Methods used in numerical analysis

Solving Equations

The following methods are iterative methods of solving equations of the form f(x) = 0.

- **Bisections method**

The function f(x) has a root at x = r if f(r) = 0

A solution is correct within p decimal places if the error is less than 0.5 x 10^(-p)

Solution error after n steps: |x-r| < (b-a)/(2^(n+1))

With n + 2 function evaluations

Guaranteed linear convergence, 1 function evaluation per step.

```
While (b-a)/2 > TOL
        c = (a+b)/2
        if f(c) = 0 stop, end
        if f(a)f(c) < 0
        b = c
        else
        a = c
        end
end
```

- **Fixed-point iteration (FPI)**

The real number r is a fixed point of the function g if g(r) = r

Wright the equation as g(x) = x then start the iteration below

x_i = initial guess
x_{i+1} = g(x_i) for i = 0,1,2,…

The sequence x_i may or **may not converge** as the number of steps goes to infinity. But if g is continuous and x_i converges to a number r, then r is a fixed point.

Assume that g is continuously differentiable, that g(r) = r, and that S =|g´(r)| < 1. Then Fixed-Point Iteration **converges linearly** with rate S to the fixed point r for initial guesses sufficiently close to r.
This is because if g`(r) is greater than 1 the vertical steps in the FPI increase in length as the iterations proceed, this results in instability and divergence. (The iterations spirals out from the fixed point). When g´(r) is less than 1 the vertical steps in the FPI decrease in length and the iteration converges to the fixed point.

The method **has locally linearly convergence when it converges**; it makes 1 function evaluation per step. May be faster or slower than Bisection method dependent if S is smaller or bigger than 1/2. **Requires a stopping criterion**, iterate until a set tolerance is reached |x_{i+1}- x_i| < TOL.

- **Conditioning**

Backward error = change in input that makes x_c the correct solution

Forward error = change in the solution that would make x_c correct, which for root-finding problems is $|x_c$ -r|

Error magnification = relative forward error/relative backward error ≈ |g(r)|/|rf¨(r)|

The condition number of a problem is defined to be the maximum error of magnification over all input changes. A problem with high condition number is called ill-conditioned, and a problem with a condition number near 1 is called well-conditioned.

- **Newton´s method**

x_0 = initial guess
$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ for i = 0,1,2,…

At a simple root, f´(r) ≠ 0, we have **quadratic convergence**. At a multiple root, f´(r) = 0, we have **linear convergence**. (Bisection and FPI)

There **is no guarantee for convergence** when using Newton´s method. (Example if f´(x_i) = 0)

- **Secant method**

If the derivative can´t be constructed the secant method could work as **a replacement for newton´s method.** In this method the tangent line is replaced with an approximation called the secant line. The method converges almost as fast as newton´s method.

Geometrically, the tangent line is replaced with a line through the two last known guesses. The intersection point of the "secant line" is the new guess.

x_0, x_1, = initial guesses
x_{i+1},= x_i, $\frac{\mathrm{f}(x_i)\cdot(x_i - x_{i-1})}{(\mathrm{f}(x_i) - \mathrm{f}(x_{i-1}))}$ for i =1,2,3,…

NOTE: two starting guesses are needed to start the Secant method iteration.

The secant method **has super linear convergence** to simple roots, meaning that it lies between linearly and quadratically convergent methods.

- **Simplified Newton´s method**

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_0)}$$

Solving systems of equations of the form Ax = b

- **The LU factorization**

The LU-factorization is a matrix representation of Gaussian elimination. It consists of writing the coefficient matrix A as a product of a lower triangular matrix L and an upper triangular matrix U.

When L and U are known we write the problem Ax = b as Lux = b. Define the vector c = Ux. Then do the following to points to get the solution:

a) Solve Lc = b for c
b) Solve Ux = c for x

Complexity of the method: Classical Gaussian elimination will require approximately kn^3/3 operations (O(n^3/3), where A is an n x n – matrix, since we have to start over at the beginning for each problem. However, the LU-factorization method requires approximately n^3/3 + kn^2, operations, when n^2 is small compared with n^3 this is a significantly better (cheaper, faster) method than classical Gaussian elimination. This is because the RHS b doesn´t enter the calculations until the elimination (LU) is finished, therefor we can solve the previous set of equations with only one elimination, followed by two back substitutions (Lc = b, Ux = c) for each new b.

- **The Jacobi method**

The Jacobi method is a form of FPI iteration for a system of equations.

Solve the i:th equation for the i:th unknown. Then, iterate as in FPI, starting with an initial guess.

x_0 = initial vector
x_{k+1} = $D^{-1}(b-(L+U)x_k)$ for k = 0,1,2,…

Where, D = diag(A), L=lowTri(A), U = upperTri(A).

The method is **convergent if** the matrix A is strictly diagonally dominant.

The n x n matrix A = (a_{ij}) is **strictly diagonally dominant** if, for each $1 \leq i \leq n$, $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$. In other words, each main diagonal entry dominates its row in the sense that it is greater in magnitude than the sum of the magnitudes of the remainder of the entries in its row.

- **Gauss-Seidel method**

The only difference between Gauss-Seidel and Jacobi is that in the former, the most recently updated values of the unknowns are used at each step, even if updating occurs in the current step.

Gauss-Seidel often converges faster than Jacobi if the method is convergent. The method is convergent if the matrix A is strictly diagonally dominant.

x(0) = initial vector
x(k+1) = $D^{-1}(b-Ux_k-Lx_{k+1})$ for k = 0,1,2,…

Interpolation

In **interpolation** we are given a set of data points, then, by different methods we want to construct a polynomial that interpolates the whole set of data points**. The main theorem of interpolation:** Let (x_1, y_1)…(x_n, y_n) be n points in the plane with distinct x_i. Then there exists one and only one polynomial P of degree n – 1 or less that satisfies P$(x_i) = y_i$ for i = 1,…,n. (that interpolates the set of data points).

- **The Vandermonde matrix**

The Vandermonde matrix evaluates a polynomial at a set of points; formally, it transforms coefficients of a polynomial $a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ to the values the polynomial takes at the points α_i. The non-vanishing of the Vandermonde determinant for distinct points α_i shows that, for distinct points, the map from coefficients to values at those points is a one-to-one correspondence, and thus that the polynomial interpolation problem is solvable with unique solution.
They are thus useful in interpolation, since solving the system of linear equations Vu = y for u with V an m × n Vandermonde matrix is equivalent to finding the coefficients u_j of the polynomial(s)

$$P(x) = \sum_{j=0}^{n-1} u_j x^j$$

of degree ≤ n − 1 which has the property

$$P(\alpha_i) = y_i \quad \text{for } i = 1, \ldots, m.$$

The Vandermonde matrix can easily be inverted in terms of Lagrange basis polynomials: each column is the coefficients of the Lagrange basis polynomial, with terms in increasing order going down. The resulting solution to the interpolation problem is called the Lagrange polynomial.

- **Lagrange interpolation**

If we are given n points (x_1, y_1)…(x_n, y_n).

For each k between 1 and n, define the term

$$L_k(x) = \frac{(x - x_1)(x - x_2) \cdot \ldots (x - x_{k-1})(x - x_{k+1}) \ldots (x - x_n)}{(x_k - x_1)(x_k - x_{k-1})(x_k - x_{k+1}) \ldots (x_k - x_n)}$$

Then define the n – 1 polynomial

$$P_{n-1}(x_k) = y_1L_1(x_k) + \cdots + y_nL_n(x_k) = y_kL_k(x_k) = y_k$$

- **Runge phenomenon**

Polynomials can fit any set of data points. However, there are some shapes that polynomials prefer over others. Try data points that cause the function to be zero at equally spaced points except for x = 0, where we set a value of 1. The polynomial that goes through points situated like this refuses to stay between 0 and 1, unlike the data points. This is called the Runge phenomenon.
It is usually used to describe extreme “polynomial wiggle” associated with high-degree polynomial interpolation at evenly-spaced points.
The characteristic of the Runge phenomenon is that the polynomial has large errors near

the outside of the interval of the data points. To fix this we can move some of the interpolation points towards the outside of the interval, where the function producing data can be better fit. We can for example use so called **Chebyshev points**.

- **Chebyshev interpolation**

Chebyshev interpolation is a way of **choosing a spacing** of the points so that we get an optimal curve-fitting. This would make the interpolation error smaller.

Define the n:th Chebyshev polynomial by $T_n(x) = \cos(n \cdot \arccos(x))$ then the recursion relation for the Chebyshev polynomials is $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$. This means that the maximum absolute value of $T_n(x)$ is 1.

Chebyshev interpolation nodes:
On the interval [a,b],

$x_i = \frac{b+a}{2} + \frac{b-a}{2}\cos\left(\frac{(2i-1)\pi}{2n}\right)$ for i = 1,…,n.
The inequality

$|(x-x_1)\cdots(x-x_{n)}| \leq \frac{\left(\frac{b-a}{2}\right)^n}{2^{n-1}}$

holds on [a,b].

- **Splines**

The idea of splines is to use several formulas, each a low-degree polynomial, to pass through the data points instead of just one as in the previous methods. The simplest example of a spline is a **linear spline**, in which one "connects the dots" with straight-line segments. The negative part with linear splines is that they lack of smoothness. However, this can be solved using cubic splines instead.

A cubic spline replaces linear functions between the data points by degree 3 (cubic) polynomials. This means that there are infinitely many cubic splines that go through any set of data points.

A cubic spline S(x) through the data points $(x_1, y_1), \dots, (x_n, y_n)$ is a set of cubic polynomials

$S_{n-1}(x) = y_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3$ on $[x_{n-1}, x_n]$

With the following **properties:**

P1: $S_i(x_i) = y_i$ and $S_i(x_{i+1}) = y_{i+1}$ for i = 1,…,n-1
P2: $S'_{i-1}(x_i) = S'_i(x_i)$ for i = 2,…,n-1
P3: $S''_{i-1}(x_i) = S''_i(x_i)$ for i = 2,…,n-1

P1 guarantees that the spline S(x) interpolates the data points. P2 forces the slopes of neighboring parts of the spline to agree where they meet, and P3 does the same for the curvature, represented by the second derivative. This means that the curve-fitting becomes smoother.

P4: A cubic spline that satisfies $S''_1(x_1) = 0$ and $S''_{n-1}(x_n) = 0$ is called a **natural cubic spline**

Curve fitting

- **Least squares**

Used when Ax=b has no solution, instead we use the least squares approximation to determine a solution.

The Least squares is a classic example of **data compression**. The input consists of a set of data points, and the output is a model that, with relatively few parameters, fits the data as well as possible. Usually, the reason for using least squares is to replace noisy data with a plausible underlying model that is then used for signal prediction or classification purposes.

Step 1: Choose a model, identify a parameterized model, such as y=a+bt, which will be used to fit the data.
Step 2: Force the model to fit the data, substitute the data points into the model. Each data point creates an equation whose unknowns are the parameters, such as a and b in the line model. This results in a system Ax=b, where the unknown represents the unknown parameters.
Step 3: Solve the normal equations, the least squares solution for the parameters will be found as the solution to the system of normal equations $A^T Ax = A^T b$.

We can now measure our success at fitting the data, we calculate the residual of the least squares solution xhat as r = b – Axhat
If the residual is the zero vector, then we have solved the original system Ax = b exactly. If not, the Euclidean length of the residual vector is a measure of how far xhat is from being a solution. The Euclidean length of a vector is called the **2-norm**.

Since input data is assumed to be subject to errors in least squares problems; it is especially important to reduce error magnification. Using the normal equations to solve the least squares problem is fine for small problems. However, the condition number cond($A^T A$) is approximately the square of the original cond(A), which will greatly increase the possibility that the problem is ill-conditioned. More sophisticated methods allow computing the least squares solution directly from A without forming $A^T A$. An example of such a method is the QR-factorization and the singular value decomposition.

- **QR factorization**

The QR factorization is a superior method compared to the normal equations. The QR factorization of the matrix A is:

$$(v_1|\dots|v_k) = (q_1|\dots|q_n)\begin{pmatrix} \begin{bmatrix} r_{11} & \cdots & r_{1k} \\ \vdots & \ddots & \vdots \\ r_{k1} & \cdots & r_{kk} \end{bmatrix} \\ \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \end{pmatrix}$$

The square matrix Q is orthogonal if $Q^T = Q^{-1}$.

Doing calculations with orthogonal matrices is preferable because they are easy to invert by definition, also they do not magnify errors hence a better conditioning.

Given the m x n inconsistent system Ax=b, factor A=QR and set Rhat=upper n x n sub matrix of R, dhat=upper n entries of d=$Q^T b$, solve Rhat·x=dhat for least squares solution x.

Numerical integration – Quadratures

Quadratures are different numerical methods for approximating the integral of a given function; this can be done by using interpolation and least squares modeling.

- **Trapezoid Rule**

This is the simplest application of interpolation-based numerical integration.

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}(y_0 + y_1) - \frac{h^3}{12}f''(c)$$

Where c is between x_0 and x_1

- **Simpson´s Rule**

Same as the trapezoid rule except that the degree 1 interpolant is replaced by a parabola.

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(y_0 + 4y_1 + y_2) - \frac{h^5}{90}f^{(iv)}(c)$$

Where h = $x_2 - x_1 = x_1 - x_0$ and c is between x_0 and x_2.

Both the trapezoid rule and Simpson´s rule are examples of “closed” Newton-Cotes formulas, because they include evaluations of the integrand at the interval endpoints. The “open” Newton-Cotes formulas are useful for circumstances where that is not possible, for example, when approximating an improper integral.

- **Composite Newton-Cotes formulas**

The trapezoid and Simpson´s rules are limited to operating on a single interval. We can evaluate an integral by dividing the interval up into several subintervals, applying the rule separately on each one, and then totaling up. This has as a result that the error is decreased. The strategy of dividing the intervals into subintervals is called **composite numerical integration.**

The composite Trapezoid Rule is simply the sum of Trapezoid Rule approximations on adjacent subintervals.

$$\int_a^b f(x)dx = \frac{h}{2}\left(y_0 + y_m + 2\sum_{i=1}^{m-1} y_i\right) - \frac{(b-a)h^2}{12}f''(c)$$

Where h = (b-a)/m and c is between a and b.

The composite Simpson´s Rule follows the same strategy:

$$\int_a^b f(x)dx = \frac{h}{3}\left(y_0 + y_{2m} + 4\sum_{i=1}^{m} y_{2i-1} + 2\sum_{i=1}^{m-1} y_{2i}\right) - \frac{(b-a)h^4}{180}f^{(iv)}(c)$$

Where c is between a and b.

- **Gaussian Quadrature**

The degree of precision of a numerical integration method is the greatest integer k for which all degree k or less polynomials are integrated exactly by the method. The Newton-Cotes methods of degree n have degree of precision n for odd n and degree of precision n+1 for even n. Gaussian Quadrature has degree of precision 2n+1 when n+1 points are used hence this is a far better method then the Newton-Cotes methods.

Gaussian Quadrature of a function is linear combinations of function evaluations at the **Legendre roots**. We achieve this by approximating the integral of the desired function by the integral of the interpolating polynomial, whose nodes are the Legendre roots.

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} c_i f(x_i)$$

Where

$$c_i = \int_{-1}^{1} L_i(x)dx\,, i = 1, \dots, n$$

The Gaussian Quadrature method, using the n:th degree Legendre polynomial on [-1,1], has degree of precision 2n-1

ODEs

- **Euler´s method**

The Euler method is a first-order numerical procedure for solving ODEs, which means that the local error (error per step) is proportional to the square of the step size, and the global error (error at a given time) is proportional to the step size. The Euler method often serves as the basis to construct more complicated methods.

Explicit Euler
calculates the state of a system at a later time from the state of the system at the current time.

Implicit Euler
finds a solution by solving an equation involving both the current state of the system and the later one

Implicit methods require an extra computation, and they can be much harder to implement. Implicit methods are used because many problems arising in practice are stiff, for which the use of an explicit method requires impractically small time steps Δt to keep the error in the result bounded. (stability)

- **Multistep methods**

Multistep methods can achieve the same order with less computational effort, usually just one function evaluation per step. Since multistep methods use more than one previous w value, they need help getting started. The start-up phase for a s-step method typically consists of a one-step method that uses w_0 to produce s-1 values $w_1, w_2, \dots, w_{s-1}$, before the multistep method can be used.

Adams-Bashforth two-step method (explicit method)

$$w_{i+1} = w_i + h\left[\frac{3}{2}f(t_i, w_i) - \frac{1}{2}f(t_{i-1}, w_{i-1})\right]$$

This method needs w_1 along with a given initial condition w_0 in order to begin. (The w can be produced using for example Trapezoid method).

Adams-Moulton two-step method(implicit method)

$$w_{i+1} = w_i + \frac{h}{12}[5f_{i+1} + 8f_i - f_{i-1}]$$

There are significant differences between the implicit and explicit methods. First, it is possible to get a stable third-order implicit method by using only two steps, unlike the explicit case. Second, the corresponding local truncation error formula is smaller for implicit methods. However, the implicit method has the inherent difficulty that extra processing is necessary to evaluate the implicit part.

For nonstiff differential equations use explicit Euler or implicit Euler with FPI.
For stiff differential equations use implicit Euler with Newton iteration.

Implicit and explicit methods are often used together, each step is the combination of a **prediction by the explicit method** and a **correction by the implicit method**. An example of a predictor-corrector pair is the explicit method Adams-Bashforth two-step method as a predictor with the implicit Adams-Moulton one-step method as the corrector. Both are second order methods.

BVPs

- **Shooting method**

This method converts the boundary value problem (BVP) into an initial value problem (IVP) by determining the missing initial values that are consistent with the boundary values.

- **Finite Difference Method**

The fundamental idea behind this method is to replace derivatives in the differential equation by discrete approximations, and evaluate on a grid to develop a system of equations.

For linear systems apply Gaussian elimination,LU- or QR factorization or Jacobi iteration to solve the resulting linear system AX = b.

For nonlinear systems apply Newton´s method to solve the resulting nonlinear system F(x) = 0.