**. $\{t^* \to 2.47335,\ T_1(t^*) \to 39.0228\}$

6.11 In one application there was a need for a high order Gauss quadrature. Find the largest root, one of the Gauss points, of the Legendre polynomial $P_5(x) = \frac{1}{8}\left(63x^5 - 70x^3 + 15x\right)$. Use Bisection method. $\{x \to 0.90618\}$

6.12 In the text we considered the function $f(x) = \tan^{-1}(x)$ to illustrate the pathological behavior of Newton's method when $x_0 \approx 1.39175$. Verify this x_0! Use Newton's method to model the situation, and then use it again to solve the model!

7. Linear algebra

7.1 Linear system of equations

System of linear equations is the basis for linear algebra as calculus of limits is for analysis. Almost every mathematical model in practice boils down to solving a system of linear equations. Let's take it from the beginning.

A linear equation in two unknowns is a relation of the form

$$ax + by = c$$

where $a, b, c \in \mathbb{R}$ (in general $\mathbb{C}$) are coefficients. The word linear means that we need to have a linear combination of the coefficients and the unknowns, for example $6x + 2y = 10$. Other forms like $6x^2 + 2y = 10$ are not allowed.

We recognize that for example $x = 1$, $y = 2$ satisfies $6x + 2y = 10$, and is therefor denoted a solution. Another one is $x = 0$, $y = 5$. You can easily find more, actually there are infinite number of them. For every given x there is a unique $y = 5 - 3x$. The reason for this is that there are more unknowns than equations.

With a system of linear equations we mean a number of linear equations that need to hold simultaneously. Even in this case there may be more equations than unknowns, less than or equal. We are talking of overdetermined, underdetermined or just a linear system of equations. They are all found in various applications. Here is a typical system with the same number of equations as unknowns

$$\begin{cases} 2x+3y-9z=5 \\ 3x+4y-5z=1 \\ 4x-2y+2z=6 \end{cases}$$

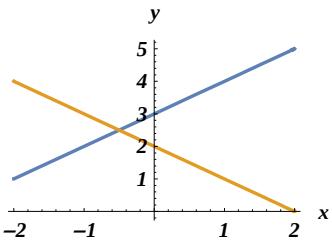
It is common to rewrite them in matrix form

$$\begin{cases} 2x+3y-9z=5 \\ 3x+4y-5z=1 \\ 4x-2y+2z=6 \end{cases} \Leftrightarrow \begin{pmatrix} 2 & 3 & -9 \\ 3 & 4 & -5 \\ 4 & -2 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix}=\begin{pmatrix} 5 \\ 1 \\ 6 \end{pmatrix} \Leftrightarrow \mathbb{A}\mathbb{x}=\mathbb{b}$$

where $\mathbb{A}$ is called the coefficient matrix, $\mathbb{b}$ the right hand side and $\mathbb{x}$ the solution vector or vector of unknowns. If $\mathbb{b}$ is the null vector the system is called homogeneous. For the moment being we will concentrate on systems where $\mathbb{A}$ is a square matrix. We have already learned that the overdetermined system is natural in the least squares method and the over determined systems will show up in optimization. The square ones dominates the picture so the naming is shortened to linear system of equations. If our intention is to talk about the other ones, there full names are used explicitly.

It looks sound to have a linear system of equations, with a nice solution due to the fact that the number of unknowns are equal to the number of equations. But, in fact, we can have a unique solution, no solution, or infinite number of solutions! It depends on the determinant $|\mathbb{A}|$ och $\mathbb{b}$. A geometrical study with a system of two equations and two unknowns x, y illuminates the scene. In this case are the equations straight lines in the 2D plane.

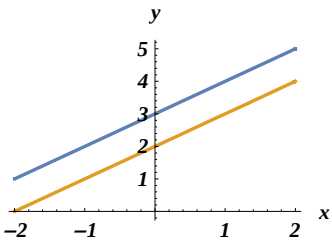
$$\begin{cases} x-y=-3 \\ x+y=2 \end{cases}$$



$|\mathbb{A}| \neq 0$

unique solution

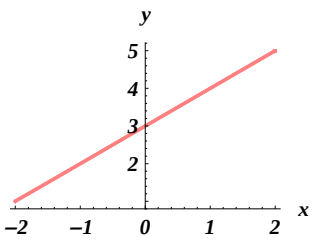
$$\begin{cases} x-y=-3 \\ x-y=-2 \end{cases}$$



$|\mathbb{A}| = 0$

no solution

$$\begin{cases} x-y=-3 \\ x-y=-3 \end{cases}$$



$|\mathbb{A}| = 0$

infinite number of solutions

↖ Which one depends on $\mathbb{b}$ ↗

If $|\mathbb{A}| \approx 0$ we say that the system is ill-conditioned. By inspection of the figures we understand that a small perturbations of the coefficients in $\mathbb{A}$ or $\mathbb{b}$ can give a new solution vector of great surprise. Friendly solvers, like **`Solve`** in *Mathematica*, signals if ill-condition is on the menu. Usually this depends on weak modelling, go back an reformulate your ideas.

Example 7.1: Study the two look-a-like systems

```
Solve[{{400, -201}, {-800, 401}} . {x, y} == {200, -200}]
```

$\{\{x \to -100, y \to -200\}\}$

```
Solve[{{401, -201}, {-800, 401}} . {x, y} == {200, -200}]
```

$\{\{x \to 40\,000, y \to 79\,800\}\}$

In the second one is the coefficient a_{11} perturbed from 400 to 401, 0.25%, making the solution to blow up compared to the first one. We are therefor talking about well- and, in this case, ill-conditioned systems.

■

Example 7.2: Solve the system

$$\begin{cases} x+y+z=50 \\ 1x+1.5y+2z=80 \\ 2x+1y+1.5z=70 \end{cases}$$

Note that **`Solve`** in *Mathematica* is not requiring a notorious error prone reload to matrix form $\mathbb{A}\mathbb{x}=\mathbb{b}$ as in the case of other computer programs, like Matlab and its \. In fact **`Solve`** can manage both of them equally well. The first form below with the equations written explicitly also opens up for storing $\mathbb{A}$ in sparse form if the coupling between the equations is weak. *Mathematica* changes the storage format dynamically between dense and sparse behind the curtain, depending on which format is most efficient on the computer at hand. Solving a sparse system with millions of unknowns on a laptop is no problems! Handy!

```
Solve[{x + y + z == 50,
       1 x + 1.5 y + 2 z == 80,
       2 x + 1 y + 1.5 z == 70}]
```

$\{\{x \to 10., y \to 20., z \to 20.\}\}$

```
Solve[{{1, 1, 1}, {1, 1.5, 2}, {2, 1, 1.5}} . {{x}, {y}, {z}} == {{50}, {80}, {70}}]
```

$\{\{x \to 10., y \to 20., z \to 20.\}\}$

Example 7.3: Solve the matrix equation $\mathbb{A}^2\mathbb{X} = 2\mathbb{X} + \mathbb{C}$ when $\mathbb{A} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}$ and $\mathbb{C}^T = (\,4 \quad 6\,)$.

Solution: Just find out that type($\mathbb{X}$) = type($\mathbb{C}$) and furnish $\mathbb{X}$ accordingly. After that it's an easy piece for *Mathematica*.

```
𝔸 = {{2, 1}, {1, 0}}; ℂ = {{4}, {6}}; 𝕏 = {{x11}, {x21}};
```

```
ekv = 𝔸.𝔸.𝕏 == 2 𝕏 + ℂ
```

$$\begin{pmatrix} 5\,x_{11} + 2\,x_{21} \\ 2\,x_{11} + x_{21} \end{pmatrix} = \begin{pmatrix} 2\,x_{11} + 4 \\ 2\,x_{21} + 6 \end{pmatrix}$$

```
𝕏 /. Solve[ekv] // First
```

$$\begin{pmatrix} \frac{16}{7} \\ -\frac{10}{7} \end{pmatrix}$$

7.2 Solution methods for linear system of equations

As linear system of equations is bread and butter for computers dealing with computational science and engineering it's important to have stable and effective algorithms to solve them, even if the number of unknowns grow up to more than 100 millions. We distinguish between direct and iterative methods. We consider them in the next chapters.

7.2.1 Gaussian elimination method for linear system of equations

The dominating direct method is the Gaussian elimination method (Carl Friedrich Gauss, 1777-1855, German mathematician) which has been used for large system as long as computers has been around. First the basics applied to a system with two unknowns

$$\begin{cases} 2x + 6y = 3 & (1) \\ 4x + 7y = 8 & (2) \end{cases}$$

and make some more or less obvious statements. The solution to the system is *not* changed if we

1. Swap two rows.
2. Multiply one row with a non zero constant.
3. Add one row to another.

They are called elementary row operations. Two systems that can be rearranged into each other by just using these row operations are called row equivalent and have of course the same solution.

Perhaps we should talk about equations instead of rows, but this vocabulary comes along later on with the implementation of the algorithm in a matrix. Now, point 1 is obvious and point 2 is a kindly reminder of that "you should do the same thing on both sides of the equation", that is for example applied to equation (2): $123(4x + 7y) = 123 \cdot 8$. Point 3 relies on the same argument, so equation (2) once again: $4x + 7y - 2 \cdot 3 = 8 - 2 \cdot 3$. But equation (1) reads $2x + 6y = 3$, which will be used to replace the number 3 on the left hand side in our latest version of equation (2): $4x + 7y - 2 \cdot (2x + 6y) = 8 - 2 \cdot 3 \Leftrightarrow -5y = 2$. By such clever use of point 2 and 3 we a capable of reducing our system of equations to a simpler one that is easy to solve. This systematic strategy is called Gaussian elimination method and a computer can easily be drilled to perform it. Hang on!

$$\begin{cases} 2x + 6y = 3 & (1) \\ 4x + 7y = 8 & (2) \end{cases}$$

Multiply equation (1) with $\frac{4}{2}$ and subtract from equation (2) giving the row equivalent system

$$\begin{cases} 2x+6y=3 & (1) \\ 4x+7y-\frac{4}{2}(2x+6y)=8-\frac{4}{2}\cdot 3 & (2) \end{cases}$$

which is simplified to

$$\begin{cases} 2x+6y=3 & (1) \\ -5y=2 & (2) \end{cases}$$

It is obvious that the maneuver using point 2 and 3 comes to success due to the clever choice of the parameter $\frac{4}{2}$ making x to be eliminated from equation (2) after multiplication of equation (1) with that parameter followed by subtraction from equation (2). The strategy is now to use elementary row operations and go through the columns of $\mathbb{A}$ in order to make it upper right triangular, that is making all coefficient below the main diagonal zero. This is the elimination step. The coefficient in the main diagonal which is so frequently used in the denominator of the clever parameter is called the pivot element. After the elimination step is finished the unknowns can easily be determined by a straight forward back substation, from below, introducing the already calculated unknowns one at a time, giving one easy equation with one unknown to solve in each step.

$$(2):\ -5y=2 \Rightarrow y=-\frac{2}{5} \Rightarrow (1):\ 2x+6\left(-\frac{2}{5}\right)=3 \Rightarrow x=\frac{27}{10}$$

The Gaussian elimination method is clearly a systematic way of finding a solution $\mathbb{x}$ to a linear system of equations $\mathbb{A}\mathbb{x}=\mathbb{b}$, without making use of the inverse of $\mathbb{A}$. It's practical to handle the process in the so called augmented matrix, consisting of $\mathbb{A}$ with $\mathbb{b}$ appended as an extra column, $[\mathbb{A}\,|\,\mathbb{b}]$. After the elimination step we are talking of row-echelon form of the augmented matrix, like a stair case. In this state we can also extract the determinant of $\mathbb{A}$ as a simple product of the elements in the main diagonal. This is actually the way computers are calculating determinants. Sound solvers taking care of error propagation and ill-condition by trying to maximize the absolute value of the pivot element via row exchange. This is called pivoting. A serious implementation, like **`Solve`** i *Mathematica*, consider this, of course.

One can actually show that Gaussian elimination method is *the* direct method that uses the least amount of floating point operations to solve a given linear system of equations. The solution time is roughly proportional to the number of unknowns raised to the power of three, $T=kn^3$, where n is the number of unknowns and k a constant depending of the computer at hand. Of course, T is going up dramatically if it's necessary to partition $\mathbb{A}$ on disk, and swap chunks of data in and out of memory. I any case this is a true "cart-horse" in many applications. We illustrate the method with a system of three equations and three unknowns, and put the "Computational Thinking" glasses on.

Example 7.4: Solve the system $\mathbb{A}\mathbb{x}=\mathbb{b}$, where $\mathbb{A}=\begin{pmatrix} 2 & 4 & -1 \\ 4 & 6 & 5 \\ 6 & 8 & 4 \end{pmatrix}$, $\mathbb{b}=\begin{pmatrix} 2 \\ 3 \\ 6 \end{pmatrix}$ and $\mathbb{x}=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ using the Gaussian elimination method.

Solution: Over to Gauss and the augmented matrix $\left(\begin{array}{ccc|c} 2 & 4 & -1 & 2 \\ 4 & 6 & 5 & 3 \\ 6 & 8 & 4 & 6 \end{array}\right)$. Use the strategy of elementary row operations explained above.

By $[\![k]\!]$ we mean the numbers in row k of the augmented matrix, and by $\sim$ we point out that two systems are row equivalent. The current pivot element is painted red and the column elements to be processed are in blue color.

Column 1: $[\![2]\!]-\frac{4}{2}[\![1]\!]$, then $[\![3]\!]-\frac{6}{2}[\![1]\!]$:

$$\left(\begin{array}{ccc|c} 2 & 4 & -1 & 2 \\ 4 & 6 & 5 & 3 \\ 6 & 8 & 4 & 6 \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & 4 & -1 & 2 \\ 4-\frac{4}{2}\cdot 2 & 6-\frac{4}{2}\cdot 4 & 5-\frac{4}{2}\cdot(-1) & 3-\frac{4}{2}\cdot 2 \\ 6-\frac{6}{2}\cdot 2 & 8-\frac{6}{2}\cdot 4 & 4-\frac{6}{2}\cdot(-1) & 6-\frac{6}{2}\cdot 2 \end{array}\right) = \left(\begin{array}{ccc|c} 2 & 4 & -1 & 2 \\ 0 & -2 & 7 & -1 \\ 0 & -4 & 7 & 0 \end{array}\right) \sim$$

Column 2: $[\![3]\!]-\frac{-4}{-2}[\![2]\!]$:

$$\left(\begin{array}{ccc|c} 2 & 4 & -1 & 2 \\ 0 & -2 & 7 & -1 \\ 0 & -4 & 7 & 0 \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & 4 & -1 & 2 \\ 0 & -2 & 7 & -1 \\ 0 & -4-\frac{-4}{-2}\cdot(-2) & 7-\frac{-4}{-2}\cdot 7 & 0-\frac{-4}{-2}\cdot(-1) \end{array}\right) = \left(\begin{array}{ccc|c} 2 & 4 & -1 & 2 \\ 0 & -2 & 7 & -1 \\ 0 & 0 & -7 & 2 \end{array}\right)$$

The elimination step is now finished, and we note that $|\mathbb{A}|=2\,(-2)\,(-7)=28$. Is it really so…

```
Det[{{2, 4, -1}, {4, 6, 5}, {6, 8, 4}}]
```

Finally $\mathbb{x}$ by back substitution, solving the simplest equation one at a time.

$$\begin{cases} \text{Eqn (3)}: \; -7x_3 = 2 \Rightarrow x_3 = -\frac{2}{7} \\ \text{Eqn (2)}: \; -2x_2 + 7\left(-\frac{2}{7}\right) = -1 \Rightarrow x_2 = -\frac{1}{2} \\ \text{Eqn (1)}: \; 2x_1 + 4\left(-\frac{1}{2}\right) - 1\left(-\frac{2}{7}\right) = 2 \Rightarrow x_1 = \frac{13}{7} \end{cases} \qquad \Rightarrow \mathbb{x} = \left(\frac{13}{7}, -\frac{1}{2}, -\frac{2}{7}\right)^T$$

What about the referee?

```
Solve[{{2, 4, -1}, {4, 6, 5}, {6, 8, 4}} . {x1, x2, x3} == {2, 3, 6}]
```

$$\left\{\left\{x_1 \to \frac{13}{7}, x_2 \to -\frac{1}{2}, x_3 \to -\frac{2}{7}\right\}\right\}$$

7.2.2 Iterative methods for linear system of equations

The advantage of Gauss's elimination method is that it in a finite number of operations gives the correct result at a given time, $T = kn^3$. However, major problems are encountered when the problem size increases and $\mathbb{A}$ does not fit into the computer's memory, but must be divided into partial arrays (partitioned) and stored on disk, and swapped in and out. This increases the solution time dramatically to much longer than the formula above suggests. Furthermore, one may ask if it is reasonable to determine a solution with full machine accuracy, maybe 15-20 digits, if one considers that in mathematical modeling you have a model of reality, perhaps full of various idealizations? In some applications it's also necessary to introduce some details "nice and smoothly" into the model, for instance increase a force in small steps to full power in a nonlinear solid mechanics model. Of course, it's no meaning to solve for full accuracy during the load step process. Then it may be time to think of an iterative solver. There are many different types of them, more or less simple to implement.

Typically, an iterative method is fed with an initial guess $\mathbb{x}^{(0)}$ and generates a sequence of better and better solutions $\{\mathbb{x}^{(k)}\}$ to $\mathbb{A}\mathbb{x} = \mathbb{b}$. As we don't know the true solution $\mathbb{x}$ we have to make a criteria for the iteration to stop as $|\mathbb{x}^{(k+1)} - \mathbb{x}^{(k)}| < \varepsilon$, which is usually reliable, and we are talking of convergence. The speed of convergence is determined as the of number of iterations k to satisfy the stop criteria. Of course there are disadvantages, in comparison to a direct Gauss. The biggest question is how many iterations k are needed before the desired accuracy ε is reached? This is determined by the so-called condition number for $\mathbb{A}$, which in turn depends on how scattered the eigenvalues of $\mathbb{A}$ are. By so-called preconditioning of $\mathbb{A}\mathbb{x} = \mathbb{b}$ with a matrix "similar" to $\mathbb{A}^{-1}$, they are gathered and performance is increasing significantly. Great effort is devoted to finding such "cheap" pre-conditioners. This research, along with the development of effective iterative methods, has been very successful in recent years. Even for small problems they have become competitive and for the really big sized problem they have been the only way forward. For iterative methods in general, due to their construction, you can iterate to the desired accuracy you need for the moment, as in the force step strategy mentioned above. This is a nice time saving!

7.2.2.1 Jacobi's method and Gauss-Seidel's method

One path goes along with the fixed point iteration, found in a previous chapter concerning solution of nonlinear equations. Shortly, a fixed point of a function f is a point x^* such that $f(x^*) = x^*$. Of course any equation $f(x) = 0$ can be brought into a fixed point equation just by writing it as $x + f(x) = x$. Then the solution to $f(x) = 0$ is equivalent to the fixed point of $g(x) = x + f(x)$.

Now we apply the same idea to solve the linear system of equations $\mathbb{A}\mathbb{x} = \mathbb{b}$. We have to bring the equation into a fixed point form, and we do it by splitting the matrix $\mathbb{A} = \mathbb{B} + (\mathbb{A} - \mathbb{B})$ with some cleverly chosen matrix $\mathbb{B}$. The success depends on this. The way to think about it is that $\mathbb{B}$ is the "main part" which will be chosen as a "nice" matrix, and we view $\mathbb{A}$ as a "small perturbation", by $\mathbb{A} - \mathbb{B}$, of the nice matrix $\mathbb{B}$. Clearly

$$\mathbb{A}\mathbb{x} = \mathbb{b} \Leftrightarrow \mathbb{B}\mathbb{x} = (\mathbb{B} - \mathbb{A})\mathbb{x} + \mathbb{b} \tag{7.1}$$

or

$$\mathbb{x} = \mathbb{B}^{-1}((\mathbb{B} - \mathbb{A})\mathbb{x} + \mathbb{b}) = \left(\mathbb{I} - \mathbb{B}^{-1}\mathbb{A}\right)\mathbb{x} + \mathbb{B}^{-1}\mathbb{b} \tag{7.2}$$

at least if $\mathbb{B}$ is invertible, hence we will have to chose $\mathbb{B}$ such that $\mathbb{B}^{-1}$ be simple. The algorithm is very primitive. Start with an initial guess $\mathbb{x}^{(0)}$, and iteratively generate

$$\mathbb{x}^{(k+1)} = \left(\mathbb{I} - \mathbb{B}^{-1}\mathbb{A}\right)\mathbb{x}^{(k)} + \mathbb{B}^{-1}\mathbb{b}, \;\; \text{for } k = 0, 1, 2, \ldots \tag{7.3}$$

and hope for the best. The computer implementation is of course (7.1) and the Gaussian elimination method.

How to predict the convergence? At least is there some sufficient condition for convergence? Let $\Delta\mathbb{x}^{(k)} = \mathbb{x} - \mathbb{x}^{(k)}$, be the error after the k-th iteration. Subtracting $\mathbb{B}$ times $\mathbb{x}^{(k)}$ in (7.3) from $\mathbb{B}\mathbb{x}$ in (7.1) we easily see that

$$\mathbb{B}\Delta\mathbb{x}^{(k)} = (\mathbb{B} - \mathbb{A})\mathbb{x}^{(k-1)} \tag{7.4}$$

that is

$$\Delta\mathbb{x}^{(k)} = \left(\mathbb{I} - \mathbb{B}^{-1}\mathbb{A}\right)\Delta\mathbb{x}^{(k-1)} = \ldots = \left(\mathbb{I} - \mathbb{B}^{-1}\mathbb{A}\right)^{k}\Delta\mathbb{x}^{(0)} \tag{7.5}$$

where $\Delta\mathbb{x}^{(0)}$ is the difference between the initial guess and the solution. Hence,

$$|\,\Delta\mathbb{x}^{(k)}\,| = |\left(\mathbb{I} - \mathbb{B}^{-1}\mathbb{A}\right)^{k}||\,\Delta\mathbb{x}^{(0)}\,| \tag{7.6}$$

It is clear that if the norm of the matrix $\left(\mathbb{I} - \mathbb{B}^{-1}\mathbb{A}\right)^{k}$ converges to zero, then the iteration converges. Moreover, this gives an estimate on the speed of the convergence. The good news is that the iteration converges very fast, in fact exponentially, as stated in a theorem we give without proof.

Theorem. If $\mathbb{A}$ and $\mathbb{B}$ are regular square matrices, then the iteration given by (7.3) converges to the solution to $\mathbb{A}\mathbb{x} = \mathbb{b}$ if all eigenvalues of $\mathbb{I} - \mathbb{B}^{-1}\mathbb{A}$ are less than one in absolute value. The speed of the convergence is exponential with a rate given by the absolute value of the largest eigenvalue of $\mathbb{I} - \mathbb{B}^{-1}\mathbb{A}$. ■

As said before, the art of iterative methods for $\mathbb{A}\mathbb{x} = \mathbb{b}$ is a good decomposition of $\mathbb{A}$. The simplest is the Jacobi's method (Carl Jacobi, 1804-1851, German mathematician) splitting $\mathbb{A}$ as

$$\mathbb{A} = \mathbb{L} + \mathbb{D} + \mathbb{U}$$

where $\mathbb{L}$ is the lower triangular matrix, containing all elements of $\mathbb{A}$ strictly below the diagonal, $\mathbb{U}$ is its upper triangular counterpart and $\mathbb{D}$ containing the main diagonal of $\mathbb{A}$. Now use the recipe above and let $\mathbb{B} = \mathbb{D}$ and $\mathbb{A} - \mathbb{B} = \mathbb{L} + \mathbb{U}$ be the decomposition. Since the diagonal matrix is easy to invert, we can easily write down the formula for Jacobi's method

$$\mathbb{x}^{(k+1)} = \mathbb{D}^{-1}\left(\mathbb{b} - (\mathbb{L} + \mathbb{U})\mathbb{x}^{(k)}\right), \text{ for } k = 0,\ 1,\ 2,\ \ldots \tag{7.7}$$

or expanded to elements

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_i^{(k)}\right), \text{ for all } i \text{ and } k = 0,\ 1,\ 2,\ \ldots \tag{7.8}$$

The initial guess may, in lack of fantasy, be chosen as the zero vector, unless we have a better a-priori guess for the true solution. Does it work? We could use the previous theorem to chek the eigenvalues of $-\mathbb{D}^{-1}(\mathbb{L} + \mathbb{U})$, but this may not be so easy. Lucky enough, there is another theorem, that is not optimal, but it provides a very simple computable condition for convergence of the Jacobi iteration

Theorem. Suppose that $\mathbb{A}$ is diagonal dominant, that is

$$|\,a_{ii}\,| > \sum_{j\neq i} |\,a_{ij}\,|, \text{ for all } i, \tag{7.9}$$

then the Jacobi iteration method converges. ■

We just remark that the condition says that in every row the diagonal element is bigger in absolute value than the sum of all the other elements in absolute value in the row. In other words, the matrix is "close" to a diagonal matrix in a sense that the matrix $\mathbb{L} + \mathbb{U}$ containing the off diagonal terms is "smaller" than $\mathbb{D}$. For the speed of the convergence we simply state that the more diagonal dominance we have, the better.

Example 7.5: Solve the system $\begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}\mathbb{x} = \begin{pmatrix} 6 \\ 8 \end{pmatrix}$, using Jacobi iteration method.

Solution: The matrix $\mathbb{A}$ is clearly diagonal dominant, so start iterating with an initial guess $\mathbb{x} = \mathbf{0}$.

```
𝔻 = {4, 3}; 𝕃 = ( 0  0 ); 𝕌 = ( 0  1 ); 𝕓 = {6, 8}; 𝕩 = {0.0, 0.0};
                ( 2  0 )      ( 0  0 )
Do[Print[𝕩];
   𝕩 = (𝕓 - (𝕃 + 𝕌).𝕩) / 𝔻, {15}]
```

{0., 0.}

{1.5, 2.66667}

{0.833333, 1.66667}

{1.08333, 2.11111}

{0.972222, 1.94444}

{1.01389, 2.01852}

{0.99537, 1.99074}

{1.00231, 2.00309}

{0.999228, 1.99846}

{1.00039, 2.00051}

{0.999871, 1.99974}

{1.00006, 2.00009}

{0.999979, 1.99996}

{1.00001, 2.00001}

{0.999996, 1.99999}

```
Solve[({{4, 1}, {2, 3}}).{{x1}, {x2}} == {{6}, {8}}]
```

$\{\{x_1 \to 1, x_2 \to 2\}\}$

Another refined method is the Gauss-Seidel method, which uses $\mathbb{B} = \mathbb{L} + \mathbb{D}$, giving the iteration formula

$$\mathbb{x}^{(k+1)} = \mathbb{D}^{-1}\left(\mathbb{b} - \mathbb{L}\mathbb{x}^{(k+1)} - \mathbb{U}\mathbb{x}^{(k)}\right), \quad \text{for } k = 0,\ 1,\ 2,\ \ldots \tag{7.10}$$

This is very similar to Jacobi iteration, but in many cases works better. Roughly speaking, Jacobi considers $\mathbb{A} = \mathbb{L} + \mathbb{D} + \mathbb{U}$ as a perturbation of its diagonal $\mathbb{D}$, while Gauss-Seidel considers $\mathbb{A} = \mathbb{L} + \mathbb{D} + \mathbb{U}$ as a perturbation of $\mathbb{L} + \mathbb{D}$. It turns out that Gauss-Seidel usually is faster as it uses the "newest" elements of $\mathbb{x}$ along the road. By changing the method slightly, it can be polished up even further. A popular variant is the successive overrelaxation (SOR). This is taking care of that $\mathbb{x}^{(k+1)}$ might be an "over-/undershoot", and speeding up the process by making an weighted average of the previous point and the new one, with a relaxation parameter ω

$$\mathbb{x}^{(k+1)} = \omega\mathbb{x}^{(k)} + (1 - \omega)\mathbb{x}^{(k+1)} \tag{7.11}$$

If $0 < \omega < 1$ we say under relaxation, and for $\omega > 1$ overrelaxation. For $\omega > 2$ the process usually diverge.

7.2.2.2 Gradient methods

Gradient based iterative methods are essentially based on minimizing the sum of squares $(\mathbb{A}\mathbb{x} - \mathbb{b})^T(\mathbb{A}\mathbb{x} - \mathbb{b})$. This can be done very cheap because the algorithms usually only requires the *vector* $\mathbb{A}\mathbb{x}$. If $\mathbb{A}$ is large, this is a dramatic reduction of memory requirements in the computer compared to $\mathbb{A}_{n\times n}$ itself, n compared to n^2 numbers. For example, this situation exists when solving partial differential equations with the so-called finite element method where $\mathbb{A}\mathbb{x}$ is easier to calculate than $\mathbb{A}$.

The underlying idea is simple as before. Guess a solution $\mathbb{x}^{(0)}$, and iterate $\mathbb{x}^{(k)}$ until desired convergence. A simple member of this class of gradient methods is usually called steepest descent and can be easily described with an analogy from real life.

Suppose you are on in the Alps and want to go down to the bottom of the valley (minimization!). Due to fog, the view is limited to a few meters. A strategy is then to go in the direction that leans the most downwards, negative gradient. Continue to go straight in this direction until it starts leaning upwards. Take a new direction downwards. Repeat (iterate) until you reach a point where it is uphill in all directions. You are at the destination!

We now dress this strategy in a mathematical costume. As pointed out, the steepest descent method is simply a method of searching the local minimum of a function. The strategy is to move from the current point $\mathbb{x}^{(k)}$ in the negative gradient direction $\mathbb{r}^{(k)} = \mathbb{A}\mathbb{x}^{(k)} - \mathbb{b}$. With a little help from the figure below you remember perhaps that this means in a direction perpendicular to the level curve you are on. The optimum step in this direction is to go as long as the function decreases. This is given by $\alpha_{k+1}\mathbb{r}^{(k)}$, where α_{k+1} is the step size. From this new point, $\mathbb{x}^{(k+1)} = \mathbb{x}^{(k)} + \alpha_{k+1}\mathbb{r}^{(k)}$, a new direction is determined, and so on until it reaches a point where no negative gradient direction can be found. We have come to the point of minimum. The method works especially well on square functions, as is the case here. The step length $\alpha_{k+1} = \frac{\mathbb{r}^{(k)}\mathbb{r}^{(k)}}{\mathbb{r}^{(k)}\mathbb{A}\mathbb{r}^{(k)}}$, whose structure we do not immerse ourselves into for the moment, is designed to be optimal in this case, that is, we take the entire downhill trip in one step only. Knowing that a quadratic function only has a single local and thus a global minimum increases safety further.

The vector $\mathbb{r}^{(k)}$ is usually referred to as the residual (error) or residual vector because it's precisely the difference between the left and right hand sides of the system of equations. So, it also represents a measure of the quality of the solution because $|\mathbb{r}|^2 = \mathbb{r}\cdot\mathbb{r} \to 0$ when we have convergence. This is illustrated in the following example.

Example 7.6: Use the steepest decent method to solve the system $\begin{pmatrix} 4 & 2 \\ 2 & 7 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 20 \\ 22 \end{pmatrix}$.

Solution: Start pessimistic, as usual, with $\mathbb{x}^{(0)} = (0, 0)^T$.

```
𝔸 = {{4, 2}, {2, 7}}; 𝕓 = {20, 22}; 𝕩 = {0.0, 0.0}; track = {𝕩}; 𝕣 = {1, 1};
While[𝕣.𝕣 > 10^-6,
  𝕣 = 𝕓 - 𝔸.𝕩;
  α = 𝕣.𝕣 / (𝕣.𝔸.𝕣);
  𝕩 = 𝕩 + α 𝕣;
  Print[{𝕩, α, 𝕣, α 𝕣, 𝕣.𝕣}];
  AppendTo[track, 𝕩];
]
```

{{2.62004, 2.88204}, 0.131002, {20., 22.}, {2.62004, 2.88204}, 884.}

{{3.73566, 1.86783}, 0.297043, {3.75578, −3.41434}, {1.11563, −1.01421}, 25.7636}

{{3.90881, 2.05829}, 0.131002, {1.32168, 1.45385}, {0.173143, 0.190457}, 3.86052}

{{3.98253, 1.99127}, 0.297043, {0.248197, −0.225634}, {0.0737253, −0.067023}, 0.112513}

{{3.99397, 2.00385}, 0.131002, {0.0873422, 0.0960764}, {0.011442, 0.0125862}, 0.0168593}

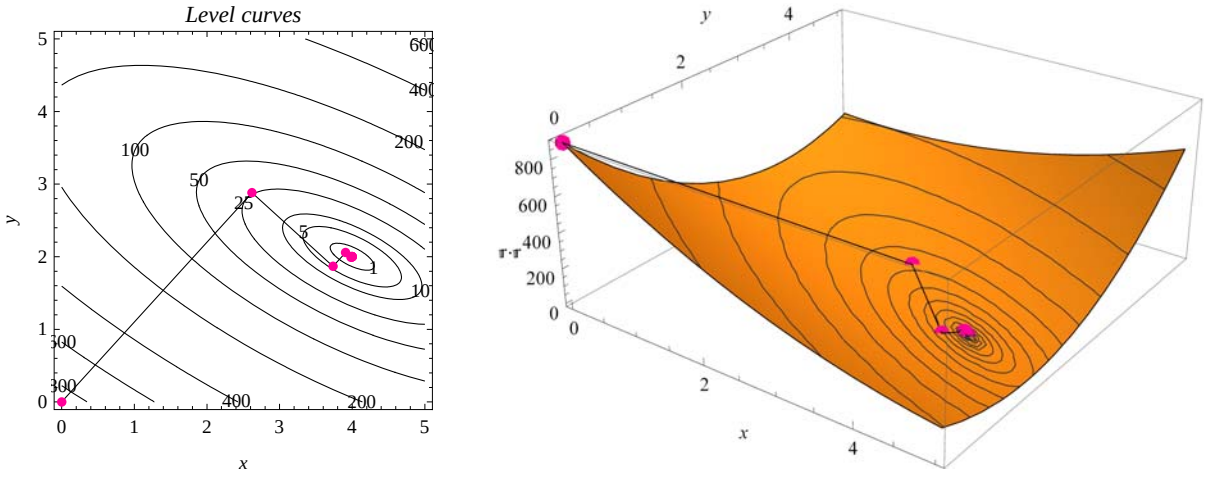
{{3.99885, 1.99942}, 0.297043, {0.0164019, −0.0149108}, {0.00487207, −0.00442915}, 0.000491354}

{{3.9996, 2.00025}, 0.131002, {0.00577193, 0.00634912}, {0.000756133, 0.000831746}, 0.0000736265}

$\{\{3.99992, 1.99996\}, 0.297043, \{0.0010839, -0.000985368\}, \{0.000321966, -0.000292697\}, 2.1458\times10^{-6}\}$

$\{\{3.99997, 2.00002\}, 0.131002, \{0.000381433, 0.000419576\}, \{0.0000499684, 0.0000549652\}, 3.21535\times10^{-7}\}$

And here is the downhill race to the bottom of the valley.



You can clearly follow the way down the valley with optimal step size. The footprints are marked with red dots in the figure. Please note that we are essentially at home after a very few iterations, which is

```
Solve[𝔸.{x, y} == 𝕓]
```

{{x → 4, y → 2}}

■

7.3 Inverse matrix

If $\mathbb{A}$ is a square matrix and there exist a (square) matrix $\mathbb{B}$ such that $\mathbb{AB} = \mathbb{BA} = \mathbb{I}$ we say that $\mathbb{A}$ is invertible and the matrix $\mathbb{B}$ is called the inverse of $\mathbb{A}$, and denote it $\mathbb{A}^{-1}$. Thus

$$\mathbb{A}\mathbb{A}^{-1} = \mathbb{A}^{-1}\mathbb{A} = \mathbb{I}$$

One can show that if $\mathbb{A}^{-1}$ exists it's unique. Given an inverse $\mathbb{B}$ to $\mathbb{A}$ and another one $\mathbb{C}$ also claiming to take this role, we have $\mathbb{C} = \mathbb{CI} = \mathbb{C}(\mathbb{AB}) = (\mathbb{CA})\mathbb{B} = \mathbb{IB} = \mathbb{B}$, stating $\mathbb{C} = \mathbb{B}$ indeed. If $\mathbb{A}$ is invertible we have $|\mathbb{A}| \neq 0$.

Using the inverse it's easy to write down the symbolical solution $\mathbb{X}$ to a matrix equation $\mathbb{AX} = \mathbb{B}$. Just premultiply with $\mathbb{A}^{-1}$

$$\mathbb{AX} = \mathbb{B} \Leftrightarrow \mathbb{A}^{-1}\mathbb{AX} = \mathbb{A}^{-1}\mathbb{B} \Leftrightarrow \mathbb{IX} = \mathbb{A}^{-1}\mathbb{B} \Leftrightarrow \mathbb{X} = \mathbb{A}^{-1}\mathbb{B}.$$

In *Mathematica* is the inverse calculated by the function **`Inverse`**.

```
Inverse[{{3, 2, -1}, {4, 6, 5}, {2, 7, 3}}]
```

$$\begin{pmatrix} \frac{17}{71} & \frac{13}{71} & -\frac{16}{71} \\ \frac{2}{71} & -\frac{11}{71} & \frac{19}{71} \\ -\frac{16}{71} & \frac{17}{71} & -\frac{10}{71} \end{pmatrix}$$

A matrix of type 2×2 should be mastered by hand,

$$\mathbb{A}^{-1} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}^{-1} = \frac{1}{|\mathbb{A}|}\begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}}\begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}$$

A simple calculation verifies it

```
1/(a11 a22 - a12 a21) {{a22, -a12}, {-a21, a11}} . {{a11, a12}, {a21, a22}}
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

A symmetrical matrix $\mathbb{A}$ is called orthogonal if $\mathbb{A}^{\mathrm{T}}\mathbb{A} = \mathbb{I}$. For such a matrix $\mathbb{A}^{-1} = \mathbb{A}^{\mathrm{T}}$. A sufficient condition for $\mathbb{A}$ to be orthogonal is that the columns should be normalized and pairwise orthogonal. Such matrices are very common in CAD systems and similar graphical applications requiring fast rotation matrix operations, like computer games.

As for determinants, inverses are rare in practice. They are costly to find and are therefor mostly found in theoretical elaborations multiplied with another matrix or vector. So, facing the statement $\mathbb{x} = \mathbb{A}^{-1}\mathbb{b}$, it should be interpreted as a job for Gauss $\mathbb{Ax} = \mathbb{b}$.

If we really need to calculate $\mathbb{A}^{-1}$ it can be arranged by letting Gauss chew $\mathbb{Ax} = \mathbb{b}$, taking $\mathbb{b}$ as the columns of $\mathbb{I}$ one at a time, giving $\mathbb{x}$ as the corresponding column in $\mathbb{A}^{-1}$. Collected into one complete scheme we call it Gauss-Jordan method. Furnish the augmented matrix in a similar way $[\mathbb{A} \mid \mathbb{I}]$ and let Gauss process it using elementary row operations until $\left[\mathbb{I} \mid \mathbb{A}^{-1}\right]$. This is justified by $\mathbb{AX} = \mathbb{I} \Leftrightarrow$ $\Leftrightarrow \mathbb{A}^{-1}\mathbb{AX} = \mathbb{A}^{-1}\mathbb{I} \Leftrightarrow \mathbb{IX} = \mathbb{A}^{-1} \Leftrightarrow \mathbb{X} = \mathbb{A}^{-1}$.

Example 7.7: Find the inverse of $\mathbb{A} = \begin{pmatrix} 3 & 2 & -1 \\ 4 & 6 & 5 \\ 2 & 7 & 3 \end{pmatrix}$.

Solution: Furnish the augmented matrix $\left(\begin{array}{ccc|ccc} 3 & 2 & -1 & 1 & 0 & 0 \\ 4 & 6 & 5 & 0 & 1 & 0 \\ 2 & 7 & 3 & 0 & 0 & 1 \end{array}\right)$. Mimic Gaussian elimination, but this time for the complete column.

Column 1: Divide ⟦1⟧ with 3, then ⟦2⟧−4⟦1⟧, finally ⟦3⟧ − 2⟦1⟧:

$$\left(\begin{array}{ccc|ccc} 3 & 2 & -1 & 1 & 0 & 0 \\ 4 & 6 & 5 & 0 & 1 & 0 \\ 2 & 7 & 3 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & \frac{2}{3} & -\frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 4 & 6 & 5 & 0 & 1 & 0 \\ 2 & 7 & 3 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & \frac{2}{3} & -\frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{10}{3} & \frac{19}{3} & -\frac{4}{3} & 1 & 0 \\ 2 & 7 & 3 & 0 & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & \frac{2}{3} & -\frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{10}{3} & \frac{19}{3} & -\frac{4}{3} & 1 & 0 \\ 0 & \frac{17}{3} & \frac{11}{3} & -\frac{2}{3} & 0 & 1 \end{array}\right) \sim$$

Column 2: Divide $[\![2]\!]$ with $\frac{10}{3}$, then $[\![1]\!] - \frac{2}{3}[\![2]\!]$, finally $[\![3]\!] - \frac{17}{3}[\![2]\!]$:

$$\left(\begin{array}{ccc|ccc} 1 & \frac{2}{3} & -\frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 1 & \frac{19}{10} & -\frac{2}{5} & \frac{3}{10} & 0 \\ 0 & \frac{17}{3} & \frac{11}{3} & -\frac{2}{3} & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 0 & -\frac{8}{5} & \frac{3}{5} & -\frac{1}{5} & 0 \\ 0 & 1 & \frac{19}{10} & -\frac{2}{5} & \frac{3}{10} & 0 \\ 0 & \frac{17}{3} & \frac{11}{3} & -\frac{2}{3} & 0 & 1 \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 0 & -\frac{8}{5} & \frac{3}{5} & -\frac{1}{5} & 0 \\ 0 & 1 & \frac{19}{10} & -\frac{2}{5} & \frac{3}{10} & 0 \\ 0 & 0 & -\frac{71}{10} & \frac{8}{5} & -\frac{17}{10} & 1 \end{array}\right) \sim$$

Column 3: Divide $[\![3]\!]$ with $-\frac{71}{10}$, then $[\![1]\!] - \left(-\frac{8}{5}\right)[\![3]\!]$, finally $[\![2]\!] - \frac{19}{10}[\![3]\!]$:

$$\left(\begin{array}{ccc|ccc} 1 & 0 & -\frac{8}{5} & \frac{3}{5} & -\frac{1}{5} & 0 \\ 0 & 1 & \frac{19}{10} & -\frac{2}{5} & \frac{3}{10} & 0 \\ 0 & 0 & 1 & -\frac{16}{71} & \frac{17}{71} & -\frac{10}{71} \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{17}{71} & \frac{13}{71} & -\frac{16}{71} \\ 0 & 1 & \frac{19}{10} & -\frac{2}{5} & \frac{3}{10} & 0 \\ 0 & 0 & 1 & -\frac{16}{71} & \frac{17}{71} & -\frac{10}{71} \end{array}\right) \sim \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{17}{71} & \frac{13}{71} & -\frac{16}{71} \\ 0 & 1 & 0 & \frac{2}{71} & -\frac{11}{71} & \frac{19}{71} \\ 0 & 0 & 1 & -\frac{16}{71} & \frac{17}{71} & -\frac{10}{71} \end{array}\right)$$

The inverse can now be extracted as $\mathbb{A}^{-1} = \begin{pmatrix} \frac{17}{71} & \frac{13}{71} & -\frac{16}{71} \\ \frac{2}{71} & -\frac{11}{71} & \frac{19}{71} \\ -\frac{16}{71} & \frac{17}{71} & -\frac{10}{71} \end{pmatrix}$.

Indeed

```
Inverse[{{3, 2, -1}, {4, 6, 5}, {2, 7, 3}}]
```

$$\begin{pmatrix} \frac{17}{71} & \frac{13}{71} & -\frac{16}{71} \\ \frac{2}{71} & -\frac{11}{71} & \frac{19}{71} \\ -\frac{16}{71} & \frac{17}{71} & -\frac{10}{71} \end{pmatrix}$$

```
Solve[{{3, 2, -1}, {4, 6, 5}, {2, 7, 3}}.{x, y, z} == #] & /@ IdentityMatrix[3]^T
```

$$\left(\{x \to \tfrac{17}{71}, y \to \tfrac{2}{71}, z \to -\tfrac{16}{71}\} \quad \{x \to \tfrac{13}{71}, y \to -\tfrac{11}{71}, z \to \tfrac{17}{71}\} \quad \{x \to -\tfrac{16}{71}, y \to \tfrac{19}{71}, z \to -\tfrac{10}{71}\}\right)$$

7.4 Eigenvalues and eigenvectors

Let $\mathbb{A}$ be a given square matrix. We call

$$\mathbb{A}\mathbb{x} = \lambda\mathbb{x} \tag{7.12}$$

an eigenvalue problem, where λ, a real or complex number, is called an eigenvalue of $\mathbb{A}$ and $\mathbb{x}$ an eigenvector according to λ. The prefix "eigen" is German and can be translated as "proper", which stems from the older nomenclature "proper values". In this context we ignore the uninteresting, so called trivial solution, that $\mathbb{x}$ is the zero vector. Furthermore, we consider complex eigenvalues a little bit odd because $\lambda\mathbb{x}$ is not in the same space as $\mathbb{x}$. Thus, with the language of vectors, one can say that an eigenvector $\mathbb{x}$ to $\mathbb{A}$ is a vector that, after the transformation $\mathbb{A}\mathbb{x}$ is parallel to itself, that is, $\mathbb{A}\mathbb{x}//\mathbb{x} \Rightarrow \mathbb{A}\mathbb{x} = \lambda\mathbb{x}$. Eigenvalue analysis is used very diligent in many applications. For example analysis of resonance in mechanical/electrical systems and in Google's page-ranking algorithm. Here is a simple observation of the linearity in (7.12) that is often used

If $\mathbb{x}_1, \mathbb{x}_2, \ldots$ are eigenvectors to $\mathbb{A}$ for the same eigenvalue λ, then a linear combination of them is also an eigenvector to $\mathbb{A}$ for the same eigenvalue λ, as $\mathbb{A}(\sum s_i\mathbb{x}_i) = \mathbb{A}s_1\mathbb{x}_1 + \mathbb{A}s_2\mathbb{x}_2 + \ldots = \lambda s_1\mathbb{x}_1 + \lambda s_2\mathbb{x}_2 + \ldots = \lambda(\sum s_i\mathbb{x}_i)$. (7.13)

After rearranging (7.12) we get a homogeneous system of equations

$$\mathbb{A}\mathbb{x} = \lambda\mathbb{x} \Leftrightarrow (\mathbb{A} - \lambda\mathbb{I})\mathbb{x} = \mathbb{0}$$

A necessary and sufficient condition for this system to have non-trivial solutions is that the determinant of the coefficient matrix is zero. This is called the characteristic equation and is a polynomial equation of degree n in λ if $\mathbb{A}$ is of order n.

$$|\mathbb{A} - \lambda\mathbb{I}| = 0 \tag{7.14}$$

This equation always has n roots if counted with multiplicity. The set of all the n eigenvalues is called the spectrum of $\mathbb{A}$.

Example 7.8: Determine the eigenvalues and corresponding eigenvectors to $\begin{pmatrix} 3 & 2 \\ 1 & 2 \end{pmatrix}$.

Solution: Solve the characteristic equation (7.14), giving all the eigenvalues

$$\left|\begin{pmatrix} 3 & 2 \\ 1 & 2 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right| = 0 \Leftrightarrow \begin{vmatrix} 3-\lambda & 2 \\ 1 & 2-\lambda \end{vmatrix} = 0 \Leftrightarrow (3-\lambda)(2-\lambda) - 2\cdot 1 = 0 \Leftrightarrow \lambda^2 - 5\lambda + 4 = 0 \Rightarrow \lambda_1 = 1 \text{ and } \lambda_2 = 4$$

Eigenvector to $\lambda_1 = 1$. Let $\mathbb{x}_{e1} = \begin{pmatrix} x \\ y \end{pmatrix}$ giving $\begin{cases} (3-1)x + 2y = 0 \\ x + (2-1)y = 0 \end{cases} \Leftrightarrow \begin{cases} x + y = 0 \\ x + y = 0 \end{cases}$. The equations are parallel as expected. Otherwise it's wrong! We can now choose for example $x = 1$ and $y = -1$. One usually choose to have "nice" numbers or to be consistent and normalize the eigenvectors. So the first pair with eigenvalue and associated eigenvector $\{\lambda_1 = 1,\ \mathbb{x}_{e1} = (1,\ -1)^T\}$.

Eigenvector to $\lambda_2 = 4$. Let $\mathbb{x}_{e2} = \begin{pmatrix} x \\ y \end{pmatrix}$ giving $\begin{cases} (3-4)x + 2y = 0 \\ x + (2-4)y = 0 \end{cases} \Leftrightarrow \begin{cases} -x + 2y = 0 \\ x - 2y = 0 \end{cases}$. OK! For example $\{\lambda_2 = 4,\ \mathbb{x}_{e2} = (2,\ 1)^T\}$.

In *Mathematica* we have **`Eigensystem`** doing it all, or **`Eigenvalues`** and **`Eigenvectors`** for portion wise delivery.

```
Eigensystem[{{3, 2}, {1, 2}}]
```

```
{{4, 1}, {{2, 1}, {-1, 1}}}
```

■

Real symmetrical matrices are very common in various applications! The eigenvalues of a real symmetrical matrix are all real and the eigenvectors are pairwise orthogonal. (7.15)

A little more about eigenvectors and eigenvalues,

(1) If $\mathbb{A}$ has an eigenvector $\mathbb{x}$ with eigenvalue λ then $\mathbb{A}^n$ has the same eigenvector $\mathbb{x}$ with the eigenvalue λ^n.
(2) If $\mathbb{A}$ has an eigenvector $\mathbb{x}$ with eigenvalue λ then $(\mathbb{A} - s\mathbb{I})$ has the same eigenvector $\mathbb{x}$ with the eigenvalue $\lambda - s$, $s \in \mathbb{R}$.
(3) If $\mathbb{A}$ is invertible $\Leftrightarrow$ all $\lambda_i \neq 0$.
(4) If $\mathbb{A}$ has an eigenvector $\mathbb{x}$ with eigenvalue λ then $(\mathbb{A} - s\mathbb{I})^{-1}$, if it exists, has the same eigenvector $\mathbb{x}$ with the eigenvalue $(\lambda - s)^{-1}$, $s \in \mathbb{R}$.
(5) If all elements in $\mathbb{A}$ are positive, $\mathbb{A}$ has at least one positive eigenvalue and all components of the corresponding eigenvector is non–negative. (*Perrons theorem*).
(6) Let $C_i = \{z :\ |\, z - a_{ii} \,| \leq \sum_{i \neq j} |\, a_{ij} \,|\}$ be circles in the complex plane. Then all eigenvalues to $\mathbb{A}$ belongs to the union of C_i, that is in $\bigcup C_i$. (*Gerschgorins theorem*).
(7) $\mathbb{A}$ satisfies its characteristic equation. (*Cayley–Hamiltons theorem*). This strange result means that if the characteristic equation is $\lambda^n + c_{n-1}\lambda^{n-1} + \ldots + c_1\lambda + c_0 = 0$ then $\mathbb{A}^n + c_{n-1}\mathbb{A}^{n-1} + \ldots + c_1\mathbb{A} + c_0\mathbb{I} = \mathbb{O}$. (7.16)

Example 7.9: Investigate $\begin{pmatrix} 2 & 3 & 1 \\ 3 & -4 & 5 \\ 5 & 8 & 7 \end{pmatrix}$ according to Gerschgorin's theorem.

Solution: Gerschgorin's three circles using the recipe (7.17)(6) above. $C_1 = \{z :\ |\, z - 2 \,| \leq (3+1)\} = \{z :\ |\, z - 2 \,| \leq 4\}$, $C_2 = \{z :\ |\, z - (-4) \,| \leq (3+5)\} = \{z :\ |\, z - (-4) \,| \leq 8\}$ and finally $C_3 = \{z :\ |\, z - 7 \,| \leq (5+8)\} = \{z :\ |\, z - 7 \,| \leq 13\}$. The eigenvalues are

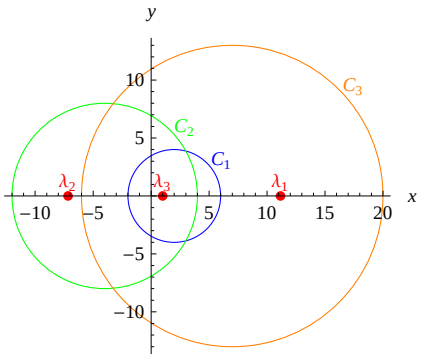
$$\left|\begin{pmatrix} 2 & 3 & 1 \\ 3 & -4 & 5 \\ 5 & 8 & 7 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right| = 0 \Leftrightarrow \begin{vmatrix} 2-\lambda & 3 & 1 \\ 3 & -4-\lambda & 5 \\ 5 & 8 & 7-\lambda \end{vmatrix} = 0 \Leftrightarrow \lambda^3 - 5\lambda^2 - 76\lambda + 80 = 0 \Leftrightarrow$$

$$(\lambda - 1)(\lambda^2 - 4\lambda - 80) = 0 \Rightarrow \lambda_1 = 2(1 + \sqrt{21}),\ \lambda_2 = 2(1 - \sqrt{21}) \text{ och } \lambda_3 = 1$$

```
Eigenvalues[{{2, 3, 1}, {3, -4, 5}, {5, 8, 7}}]
```

$$\left\{2\left(1+\sqrt{21}\right), 2\left(1-\sqrt{21}\right), 1\right\}$$

Now it's time to draw the picture.



Seems to be OK! All the eigenvalues λ_i to $\mathbb{A}$ are in the union $\bigcup_{i=1}^{3} C_i$.

So far we have obtained eigenvalues and eigenvectors for low order matrices, 2×2 and 3×3. This involved firstly solving the characteristic equation (7.14) for a given $n \times n$ matrix $\mathbb{A}$. This is an n:th order polynomial equation and, even for n as low as 3, solving it is not always straightforward. For large n even obtaining the characteristic equation may be difficult, and solving it is far beyond any hope of success if n is really big, say 10 million. There are many intricate direct methods around giving all eigenvalues and corresponding eigenvectors; QR with Householder or Gram-Schmidt orthonormalization, singular value decomposition...but, there are simpler alternatives.

Consequently we give a brief introduction to alternative iterative methods for obtaining eigenvalues and eigenvectors of $\mathbb{A}_{n\times n}$ one pair at a time. This is good news because we remember that in almost every application is the dominate eigenvalue, the largest in magnitude, of most interest. Thus, let us denote them in order of their magnitude, $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n|$.

7.4.1 The Power method

The Power method is designed to extract the dominate eigenvalue λ_1 and the corresponding eigenvector $\mathbb{x}$. That's right on the spot!

Let $\mathbb{x}^{(0)}$ be "almost" any non-zero start vector and

iterate for $k = 0, 1, 2, \ldots$

$$\mathbb{z}^{(k+1)} = \mathbb{A}\mathbb{x}^{(k)}$$
$$\lambda^{(k+1)} = \|\mathbb{z}^{(k+1)}\|_\infty$$
$$\mathbb{x}^{(k+1)} = \frac{1}{\lambda^{(k+1)}}\mathbb{z}^{(k+1)}$$

then $\lambda^{(k)} \to \lambda_1$ and $\mathbb{x}^{(k)} \to \mathbb{x}_{e1}$

Just for information we point out that the generated sequence $\mathbb{x}^{(0)}$, $\mathbb{x}^{(1)}$, … is the basis vectors that spans the so called Krylov subspace, $\mathcal{K}$. Many advanced methods start here for further sophistication. Namely, at a given iteration count k the best eigenvector so far is found as a linear combination of the basis functions in $\mathcal{K}_k$ extracted so far. It is a matter of speed to convergence. Rayleigh-Ritz, Arnoldi and Lanczos are typical methods that can be traced behind the curtain of many software packages, like **`eigs`** in Matlab. *Mathematica* has a bouquet of methods picking a suitable one depending on the question asked, matrix size, sparse/dense format and the computer hardware at hand. We will not dwell further into the vast amount of theory behind it, but give some important observations, which also holds for the variants of the power methods to come.

\+ The algorithm is surprisingly simple and easy to implement.
\+ The algorithm only needs the *vector* $\mathbb{A}\mathbb{x}^{(k)}$. This means that $\mathbb{A}$ need not be assembled before the multiplication with $\mathbb{x}^{(k)}$. This is a huge time and memory saving in many important applications, for instance in the finite element method.
\+ As for iterative methods in general, you can iterate to the desired accuracy you need for the moment.
\+ It gives only the eigenvector corresponding to λ_1 as well as λ_1 itself, although this is often the most important in applications. Many other iterative methods require separate calculation to obtain the eigenvector.
± It is possible to show that the speed of convergence depends on the ratio $\left|\frac{\lambda_2}{\lambda_1}\right|$, the smaller the faster. If this quantity is close to 1 the method is slow to converge.
– If $\mathbb{x}^{(0)}$ is an eigenvector of $\mathbb{A}$ other than that corresponding to the dominant eigenvalue then the method will fail since the iteration will converge to the wrong eigenvalue, say λ_m, after only one iteration, because $\mathbb{A}\mathbb{x}^{(0)} = \lambda_m\mathbb{x}^{(0)}$ in this case.
– If $\mathbb{x}^{(0)}$ resides in an invariant subspace of $\mathbb{A}$, there will be no convergence. So these two last observations, indicates that it is wise to choose $\mathbb{x}^{(0)}$ randomly.
– In particular if the dominant eigenvalue λ_1 is complex the method will fail completely to converge because the complex conjugate $\bar{\lambda}_1$ will also be an eigenvalue and $\lambda_1 = |\bar{\lambda}_1|$.

Example 7.10: Let us try it for the matrix in **Example 7.8**.

Solution: Follow the recipe just delivered.

```
𝔸 = ( 3  2 ); 𝕩 = RandomReal[{1, 2}, 2];
    ( 1  2 )
Do[𝕫 = 𝔸.𝕩;
 λ = MaximalBy[𝕫, Abs][[1]];
 𝕩 = 𝕫 / λ;
 Print[{i, λ, 𝕩}];
 , {i, 10}]
```

{1, 7.12569, {1., 0.609304}}

{2, 4.21861, {1., 0.52591}}

{3, 4.05182, {1., 0.506395}}

{4, 4.01279, {1., 0.501594}}

{5, 4.00319, {1., 0.500398}}

{6, 4.0008, {1., 0.500099}}

{7, 4.0002, {1., 0.500025}}

{8, 4.00005, {1., 0.500006}}

{9, 4.00001, {1., 0.500002}}

{10, 4., {1., 0.5}}

```
Eigensystem[𝔸]
```

$$\begin{pmatrix} 4 & 1 \\ \{2, 1\} & \{-1, 1\} \end{pmatrix}$$

■

7.4.2 The Inverse power method

This one is designed to find the smallest eigenvalue, in magnitude, and the associated eigenvector. Recall (7.16)(4) with $s = 0$. If $\mathbb{A}$ has a dominant eigenvalue λ_1 then its inverse $\mathbb{A}^{-1}$ has an eigenvalue λ_1^{-1}. Clearly λ_1 will be the smallest eigenvalue, in magnitude, of $\mathbb{A}^{-1}$. Conversely if we obtain the largest magnitude eigenvalue, say $\hat{\lambda}_1$, of $\mathbb{A}^{-1}$ by the power method then the smallest eigenvalue of $\mathbb{A}$ is $\lambda_n = \hat{\lambda}_1^{-1}$. So, it's all about using the Power method in a smart manner.

Let $\mathbb{x}^{(0)}$ be "almost" any non-zero start vector and

iterate for $k = 0, 1, 2, \ldots$

$$\mathbb{z}^{(k+1)} = \mathbb{A}^{-1}\mathbb{x}^{(k)}$$
$$\lambda^{(k+1)} = \|\mathbb{z}^{(k+1)}\|_\infty$$
$$\mathbb{x}^{(k+1)} = \frac{1}{\lambda^{(k+1)}}\mathbb{z}^{(k+1)}$$

then $\lambda^{(k)} \to \lambda_n^{-1}$ and $\mathbb{x}^{(k)} \to \mathbb{x}_{en}$

We have a number of times pointed out that $\mathbb{A}^{-1}$ should newer be used explicitly, so in most cases ought the first line in the iteration block be implemented as $\mathbb{A}\mathbb{z}^{(k+1)} = \mathbb{x}^{(k)}$, and solved by Gaussian elimination method. In fact preprocessed with LU decomposition as there are many right hand sides to consider, one for each iteration. It's really a matter of speed to convergence between the time consuming calculation of $\mathbb{A}^{-1}$ once before the iteration begins, and the number of iterations to desired convergence.

Example 7.11: Let us try it for the matrix in **Example 7.8** once again.

Solution: Follow the recipe just delivered.

```
𝔸 = ( 3  2 ); 𝕩 = RandomReal[{1, 2}, 2];
    ( 1  2 )
Do[𝕫 = Inverse[𝔸].𝕩;
 λ = MaximalBy[𝕫, Abs][[1]];
 𝕩 = 𝕫 / λ;
 Print[{i, λ, 𝕩, 1 / λ}];
 , {i, 10}]
```

{1, 0.671754, {0.00848184, 1.}, 1.48864}

{2, 0.74788, {−0.662886, 1.}, 1.33711}

{3, 0.915722, {−0.907965, 1.}, 1.09203}

{4, 0.976991, {−0.976449, 1.}, 1.02355}

{5, 0.994112, {−0.994077, 1.}, 1.00592}

{6, 0.998519, {−0.998517, 1.}, 1.00148}

{7, 0.999629, {−0.999629, 1.}, 1.00037}

{8, 0.999907, {−0.999907, 1.}, 1.00009}

{9, 0.999977, {−0.999977, 1.}, 1.00002}

{10, 0.999994, {−0.999994, 1.}, 1.00001}

```
Eigensystem[𝔸] // N
```

$$\begin{pmatrix} 4. & 1. \\ \{2., 1.\} & \{-1., 1.\} \end{pmatrix}$$

7.4.3 The Inverse power method with shift

This is used for obtaining the eigenvalue closest to a given number s. This time we rely on (7.16)(2)&(4). Suppose λ_i is the unknown sought eigenvalue of $\mathbb{A}$ *closest* to s. Then $\lambda_i - s$ will be the smallest, in absolute magnitude, to $\mathbb{A} - s\mathbb{I}$, and $(\lambda_i - s)^{-1}$ will be the largest eigenvalue, in absolute magnitude, to $(\mathbb{A} - s\mathbb{I})^{-1}$. Hence if we apply the power method to $(\mathbb{A} - s\mathbb{I})^{-1}$ we can obtain λ_i. The method is called the shifted inverse power method.

7.5 Problems

7.1 Three fluids are mixed in the volume ratio 1:2:4 giving the mixture the density 0.9. If the mixing ratio is 2:3:1 the density of the mixture becomes 0.8, and finally mixing 3:1:5 gives the density 1.1. Find the densities for the fluids involved in the experiment. Use **Solve**. $\{\{\rho_1 \to 1.41429,\ \rho_2 \to 0.3,\ \rho_3 \to 1.07143\}\}$

7.2 In order to find the temperature in a square formed plate we divide it into 4×4 smaller ones. The temperature in the plates along the boundary is known. Now, apply the model that the temperature for an interior plate is the mean value of its edge neighbors. Furnish a system of linear equations for this model and solve it for the requested temperatures T_i. We call it the finite difference method (FDM) applied to the Laplace differential equation, which is heavily used in many applications dealing with potential type models. Use **Solve** and visualize your findings with **ListContourPlot** as a fringe plot with labeled contour lines.

38	46	20	19
35	T_1	T_2	15
10	T_3	T_4	5
8	5	0	1

[°C]

7.3 Write a small snippet in *Mathematica* implementing the Gaussian elimination method. Test it on some of your favorite linear system of equations. Include the possibility to make educational printouts along the journey, as in the text above. Compare with **Solve**.

7.4 Consider the problem in **Example 7.5** and apply the Gauss-Seidel method (7.10). Try to speed up the convergence using relaxation with a suitable ω according to (7.11). Compare with the Jacobi method in the example.

7.5 Apply your snippet from **Problem 7.3** to invert a given square matrix $\mathbb{A}$ using Gauss-Jordan's method. Wrap it in one input cell prepared for eating $\mathbb{A}$. Test it on some of your favourite matrices. Include the possibility to make educational printouts along the journey, as in the text above. Compare with **Inverse**.

7.6 Let $\mathbb{A} = \begin{pmatrix} 2 & 4 & 1 & 2 \\ 4 & 6 & 8 & 3 \\ 1 & 8 & 4 & 6 \\ 2 & 3 & 6 & 5 \end{pmatrix}$ and use the three flavours of the power metod to extract the four eigenvalues and associated eigenvectors.

Let Gerschgorin help you to cover the range of search. Paint the Gerschgorin circles together with your findings. Compare with **Eigensystem**.

8. Introduction to Optimization

8.1 Motivation

The field of optimization encompasses the use of mathematical models and methods to find the "best alternative" in decision making situations. The word optimum originates from the Latin word "optimus" which means "best, very good". Optimization is the science of making the best possible decision. The typical optimization model consists of an objective function that depends on design variables. The task is to minimize/maximize this objective function according to the design variables. The motivations for such models are very strong and the application areas are vast, for example minimize weight, minimize energy consumption, or maximize profit.

If the design variables are free to take any value, we are talking about unconstrained optimization, but this is seldom the case in practice. In most cases there are also natural constraints to be taken into consideration. These can be on the design variables themselves or arising as special functions of the design variables. The intersection of these constraints forms the feasible region where the solution is to be sought. This more general case of optimization is called constrained optimization, and we use the phrase minimize/-maximize the objective function subject to (s.t.) the constraints. The functions in the models may be more or less nice to deal with. The complications in solving the models runs from fairly easy to hard when the objective function and the constraints goes from linear to nonlinear.

When we have a linear objective function and linear constraints, referred to as linear programming (LP-problems), the strategies has been driven very far with several advanced solution methods to possibly handle millions of design variables and constraints. Such models are very common in a variety of applications.

In the following chapters we will concentrate on more general models. Still, as for solving nonlinear equations, note that no guarantees can be given for a global optimum, or even a local optimum.

Example 8.1: Solve the problem $\max xy^2$

$$s.t. \begin{cases} x^2 + y^2 \leqslant 1 \\ x \geqslant 0,\ y \geqslant 0 \end{cases}$$

Solution: In words, maximize the objective function xy^2 subject to (s.t.) the three constraints $x^2 + y^2 \leqslant 1$, $x \geqslant 0$, and $y \geqslant 0$, which forms the first quadrant of the unit circle as the feasible region. *Mathematica* has by default a set of very powerful, and easy to use functions to deal with all types of (un)constrained optimization mentioned above. For other software packages, like Matlab you need to buy a special Optimization Toolbox.

```
Maximize[{x y^2, x^2 + y^2 <= 1, x >= 0, y >= 0}, {x, y}]
```

$$\left\{\frac{2}{3\sqrt{3}}, \left\{x \to \frac{1}{\sqrt{3}}, y \to \sqrt{\frac{2}{3}}\right\}\right\}$$

■

8.2 Some mathematical preliminaries

A frequent requirement for the functions involved will be that they are differentiable. For a function $f : R^n \to R$, we write $f \in C^k(\Omega)$ if f has k times continuous derivatives on $\Omega \subseteq R^n$. The directional derivative (Gâteaux derivative) of order k at point x in the normed direction y is defined as

$$\frac{d^k}{d\tau^k} f(x + \tau y)\,|_{\tau=0}$$

For $k = 1$, using the chain rule, we obtain $\frac{d}{d\tau} f(x + \tau y)\,|_{\tau=0} = \nabla f(x)^{\mathrm{T}} y$, where

$$G = \nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n}\right)^{\mathrm{T}} \tag{8.1}$$

is the gradient to f. If f is k times differentiable there exists a directed derivative of order k in all directions. We also define the hessian matrix H for f to be the matrix having the elements

$$h_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{8.2}$$

Example 8.2: Calculate G and H to the function $f(x_1, x_2) = 3x_1^2 + x_1x_2 - x_2^2$ at the point (2, 3), and the directional derivative in the direction (1, 2).

Solution: Define the function and use the definitions.

```
f = 3 x1^2 + x1 x2 - x2^2;
y = {1, 2};

G = D[f, {{x1, x2}}]
```

$\{6\,x_1 + x_2,\, x_1 - 2\,x_2\}$

```
G /. {x1 → 2, x2 → 3}
```

$\{15, -4\}$

```
H = D[f, {{x1, x2}, 2}]
```

$$\begin{pmatrix} 6 & 1 \\ 1 & -2 \end{pmatrix}$$

The directional derivative by chain rule, or by definition of directional derivative. Note that y must be a normalized direction.

```
G.y / Sqrt[y.y] /. {x1 → 2, x2 → 3}
```

$$\frac{7}{\sqrt{5}}$$

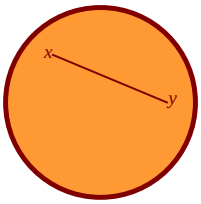
```
D[f /. Thread[{x1, x2} → {2, 3} + τ y/Sqrt[y.y]], τ] /. τ → 0
```

$$\frac{7}{\sqrt{5}}$$

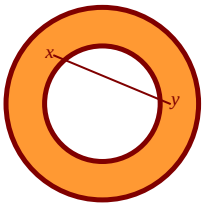
There are a lot of technicalities involved in the theory of optimization. Here are some definitions.

Definition. A matrix $A(x)$ of order $n \times n$ is positive semi-definite at x if $y^T A(x) y \geq 0$, $\forall y \in R^n$, and positive definite at x if $y^T A(x) y > 0$, $\forall y \in R^n$, $y \neq 0$. ■

Definition. The set $C \subseteq R^n$ is convex if $x, y \in C$, $\lambda \in [0, 1] \Rightarrow (1 - \lambda)x + \lambda y \in C$. That is, if all points on the line segment connecting two arbitrary points x and y in C lies in C.



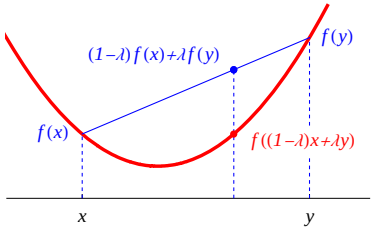
A convex set



Not a convex set

That is, if all points on the line segment connecting two arbitrary points x and y in C lies in C. ■

Definition. The function $f : C \to R$, C convex, is convex if $x, y \in C$, $\lambda \in [0, 1] \Rightarrow f((1 - \lambda)x + \lambda y) \leq (1 - \lambda) f(x) + \lambda f(y)$. That is, the function curve is under its chords. The counterpart in a multi-dimensional setting is that f is convex if we have $x, y \in C \Rightarrow f(y) \geq f(x) + \nabla f(x)^T (y - x)$. If we happens to have $<$ rather than $\leq$ we say strictly convex. ■



Theorem. If C_1 and C_2 are convex sets, then the intersection is also convex. ■

The function f is called concave if $-f$ is convex and we have similar definitions. Convex sets and functions play a crucial role when it comes to scrutinizing points as local or global optima. We have the following rule of thumb.

Necessary conditions for minimum becomes sufficient conditions if the sets and functions involved are convex.

8.3 Problem definition, conditions for local and global minima

In this chapter we will study the characteristics and conditions of solutions to general optimization problems

$$(G) \qquad \begin{aligned} &\min f(x) \\ &s.t.\ x \in F \end{aligned}$$

and mathematical optimization problem

$$(M) \qquad \begin{aligned} &\min f(x) \\ &s.t. \begin{cases} g(x) \le 0 \\ x \in X \end{cases} \end{aligned}$$

Any optimization problem can be written in the general form (G). For our needs, it is sufficient to consider $F \subseteq R^n$ and $f : F \to R$. The problem of maximizing f may of course be rewritten in terms of minimizing as $\max f = -\min(-f)$. Note that the direction of the relational operator in $g(x) \le 0$ is crucial for the theory to be consistent, thus, $g(x) \ge 0$ should be rewritten as $-g(x) \le 0$.

F is called the feasible region and f is the objective function. If $F = R^n$ we have unconstrained optimization, and if $F \subset R^n$ constrained optimization. A point $\hat{x}$ that solves (G) or (M) is called an optimal point and the corresponding function value $f(\hat{x})$ is the optimal value. In (G) we also talk about extreme point and extreme value. A point $\hat{x}$ where $\nabla f(\hat{x}) = 0$ is sometimes called a stationary point.

The form (M) is the typical prototype for mathematical optimization problems. We assume that $X \subseteq R^n$, $g : X \to R^m$ and $f : X \to R$. In (M) is the feasible region $F = \{x \in X \mid g(x) \le 0\}$ where $g(x) \le 0$ means $g_i(x) \le 0$ for $i = 1, \ldots, m$.

Definition. $\hat{x}$ is a global minimum of (G) if $f(\hat{x}) \le f(x)$, $\forall x \in F$. ■

Definition. $\hat{x}$ is a local minimum of (G) if $\exists \varepsilon > 0$ such that $f(x) \ge f(\hat{x})$, $\forall x \in F$ with $\|x - \hat{x}\| < \varepsilon$. ■

If we replace $\le$ with $<$ we have a strict minimum.

Definition. $d \in R^n$ is called a feasible direction of (G) at x if $\exists \tau_0 > 0$ such that $x + \tau d \in F$ for $0 < \tau < \tau_0$. ■

Definition. $d \in R^n$ is called a descent direction of f at x if $\exists \tau_1 > 0$ such that $f(x + \tau d) < f(x)$ for $0 < \tau < \tau_1$. ■

Theorem. If f is differentiable at x, then d is a descent direction at x if $\nabla f(x)^T d < 0$. ■

Theorem. There are no descent directions at $\hat{x}$. ■

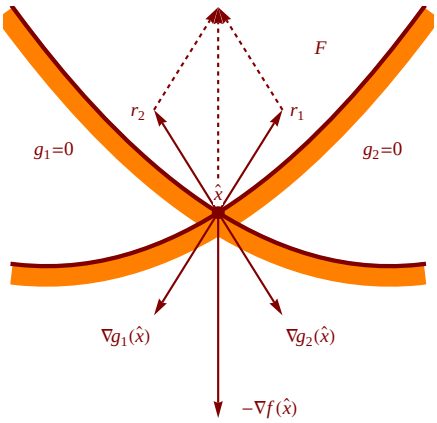
Theorem. Assume F and f convex and f differentiable. If for $\hat{x} \in F$ holds that $\nabla f(\hat{x})^T d \ge 0$ is a feasible directions d at $\hat{x}$ then $\hat{x}$ is a global minimum of (G). ■

For the mathematical problem

$$(M) \qquad \begin{aligned} &\min f(x) \\ &s.t. \begin{cases} g_i(x) \le 0,\ i = 1, \ldots, m \\ x \in X \end{cases} \end{aligned}$$

there exists special optimality conditions, the so called Kuhn-Tucker conditions (KT-conditions). We shall motivate them using a mechanical analogy.

Local minima to (M) can be interpreted as stable equilibrium points for a particle under the action of a potential field f in X. The freedom to move is constrained by smooth walls $V_i = \{x \mid g_i(x) = 0\}$, see figure



Now, let the particle be at rest at the point $\hat{x}$, within the feasible region $F \subset X$. It is supported by reaction forces r_i from the walls. These are perpendicular to the walls, and directed into F. The gradients ∇g_i are perpendicular as well, but pointing outwards. Thus, if $\nabla g_i(\hat{x}) \neq 0$ we have for $\lambda_i \geq 0$,

$$r_i = -\lambda_i \nabla g_i(\hat{x}),\ i = 1,\ \ldots,\ m$$

As we have forces from the walls in contact with the particle, it must hold that $\lambda_i = 0$ when $g_i(\hat{x}) \neq 0$, an inactive constraint,

$$\lambda_i g_i(\hat{x}) = 0,\ i = 1,\ \ldots,\ m$$

Finally, invoking force equilibrium

$$\nabla f(\hat{x}) = \sum_{i=1}^{m} r_i = -\sum_{i=1}^{m} \lambda_i \nabla g_i(\hat{x})$$

Putting things together forming the fundamental Kuhn-Tucker conditions

$$\begin{aligned} &\text{(KT, } i\text{)} \quad g_i(\hat{x}) \leq 0 \\ &\text{(KT, } ii\text{)} \quad \lambda_i g_i(\hat{x}) = 0 \\ &\text{(KT, } iii\text{)} \quad \lambda_i \geq 0 \\ &\text{(KT, } iv\text{)} \quad \nabla f(\hat{x}) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\hat{x}) = 0 \end{aligned} \tag{8.3}$$

These conditions are necessary for local minimum, with the exception of some pathological cases. However, our intention is not to dwell on concepts like constraints qualifications and Farkas lemma. These cases are beyond the scope of this introductory presentation.

9. Unconstrained Optimization

Consider the problem

$$\min f(x), \text{ where } f : R^n \to R$$

9.1 Conditions for minimum

The Kuhn-Tucker conditions (8.3) are called first order conditions, due to regularity conditions on the function f. We now add second order conditions.

Theorem. Necessary conditions for $\hat{x}$ to be a local minimum to f are
(1) $\nabla f(\hat{x}) = 0$
(2_N) The hessian matrix is positive semi-definite ■

Theorem. Sufficient conditions for $\hat{x}$ to be a strict local minimum to f are
(1) $\nabla f(\hat{x}) = 0$
(2_S) The hessian matrix is positive definite ■

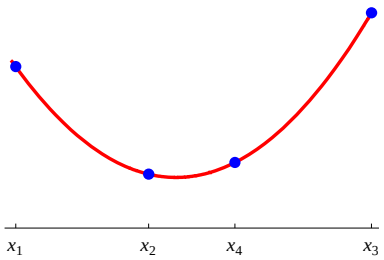
Theorem. If f is convex and $\nabla f(\hat{x}) = 0$ then $\hat{x}$ is a global minimum. ■

Clearly, every stationary point of a convex function is a global minimum.

9.2 Minimization of a function in one variable

9.2.1 Golden section search

The golden section search is a technique for finding the minimum of a strictly unimodal function by successively narrowing the range of the interval in which the extremum is known to exist. The technique derives its name from the fact that the algorithm maintains the function values for triples of points whose distances form a golden ratio.



The diagram above illustrates a single step in the technique for finding a minimum. The value of $f(x)$ has already been evaluated at the three points x_1, x_2 and x_3. Since $f(x_2)$ is smaller than either $f(x_1)$ or $f(x_3)$, it is clear that a minimum lies inside the interval from x_1 to x_3. The next step in the minimization process is to "probe" the function by evaluating it at x_4. It is most efficient to choose x_4 somewhere inside the largest interval, i.e. in this case between x_2 and x_3. From the diagram, it's almost clear that if $f(x_4) > f(x_2)$, then a minimum lies between x_1 and x_4, and the new triplet of points will be x_1, x_2, and x_4. However if $f(x_4) \le f(x_2)$, we pick the new triplet of points as x_2, x_4, and x_3. Thus, in either case, we can construct a new more narrow search interval that is guaranteed to contain the minimum. This forms the iterative process, until some terminating criterion, for example $|x_3 - x_1| < \varepsilon$.

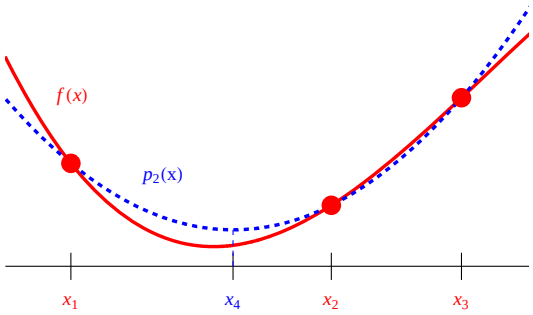
Now, if the location of the internal points x_2 and x_4 are cleverly chosen, only one new function evaluation need to be done in each iteration. The answer to the question of where they should be placed is given by the golden ratio φ.

$$\frac{x_2-x_1}{x_3-x_2} = \frac{x_3-x_2}{x_3-x_1} = \frac{x_3-x_4}{x_4-x_1} = \frac{x_4-x_1}{x_3-x_1} = \varphi = \frac{1}{2}\left(-1+\sqrt{5}\right) \Rightarrow \begin{cases} x_2 = \varphi\, x_1 + (1-\varphi)\, x_3 \\ x_4 = (1-\varphi)\, x_1 + \varphi\, x_3 \end{cases}$$

The golden ratio ensures that the relative position between the points are maintained from interval to interval throughout the algorithm, and therefore only $f(x_4)$ needs to be calculated, the other ones are inherited via bookkeeping. It is also guaranteed that the interval width shrinks by the same constant proportion φ in each iteration. This gives a substantial saving in the number of function calls, and no derivatives are required.

9.2.2 Polynomial approximation

This method utilizes the same techniques we used for differentiation i chapter 4.2. Assume that f is convex. Start by calculating f at three points $x_1 < x_2 < x_3$, selected such that the minimum is covered by the interval $[x_1, x_3]$. Then calculate the interpolation polynomial of order two for these three points. Finally approximate the minimum of f by the minimum of $p_2(x)$, at the point x_4, see figure.



One can show that (do it ;-)

$$x_4 = \frac{f_3\,(x_2^2-x_1^2)+f_2\,(x_1^2-x_3^2)+f_1\,(x_3^2-x_2^2)}{2\,(f_3\,(x_2-x_1)+f_2\,(x_1-x_3)+f_1\,(x_3-x_2))}$$

$$p_2(x_4) = -\Big(\big(f_2^2\,(x_1-x_3)^4 - 2\,f_2\,\big(f_3\,(x_1-x_2)^2 + f_1\,(x_2-x_3)^2\big)(x_1-x_3)^2 + \big(f_3\,(x_1-x_2)^2 - f_1\,(x_2-x_3)^2\big)^2\big)\Big/ \\ \big(4\,(x_1-x_2)\,(x_1-x_3)\,(x_2-x_3)\,(f_3\,(x_1-x_2) + f_1\,(x_2-x_3) + f_2\,(x_3-x_1))\big)\Big)$$

The method proceeds iteratively with shrinking interval size until a predefined stopping criterion, for example $|x_3 - x_1| < \varepsilon$. The three new points for the next interval are, of course, selected in such a way that the minimum stays inside that interval. In the figure above $x_1 < x_4 < x_2$, are selected as the new interpolation points. Note, that only one new function evaluation is needed for each iteration, if the bookkeeping is done properly, and that no derivative is required.

9.2.3 Bolzano search

Let f' exist and be continuous. Start with an interval

$$I = [a, b]$$

covering the minimum. Calculate $f'(c)$, where $c = \frac{1}{2}(a+b)$ is the midpoint of the interval. Now, if $|f'(c)| \le \varepsilon$, where ε is a predefined tolerance parameter, we accept c as minimum. If not, continue iteratively with

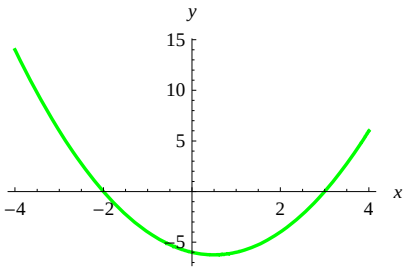
$$I = \begin{cases} [a, c] \text{ if } f'(c) > \varepsilon \\ [c, b] \text{ if } f'(c) \le \varepsilon \end{cases}$$

The method converges to a global minimum if f is convex, otherwise a local minimum if it's a stationary point.

Example 9.1: Minimize the function $y = f(x) = x^2 - x - 6$.

Solution: First function and a plot, then Bolzano search. Start with $[a, b] = [0, 1]$.

```
f[x_] := x^2 - x - 6
Plot[f[x], {x, -4, 4}, PlotStyle → Green, AxesLabel → {"x", "y"}]
```




```
{a, b} = {0, 1};
Do[c = 0.5 (a + b); Print[{a, c, b}]; If[f'[c] > 0, b = c, a = c], {6}]
```

{0, 0.5, 1}

{0.5, 0.75, 1}

{0.5, 0.625, 0.75}

{0.5, 0.5625, 0.625}

{0.5, 0.53125, 0.5625}

{0.5, 0.515625, 0.53125}

9.2.4 Newton's method

When f has two continuous derivatives, Newton's method can be used to search for a minimum. This is in analogy with Newton's method for solving nonlinear equations (6.3), $f'(x) = 0$. Iterate

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

The secant method is applicable, in the same manner.

Example 9.2: Minimize the function $y = f(x) = x^2 - x - 6$.

Solution: Same function as in previous example. Start Newton at $x_0 = -2$. Since f is quadratic we expect convergence in one step only!

```
x - f'[x]/f''[x] /. x → -2.0
```

0.5

9.3 Minimization of functions with several variables

A common class of methods are search methods, defined as the iteration

$$x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)},\ k = 0,\ 1,\ 2,\ \ldots \tag{9.1}$$

where $x^{(0)}$ is the starting point, α_k the step size and $d^{(k)}$ the search direction. Continue until $x^{(k+1)}$ is close to minimum in some sense, for instance $|x^{(k+1)} - x^{(k)}| < \varepsilon$, where ε is a predefined tolerance parameter.

The choice of α_k and $d^{(k)}$ defines different methods. Let's first take a look at α_k, which can be seen as a sub-problem independent of the choice of $d^{(k)}$.

If f is quadratic, α_k can be selected in an optimal way

$$\alpha_k = -\frac{\nabla f(x^{(k)})^T d^{(k)}}{d^{(k)} H d^{(k)}} \tag{9.2}$$

where H is the hessian matrix to f. As f is quadratic H is constant.

In a general setting α_k is to be found approximately. The problem to find α_k after the $d^{(k)}$ has been sorted out as a one-dimensional problem called line search. That is an unconstrained problem in one variable, and any of the methods from chapter 9.2 may be used for this purpose. The formulation is

$$\min_{\alpha} f\left(x^{(k)} + \alpha d^{(k)}\right) \tag{9.3}$$

9.3.1 Steepest descent (SD)

Let the search direction be given by the negative gradient, often together with the streetwise choice $\alpha_k = 1$.

$$d^{(k)} = -\nabla f\left(x^{(k)}\right) \tag{9.4}$$

9.3.2 Newton's method

Here we have

$$d^{(k)} = -H\left(x^{(k)}\right)^{-1} \nabla f\left(x^{(k)}\right) \text{ and } \alpha_k = 1. \tag{9.5}$$

The method is a direct generalization of Newton's method in one dimension, i.e. solving the nonlinear system of equations $\nabla f = 0$. If the hessian matrix H is non-singular we have quadratic convergence. Sometimes when the start point is far away from $\hat{x}$, it's wise to let $0 < \alpha_k < 1$ in the first iterations. This is the damped Newton's method. The same advice goes for the Steepest Descent method.

Newton's method for solving a system of nonlinear equations $f = 0$ in chapter 6.4 is the counterpart to Newton's method for unconstrained minimization of the length of the residual vector $\frac{1}{2}\| f \|^2$. Highly specialized methods has been developed for this type of (non)linear least squares problems, which are commonly used when fitting parameters in a mathematical model to data, recall the least squares method.

Example 9.3: Typical Newton downhill road trip in the Rosenbrock's banana valley $f(x, y) = (x + y)^2 + \left(2\left(x^2 + y^2 - 1\right) - 1\right)^2$.

Solution: First the valley. Then gradient vector and hessian matrix.

```
f = (x + y)^2 + (2 (x^2 + y^2 - 1) - 1)^2;
```

```
G = D[f, {{x, y}}]
```

$$\left\{8x\left(2\left(x^2 + y^2 - 1\right) - 1\right) + 2(x + y),\ 8y\left(2\left(x^2 + y^2 - 1\right) - 1\right) + 2(x + y)\right\}$$

```
H = D[f, {{x, y}, 2}]
```

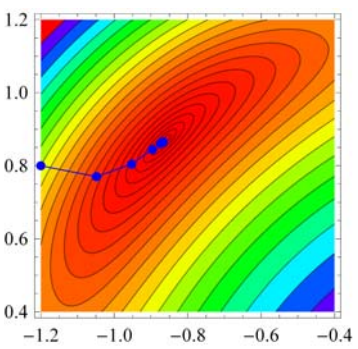
$$\begin{pmatrix} 32x^2 + 8\left(2\left(x^2 + y^2 - 1\right) - 1\right) + 2 & 32xy + 2 \\ 32xy + 2 & 32y^2 + 8\left(2\left(x^2 + y^2 - 1\right) - 1\right) + 2 \end{pmatrix}$$

Start at $(-1.2,\ 0.8)^T$, and apply the damped Newton's method, with $\alpha_k = 0.8$, in order to have a nice and smooth journey. Also, take the opportunity to GPS monitor the road through the countryside for later visualization.

```
{x, y} = {-1.2, 0.8}; route = {{x, y}};
Do[{x, y} = {x, y} - 0.8 Inverse[H].G; AppendTo[route, {x, y}], {6}]; route^T
```

$$\begin{pmatrix} -1.2 & -1.04846 & -0.952914 & -0.895856 & -0.873532 & -0.867628 & -0.866351 \\ 0.8 & 0.770187 & 0.804506 & 0.844932 & 0.860841 & 0.864926 & 0.865803 \end{pmatrix}$$

```
ContourPlot[f, {x, -1.2, -0.4}, {y, 0.4, 1.2},
 Contours → 6 (Range[0, 6, 0.25] / 6)^4, PlotPoints → 50, ColorFunction → Hue,
 Epilog → {Blue, Line[route], PointSize[0.03], Point /@ route}]
```



9.3.3 Conjugate gradient

In this case the search direction is a linear combination of the previous one and the current negative gradient.

$$d^{(k)} = -\nabla f\left(x^{(k)}\right) + \beta_k d^{(k-1)},\ k = 0,\ 1,\ 2,\ \ldots \tag{9.6}$$

where

$$\beta_k = \frac{\nabla f\left(x^{(k)}\right)^{\mathrm{T}} \nabla f\left(x^{(k)}\right)}{\nabla f\left(x^{(k-1)}\right)^{\mathrm{T}} \nabla f\left(x^{(k-1)}\right)},\quad \beta_0 = 0. \tag{9.7}$$

If $f : R^n \to R$ is quadratic the minimum is found after max n iterations.

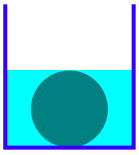
9.4 Problems

9.1 $\min_{x \in R^+} x\ln(x)$. Use **a)** Bolzano search, **b)** Newton's method. Visualize with **`Plot`**. Compare with **`Minimize`** and **`FindMinimum`**.
$\{-0.367879,\ \{x \to 0.367879\}\}$

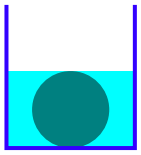
9.2 $\min_{x \in R^2} xy^2 + x^2 + 1.5y^2 - 2x + 3y$. Use method of steepest descent. Visualize with **`Plot3D`**. Compare with **`Minimize`** and **`FindMinimum`**. $\{-1.93846,\ \{x \to 0.784425,\ y \to -0.65662\}\}$

9.3 The linear system of equations $\mathbb{A}\mathbb{x} = \mathbb{b}$, can be solved by minimizing the sum of squares $\frac{1}{2}\|\mathbb{A}\mathbb{x} - \mathbb{b}\|^2$. Use this and Newton's method to solve $\begin{cases} 2x - 3y = 1 \\ 4x + 7y = 5 \end{cases}$. $\{x \to 0.846154,\ y \to 0.230769\}$

9.4 A cylindrical vessel has a base radius of 12 cm and a height exceeding 24 cm. A sphere of metal with a radius of less than 12 cm is placed at the bottom. Water is then poured into the vessel so that it just covers the sphere. Determine the maximum volume required. Use Bolzano search. $\{V \to 5118.2\}$



9.5 An open box with 10 cm square shaped bottom is filled with water to the depth of 4 cm. When a steel cylinder with the same diameter as height is put into the water it raises to just cover the cylinder. Find the dimensions and volume of the cylinder. Use Newton' s method.



9.6 An ordinary A4–paper has the width a and height $b = a\sqrt{2}$. The lower right corner is folded over so it just touches the left edge in such a way that the fold runs from the bottom edge to the right edge. How should this be done in order to minimize the length of the fold. Use Newton' s method. $\left\{L \to \frac{3\sqrt{3}\,a}{4}\right\}$

9.7 We want to minimize the cost of making a rectangular box with volume 2 m^3, given that its front and back cost \$1/m^2, top and bottom \$2/m^2, and two ends \$3/m^2. Find its length x, width y, and height z. Eliminate one of its dimensions using the volume constraint and let Newton's method solve the unconstrained minimization problem.
$\{\text{price} \to \$8.6535,\ \{x \to 2.08008,\ y \to 0.693361,\ z \to 1.38672\}\}$

9.8 A rectangle has two of its corners on the x-axis and the other two on the curve $y = x(1 - x)$, $0 < x < 1$. Find the dimensions that maximize the area. $\{\{0.096225,\ \{x \to 0.211325,\ y \to 0.166667\}\}$

9.9 Implement the polynomial approximation method and solve the task in **Problem 9.1**.

9.10 Show the results in chapter **9.2.2**.

10. Constrained Optimization

In this chapter we address the following class of problem

$$\begin{aligned} &\min f(x) \\ &s.t.\ x \in F = \{x \in R^n \mid g(x) \le 0\} \end{aligned}$$

where $f : R^n \to R$ and $g : R^n \to R^m$. Note that equality constraints $h(x) = 0$ can, at least mathematically, be included in the list of inequality constraints as $\{g_1(x) \le 0, \ldots, g_m(x) \le 0, h_1(x) \le 0, -h_1(x) \le 0, \ldots, h_q(x) \le 0, -h_q(x) \le 0\}$ and replace R^m by R^{m+2q}. This fiddle is of course not necessary in *Mathematica*, just mix $\le, \ge$ and = as you wish.

When both the object function and the constraints are linear, we are talking about linear programming, or a LP-problem

$$\begin{aligned} &\min \mathfrak{c}^T \mathfrak{x} \\ &s.t.\ \mathbb{A}\mathfrak{x} \le \mathfrak{b} \end{aligned}$$

where $\mathfrak{c}$, $\mathfrak{x}$, and $\mathfrak{b}$ are vectors, and $\mathbb{A}$ a matching matrix. This class of problem is very common in many applications, because different types of planning and scheduling are handled with this formulation. For example, scheduling air traffic or in which order a truck with goods should visit a number of shops to minimize the traveling cost. Because much money is involved, solution algorithms for this particular formulation have been driven far away into sophistication. Today, problems are routinely solved with millions of unknowns in $\mathfrak{x}$ and number of constraints, the number of rows in $\mathbb{A}$.

We will not immerse ourselves in specialized methods for solving linear programming problems, like Simplex and Karmarkar, but simply refer to *Mathematica*'s functions **`Minimize/Maximize`**, which directly identifies if LP is the case and solves them effectively with the most modern algorithms. We have in this booklet seen that most nonlinear problems are solved iteratively by linearization at the current point using Taylor expansion. So, there is linear programming behind the curtain, even in this section with nonlinear object function and nonlinear constraints.

10.1 Lagrange relaxation

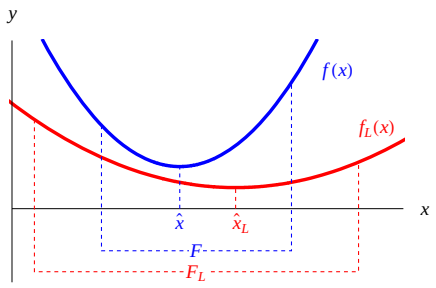
Relaxation means to transform, under certain conditions, the original problem to a simpler one. Consider the two problems

$$(G) \qquad \begin{aligned} &\min f(x) \\ &s.t.\ x \in F \end{aligned}$$

$$(G_L) \qquad \begin{aligned} &\min f_L(x) \\ &s.t.\ x \in F_L \end{aligned}$$

Definition. (G_L) is a relaxation to (G) if
(i) $F_L \supseteq F$
(ii) $f_L(x) \le f(x)$ when $x \in F$ ■

Rule of thumb, L in (i) and (ii) stands for "larger" and "lower", respectively, see figure for a relaxed situation.



Theorem. Suppose that (G_L) is a relaxation of (G). If $\hat{x}$ and $\hat{x}_L$ are solutions of (G) and (G_L), respectively, and $\hat{x}_L \in F$ then $f_L(\hat{x}_L) \le f(\hat{x}) \le f(\hat{x}_L)$. ■

Theorem. Suppose that (G""$_L$) is a relaxation of (G) and that $\hat{x}_L$ is a solution of (G_L). If
(i) $\hat{x}_L \in F$
(ii) $f_L(\hat{x}_L) = f(\hat{x}_L)$
then $\hat{x}_L$ is also a solution of (G). ■

For the mathematical problem

$$(M)\qquad \begin{aligned} &\min f(x)\\ &s.t.\ \begin{cases} g_i(x) \le 0,\ i = 1,\ \ldots,\ m\\ x \in X \end{cases} \end{aligned}$$

the so-called Lagrange relaxation is used, which means that we form the problem

$$(M_L)\qquad \begin{aligned} &\min f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)\\ &s.t.\ x \in X \end{aligned}$$

λ_i, $i = 1,\ \ldots,\ m$ are called Lagrange multipliers and the new objective function

$$f_L(x) = L(x,\ \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

is called the Lagrange function $L(x,\ \lambda)$. Note, that this indicates unconstrained minimization, and the λ_i are the same variables as in the Kuhn-Tucker conditions (8.3). When is (M_L) a relaxation to (M)?

Theorem. (M_L) is a relaxation of (M) if $\lambda \ge 0$. ■

For equality constraints in the mathematical optimization problem

$$(M')\qquad \begin{aligned} &\min f(x)\\ &s.t.\ \begin{cases} g_i(x) = 0,\ i = 1,\ \ldots,\ m\\ x \in X \end{cases} \end{aligned}$$

there is no sign restriction on λ for (M_L) to be a relaxation.

Theorem. (M_L) is a relaxation of (M') for every λ. ■

Using the general theorem above concerning relaxation, we recognize the Kuhn-Tucker conditions.

Theorem. Let $\hat{x}_L$ be a solution of (M_L). Then $\hat{x}_L$ is also a solution of (M) if
(i) $g_i(\hat{x}_L) \leqslant 0$
(ii) $\lambda_i\, g_i(\hat{x}_L) = 0$
(iii) $\lambda_i \ge 0$ ■

Suppose X and f_L are convex, then

$$\nabla f_L(\hat{x}_L) = 0 \Rightarrow \hat{x}_L \text{ is a solution of } (M_L).$$

If f and g_i are convex (and $\lambda \ge 0$) then f_L is also convex. Further on $\nabla f_L(x) = \nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x)$, so

$$\nabla f(\hat{x}_L) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\hat{x}_L) = 0 \Rightarrow \hat{x}_L \text{ is a solution of } (M_L).$$

Finally, the main result

Theorem. Suppose X, f and g_i are convex, and that f and g_i are differentiable. Then the Kuhn-Tucker conditions are sufficient for the point $\hat{x} \in X$ to be a solution (global minimum) of (M). ■

10.2 Methods for nonlinear constraints

The methods are divided into the Lagrange method and penalty methods.

10.2.1 Lagrange method for equality constraints

We have the problem

$$(L)\qquad \begin{aligned} &\min f(x)\\ &s.t.\ \begin{cases} g_i(x) = 0,\ i = 1,\ \ldots,\ m\\ x \in X \end{cases} \end{aligned}$$

and the associated Lagrange function $L(x,\ \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$ with no sign restriction on λ. In other words, the problem is transferred to the solution of a system of nonlinear equations,

$$\nabla_{x,\lambda} L(x,\ \lambda) = 0 \quad \Leftrightarrow \quad \begin{cases} \nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) = 0\\ g_i = 0 \end{cases} \tag{10.1}$$

and the Newton's method of chapter 6.4 is applicable.

Example 10.1: Find the point on the plane $x - y + 2z = 5$ which is closest to the origin.

Solution: We have the following problem to consider

$$(L) \qquad \begin{aligned} &\min f(x, y, z) = x^2 + y^2 + z^2 \\ &s.t.\ g(x, y, z) = x - y + 2z - 5 = 0 \end{aligned}$$

Apply the Lagrange formulation and solve (10.1) with no sign restriction on λ

$$\begin{cases} \nabla f + \lambda \nabla g = 0 \\ g = 0 \end{cases} \iff \begin{cases} 2x + \lambda = 0 \\ 2y - \lambda = 0 \\ 2z + 2\lambda = 0 \\ x - y + 2z - 5 = 0 \end{cases}$$

Solve the system of, in this particular case, linear equations.

```
ekv = D[x^2 + y^2 + z^2 + λ (x - y + 2 z - 5), {{x, y, z, λ}}] == 0
```

$\{\lambda + 2x, 2y - \lambda, 2\lambda + 2z, x - y + 2z - 5\} = 0$

```
Solve[ekv]
```

$$\left\{\left\{x \to \frac{5}{6}, y \to -\frac{5}{6}, z \to \frac{5}{3}, \lambda \to -\frac{5}{3}\right\}\right\}$$

This is in accordance to what we already know from linear algebra. The distance vector $\left(\frac{5}{6}, -\frac{5}{6}, \frac{5}{3}\right)^T$ is parallel to the normal of the plane $\left(\frac{5}{6}, -\frac{5}{6}, \frac{5}{3}\right)^T = \frac{5}{6}(1, -1, 2)^T = \frac{5}{6}\nabla g$.

Example 10.2: Solve the equality constrained minimization problem

$$\begin{aligned} &\min f = x^2 + y^2 + z^2 \\ &s.t. \begin{cases} g_1 = x + y + z - 1 = 0 \\ g_2 = x + 2y + 3z - 6 = 0 \end{cases} \end{aligned}$$

Solution: First his master's voice.

```
Minimize[{x^2 + y^2 + z^2, x + y + z - 1 == 0, x + 2 y + 3 z - 6 == 0}, {x, y, z}]
```

$$\left\{\frac{25}{3}, \left\{x \to -\frac{5}{3}, y \to \frac{1}{3}, z \to \frac{7}{3}\right\}\right\}$$

Then solve it using the Lagrange function $L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$ formulation

```
L = x^2 + y^2 + z^2 + λ1 (x + y + z - 1) + λ2 (x + 2 y + 3 z - 6);
vars = {x, y, z, λ1, λ2};
```

and, looking for the solution to the formulation (10.1), we need both the gradient vector G and the hessian matrix H.

```
G = D[L, {vars}]
```

$\{\lambda_1 + \lambda_2 + 2x, \lambda_1 + 2\lambda_2 + 2y, \lambda_1 + 3\lambda_2 + 2z, x + y + z - 1, x + 2y + 3z - 6\}$

```
H = D[L, {vars, 2}]
```

$$\begin{pmatrix} 2 & 0 & 0 & 1 & 1 \\ 0 & 2 & 0 & 1 & 2 \\ 0 & 0 & 2 & 1 & 3 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 \end{pmatrix}$$

Solve the system (10.1) of, in this particular case, linear equations in five unknowns.

```
Solve[H.vars == H.vars - G]
```

$$\left\{\left\{x \to -\frac{5}{3}, y \to \frac{1}{3}, z \to \frac{7}{3}, \lambda_1 \to \frac{22}{3}, \lambda_2 \to -4\right\}\right\}$$

Example 10.3: Solve, again, the equality constrained minimization problem

$$\min f = x^2 + y^2 + z^2$$
$$s.t. \begin{cases} g_1 = x + y + z - 1 = 0 \\ g_2 = x + 2y + 3z - 6 = 0 \end{cases}$$

Solution: We found that the hessian matrix H was constant. That means that (10.1) is a linear system of equations. This is of course not always the case, and we have to invoke a general optimizer, namely Newton's method. Constant H also indicate that the Lagrange function is quadratic, which is Newton's favorite lunch, promising convergence in one iteration only!

```
vars - Inverse[H].G /. Thread[Rule[vars, 1]]
```

$$\left\{-\frac{5}{3}, \frac{1}{3}, \frac{7}{3}, \frac{22}{3}, -4\right\}$$

Example 10.4: Consider the previous example once again, but now with (10.1) as a least squares problem $\frac{1}{2}||\nabla L||^2$.

$$\min f = x^2 + y^2 + z^2$$
$$s.t. \begin{cases} g_1 = x + y + z - 1 = 0 \\ g_2 = x + 2y + 3z - 6 = 0 \end{cases}$$

Solution: The quantity $S = \frac{1}{2}||\nabla L||^2 = \frac{1}{2}||G||^2$, is easily calculated as a scalar product, using G from **Example 10.2**.

```
S = 1/2 G.G
```

$$\frac{1}{2}\left((\lambda_1 + \lambda_2 + 2x)^2 + (x + y + z - 1)^2 + (x + 2y + 3z - 6)^2 + (\lambda_1 + 2\lambda_2 + 2y)^2 + (\lambda_1 + 3\lambda_2 + 2z)^2\right)$$

```
H = D[S, {vars, 2}]
```

$$\begin{pmatrix} 6 & 3 & 4 & 2 & 2 \\ 3 & 9 & 7 & 2 & 4 \\ 4 & 7 & 14 & 2 & 6 \\ 2 & 2 & 2 & 3 & 6 \\ 2 & 4 & 6 & 6 & 14 \end{pmatrix}$$

Solve the system (10.1) of, in this particular case, linear equations in five unknowns.

```
Solve[H.vars == H.vars - G]
```

$$\left\{\left\{x \to -\frac{5}{3}, y \to \frac{1}{3}, z \to \frac{7}{3}, \lambda_1 \to \frac{22}{3}, \lambda_2 \to -4\right\}\right\}$$

10.2.2 Penalty methods for nonlinear equality constraints

For the problem

$$(P_E) \qquad \begin{aligned} &\min f(x) \\ &s.t.\ g(x) = 0 \end{aligned}$$

where $f : R^n \to R$ and $g : R^n \to R^m$, we furnish a penalty function with a penalty parameter $r > 0$

$$P(x, r) = f(x) + \frac{1}{r} g(x)^T g(x). \tag{10.2}$$

The idea is to punish P if the constraints are not satisfied during the minimization process. As the penalty parameter $r \downarrow 0$ the penalty increases. In other words, the constrained problem is converted to an unconstrained one. Still, in general, the limiting process is not possible to achieve in practice. This means that we have to repeat the minimization with decreasing r until there is sufficiently small changes in the solution. That is

$$(P_k) \qquad \min_{x \in R^n} P(x, r_k)$$

for a sequence r_k, $r_k \downarrow 0$ as $k \to \infty$. Under certain soft conditions on f and g, we have that $\hat{x}^{(k)} \to \hat{x}$ as $r_k \downarrow 0$, where $\hat{x}$ is a strict local minimum to (P_E). It can also be shown that for the corresponding Lagrange parameters

$$\frac{2}{r_k} g(\hat{x}^{(k)}) \to \hat{\lambda},\ r_k \downarrow 0.$$

The great advantage with the penalty formulation is, of course, that we transferred the constrained minimization to an unconstrained one. There are still drawbacks, besides solving a sequence of problems, the process $r_k \downarrow 0$ will sooner or later cause the numerics to be ill-conditioned, due to large condition number for the hessian matrix.

Example 10.5: Solve the problem using penalty formulation

$$\begin{aligned} &\min \tfrac{1}{2}\left(x_2^2 - x_1^2\right) \\ &s.t.\ \ x_1 = 1 \end{aligned}$$

Solution: Form the penalty function (10.2)

$$P(x,\, r) = \tfrac{1}{2}\left(x_2^2 - x_1^2\right) + \tfrac{1}{r}(x_1 - 1)^2$$

and we have the gradient $P_x(x,\, r) = \begin{pmatrix} -x_1 + \frac{2\,(x_1 - 1)}{r} \\ x_2 \end{pmatrix}$ and hessian matrix $P_{xx}(x,\, r) = \begin{pmatrix} -1 + \frac{2}{r} & 0 \\ 0 & 1 \end{pmatrix}$. With $P_x\left(\hat{x}(r),\, r\right) = 0 \Leftrightarrow$ $\hat{x}(r) = \left(\frac{2}{2-r},\, 0\right)^{\mathrm{T}}$ and we can see that $\hat{x}(r)$ is a minimum when $r < 2$ since $P_{xx}(x,\, r)$ is positive definite. In this simple example

$$\lim_{r \downarrow 0} \hat{x}(r) = (1,\, 0)^{\mathrm{T}} = \hat{x}$$

and, if we wish

$$\lim_{r \downarrow 0} \tfrac{2}{r} g\left(\hat{x}(r)\right) = \lim_{r \downarrow 0} \tfrac{2}{r}\left(\tfrac{2}{2-r} - 1\right) = 1 = \hat{\lambda}$$

Finally, for the condition number $\kappa(P_{xx}) = -1 + \frac{2}{r} \to \infty$ as $r \downarrow 0$.

Example 10.6: Consider **Example 10.2** once again, but now with the penalty formulation.

$$\begin{aligned} &\min f = x^2 + y^2 + z^2 \\ &s.t. \begin{cases} g_1 = x + y + z - 1 = 0 \\ g_2 = x + 2\,y + 3\,z - 6 = 0 \end{cases} \end{aligned}$$

Solution: Form the penalty function (10.2). Also apply a "small" r.

```
P = x^2 + y^2 + z^2 + 1/r ((x + y + z - 1)^2 + (x + 2 y + 3 z - 6)^2) /. r → 10^-5
```

$$x^2 + 100\,000\left((x + y + z - 1)^2 + (x + 2\,y + 3\,z - 6)^2\right) + y^2 + z^2$$

And brute force using (10.1). Form gradient vector G and hessian matrix H.

```
G = D[P, {{x, y, z}}]
```

$$\{100\,000\,(2\,(x + y + z - 1) + 2\,(x + 2\,y + 3\,z - 6)) + 2\,x,\\ 100\,000\,(2\,(x + y + z - 1) + 4\,(x + 2\,y + 3\,z - 6)) + 2\,y,\ 100\,000\,(2\,(x + y + z - 1) + 6\,(x + 2\,y + 3\,z - 6)) + 2\,z\}$$

```
H = D[P, {{x, y, z}, 2}]
```

$$\begin{pmatrix} 400\,002 & 600\,000 & 800\,000 \\ 600\,000 & 1\,000\,002 & 1\,400\,000 \\ 800\,000 & 1\,400\,000 & 2\,000\,002 \end{pmatrix}$$

Solve the system (10.1) of, in this particular case, linear equations in three unknowns.

```
NSolve[H.{x, y, z} == H.{x, y, z} - G]
```

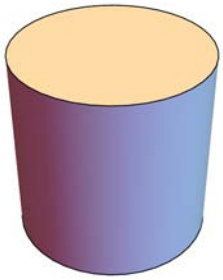
$$\{\{x \to -1.66661,\ y \to 0.333346,\ z \to 2.3333\}\}$$

This is in good agreement with previous results.

```
{x → -5/3, y → 1/3, z → 7/3} // N
```

$$\{x \to -1.66667,\ y \to 0.333333,\ z \to 2.33333\}$$

Example 10.7: Minimize the sheet metal needed to fabricate a cylindrical can of given volume $V = 3$.



Solution: We have the total area $A = 2\pi r^2 + 2\pi rh$, and the volume $V = \pi r^2 h$, furnishing the following optimization problem with a nonlinear objective function and nonlinear equality constraint.

$$\begin{aligned} &\min\ f(r, h) = 2\pi r^2 + 2\pi rh \\ &s.t.\ g(r, h) = \pi r^2 h - 3 = 0 \end{aligned}$$

Yes, yes, I know, simply eliminate h, and we have an unconstrained problem $f_*(r)$. However, our intention here is to exercise the penalty method for equality constraints. Rename the penalty parameter to λ, avoiding a name conflict with the cylinder radius. Form penalty function (10.2), and calculate the gradient vector G and hessian matrix H.

```
P = 2 π r^2 + 2 π r h + 1/λ (π r^2 h - 3)^2;
```

```
G = D[P, {{r, h}}]
```

$$\left\{\frac{4\pi h r\left(\pi h r^2-3\right)}{\lambda}+2\pi h+4\pi r,\ \frac{2\pi r^2\left(\pi h r^2-3\right)}{\lambda}+2\pi r\right\}$$

```
H = D[P, {{r, h}, 2}]
```

$$\begin{pmatrix} \frac{8h^2\pi^2 r^2}{\lambda}+\frac{4h\pi\left(h\pi r^2-3\right)}{\lambda}+4\pi & \frac{4h\pi^2 r^3}{\lambda}+\frac{4\pi\left(h\pi r^2-3\right)r}{\lambda}+2\pi \\ \frac{4h\pi^2 r^3}{\lambda}+\frac{4\pi\left(h\pi r^2-3\right)r}{\lambda}+2\pi & \frac{2\pi^2 r^4}{\lambda} \end{pmatrix}$$

Hessian is not constant, call Newton for unconstrained minimization. Let's start at the optimistic point $(r, h) = (0.8, 1.5)$, with $\lambda = 10^{-k}$, $k = 3, 4, 5$.

```
{r, h} = {0.8, 1.5};
Do[Print["Penalty λ = ", λ = 10.0^-k, ". Five Newton steps follows..."];
   Do[Print[{r, h} = {r, h} - Inverse[H].G], {5}]
 , {k, 3, 5}]
```

Penalty $\lambda = 0.001$. Five Newton steps follows…

{0.799155, 1.49464}

{0.779137, 1.56948}

{0.779811, 1.56967}

{0.781511, 1.56282}

{0.781484, 1.56295}

Penalty $\lambda = 0.0001$. Five Newton steps follows…

{0.781584, 1.56316}

{0.781582, 1.56316}

{0.781582, 1.56316}

{0.781582, 1.56316}

{0.781582, 1.56316}

Penalty $\lambda = 0.00001$. Five Newton steps follows…

{0.781592, 1.56318}

{0.781592, 1.56318}

{0.781592, 1.56318}

{0.781592, 1.56318}

{0.781592, 1.56318}

Let us compare with the results given by *Mathematica*.

```
NMinimize[{2 π r^2 + 2 π r h, 3 == π r^2 h, r > 0, h > 0}, {r, h}]
```

{11.5149, {$r \to 0.781586$, $h \to 1.56321$}}

```
AÅh = Solve[{A == 2 π r^2 + 2 π r h, V == π r^2 h}, {A, r}] // Last
```

$$\left\{A \to \frac{2\left(\sqrt{\pi}\ h^{3/2}\sqrt{V} + V\right)}{h},\ r \to \frac{\sqrt{V}}{\sqrt{\pi}\ \sqrt{h}}\right\}$$

```
hOpt = Solve[D[A /. AÅh, h] == 0, h] // Last
```

$$\left\{h \to \frac{2^{2/3}\sqrt[3]{V}}{\sqrt[3]{\pi}}\right\}$$

```
AÅh /. hOpt
```

$$\left\{A \to 3\sqrt[3]{2\pi}\ V^{2/3},\ r \to \frac{\sqrt[3]{V}}{\sqrt[3]{2\pi}}\right\}$$

```
{A, r, h} /. AÅh /. hOpt /. V → 3.0
```

{11.5149, 0.781593, 1.56319}

10.2.3 Penalty methods for nonlinear inequality constraints

For the problem

$$(P_I) \qquad \begin{array}{l} \min\ f(x) \\ s.t.\ g(x) \le 0 \end{array}$$

where $f : R^n \to R$ and $g : R^n \to R^m$, we furnish a similar penalty function with a penalty parameter $r > 0$ as we did in the equality constraint situation.

$$P(x, r) = f(x) + \frac{1}{r} g^+(x)^T g^+(x) \text{ where } g_i^+(x) = \begin{cases} g_i(x) \text{ if } g_i(x) > 0 \\ 0 \text{ otherwise} \end{cases} \tag{10.3}$$

In other words, we are punished if we are outside the feasible region. This type of penalty is therefore called outer penalty method, since we are approaching the minimum from the outside. If it's not possible to evaluate f or g outside the feasible domain, we can switch to an inner penalty method or barrier method. Examples of such functions are

$$P(x, r) = f(x) - r \sum_{i=1}^{m} g_i(x)^{-1} \tag{10.4}$$

or

$$P(x, r) = f(x) - r \sum_{i=1}^{m} \ln(-g_i(x)) \tag{10.5}$$

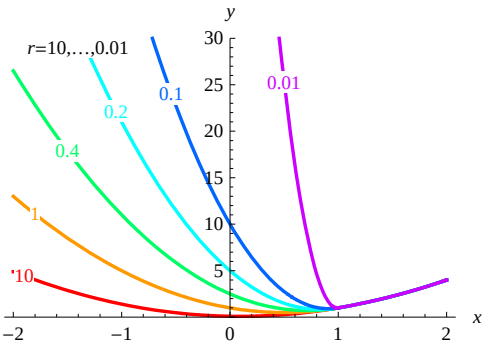
Also in this case $\hat{x}(r) \to \hat{x}$ when $r \downarrow 0$, where $\hat{x}(r)$ is the solution to (10.4) or (10.5). Further on

$$\frac{r}{g_i(\hat{x}(r))^2} \to \hat{\lambda}_i \text{ when } r \downarrow 0 \text{ in (10.4) and } -\frac{r}{g_i(\hat{x}(r))} \to \hat{\lambda}_i \text{ when } r \downarrow 0 \text{ in (10.5).}$$

Example 10.8: Illustrate the penalty functions for the problem

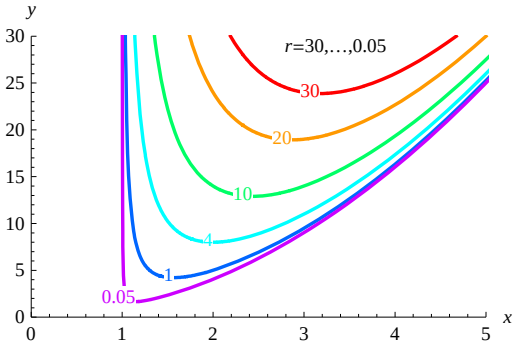
$$\begin{array}{l} \min\ x^2 \\ s.t.\ x \ge 1 \end{array}$$

Solution: Note that we have to rewrite the constraint $x \ge 1$ as $1 - x \le 0$ to fit our prototype formulation. As we can see in the figure below, $P(x, r)$ approaches its extreme point $\hat{x} = 1$ when $r \downarrow 0$.



$P(x, r) = f(x) + \frac{1}{r} g^+(x)^T g^+(x) = x^2 + \frac{1}{r}((1-x)^+)^2$

Outer penalty function according to (10.3)



$P(x, r) = f(x) - r \sum_{i=1}^{m} g_i(x)^{-1} = x^2 - \frac{r}{1-x}$

Inner penalty according to (10.4)

Indeed, the results indicated in the figure also hold analytically, and of course also for the inner penalty function (10.5) not visualized.

```
dPdx = D[x^2 - r Log[-(1 - x)], x]
```

$$2x - \frac{r}{x-1}$$

```
x̂ = Solve[dPdx == 0, x]
```

$$\left\{\left\{x \to \frac{1}{2}\left(1 - \sqrt{2r+1}\right)\right\}, \left\{x \to \frac{1}{2}\left(\sqrt{2r+1} + 1\right)\right\}\right\}$$

```
x̂ /. r → 0
```

$\{\{x \to 0\}, \{x \to 1\}\}$

Here $x = 1$ is the only feasible point.

Example 10.9: Solve the inequality constrained problem.

$$\min_{(x,y)\in R^+} (x-2)^4 + (x-2y)^2$$
$$s.t.\ x^2 - y \leqslant 0$$

Solution: Use for instance the inner penalty function (10.4), $P = (x-2)^4 + (x-2y)^2 - r\left(x^2 - y\right)^{-1}$. A damped Newton will do the job, in a similar way as in **Example 10.7**. Form inner penalty function P, calculate the gradient vector G and hessian matrix H.

```
P = (x - 2)^4 + (x - 2y)^2 - r (x^2 - y)^-1
```

$$-\frac{r}{x^2 - y} + (x - 2y)^2 + (x-2)^4$$

```
G = D[P, {{x, y}}]
```

$$\left\{\frac{2rx}{(x^2-y)^2} + 2(x-2y) + 4(x-2)^3, -\frac{r}{(x^2-y)^2} - 4(x-2y)\right\}$$

```
H = D[P, {{x, y}, 2}]
```

$$\begin{pmatrix} 12(x-2)^2 + \frac{2r}{(x^2-y)^2} - \frac{8rx^2}{(x^2-y)^3} + 2 & \frac{4rx}{(x^2-y)^3} - 4 \\ \frac{4rx}{(x^2-y)^3} - 4 & 8 - \frac{2r}{(x^2-y)^3} \end{pmatrix}$$

```
{x, y} = {0.8, 0.9};
Do[Print["Penalty r = ", r = 10.0^-k, ". Ten Newton steps follows..."];
   Do[{x, y} = {x, y} - 0.25 Inverse[H].G, {10}];
   Print[{x, y}]
 , {k, 1, 8}]
```

Penalty r = 0.1. Ten Newton steps follows…

{0.895085, 0.959428}

Penalty r = 0.01. Ten Newton steps follows…

{0.929553, 0.916959}

Penalty r = 0.001. Ten Newton steps follows…

{0.942495, 0.901183}

Penalty r = 0.0001. Ten Newton steps follows…

{0.943962, 0.896506}

Penalty r = 0.00001. Ten Newton steps follows…

{0.946028, 0.893754}

Penalty r = $1.\times10^{-6}$. Ten Newton steps follows…

{0.945756, 0.893905}

Penalty r = $1.\times10^{-7}$. Ten Newton steps follows…

{0.945533, 0.89418}

Penalty r = $1.\times10^{-8}$. Ten Newton steps follows…

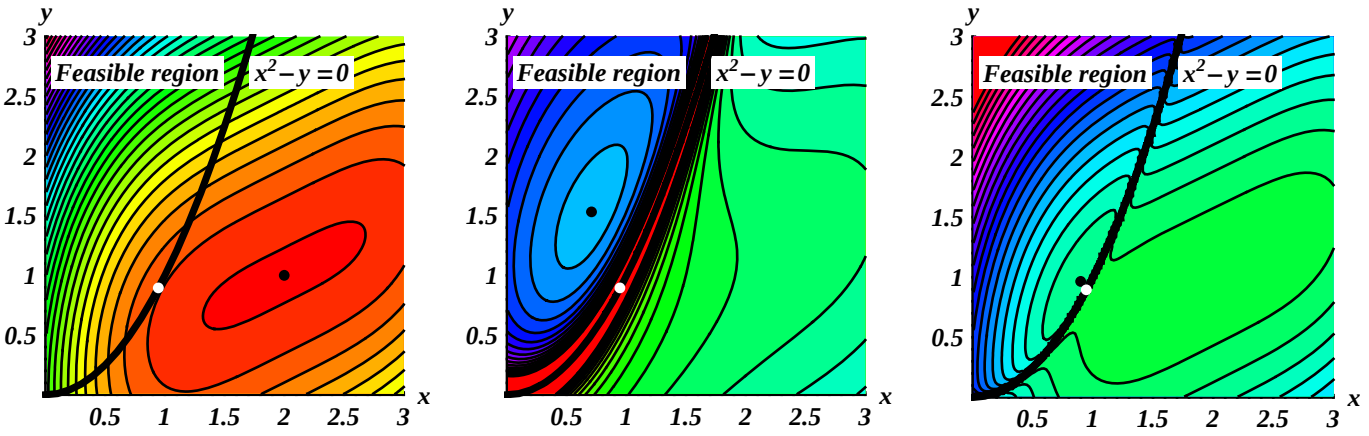
{0.945568, 0.894149}

Or why not give the last word to *Mathematica*.

```
NMinimize[{(x - 2)^4 + (x - 2 y)^2, x^2 - y <= 0, x > 0, y > 0}, {x, y}]
```

{1.94618, {$x \to 0.945583$, $y \to 0.894127$}}

A visualization of the scenario taking place close to the constraint follows. Clearly, the penalty term makes a nonlinear transformation of the map. The current minimum $\hat{x}(r)$, the black dot, initially not in the feasible region, moves from the inside as $r \downarrow 0$ towards the final local minimum, to fill the white dot gap in the fence. From the left figure it's also clear that (10.1) holds, the constraint curve is tangent to a level curve of f at $\hat{x}$; $\nabla f(\hat{x})$ is parallel to $\nabla g(\hat{x})$. As indicated in the right figure, this is also the case for $P(x, r)$ as $r \downarrow 0$.



Objective function | Penalty parameter $r = 10$ | Penalty parameter $r = 0.1$

10.3 Problems

In **10.1** and **10.2** use the methods presented and compare with **Minimize**.

10.1 Solve **a)** $\min x_1^2 + 3x_2^2 + 2x_2$ s.t. $x_1 + x_2 = 0$ **b)** $\min 3x + 4y$ s.t $x^2 + y^2 = 1$

a) {−0.25, {$x_1 \to 0.25$, $x_2 \to -0.25$}}
b) {−5., {$x \to -0.6$, $y \to -0.8$}}

10.2 Solve **a)** $\min 3x_1^2 + x_2^2 - 2x_1x_2$ s.t. $(x_1 - 2)^2 + (x_2 - 1)^2 \le 1$

b) $\min (x_1 - 2)^2 + (x_2 - 2)^2$ s.t. $\begin{cases} x_1 + x_2 \le 2 \\ 2x_1 \le 1 \end{cases}$

a) {2., {$x_1 \to 1.$, $x_2 \to 0.99998$}}
b) {2.5, {$x_1 \to 0.5$, $x_2 \to 1.5$}}

In the following use **Minimize/Maximize**.

10.3 Find the shortest distance from the point (0, 1) to the curve $y = x^2 - 2$. {1.65831, {$x \to 1.58114$}}

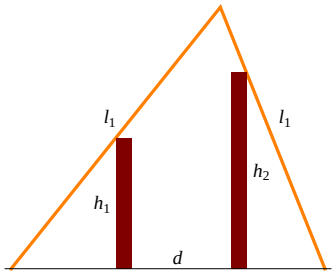
10.4 A sawmill is to cut out a beam, with rectangular cross section $b \times h$, from a circular log with diameter 0.5 m. Determine b and h that will maximize the bending moment of inertia $I = \frac{1}{12} b h^3$ for the beam. {0.00169146, {$b \to 0.25$, $h \to 0.433013$}}

10.5 A fish tank of volume $1\,\mathrm{m}^3$ with square bottom is to be constructed. Find the minimum amount of sheet glass needed. {4.7622, {$a \to 1.25992$, $h \to 0.629961$}}

10.6 A manufacturer want to minimize the cost of making a rectangular box with volume $2\,\mathrm{m}^3$, given that its front and back cost $\$1/\mathrm{m}^2$, top and bottom $\$2/\mathrm{m}^2$, and two ends $\$3/\mathrm{m}^2$. Find its length x, width y, and height z. {See **Problem 9.7**}

10.7 Find the dimensions for a circular can that maximize its volume for a given area of one unit.
{$V \to 0.0767765$, {$r \to 0.230329$, $h \to 0.460659$}}

10.8 The distance between two walls is d. Towards each of the walls, with heights h_1 and h_2, a ladder is leaned so that the ladders exactly meet each other, see figure below. Minimize the sum of the lengths of the ladders, $l_1 + l_2$. Data; $h_1 = 3$, $h_2 = 4$ and $d = 6$.



10.9 A student is trying to decide on lowest cost diet per day that provide a sufficient amount of vitamin, with six choices:

Product	1	2	3	4	5	6
Cost/Unit	2	3	4	1	2	3
Vitamin 1 IE/Unit	2	1	7	1	0	4
Vitamin 2 IE/Unit	3	0	2	8	6	4

A proper diet, require at least 50 IE of each vitamin per day. Formulate the LP-problem and find the optimal eating strategy.

10.10 At a paper mill, large paper rolls are produced. These are called tambours, typically of a few meters in width and a paper length of up to 30 km. When a tambour is produced it is moved to a machine which cuts it into smaller widths. These new rolls are called product rolls and have a typical customer demand. Each possible way to cut a tambour is called a pattern. One paper mill has concentrated on the following four patterns for their 3.9 m wide tambours.

Pattern	Widths [m]	Waste [m]
1	1.25, 1.0, 2×0.8	0.05
2	1.0, 3×0.8	0.5
3	2×1.25, 0.8	0.6
4	3×1.0, 0.8	0.1

The LP-problem is to choose cutting patterns minimizing the total number of tambours needed to satisfy a customer demand of rolls. One day a customer requested 35 of width 1.25 m, 171 of width 1.0 m and 133 of width 0.8 m. Help the paper mill company to sort out the cutting problem. Be sure that the number of tambours used are integer valued!

10.11 A major department in a hospital will make a rolling week schedule, one week periodic plan, for the nurses' night shift. The number of nurses needed each night is given by the attached table. Every nurse works three nights in a row and is then free for four nights. Determine a plan that meets the need and minimizes the number of nurses. That is to formulate the LP–problem and solve it. Is it overcapacity some night?

Night	Need
Sun	24
Mon	20
Tue	18
Wed	18
Thu	22
Fri	28
Sat	24

11. Begynnelsevärdesproblem

11.1 Introduktion

Med en differentialekvation (DE) menar vi en ekvation som innehåller en sökt funktion och dess derivator. Givetvis kan man ha ett system av (DE) likt ett vanligt ekvationssystem i linjär algebra. Högsta derivatan som förekommer anger ordningen på (DE). Om funktionen har endast en oberoende variabel kallas (DE) för en ordinär differentialekvation (ODE), annars partiell (PDE). Vi ska här koncentrera oss på en (ODE) och utgår från att den är alldeles för svår att lösa analytiskt med de metoder vi känner från en kurs i analys. Om vi låter $f(x,\, y) : \mathbb{R}^2 \to \mathbb{R}$ beteckna en given reellvärd funktion, har vi en första ordningens (ODE) på explicit form

$$y'(x) = f(x,\, y) \tag{11.1}$$

där $y(x)$ är den sökta funktionen, och x den oberoende variabeln. Eftersom (DE) används flitigt för att beskriva tidsberoende skeenden omkring oss, är den oberoende variabeln ofta tiden t. Vi låter *Mathematica* visa att (11.1) har oändligt med lösningar.

Exempel 11.1: Låt $f(x,\, y) = y$ och lös (ODE) $y'(x) = y(x)$ med **DSolve**.

```
DSolve[y'[x] == y[x], y[x], x]
```

$\{\{y(x) \to c_1\, e^x\}\}$

■

Uppenbarligen oändligt med lösningar beroende på val av konstanten c_1. Man får alltid lika många c_i som ordningen på (ODE). För att fixera en av dessa, och representera en lösning till vårt ingenjörsproblem, måste c_1 bestämmas av tillståndet i en ögonblicksbild. Detta väljs oftast till det man observerar då man börjar studera systemet, så kallat begynnelsevärde (BV). Därav namnet begynnelsevärdesproblem (BVP)

$$\text{(BVP)} \begin{cases} y'(x) = f(x,\, y) & \text{(ODE)} \\ y(x_0) = y_0 & \text{(BV)} \end{cases} \tag{11.2}$$

eller då vi har ett system av dem

$$\text{(BVP)} \begin{cases} y'(x) = f(x,\, y,\, v), & y(x_0) = y_0 \\ v'(x) = f(x,\, y,\, v), & v(x_0) = v_0 \\ \quad\vdots \end{cases} \tag{11.2'}$$

Dessa löses parallellt med de metoder som beskrivs nedan för att lösa (11.2).

Exempel 11.2: Vi tar **Exempel 11.1** igen med (BV) $y(0) = 1$.

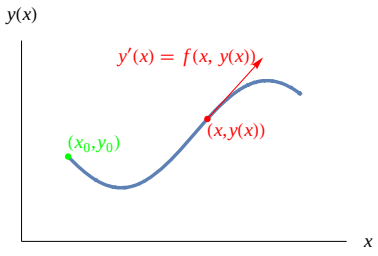
```
DSolve[{y'[x] == y[x], y[0] == 1}, y[x], x]
```

$\{\{y(x) \to e^x\}\}$

■

Först som sist; *Mathematica* har ett helt batteri av metoder i **DSolve** för direkt analytisk lösning av såväl linjära som en del olinjära, högre ordningens (ODE) samt system av dem. Även (PDE) kan hanteras. Då det blir för svårt får man även här tillgripa den rent numeriska lösaren **NDSolve** som är väl rustad med bland annat de metoder som omnämns nedan. Vid behov byter den dessutom mellen dessa under resans gång.

Vårt intresse i detta kapitel är att studera olika numeriska metoder för att lösa (11.2), det vill säga att söka numeriska approximationer till lösningskurvan $y(x)$, $x \geq x_0$. Vi säger att vi söker en numerisk lösning till (11.2). Situationen åskådliggörs i figuren nedan, efter det att vi kommit "liten" bit på vägen.



Det finns omfattande teori kring (ODE). Från existens av lösningar till något lite om konstruktionen av dem. Speciellt svårt blir det ju mer olinjär f är.

Sats. Om $f(x,\, y)$ är kontinuerlig så existerar en lösning till $y'(x) = f(x,\, y)$. (Augustin Louis Cauchy, 1789-1857, fransk matematiker, Giuseppe Peano, 1858-1932, italiensk matematiker.) ■

Sats. Låt $f(x, y)$ vara kontinuerlig för alla punkter $(x, y) : a \le x \le b, -\infty \le y \le \infty$, där a, b är ändliga. Om f satisfierar ett Lipschitzvillkor. (Rudolf Lipschitz, 1832-1903, tysk matematiker.)

$$| f(x, y) - f(x, y^*) | \le L | y - y^* | \tag{11.3}$$

för $a \le x \le b$ och alla y, y^*, så existerar en entydig lösning till (11.2) för varje begynnelsevärde $y(x_0) = y_0$, $x_0 \in [a, b]$. Konstanten L kallas Lipschitzkonstant. ■

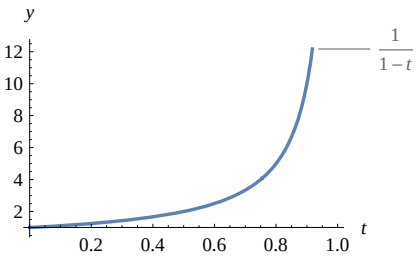
Tyvärr uppfyller de flesta f inte något Lipschitsvillkor (eller är Lipschitzkontinuerliga).

Exempel 11.3: Blow-up i ändlig tid saknar Lipschitzkonstant.

```
yOft = DSolve[{y'[t] == y[t]^2, y[0] == 1}, y[t], t]
```

$$\left\{\left\{y(t) \to \frac{1}{1-t}\right\}\right\}$$

```
Plot[Evaluate[y[t] /. yOft], {t, 0, 1}, AxesLabel → {t, y}, PlotLabels → Automatic]
```



Andra problem uppfyller alltid ett Lipschitzvillkor

Sats. Systemet av (ODE) $y' = \mathbb{A}y$ där $\mathbb{A}$ är en konstant matris har Lipschitzkonstanten $L = \max_{\|\mathbb{x}\|=1} \|\mathbb{A}\mathbb{x}\| = \|\mathbb{A}\|_\infty = \max_i \sum_j |a_{ij}|$. Det finns flera matrisnormer att välja på, men max absolut rad summa duger bra. Den är dessutom lätt att beräkna. ■

Exempel 11.4: Studera

$$\text{(BVP)} \begin{cases} y'(x) = f(x, y) = xy & \text{(ODE)} \\ y(0) = 1 & \text{(BV)} \end{cases}$$

och antag att vi kan lösa (BVP) i $x \in [0, 1]$. Vi får direkt

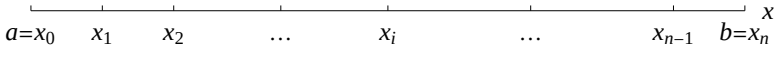
$$| xy - xy^* | = | x | | y - y^* | \le 1 | y - y^* |$$

med Lipschitzkonstanten $L = 1$. Allmänt kan man visa att om $\left|\frac{\partial f}{\partial y}\right| \le L$ i det aktuella intervallet, så kan man använda L som Lipschitzkonstant.

Vi noterar att Lipschitzkontinuitet är ett *konstruktivt* begrepp som kan användas för att exempelvis visa konvergens och studera konvergenshastighet hos iterativa algoritmer, till skillnad från Cauchy's rent *existensiella* ε, δ-exercis som är så vanlig i de flesta analyskurser. Lipschitzkontinuitet är en förbättring och strängare än denna, och helt dominerande i "Computational Mathematics". (R. Lipschitz, Lehrbuch der Analysis, Vol II, 1877)

Som vi antydde tidigare kan vi i allmänhet inte förvänta oss en analytisk lösning till ett (BVP), utan får nöja oss med en ren numerisk lösning, vilket är väl så bra ur många aspekter. Sagan går såhär…

Lösningsidé: Alla numeriska metoder går ut på att i intervallet $I = [a, b]$, där vi söker lösningen till (11.2), ge punktvisa numeriska approximationer y_i i ett antal punkter $x_i \in I$, så att $y_i \approx y(x_i)$ är så bra som möjligt i någon mening, så kallad numerisk lösning. Placeringen av alla x_i kallas diskretisering, eller partionering, av I. Avståndet mellan dem kallas steglängd, h. Ofta väljes denna konstant, det vill säga x_i är ekvidistanta. Så för $n + 1$ punkter har vi då



$x_i = x_0 + ih$, $i = 0, 1, 2, \ldots, n$, där $h = \frac{b-a}{n}$

Man säger att metoderna stegar fram från punkt till punkt, med start i det kända (BV) (x_0, y_0), och levererar till slut den numeriska lösningen i I som en följd av punkter $\{(x_i, y_i)\}_{i=0}^n$. Avancerade algoritmer väljer naturligtvis $h = h_i$ adaptivt vid x_i beroende på $f(x_i, y_i)$. Så gör *Mathematica* alltid. Metoder som stegar på med friskt mod kallas explicita, medan de lite mer försiktiga kallas implicita. Men först en kär gammal vän, som kommer att följa oss som ett fundament genom hela kapitlet.

11.2 Taylors metod

Här får vi lösningen till (11.2) med Taylorutveckling kring x_0 (Brook Taylor, 1685-1731, engelsk matematiker, “I’m the best!”)

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + \frac{(x-x_0)^2}{2!}y''(x_0) + \ldots + \frac{(x-x_0)^{n-1}}{(n-1)!}y^{(n-1)}(x_0) + \frac{(x-x_0)^n}{n!}y^{(n)}(\xi), \quad \xi \in [x_0,\, x] \tag{11.4}$$

Tydligen handlar det om att derivera och lösa ekvationssystem. Det är *Mathematica* bra på! Så låt oss ta ett

Exempel 11.5: Studera något som ser förädiskt enkelt ut

$$(\text{BVP}) \begin{cases} y'(x) = 1 - 2xy & (\text{ODE}) \\ y(0) = 0 & (\text{BV}) \end{cases}$$

Först den analytiska lösningen.

```
yAx = DSolve[{y'[x] == 1 - 2 x y[x], y[0] == 0}, y[x], x]
```

$$\left\{\left\{y(x) \to \frac{1}{2}\sqrt{\pi}\, e^{-x^2}\, \text{erfi}(x)\right\}\right\}$$

Sedan numeriska lösningen som ett femtegradspolynom med hjälp av Taylor.

```
dODE = D[y'[x] == 1 - 2 x y[x], {x, #}] & /@ Range[0, 5]
```

$$\{y'(x) = 1 - 2\,x\,y(x),\ y''(x) = -2\,x\,y'(x) - 2\,y(x),\ y^{(3)}(x) = -2\,x\,y''(x) - 4\,y'(x),\ y^{(4)}(x) = -2\,x\,y^{(3)}(x) - 6\,y''(x),\ y^{(5)}(x) = -2\,x\,y^{(4)}(x) - 8\,y^{(3)}(x),\ y^{(6)}(x) = -2\,x\,y^{(5)}(x) - 10\,y^{(4)}(x)\}$$

```
dydx = Solve[dODE /. {y[x] → 0, x → 0}] // First
```

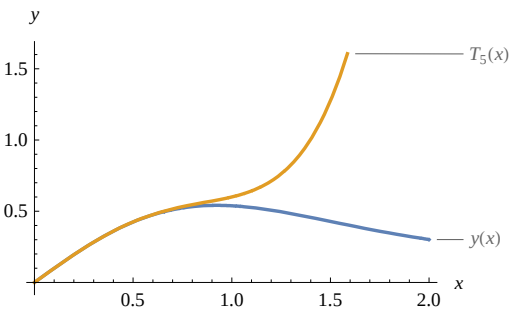
$$\{y'(0) \to 1,\ y''(0) \to 0,\ y^{(3)}(0) \to -4,\ y^{(4)}(0) \to 0,\ y^{(5)}(0) \to 32,\ y^{(6)}(0) \to 0\}$$

```
yTx = Series[y[x], {x, 0, 5}] /. dydx /. y[0] → 0
```

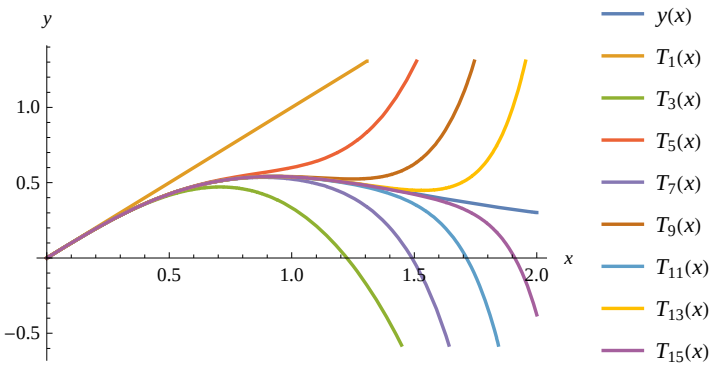
$$x - \frac{2\,x^3}{3} + \frac{4\,x^5}{15} + O\left(x^6\right)$$

Nu är det bara att rita lite.

```
Plot[Evaluate[{y[x] /. yAx, Normal[yTx]}],
 {x, 0, 2}, AxesLabel → {x, y}, PlotLabels → {y[x], T5[x]}]
```



Som väntat får vi en allt bättre numerisk lösning som också “håller” allt längre ju högre gradtal vi väljer på Taylorutvecklingen.



I allmänhet kan en exakt uppskattning av trunkeringsfelet i Taylorutvecklingen (11.4) bli besvärlig, eftersom vi direkt endast får derivatorna i x_0. Men då trunkeringsfelet innehåller faktorn $(x - x_0)^{n+1}$ används metoden vanligtvis för x nära x_0.

■

11.3 Explicita lösningsmetoder

Vi ska behandla några vanliga stegmetoder. Den enklaste av dessa, Eulers metod, stegar hela tiden i aktuell tangentriktning. De mer sofistikerade Heuns metod, Mittpunktsmetoden och Runge-Kuttas metod, väljer ett mer raffinerat sätt att stega. Man skulle kunna säga att de gör en rekognosering och tar ut riktningar i en eller flera punkter utefter tänkbara vägar innan de bestämmer sig för den slutliga riktningen för steget. I metoderna nedan har vi för enkelhets skull valt en konstant steglängd h, men självklart är det enkelt att beakta ett lokalt h_i.

11.3.1 Eulers metod

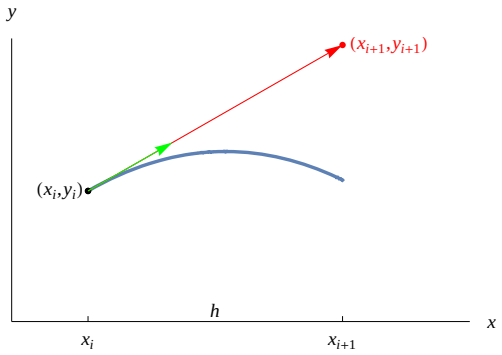
I begynnelsepunkten $(x_0,\ y_0)$ får vi en tangentriktning $y'=f(x_0,\ y_0)$ ur (ODE). Går vi steget h i x med denna riktning, kommer vi till punkten $(x_1,\ y_1)$, där $x_1=x_0+h$ och $y_1=y_0+h\,f(x_0,\ y_0)$. Vi hoppas då att $y_1\approx y(x_1)$. Ta ut en ny riktning $y'=f(x_1,\ y_1)$ där med (ODE), fortsätta steget h i x med denna nya riktning till punkten $(x_2,\ y_2)$, där $x_2=x_1+h=x_0+2h$ och $y_2=y_1+h\,f(x_1,\ y_1)$ och hoppas att $y_2\approx y(x_2)$, och så vidare. Återigen utnyttnjas Taylorutveckling (11.4). Här trunkeras direkt efter första derivatan så att vi i punkten x_i ersätter $y'(x_i)$ med framåtdifferens, varvid differentialekvationen (11.1) övergår i

$$\frac{y(x_{i+1})-y(x_i)}{h}\approx y'(x_i)=f(x_i,\ y_i)$$

Första och sista leden ger en klassisk metod för numerisk lösning av (BVP) och heter Eulers metod (Leonhard Euler, 1707-1783, schweizisk matematiker),

$$y_{i+1}=y_i+h\,f(x_i,\ y_i),\ i=0,\ 1,\ 2,\ \ldots \tag{11.5}$$

Idén bakom Eulers metod med enklast tänkbara och naiva stegande kan åskådliggöras schematiskt



Av Taylorutvecklingen (11.4) framgår, att det lokala trunkeringsfelet $e_l=y_{i+1}-y(x_i+h)$ *då* $y_i=y(x_i)$, det vill säga vi antas starta lokala steget i rätt punkt på lösningskurvan, är $O(h^2)$. Ty om alla derivator är begränsade har vi

$$e_l=y_i-y(x_i+h)=\{\text{Euler}-\text{Taylor}\}=y(x_i)+h\,y'(x_i)-\left(y(x_i)+h\,y'(x_i)+\frac{h^2}{2}y''(x_i)+\ldots\right)=-\frac{h^2}{2}y''(x_i)-\frac{h^3}{6}y'''(x_i)-\ldots=O(h^2).$$

Så e_l minskar alltså till $\left(\frac{1}{2}\right)^2 e_l$ om h halveras. Om vi går från $x=a$ till $x=b$ med steglängden h, är antalet steg $\frac{b-a}{h}$ lika med antalet lokala trunkeringsfel. Man inser, att Eulers metod då har ett totalt trunkeringsfel av $O\left(\frac{1}{h}\times h^2\right)=O(h)$, eller som man brukar säga, ett globalt trunkeringsfel $e_g=y_n-y(x_n)$ av $O(h)$, alltså av första ordningen. Så inte speciellt noggrann. Vi säger att metoden är konvergent om $\|e_g\|\to 0$ då $h\downarrow 0$, annars divergent. En divergent lösning yttrar sig i att y_i avviker alltmer från $y(x_i)$, antingen monotont eller genom allt mer kraftiga oscillationer. Man kan visa att Eulers metod är konvergent, men i praktiken skulle det innebära mycket små steglängder, med lösningstider där man får vänta "för gott". En metod kallas stabil om en liten störning i indata resulterar i en liten störning av den numeriska lösningen, annars instabil. Instabila modeller sägs vara känsliga för "fjärilseffekt". Även instabilitet beror av h. Vi tar några exempel

Exempel 11.6: Vi tar ett handräkningsexempel

$$\text{(BVP)}\begin{cases} y'(x)=x+y & \text{(ODE)}\\ y(0)=1 & \text{(BV)}\end{cases}$$

Vi är intresserade av att hitta $y(0.4)$ med steglängden $h=0.1$. Den analytiska lösningen är

```
DSolve[{y'[x] == x + y[x], y[0] == 1}, y[x], x]
% /. x → 0.4
```

$\{\{y(x)\to -x+2\,e^x-1\}\}$

$\{\{y(0.4)\to 1.58365\}\}$

Räkningarna genomförs enklast i ett räkneschema

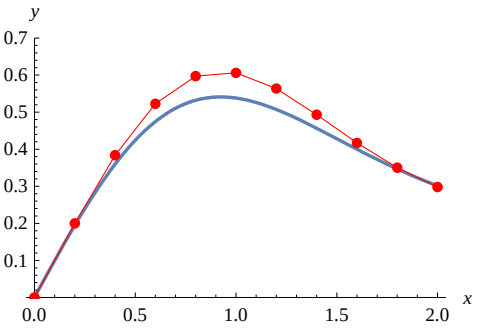
x	y	$y' = x + y$	hy'
0	1	1	0.1
0.1	1.1	1.2	0.12
0.2	1.22	1.42	0.142
0.3	1.362	1.662	0.1662
0.4	1.5282		

Exempel 11.7: Känt exempel.

$$(\text{BVP}) \begin{cases} y'(x) = 1 - 2xy & (\text{ODE}) \\ y(0) = 0 & (\text{BV}) \end{cases}$$

Vi är intresserade av att hitta lösningen i $x \in [0, 2]$. Steglängen väljer vi till $h = 0.2$, sedan är det bara att stega på i tangentens riktning!

```
n = 10; h = 2.0/n;
f[x_, y_] := 1 - 2 x y
g[x_, y_] := {x + h, y + h f[x, y]};
yEx = NestList[g @@ # &, {0, 0}, n];
Plot[y[x] /. yAx, {x, 0, 2}, Epilog → {PointSize[0.025], Red, Line[yEx], Point[yEx]},
 PlotRange → {0, 0.7}, AxesLabel → {x, y}]
```



Exempel 11.8: Logistiska ekvationen i biologi beskriver bland annat hur en djurpopulation utvecklas under begränsande levnadsvillkor.

$$(\text{BVP}) \begin{cases} y'(t) = \frac{1}{3} y(t)(8 - y(t)) & (\text{ODE}) \\ y(0) = 1 & (\text{BV}) \end{cases}$$

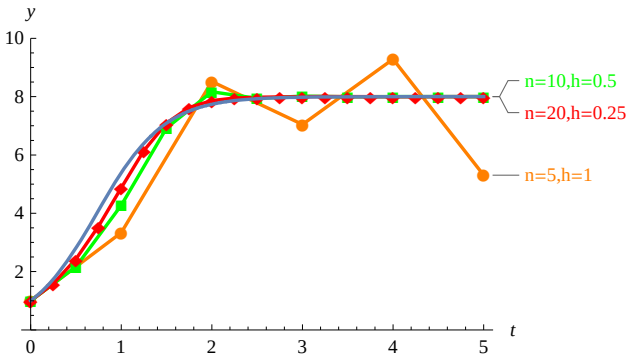
Vi är intresserade av att hitta lösningen i $x \in [0, 5]$. Steglängen väljer vi till $h = \frac{5}{n}$, $n = 5, 10, 20$. Först analytiska lösningen…

```
yAt = DSolve[{y'[t] == 1/3 y[t] (8 - y[t]), y[0] == 1}, y[t], t]
```

$$\left\{\left\{y(t) \to \frac{8\, e^{8t/3}}{e^{8t/3} + 7}\right\}\right\}$$

…sedan är det bara att stega på i tangentens riktning med olika val av h!

```
f[t_, y_] := 1/3 y (8 - y)
g[t_, y_] := {t + h, y + h f[t, y]};
yEt = {}; Do[h = 5.0/n;
 AppendTo[yEt, NestList[g @@ # &, {0, 1}, n]], {n, {5, 10, 20}}];
Show[ListLinePlot[yEt, PlotStyle → {Orange, Green, Red},
   PlotMarkers → Automatic, PlotRange → {0, 10}, AxesLabel → {t, y},
   PlotLabels → {"n=5,h=1", "n=10,h=0.5", "n=20,h=0.25"}],
 Plot[y[t] /. yAt, {t, 0, 5}, PlotRange → {0, 10}, AxesLabel → {t, y}]]
```



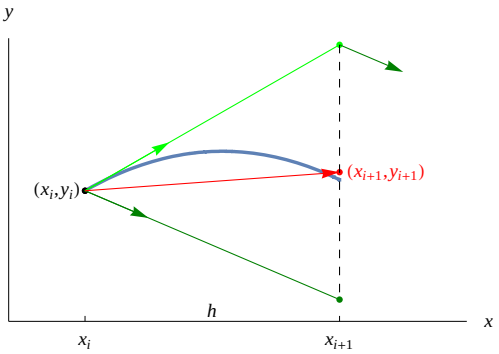
Vi ser i exemplen att i varje tangentsteg flyttar vi oss oundvikligen till en ny trajektoria till (ODE), med annat (BV). Att vi ibland hamnar väldigt rätt till slut är ren tur och beror naturligtvis på hur (ODE) ser ut och vilken steglängd h vi valt! Vi kan naturligtvis minska steglängden för att få bättre noggrannhet. Har man lösningar för h och $h/2$ kan man förbättra resultatet avsevärt, se senare avsnitt. Men det är i allmänhet lämpligare, med tanke på beräkningsarbetet, att gå över till en noggrannare metod, där vi väljer ett mera raffinerat sätt att räkna ut riktningen i varje steg.

11.3.2 Heuns metod

Strategin är att gå från punkten (x_i, y_i) i riktningen $y' = f(x_i, y_i)$ till punkten $(x_i + h, y_i + h f(x_i, y_i))$ och där ta ut riktningen $f(x_i + h, y_i + h f(x_i, y_i))$. Vi går inte i denna riktning, utan går tillbaka till (x_i, y_i) och bildar medelvärdet av de två riktningarna när vi så tar steget h i x till nästa punkt (x_{i+1}, y_{i+1}) som blir nästa punkt på vår numeriska lösning till (BVP). Denna blir sedan utgångspunkt för nästa steg. Heuns metod (Karl Heun, 1859-1929, tysk matematiker) fungerar alltså enligt

$$y_{i+1} = y_i + h\,\tfrac{1}{2}\left(f(x_i, y_i) + f(x_i + h, y_i + h f(x_i, y_i))\right),\ i = 0, 1, 2, \ldots \tag{11.6}$$

Idén bakom Heuns metod kan åskådliggöras schematiskt



Beteckar vi, som brukligt i branschen, med “tangenttillskott” kan Heuns metod skrivas

$$\begin{cases} k_1 = h\,f(x_i, y_i) \\ k_2 = h\,f(x_i + h, y_i + k_1) \end{cases} \Rightarrow y_{i+1} = y_i + \tfrac{1}{2}(k_1 + k_2),\ i = 0, 1, 2, \ldots \tag{11.7}$$

Vi återkommer till noggrannhet i samband med nästa metod.

Exempel 11.9: Vi tar samma handräkningsexempel som ovan

$$\text{(BVP)} \begin{cases} y'(x) = x + y & \text{(ODE)} \\ y(0) = 1 & \text{(BV)} \end{cases}$$

Vi är intresserade av att hitta $y(0.4)$ med steglängden $h = 0.2$. Vi får följande räkneschema

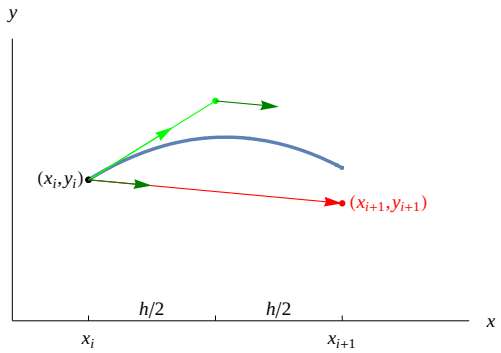
x	y	$y' = x + y$	hy'
$\underline{0}$	$\underline{1}$	1	0.2
0.2	1.2	1.4	0.28
			$\frac{1}{2}(0.2 + 0.28) = 0.24$
$\underline{0.2}$	$\underline{1.24}$	1.44	0.288
0.4	1.528	1.928	0.3856
			$\frac{1}{2}(0.288 + 0.3856) = 0.3368$
$\underline{0.4}$	$\underline{1.5768}$		

11.3.3 Mittpunktsmetoden

Även här går vi från punkten (x_i, y_i) på rekognosering i riktningen $y' = f(x_i, y_i)$ men bara till mittpunkten $\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} f(x_i, y_i)\right)$ och där ta ut en ny riktning, som vanligt genom insättning i (ODE) $f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} f(x_i, y_i)\right)$. Vi återvänder sedan till (x_i, y_i) och använder denna "mittpunktsriktning" för nästa steg h i x till nästa punkt (x_{i+1}, y_{i+1}) på vår numeriska lösning till (BVP). Denna blir sedan utgångspunkt för nästa steg. Mittpunktsmetoden fungerar alltså enligt

$$y_{i+1} = y_i + h f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} f(x_i, y_i)\right), \; i = 0, 1, 2, \ldots \tag{11.8}$$

Även här låter vi Mittpunktsmetoden åskådliggöras schematiskt



Beteckar vi även här "tangenttillskottet" med $k_1 = h f(x_i, y_i)$ kan Mittpunktsmetoden skrivas

$$y_{i+1} = y_i + h f\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right), \; i = 0, 1, 2, \ldots \tag{11.9}$$

Exempel 11.10: Handräkningsexemplet igen

$$\text{(BVP)} \begin{cases} y'(x) = x + y & \text{(ODE)} \\ y(0) = 1 & \text{(BV)} \end{cases}$$

Vi är intresserade av att hitta $y(0.4)$ med steglängden $h = 0.2$. Vi får följande räkneschema

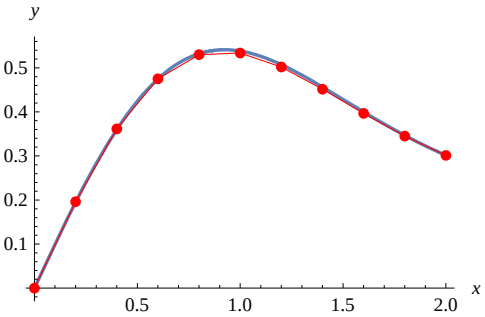
x	y	$y' = x + y$	hy'
$\underline{0}$	$\underline{1}$	1	0.2
0.1	1.1	1.2	0.24
$\underline{0.2}$	$\underline{1.24}$	1.44	0.288
0.3	1.384	1.684	0.3368
$\underline{0.4}$	$\underline{1.5768}$		

Exempel 11.11: Vi tar välkänt exempel ännu en gång.

$$\text{(BVP)} \begin{cases} y'(x) = 1 - 2xy & \text{(ODE)} \\ y(0) = 0 & \text{(BV)} \end{cases}$$

Vi är intresserade av att hitta lösningen i $x \in [0, 2]$. Steglängen väljer vi till $h = 0.2$, sedan är det bara att stega på i tangentens riktning!

```
n = 10; h = 2.0/n;
f[x_, y_] := 1 - 2 x y
g[x_, y_] := {x + h, y + h f[x + h/2, y + h/2 f[x, y]]}
yMx = NestList[g @@ # &, {0, 0}, n];
Plot[y[x] /. yAx, {x, 0, 2},
 Epilog → {PointSize[0.025], Red, Line[yMx], Point[yMx]}, PlotRange → All, AxesLabel → {x, y}]
```



Om man ansätter $y_{i+1} = y_i + a_1k_1 + a_2k_2$, där $k_1 = h\,f(x_i,\, y_i)$ och $k_2 = h\,f(x_i + \alpha h,\, y_i + \beta k_1)$, och jämför Taylorutvecklingen av y_{i+1} omkring punkten $(x_i,\, y_i)$ med Taylorutvecklingen av $y(x)$ omkring punkten x_i för $x = x_{i+1}$, så visar det sig att Taylorutvecklingarna kan fås att överensstämma upp till och med h^2-termerna för vissa val av a_1, a_2, α och β, bland annat de val som ger Heuns metod och Mittpunktsmetoden. Så för dessa gäller alltså att lokala trunkeringsfelet e_l är $O(h^3)$. Eftersom antalet steg till en viss punkt är omvänt proportionellt mot h har de ett globalt trunkeringsfel e_g som är $O(h^2)$. Så Heuns metod och Mittpunktsmetoden är $O(h^2)$, jämfört med Eulers metod som är $O(h)$.

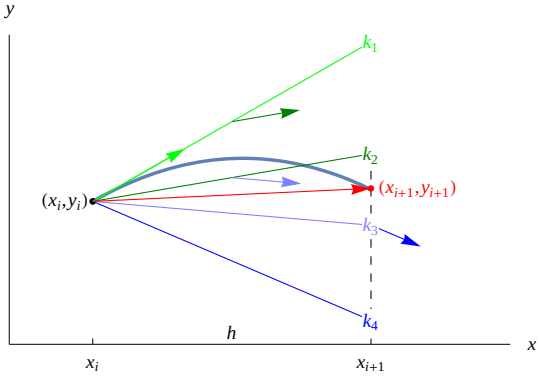
11.3.4 Runge-Kuttas metod

Med samma förfarande som antyddes i slutet av avsnittet om Mittpunktsmetoden kan man härleda medoder av godtycklig ordning, $O(h^p)$, speciellt enkelt blir det med symbolhanterande program som *Mathematica*. Den vanligaste med globalt trunkeringsfel som är $O(h^4)$ kallas Runge-Kuttametoden (Carl Runge, 1856-1927, Wilhelm Kutta, 1867-1944, tyska matematiker), men alla metoder av denna typ kallas Runge-Kutta-metoder. Man brukar ibland även kalla de tidigare nämnda för Runge-Kutta metoder eftersom de alla kan härledas på samma generella sätt. Så Eulers metod är en Runge-Kutta metod av första ordningen samt Heuns och Mitttpunktsmetoden Runge-Kutta metoder av andra ordningen.

Medan vi i Heuns metod bildade medelvärdet av två riktningar, får vi här den slutliga riktningen för varje steg som medelvärde av fyra riktningar, en i utgångspunkten och tre i rekognoseringspunkter. Eftersom varje y'-värde multiplicerat med steglängden h ger tillskott, eventuellt negativt, kan vi också säga att vi får det slutliga tillskottet som ett vägt medelvärde av fyra preliminära tillskott. Även här är det då naturligtvis brukligt att teckna metoden tillskottsmässigt som "tangenttillskott". Så *den* klassiska Runge-Kuttas metod (RK4) där globala trunkeringsfelet e_g alltså är $O(h^4)$.

$$y_{i+1} = y_i + a_1k_1 + a_2k_2 + a_3k_3 + a_4k_4 = y_i + k,\ i = 0,\ 1,\ 2,\ \ldots$$

$$\text{där}\ \begin{cases} k_1 = h\,f(x_i,\, y_i) & a_1 = \frac{1}{6} \\ k_2 = h\,f\left(x_i + \frac{h}{2},\, y_i + \frac{k_1}{2}\right) & a_2 = \frac{2}{6} \\ k_3 = h\,f\left(x_i + \frac{h}{2},\, y_i + \frac{k_2}{2}\right) & a_3 = \frac{2}{6} \\ k_4 = h\,f(x_i + h,\, y_i + k_3) & a_4 = \frac{1}{6} \end{cases} \quad \text{och}\ k = \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{11.10}$$



Exempel 11.12: Samma gamla handräkningsexempel som ovan

$$(\text{BVP}) \begin{cases} y'(x) = x + y & (\text{ODE}) \\ y(0) = 1 & (\text{BV}) \end{cases}$$

Vi är intresserade av att hitta $y(0.4)$ med steglängden $h = 0.2$. Vi får följande räkneschema

x	y	$y' = x + y$	hy'
0	1	1	0.2
0.1	1.1	1.2	0.24
0.1	1.12	1.22	0.244
0.2	1.244	1.44	0.2888
			$\frac{1}{6} \cdot 1.4568 = 0.2428$
0.2	1.2428	1.4428	0.28856
0.3	1.38708	1.68708	0.337416
0.3	1.411508	1.711508	0.342302
0.4	1.585102	1.985102	0.397020
			$\frac{1}{6} \cdot 2.045016 = 0.340836$
0.4	1.583636		

11.4 Implicita metoder

Implicita metoder får vi genom att välja bakåtdifferens, som då kommer att introducera den sökta y_{i+1} även på "höger" sida i formlerna ovan. Vi kan då inte lägre stega fram h explicit, utan blir istället tvungna att lösa en ekvation, inte sällan olinjär, i varje steg för att finna y_{i+1}. Den extra kostnaden för detta motiveras av att vi kan ha större steglängd h. Dessa kommer till användning när stabilitetsvillkoren gör att de explicita varianterna inte klarar uppgiften. Vi sammanfattar

$$\begin{aligned} &\text{Implicit Eulers metod } y_{i+1} = y_i + h f(x_{i+1}, y_{i+1}),\ i = 0, 1, 2, \ldots \\ &\text{Implcit Trapetsmetod } y_{i+1} = y_i + \tfrac{h}{2}(f(x_i, y_i) + f(x_{i+1}, y_{i+1})),\ i = 0, 1, 2, \ldots \\ &\text{Implicit Mittpunktsmetod } y_{i+1} = y_i + h f\left(\tfrac{x_i + x_{i+1}}{2}, \tfrac{y_i + y_{i+1}}{2}\right),\ i = 0, 1, 2, \ldots \end{aligned} \tag{11.11}$$

Vi känner igen Trapetsmetoden som följer direkt efter en formell integration över intervallet $\int_{x_i}^{x_{i+1}} y'(x)\,dx = \int_{x_i}^{x_{i+1}} f(x, y)\,dx$ med användande av just trapetsmetoden. Implicita metoder är till skillnad från explicita nästan alltid stabila för "alla" h, medan explicita metoder oftast kräver små h. Differentialekvationer som inte kan skrivas explicit enligt (11.1) utan endast på implicit form $f(x, y(x), y'(x)) = 0$ hamnar naturligt under denna rubrik.

En (ODE) kallas styv om det finns mycket olika skalor i den oberoende variabeln. Ett annat sätt att se på det är att den homogena lösningen är kraftigt dämpad. Man bör då välja en implicit metod för att säkerställa stabilitet. Explicita behöver små h, om ens möjligt praktiskt att få dem stabila.

11.5 Något lite om fel...

Det finns tydligen en del teori om (ODE), men de är begränsade till enkla sådana, t.ex. typexemplet framför andra $y'(x) = \lambda y(x)$, som man brukar mäta sig med när det gäller erforderlig steglängd i explicita metoder. I allmänhet är det mycket besvärligt! Betänk bara att vi alltid tar ett linjärt steg från den punkt vi är i (x_i, y_i) till nästa (x_{i+1}, y_{i+1}), vilka i princip aldrig ligger på den sanna lösningskurvan. Det kan ju faktiskt vara det inte helt otänkbara scenariot att $(x_{i+1}, y_{i+1}) \notin D_f$! Vad som återstår är laborativ matematik, som är enkelt i dessa dagar! Prova olika metoder, ändra steglängd, och se vad som fungerar. Till sist är det naturligtvis alltid en fördel att veta det fysikaliska problem man arbetar med för att bedömma rimlighet i lösningar. I tidskritiska system kan det krävas både fingertoppskänsla och specialmetoder.

Genom att halvera h kan man systematiskt förfina lösningen med så kallad **Richardsonextrapolation** (Lewis Fry Richardson, 1881-1953, engelsk matematiker), eller gärna upprepad sådan. Här applicerat på explicit Euler för några halveringar av h i **Exempel 11.6**. Eftersom globala trunkeringsfelet e_g är $O(h)$ får vi med gott resultat

h	$y(0.4)$	2^1-1	2^2-1	2^3-1
0.4	1.40			
0.2	1.48	$1.56 = \frac{2^1\cdot 1.48-1.40}{2^1-1}$		
0.1	1.5282	1.5764	$1.5819 = \frac{2^2\cdot 1.5764-1.56}{2^2-1}$	
0.05	1.5549	1.5816	1.5833	$1.5835 = \frac{2^3\cdot 1.5833-1.5819}{2^3-1}$

För Runge-Kutta är globala trunkeringsfelet e_g av $O(h^4)$ då får vi istället viktningen 2^4-1, 2^5-1, 2^6-1, …

11.6 Differentialekvation av högre ordning

En differentialekvation av högre ordning, såväl som system av dem, kan överföras till ett system av första ordningen (11.2'), genom införande av hjälpfunktioner. Exempelvis ett andra ordningens

$$\text{(BVP)}\begin{cases} y''(x) = f(x,\, y(x),\, y'(x)) & \text{(ODE)} \\ y(x_0) = y_0,\; y'(x_0) = y_0' & \text{(BV)} \end{cases} \Leftrightarrow \text{(BVP')}\begin{cases} y'(x) = v(x), & y(x_0) = y_0 \\ v'(x) = f(x,\, y(x),\, v(x)), & v(x_0) = v_0 = y_0' \end{cases} \tag{11.12}$$

Om systemet är linjärt brukar det skrivas på matrisform $u'(x) = \mathbb{A}(x)u(x) + \mathbb{B}(x)$ med $u(x) = (y(x),\, v(x))^{\mathrm{T}}$ för att underlätta den parallella bearbetningen. Dessutom innehåller $\mathbb{A}$ mycket information om systemets egenskaper.

11.7 Något om andra metoder

Vi noterar att (11.2) efter en formell integrering är ekvivalent med integralekvationen

$$y(x) = y_0 + \int_{x_0}^{x} f(x,\, y(x))dx \tag{11.13}$$

Att integrera är alltid en lugnare operation än derivering, "gjuta olja på vågorna". Så genom att utgå från denna integralformulering kan man få en följd av allt bättre numeriska lösningar till (11.2) som *funktioner* $\hat{y}_i(x)$, $i = 0, 1, \ldots$. Detta kallas Picards metod (Emile Picard, 1856-1941, fransk matematiker). Som första funktion använder vi begynnelsevärdet y_0.

$$\begin{cases} \hat{y}_0(x) = y_0 \\ \hat{y}_1(x) = y_0 + \int_{x_0}^{x} f\big(x,\, \hat{y}_0(x)\big)dx \\ \hat{y}_2(x) = y_0 + \int_{x_0}^{x} f\big(x,\, \hat{y}_1(x)\big)dx \\ \quad\vdots \\ \hat{y}_n(x) = y_0 + \int_{x_0}^{x} f\big(x,\, \hat{y}_{n-1}(x)\big)dx \end{cases} \tag{11.14}$$

Man kan visa att att följden $\hat{y}_i(x)$, $i = 0, 1, \ldots$, under vissa förutsättningar, konvergerar mot lösningen $y(x)$ till integralekvationen (11.13). I själva verket används metoden för att visa existens av lösningar och för avstamp för vidare analys av randvärdesproblem. I detta fallet är metoden inte alltid så lämplig för numeriskt arbete eftersom integralerna kan bli besvärliga. Visserligen har vi återfört problemet med att lösa (ODE) till att numeriskt bestämma integraler, vilket vi avhandlat i ett tidigare kapitel.

Exempel 11.13: Lös med Picards metod

$$\text{(BVP)}\begin{cases} y'(x) = 1 + y(x)^2 & \text{(ODE)} \\ y(0) = 0 & \text{(BV)} \end{cases}$$

Lösning: Räkna på enligt receptet (11.14)

$$\begin{cases} \hat{y}_0(x) = 0 \\ \hat{y}_1(x) = 0 + \int_0^x \big(1 + 0^2\big)dx = x \\ \hat{y}_2(x) = 0 + \int_0^x \big(1 + x^2\big)dx = x + \frac{1}{3}x^3 \\ \hat{y}_3(x) = 0 + \int_0^x \Big(1 + \big(x + \frac{1}{3}x^3\big)^2\Big)dx = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{1}{63}x^7 \\ \quad\vdots \end{cases}$$

Eftersom exakta lösningen till (BVP) är $y(x) = \tan(x)$, kan vi jämföra med Taylorutveckling av denna, och se att de stämmer väl överens.

```
Series[Tan[x], {x, 0, 7}]
```

$$x + \frac{x^3}{3} + \frac{2\,x^5}{15} + \frac{17\,x^7}{315} + O(x^8)$$

Om $f(x, y(x)) = f(x)$ i (11.12) har vi ett vanligt integrationsproblem $y(x) = y_0 + \int_{x_0}^{x} f(x)dx$. Då är Heuns metod lika med Trapetsformeln och Runge-Kuttas metod med Simpsons formel. Så visst utnyttjar vi gamla kunskaper!

Vi har såhär långt behandlat enstegsmetoder där man utgår endast från punkten (x_i, y_i) för beräkningen av nästa punkt (x_{i+1}, y_{i+1}). Eulers metod, Heuns metod, Mittpunktsmetoden och Runge-Kuttas metod är alla exempel på sådana metoder. I en flerstegsmetod utnyttjar man förutom punkten (x_i, y_i) även en eller flera av de tidigare punkterna (x_{i-1}, y_{i-1}), (x_{i-2}, y_{i-2}), … för beräkningen av (x_{i+1}, y_{i+1}). För att starta en flerstegsmetod krävs alltså fler punkter än den givna i (BV). Dessa måste beräknas med någon enstegsmetod.

Antag att vi känner tidigare punkter (x_i, y_i), (x_{i-1}, y_{i-1}), (x_{i-2}, y_{i-2}), (x_{i-3}, y_{i-3}). Vi kan då konstruera interpolationspolynomet $p(x)$ genom dessa och integrera detta istället för $y'(x)$ i det triviala sambandet $y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} y'(x)dx$. Vi får då Adams-Bashforths prediktorformel med konstant steglängd h

$$y_{i+1} = y_i + \frac{h}{24}(55\,p'(x_i) - 59\,p'(x_{i-1}) + 37\,p'(x_{i-2}) - 9\,p'(x_{i-3})) \tag{11.15}$$

Vi har nu tillgång till ett första närmevärde till y_{i+1}. Om vi sedan skapar ett nytt interpolationspolynom $q(x)$ genom punkterna (x_{i+1}, y_{i+1}), (x_i, y_i), (x_{i-1}, y_{i-1}), (x_{i-2}, y_{i-2}) och integrerar detta får vi en Adams-Moultons korrektorformel

$$y_{i+1} = y_i + \frac{h}{24}(9q'(x_{i+1}) - 19q'(x_i) - 5q'(x_{i-1}) - q'(x_{i-2})) \tag{11.16}$$

Som namnen antyder ger prediktorn ett första y_{i+1}, som sedan används av korrektorn för ett förbättrat y_{i+1}. Eventuellt kan korrektorn användas flera gånger. Adams prediktor-korrektormetod är $O(h^4)$. Liksom vid de tidigare metoderna kan tangenttillskottet uppfattas som en steglängd h gånger ett vägt medelvärde av olika riktningar. Med *Mathematica* är det lätt att utöka metoden till icke ekvidistanta punkter för att hantera ett adaptivt h, eller till en högra ordningens metod.

11.8 Något lite om Randvärdesproblem

Vi ska använda en så kallad Finita differensmetod (FDM) för att lösa en differentialekvation. Detta är en metod för att söka en numerisk lösning till ett begynnelse- eller randvärdesproblem som av olika skäl är så besvärligt att vi inte klarar av att lösa det analytiskt med metoder vi känner från tidigare. Detta är den "vanligaste" situationen för en ingenjör i verkliga livet! Speciellt har vi ju då oftast geometrier i 2D/3D. Vi testar den på ett enkelt 1D

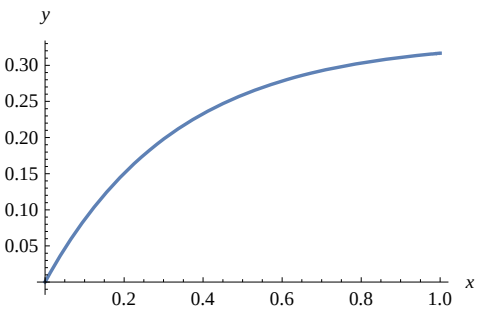
$$\text{(BVP)} \begin{cases} y'(x) + 3y(x) = 1 & \text{(ODE)} \\ y(0) = 0 & \text{(BV)} \end{cases}.$$

Först analytiska lösningen så att har vi något att jämföra med

```
yAx = DSolve[{y'[x] + 3 y[x] == 1, y[0] == 0}, y[x], x] // First // Simplify
```

$$\left\{y(x) \to \frac{1}{3} - \frac{e^{-3x}}{3}\right\}$$

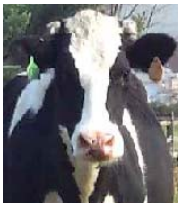
```
Plot[y[x] /. yAx, {x, 0, 1}, AxesLabel → {x, y}]
```



Vi söker nu en numerisk lösning i ett område, säg $x \in [0, 1]$. Idén är som tidigare att göra en diskretisering av området. Vi kallar nu "steglängden" för element. Gränspunkterna x_i mellan elementen kallas noder. I de inre noderna ersätts sedan $y'(x)$ med sin differenskvot. Vi får då ett ekvationssystem ur vilket y_i kan bestämmas i varje nod x_i. Resultatet är en numerisk lösning som är styckvis linjär över elementen. Det typiska är nu, som tidigare, att ju finare indelning vi gör ju bättre blir den numeriska lösningen. Metoden kan förfinas till den så kallade Finita elementmetoden (FEM) vilken är det helt dominerande redskapet för datorbaserad beräkning av

exempelvis elektromagnetism, hållfasthet för en maskindetalj, lyftkraften hos en flygplansvinge eller väderprognoser. Man kan jämföra (FEM) med “Söndra och Härska”. I analogi med Lego, se nedan, approximerar, diskretiserar, man en kontinuerlig modell med standardiserade bitar, finita element, som är enkla att räkna på, “söndra”. När de sätts samman, assembleras, till en diskret modell kommer varje bit bara att “prata” med sina närmaste grannbitar med krav på kontinuitet i fysikaliska storheter, vilket leder till ett ekvationssystem, “härska”. Det blir i allmänhet mycket stort, kanske 100 miljoner obekanta! Det måste då lagras glest, eller så använder man kanske hellre någon iterativ metod eftersom det inte sällan handlar om olinjära fysikmodeller, t.ex. flödesanalys kring en flygpansvinge eller biomekanik.

Kontinuerlig modell



$\int_\Omega 2\mu \boldsymbol{e}(\boldsymbol{u}):\boldsymbol{e}(\boldsymbol{v})d\Omega$

Diskretisera ⟹

Finita element

$\mathbb{k}_{ij} = \int_{\Omega_e} \nabla\varphi_i \cdot \nabla\varphi_j d\Omega_e$

$\mathbb{f}_i = \int_{\Omega_e} f\,\varphi_i d\Omega_e$

Assemblera ⟹

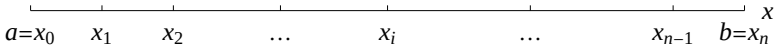
Diskret modell

$\mathbb{K}\mathbb{u} = \mathbb{F}$

Nu åter till (BVP) i 1D. Antag att vi delar in vårt område i n st lika långa element. Elementen behöver inte vara lika långa, snarare tvärtom, där det "händer" mycket måste man använda mindre element för att kunna följa lösningen. Omvänt gäller då större element i "lugnare" områden. Vi börjar med en grov indelning som har lika stora element. Dessa blir då h långa.

```
n = 3; h = 1.0/n;
```

Vi har alltså indelningen med nodernas lägen x_i och de eftersökta nodvärdena y_i. Här vet vi på grund av (BV) att $y(0) = y_0 = 0$.



$$\begin{pmatrix} x_0 = 0 & y_0 = 0 \\ x_1 = x_0 + h & y_1 \\ x_2 = x_1 + h & y_2 \\ \vdots & \vdots \\ x_n = 1 & y_n \end{pmatrix}$$

Approximera nu derivatan i varje inre nod med bakåtdifferens, det vill säga $y'(x_i) \approx \frac{y_i - y_{i-1}}{h}$. Detta leder till ett ekvationssystem där varje rad motsvarar differentialekvationen applicerad i tur och ordning på de inre noderna.

```
ekv = Table[(yi - yi-1)/h + 3 yi == 1, {i, 1, n}]
```

$\{3\,y_1 + 3.\,(y_1 - y_0) = 1,\ 3\,y_2 + 3.\,(y_2 - y_1) = 1,\ 3\,y_3 + 3.\,(y_3 - y_2) = 1\}$

Nu är detta ekvationssystem singulärt och kräver liksom tidigare ett begynnelsevärde. Jämför bestämning av c_1 för att fixera en fysikalisk lösning till differentialekvationen! I vårt fall $y(0) = y_0 = 0$. Sedan är det bara att lösa det.

```
lösning = Solve[ekv /. y0 → 0] // First
```

$\{y_1 \to 0.166667,\ y_2 \to 0.25,\ y_3 \to 0.291667\}$

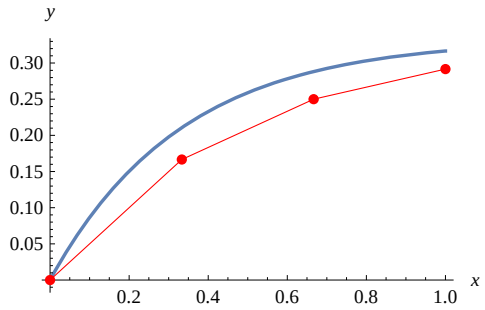
eller i tabellform $(x_i,\ y_i)$.

```
xyFDM = Table[{i/n, yi}, {i, 0, n}] /. lösning /. y0 → 0
```

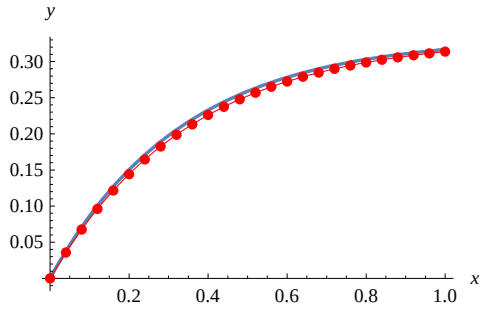
$$\begin{pmatrix} 0 & 0 \\ \frac{1}{3} & 0.166667 \\ \frac{2}{3} & 0.25 \\ 1 & 0.291667 \end{pmatrix}$$

Slutligen en jämförelse med den analytiska lösningen.

```
Plot[y[x] /. yAx, {x, 0, 1}, PlotRange → All, AxesLabel → {x, y},
    Epilog → {Red, PointSize[0.025], Line[xyFDM], Point[xyFDM]}]
```

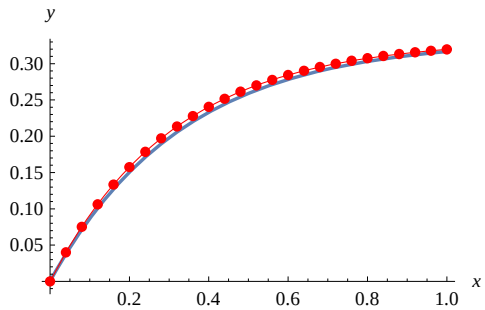


Som tidigare nämnts kan man förvänta sig bättre noggrannhet om en finare diskretisering väljs. I figuren nedan ses en jämförelse mellan analytisk lösning och en modell med 25 element. Man kan visa att felet i varje nod är proportionellt mot *h*.



Metoden ovan är en variant av Eulers metod eftersom man egentligen inte behöver lösa ett ekvationssystem. Metoden kan användas då derivatan kan räknas ut explicit $y'(x) = f(x, y)$. I vårt modellexempel ovan är tydligen detta fallet, eftersom $y'(x) = 1 - 3y(x)$. Den Euler vi känner får vi ju med framåtdifferens, $y'(x_i) \approx \frac{y_{i+1}-y_i}{h} = 1 - 3y_i \Rightarrow y_{i+1} = y_i + h(1 - 3y_i)$, vilket visar att varje ny punkt på kurvan kan beräknas genom att stega fram i tangentens riktning med start i (BV) som tidigare. Jämfört med (FDM) hamnar vi på "andra sidan" om den analytiska lösningen…

```
n = 25;
h = 1.0/n;
xyEuler = Table[{(i + 1)/n, If[i == -1, y0 = 0, yi+1 = yi + h (1 - 3 yi)]}, {i, -1, n - 1}];
Plot[y[x] /. yAx, {x, 0, 1}, PlotRange → All, AxesLabel → {x, y},
    Epilog → {Red, PointSize[0.025], Line[xyEuler], Point[xyEuler]}]
```



När vi har ett äkta andra ordningens randvärdesproblem, är det inte så enkelt att stega. Även om det finns inskjutningsalgoritmer, så väljer man i princip alltid att lösa dem med (FDM) eller bättre med (FEM). Vid högre ordningens differentialekvationer eller mer förfinad approximation av derivatan får vi en allt starkare koppling mellan ekvationerna som då kräver ekvationslösning, Vi tar ett exempel

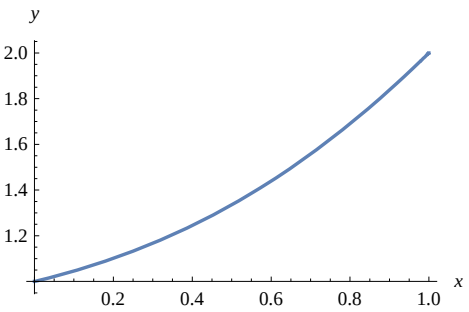
$$\text{(RVP)} \quad \begin{cases} y''(x) - 2y'(x) + y(x) = 1 - x^2 & \text{(ODE)} \\ y(0) = 1 \\ y(1) = 2 & \text{(RV)} \end{cases}$$

med analytisk lösning

```
yAx = DSolve[{y''[x] - 2 y'[x] + y[x] == 1 - x^2,
              y[0] == 1, y[1] == 2}, y[x], x] // First // Simplify
```

$$\{y(x) \to -x^2 + 12\, e^{x-1}\, x - 4\, x - 6\, e^x\, (x-1) - 5\}$$

```
Plot[y[x] /. yAx, {x, 0, 1}, AxesLabel → {x, y}]
```



Antag att vi delar in vårt område i n st lika långa element.

```
n = 5;
```

Dessa blir då h långa.

```
h = 1.0 / n
```

0.2

Andraderivatan approximeras med framåtdifferens och bakåtdifferens $y''(x_i) \approx \frac{\frac{y_{i+1}-y_i}{h} - \frac{y_i-y_{i-1}}{h}}{h} = \frac{y_{i+1}-2\,y_i+y_{i-1}}{h^2}$, vilket vi känner igen som centraldifferens. Vi ser att denna blir symmetrisk som sig bör. Här har vi även möjlighet att göra approximationen av derivatan symmetrisk genom att använda centraldifferens, det vill säga $y'(x_i) \approx \frac{y_{i+1}-y_{i-1}}{2h}$. Detta leder till ett ekvationssystem där varje rad motsvarar differentialekvationen applicerad i tur och ordning på de inre noderna.

```
ekv = Table[(y_{i+1} - 2 y_i + y_{i-1})/h^2 - 2 (y_{i+1} - y_{i-1})/(2 h) + y_i == 1 - (i h)^2, {i, 1, n - 1}]
```

$\{y_1 - 5.\,(y_2 - y_0) + 25.\,(y_0 - 2\,y_1 + y_2) = 0.96,\ y_2 - 5.\,(y_3 - y_1) + 25.\,(y_1 - 2\,y_2 + y_3) = 0.84,$
$y_3 - 5.\,(y_4 - y_2) + 25.\,(y_2 - 2\,y_3 + y_4) = 0.64,\ y_4 - 5.\,(y_5 - y_3) + 25.\,(y_3 - 2\,y_4 + y_5) = 0.36\}$

Nu är detta ekvationssystem singulärt och kräver liksom tidigare randvärde (RV), här $\begin{cases} y(0) = y_0 = 1 \\ y(1) = y_n = 2 \end{cases}$. Sedan är det bara att lösa det.

```
lösning = Solve[ekv /. {y_0 → 1, y_n → 2}] // First
```

$\{y_1 \to 1.1,\ y_2 \to 1.24299,\ y_3 \to 1.43733,\ y_4 \to 1.68898\}$

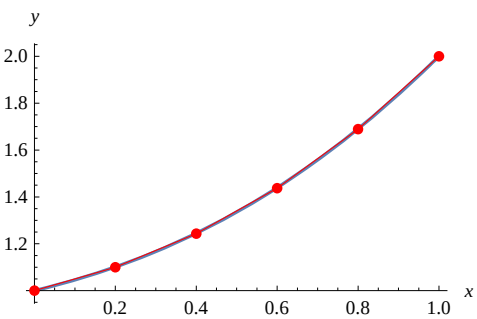
eller i tabellform $(x_i,\ y_i)$.

```
xyFDM = Table[{i/n, y_i}, {i, 0, n}] /. lösning /. {y_0 → 1, y_n → 2}
```

$$\begin{pmatrix} 0 & 1 \\ \frac{1}{5} & 1.1 \\ \frac{2}{5} & 1.24299 \\ \frac{3}{5} & 1.43733 \\ \frac{4}{5} & 1.68898 \\ 1 & 2 \end{pmatrix}$$

Slutligen en jämförelse med den analytiska lösningen.

```
Plot[y[x] /. yAx, {x, 0, 1}, AxesLabel → {x, y},
    Epilog → {Red, PointSize[0.025], Line[xyFDM], Point[xyFDM]}]
```



Som tidigare nämnts kan man förvänta sig bättre noggrannhet om en finare indelning väljs. Eftersom vi har symmetriska approximationer kan man visa att felet i varje nod är proportionellt mot h^2.

11.9 Uppgifter

11.1 Lös (BVP) $\begin{cases} y'(x) = y(x)\cos(x) & \text{(ODE)} \\ y(0) = 1 & \text{(BV)} \end{cases}$ för $x \in [0, 20]$ med $h = \frac{20}{n}$, $n \in \{50, 100, 200, 400\}$ och explicit Eulers metod. Jämför med **DSolve**. Rita i samma firgur. Jämför!

11.2 Lös (BVP) $\begin{cases} y'(x) = \sin(x\,y(x)) & \text{(ODE)} \\ y(0) = 6 & \text{(BV)} \end{cases}$ för $x \in [0, 35]$ med $h = 0.2$ och explicit Eulers metod. Jämför med **NDSolve**. Rita i samma firgur. Upprepa för $h = 0.1$. Jämför!

11.3 Uppgift 11.1 men med Heuns metod.

11.4 Övertyga dig om att du förstått Runge-Kuttas metod genom att återge den schematiskt på samma sätt som finns i texten för Heuns metod och Mittpunktsmetoden.

11.5 Lös (BVP) $\begin{cases} y'(x) = y(x) + x & \text{(ODE)} \\ y(0) = 1 & \text{(BV)} \end{cases}$ för $x \in [0, 1]$ med $h = 0.2$ med Runge-Kuttas metod. Jämför med **DSolve**. Rita i samma firgur. Upprepa för $h = 0.1$. Jämför!

11.6 Lös (BVP) $\begin{cases} y''(x) = x + 2y(x) + y'(x) & \text{(ODE)} \\ y(0) = 2,\ y'(0) = 1 & \text{(BV)} \end{cases}$ för $x \in [0, 1]$ med $h = 0.2$ och explicit Euler. Låt $u(x) = (y(x), v(x))^T$ och skriv om på matrisform (BVP) $\begin{cases} u'(x) = \mathbb{A}u(x) + \mathbb{B}(x) & \text{(ODE)} \\ u(0) = (2, 1)^T & \text{(BV)} \end{cases}$, där $\mathbb{A}_{2\times 2}$ är en konstant matris och $\mathbb{B}_{2\times 1}(x)$. Jämför med **DSolve**. Rita dem i samma figur.

11.7 Lös (BVP) $\begin{cases} y'(x) = -20(y(x) - \cos(x)) & \text{(ODE)} \\ y(0) = 0 & \text{(BV)} \end{cases}$ för $x \in \left[0, \frac{\pi}{2}\right]$. Systemet är mycket styvt, eftersom homogena lösningen snabbt går mot noll. Jämför explicit Euler för allt minskande $h = \frac{\pi}{2n}$, $n = 10, 20, \ldots, 50$, med implicit Euler. Jämför med **DSolve**. Rita dem i samma figur.

11.8 Lös (BVP) $\begin{cases} y''(x) + 101y'(x) + 100y(x) = 0 & \text{(ODE)} \\ y(0) = 1,\ y'(0) = -11 & \text{(BV)} \end{cases}$ för $x \in [0, 1]$. Låt $u(x) = (y(x), v(x))^T$ och skriv om på matrisform (BVP) $\begin{cases} u'(x) = \mathbb{A}u(x) & \text{(ODE)} \\ u(0) = (1, -11)^T & \text{(BV)} \end{cases}$, där $\mathbb{A}_{2\times 2}$ är en konstant matris. Visa att egenvärdena till $\mathbb{A}$ är vitt skilda, så systemet är styvt! Låt även **DSolve** visa att $y(x)$ innehåller en snabbt och en långsamt avtagande term. Implicita trapetsmetoden är mycket lämplig här. Låt steglängden vara h och visa att implicita trapetsmetoden, i detta fall, kan skrivas $\left(\mathbb{I} - \frac{h}{2}\mathbb{A}\right)u_{i+1} = \left(\mathbb{I} + \frac{h}{2}\mathbb{A}\right)u_i$. Välj lite olika h och stega på! Egentligen behövs bara små h "i början" under det att den snabba termen klingar ut. Jämför med **DSolve**. Rita dem i samma figur.

11.9 En sportbilstillverkare begränsar prestandan för en av sina modeller genom att vid full gas styra bränsletillförseln så att accelerationen i varje ögonblick är proportionell mot skillnaden mellan önskad toppfart 80 m/s (≈ 288 km/h) och aktuell fart med proportionalitetskonstanten 0.1 s^{-1}. Försumma luftmotståndet och använd Newtons accelerationslag $m\ddot{x} = F$, som löses med **DSolve**.

a) Formulera och lös (BVP) som bestämmer bilens läge $x(t)$.
b) Vilken fart har bilen efter 20 s om den startar från stillastående med gasen i botten?
c) Hur långt har den kört då?
d) Hur lång tid tar det till 50 m/s (= 180 km/h) och hur långt har den då kört?

11.10 Ett så kallat bungy jump är välbekant för de flesta. Från en hoppställning högt ovan mark kastar sig en person handlöst ut. Livlinan utgörs av en gummisnodd vars ena ände är fäst vid uthoppsplatsen och den andra surrad runt vristerna på offret. Placera en y–axel med origo vid marken och pekande uppåt så har vi återigen Newton, $m\ddot{y} = F$. Krafterna som verkar är tyngdkraften och den från gummisnodden. Eftersom en gummisnodd, för enkelhets skull antar vi, fungerar ungefär som en fjäder frånsett att den inte kan ta trycklaster, är den lite kinkig att simulera och lösa analytiskt. Men med **NDSolve** går det både snabbt och enkelt. För att offret ska slippa svänga upp och ned i all oändlighet, vilket är fallet med en fjäder, lägger vi in lite dämpning i snodden. Med fjäderkonstanten k, dämpning c och den naturliga längden L gäller om hoppställningen har höjden H att kraften i snodden

$$F_{\text{gs}} = \begin{cases} k(H-y-L) - c\dot{y} & \text{om } (H-y-L) > 0 \\ 0 & \text{om } (H-y-L) \le 0 \end{cases}$$. Vi får då till slut (BVP) $$\begin{cases} m\ddot{y} = -mg + F_{\text{gs}} & \text{(ODE)} \\ y(0) = H & \\ \dot{y}(0) = 0 & \text{(BV)} \end{cases}$$.

Välj nu t.ex. $m = 75$ kg, $H = 50$ m, $L = 30$ m, $k = 125$ N/m och $c = 50$ Ns/m. Gör sedan en simulering av spektaklet de $T = 15$ första sekunderna. Rita $y(t)$ och $\dot{y}(t)$.

11.11 En fallskärmshoppare, vars vikt är 75 kg, hoppar ut från en helikopter 4000 m ovan markytan. Antag att de enda krafterna som påverkar rörelsen är tyngdkraften och luftmotståndet från fallskärmen som är proportionellt mot hopparens fart. Proportionalitetsfaktorn är $k_1 = 15$ kg/s när fallskärmen är outlöst och $k_2 = 105$ kg/s när fallskärmen är utvecklad. Antag att fallskärmen utvecklas 60 s efter uthoppet. Sök läge $y(t)$ och hastighet $\dot{y}(t)$ under resan samt bestäm hur lång tid hoppet tar. Använd **`NDSolve`** och byt $k_1 \to k_2$ i flykten med **`If`** (smidigt jämfört med att lösa två (BVP)) och **`WhenEvent`** för att hålla koll på landningstiden. Rita läge och hastighet!

11.12 När tomtarna åker på utflykt glömmer de stänga ett fönster på andra våningen i stugan. Plötsligt sker ett utsläpp av giftig gas i Tomteland så att koncentrationen i luften utanför stugan är konstant 0.1 mg/m^3. Låt flödet genom fönstret vara 4 m^3/min och mellan våningarna 2 m^3/min. Första våningen i stugan är på 100 m^3 och andra på 75 m^3. Antag perfekt omrörning på våningarna.

a) Härled det system av (BVP) som bekriver hur koncentrationen av den giftiga gasen utvecklas på första $f(t)$ respektive andra $a(t)$ våningen.

b) Använd **`DSolve`** för att lösa (BVP).

c) Rita och se om utvecklingen stämmer med hur ni tänkt er den!

d) Efter 3 h är luften åter helt ren utanför stugan. Modifiera (BVP) i **a)** med **`If`** för att hantera den plötsliga ändringen. Lös (BVP) med **`NDSolve`**.

e) Rita och se om utvecklingen stämmer med hur ni tänkt er den!

f) Bestäm högsta koncentrationen på första och andra våningen, samt vid vilka tidpunkter det inträffar!

g) När har nivån sjunkit under 0.005 mg/m^3, så tomtarna kan återvända?

11.13 Avsluta Picards metod i **Exempel 11.13** så att $\hat{y}_n(x)$ åtminstone stämmer med den jämförande Taylorutvecklingen.

11.14 Visa Adams korrektor-prediktorformler (11.15) och (11.16).

11.15 Visa Heuns metod (11.6) med hjälp av receptet som står i slutet av avsnittet om Mittpunktsmetoden.