Numerical analysis - Summary

databogdan

24 March, 2020

Contents

1 General Concepts and Taylor (L1)

In math a few problems can be solved exactly. For example

$$x + y = 1 \tag{1}$$
$$x - y = 0 \tag{2}$$

Most real life problems must be solved numerically, i.e with an approximation and an error. These are typically **Non-linear equations** or **Differential equations**.

> "Numerical analysis is about constructing and analyzing quantitative methods for the computation of solutions to mathematical problems."

The main concerns of numerical analysis are:

1. **Approximating** the solution
2. **Cost** of the approximation
3. **Error** of the approximation

With cost we mean number of operations (NOO), such as addition, multiplication etc. Different factors in real life such as computing power, type of processor etc may also be needed to taken into account, but this is math and therefore only NOO.

Because we get approximation, we are not interested in exact answers but instead approximations within a given tolerance. This tolerance is commonly our **Stopping Criteria**, i.e stop iterating if $E < TOL$.

1.1 Error of Approximation

Absolute Error

Assume that the approximation $\hat{p}(x)$ of the real solution $f(x)$. Then the absolute error is given by

$$E = |f(x) - \hat{p}(x)| \tag{3}$$

Relative Error

Another type of error is the relative error which is a percentage given by

$$E = \frac{|f(x) - \hat{p}(x)|}{|\hat{p(x)}|} \tag{4}$$

Truncation Error

A truncation error is for example only keeping Taylor terms up to a certain degree, e.g

$$e^x \approx 1 + x \tag{5}$$

Round-off Error

Computers can not represent all real numbers exactly because they are discrete. This leads to an inevitable round off error, e.g 2.1234567 may be rounded up to 2.1 in a computer if only few bits can represent the value.

Forward Error

Given by $|r-x_n|$, also called the horizontal distance since it measures distance in x-direction.

Backward Error

Given by $|f(x_n)|$, also called the vertical distance since it measures distance in y-direction.

1.2 Convergence Order

How fast a given algorithm, method, scheme converges to the correct value is important.

Definition:

Let $e_n = |r - x_n|$ be the error of an iterative method. If

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^k} = S < 1 \tag{6}$$

then the method is said to converge with *order* k and *rate* S

Different orders and rates

If the order $k = 1$ then we have the following rates:

1. if $S \to 0$ - **super linear** convergence
2. if $S \to 1$ - **sub linear** convergence
3. if $S = 1$ - **logarithmic** convergence
4. if $S > 1$ - **diverge**

For orders $k = 2, 3$, i.e **quadratic**- and **cubic** convergence, or higher then the method converges with rate S if and only if $0 < S < \infty$. To be clear, this means that if S diverges to ∞ then the method also diverges for that convergence order.

1.3 Taylor's Theorem 1D

Let f be $k+1$ times continuously differentiable function on the interval between x and x_0 for given real numbers x and x_0. Then there exist a number ζ in the interval such that

$$f(x) = P(x) + R(x) \tag{7}$$

where the polynomial $P(x)$ (approximation term) of degree k is given by,

$$f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + ... + \frac{f^k(x_0)}{k!}(x - x_0)^k \tag{8}$$

and the remainder $R(x)$ (error term) of degree $k+1$ is given by

$$\frac{f^{k+1}(\zeta)}{(k+1)!}(x-x_0)^{k+1} \tag{9}$$

Taylor's Theorem 2-D Example

Let $f(x,y)$ have 3 continuous partial derivatives on the interval between (x,y) and (x_0,y_0) for given real points (x,y) and (x_0,y_0). Then there exist a point (α,β) in the interval such that

$$f(x,y) = P(x,y) + R(x,y) \tag{10}$$

where the polynomial $P(x,y)$ (approximation term) is given by,

$$f(x_0,y_0) + f'_x(x_0,y_0)(x-x_0) + f'_y(x_0,y_0)(y-y_0) +$$
$$\frac{1}{2!}[f''_{xx}(x_0,y_0)(x-x_0)^2 + 2f''_{xy}(x_0,y_0)(x-x_0)(y-y_0) + f''_{yy}(x_0,y_0)(y-y_0)^2]$$

and the remainder $R(x,y)$ (error term) is given by

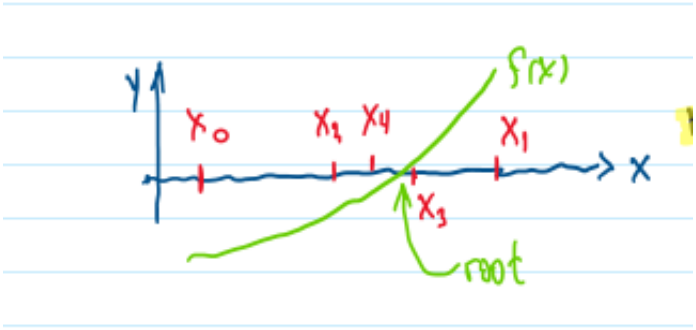
$$\frac{1}{3!}[f'''_{xxx}(\alpha,\beta)(x-x_0)^3 + 3f'''_{xxy}(\alpha,\beta)(x-x_0)^2(y-y_0) + \tag{11}$$
$$3f'''_{xyy}(\alpha,\beta)(x-x_0)(y-y_0)^2 + f'''_{yyy}(\alpha,\beta)(y-y_0)^3] \tag{12}$$

---

2 Root Finding (L2, L3, L4)

2.1 Bisection Method

$$x_{n+1} = \frac{x_n + x_{n-1}}{2} \tag{13}$$



- **What** - Solves $f(x) = 0$
- **Requires** - Continuous $f(x)$, two starting points $[a_0, b_0]$
- **How** - Divide interval until number of iterations or tolerance have been reached
- **Advantages** - Simple, always converge, precise formulation of error
- **Disadvantages** - Slow convergence, may discard early good approximation

Bisection Theorem:

Suppose the following

1. f is continuous
2. $f(r) = 0$ for some $r \in [a, b]$
3. $f(a)f(b) < 0$, i.e opposite signs

Then the sequence produced by the Bisection Method converges to the root r of f

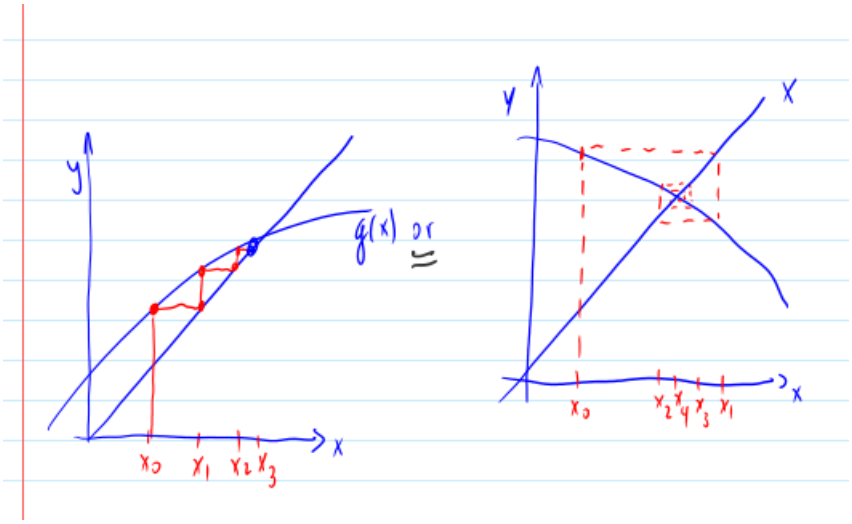
$$\lim_{n\to\infty} x_n = r \tag{14}$$

and the absolute error is given by

$$|r - x_n| \leq \frac{b_n - a_n}{2} = \ldots = \frac{b_0 - a_0}{2^{n+1}} \tag{15}$$

2.2 Fixed Point Iteration (FPI)

$$x_{n+1} = g(x_n) \tag{16}$$



- **What** - Solves $g(x) = x$
- **Requires** - Continuous $g(x)$, one starting points x_0
- **How** - Iterate from starting point
- **Advantages** - Simple
- **Disadvantages** - Does not always converge, rate of convergence depends on $g'(x)$

Contractive:

A continuous g is contractive on an interval $[a, b]$ if

$$|g'(x)| < 1, \text{ for all } x \in (a, b) \tag{17}$$

FPI Theorem:

The function $g : [a, b] \to \mathbf{R}$ has a unique fixed points if

1. $g : [a, b] \to [a, b]$ (existence)
2. g is contractive on (a, b)

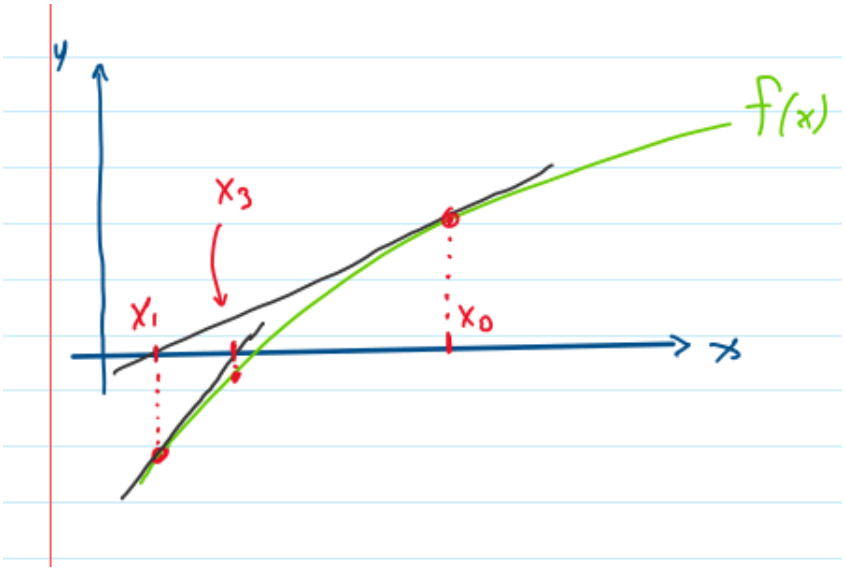
Then any initial $x_0 \in [a, b]$ will converge to the root. The absolute error is then given by

$$|x_n - r| \leq \frac{c^n}{1 - c}|x_1 - x_0| \tag{18}$$

where $c = max|g'(x)| < 1$ on the interval.

2.3 Newton Method 1D

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{19}$$



For this to work, our initial x_0 needs to be close to the root we are trying to approximate.

- **What** - Solves $f(x) = 0$ (linear and nonlinear f)
- **Requires** - Continuous $f(x)$, one starting points x_0
- **How** - Iterate from starting point by following the tangent line of $f(x_n)$ down to find the next x_{n+1} value
- **Advantages** - Fast (can be quadratic near root), error estimate and number of iterations given a tolerance can be calculated
- **Disadvantages** - Does not always converge (depends on starting point and f), costly to calculate $f'(x)$, problems if $f'(x) = 0$

The convergence rate of Newton near a root can be found by Taylor expanding $f(x)$ with the second derivative as error term. The result is **quadratic** convergence

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^2} = |-\frac{f''(\zeta)}{2f'(x_n)}| \approx |\frac{f''(r)}{2f'(r)}| \tag{20}$$

Assuming that $f'(x_n) \neq 0$ for all x_n and $f''(r) < \infty$.

2.4 Secant Method

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{21}$$

- **What** - Solves $f(x) = 0$ (linear and nonlinear f)
- **Requires** - Continuous $f(x)$, two starting points x_0, x_1
- **How** - Iterate from starting points by following the secant of $f(x_n)$ and $f(x_{n-1})$ down to find the next x_{n+1} value
- **Advantages** - Not too difficult to implement, no need to calculate derivatives, faster convergence than the Bisection Method

- **Disadvantages** - Does not always converge, costly function evaluations, may be slower than the Newton Method

The convergence rate of Secant Method can be shown to be **super linear**.

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^k} = S < \infty, \text{ where } k = \frac{1+\sqrt{5}}{2} \approx 1.62 \tag{22}$$

---

3 Linear systems, eigenvalues (L5, L6, L7, L8, L9, L10, L11)

This chapter will cover methods to solve $Ax = b$. These types of linear systems are found everywhere. If the equations describing a system is hard, i.e non-linear, we can always simplify with Taylor expansion. Keeping the first order terms for our approximation and the quadratic terms as error. After the Taylor expansion, we are back to a system that can be described as $Ax = b$.

Matrix Algebra

3.1 Vector and Matrix Norms

A vector or matrix norm, denoted $||\cdot||$, is a way to define distance in a space. The common used norm i Euclidean, but there exists more. Mathematically a vector or matrix norm is a real valued function which takes vectors or matrices as inputs and outputs a scalar.

Any given **vector** norm have these properties:

1. $||x|| \geq 0$ for all $x \in \mathbf{R^n}$
2. $||x|| = 0$ if and only if $x = \vec{0}$
3. $||kx|| = |k| \cdot ||x||$ for all $k \in \mathbf{R}$ and $x \in \mathbf{R^n}$
4. $||x + y|| \leq ||x|| + ||y||$ for all $x, y \in \mathbf{R^n}$

Similarly a **matrix** norm have these properties:

1. $||A|| \geq 0$ for all $A : N \times M$
2. $||A|| = 0$ if and only if $A = 0$
3. $||kA|| = |k| \cdot ||A||$ for all $k \in \mathbf{R}$
4. $||A + B|| \leq ||A|| + ||B||$
5. $||AB|| \leq ||A|| \cdot ||B||$

l_1-norm

For a column **vector**, $n \times 1$ the norm is defined as

$$||x||_1 = \sum_{i=1}^{n} |x_i| \tag{23}$$

For a **matrix** A, $r \times c$ this norm is defined as

$$||A||_1 = \max_{1 \leq j \leq c} \sum_{i=1}^{r} |a_{ij}| \tag{24}$$

l_2-norm

Also know as the Euclidean and is the typical length of a row or column vector we are use too.

$$||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{25}$$

Notice that this equation is only defined for row or column vectors.

l_∞-norm

For a column **vector**, $n \times 1$ the norm is defined as

$$||x||_\infty = \max_{1 \leq i \leq n} |x_i| \tag{26}$$

For a **matrix** A, $r \times c$ this norm is defined as

$$||A||_\infty = \max_{1 \leq i \leq r} \sum_{j=1}^{c} |a_{ij}| \tag{27}$$

Spectral Radius Definition:

Let A be a $n \times n$ matrix with complex or real elements with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. Then the spectral radius is defined as

$$\rho(A) = \max_{1 \leq i \leq n} |\lambda_i| \tag{28}$$

Theorem:

Suppose that A is an $n \times n$ matrix. Then

$$\sqrt{\rho(A^T A)} = ||A||_2 \tag{29}$$

$$\rho(A) \leq ||A|| \text{ for any matrix norm} \tag{30}$$

$$\frac{1}{\min |\lambda|} = ||A^{-1}||_2 \text{ for A symmetric} \tag{31}$$

3.2 Condition Number of a Matrix

A condition number of a matrix measures how sensitive the answer is to small changes in the input data and to round-off errors made during the solution process.

This is important, as it informs us on how much we should trust our approximations. A large condition number is bad for our approximation.

Definition:

$$k(A) = ||A|| \cdot ||A^{-1}|| \tag{32}$$

where $1 \leq k(A) \leq \infty$. If A is symmetric, we have

$$k(A) = \frac{\max |\lambda|}{\min |\lambda|} \tag{33}$$

Practical Understanding of k(A):

Suppose we have a computer which rounds numbers to 10^{-15}, i.e 15 decimals. Suppose that the condition number of your matrix A is 10^{10}. The condition number tells you that out of the 15 decimals, you can only trust 5 in your answer.

3.3 Eigenvalues and Eigenvectors

For a square matrix A, $n \times n$, we define eigenvalues, λ, and eigenvectors, x, as follows:

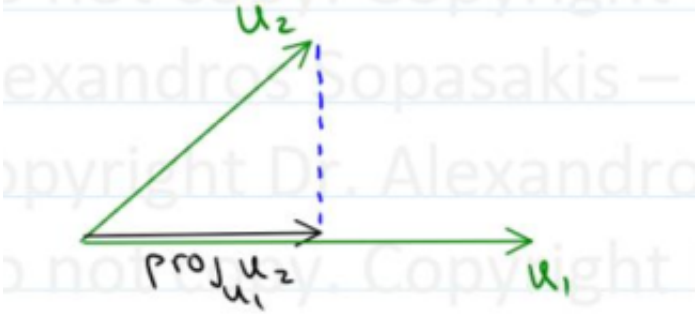
$$Ax = \lambda x \tag{34}$$

Theorem:

Let the eigenvalues of the square matrix A be $\lambda_1, \lambda_2, \ldots, \lambda_n$. Then:

1. The eigenvalues of the inverse matrix A^{-1} are $\lambda_1^{-1}, \lambda_2^{-1}, \ldots, \lambda_n^{-1}$ given that A^{-1} exists. The eigenvectors are the same
2. The eigenvalues of the shifted matrix $A - sI$ are $\lambda_1 - s, \lambda_2 - s, \ldots, \lambda_n - s$. The eigenvectors are the same

3.4 Projection



Given two vectors u_1, u_2 we can find the orthogonal projections u_2' of u_2 onto u_1 by the following formula:

$$u_2' = \frac{u_2 \cdot u_1}{||u_1||^2} u_1 \tag{35}$$

And to find the vector from u_2' tip to the tip of u we simply take $u_2 - u_2'$. We have now for vectors that are orthogonal that we also can normalize.

Gram-Schmidt Process

The goal of this method is to othonormalize a set of given vector. This is done with repeated uses of the projection formula. Given the vectors $u_1, u_2, u_3, \ldots, u_k$ we do the following:

$$v_1 = u_1 \tag{36}$$
$$v_2 = u_2 - \frac{u_2 \cdot v_1}{||v_1||^2} v_1 \tag{37}$$
$$v_3 = u_3 - \frac{u_3 \cdot v_2}{||v_2||^2} v_2 - \frac{u_3 \cdot v_1}{||v_1||^2} v_1 \tag{38}$$
$$\vdots \tag{39}$$

Finally all orthogonal vectors are normalized $q_j = \frac{v_j}{||v_j||}$. The **cost** is $\mathcal{O}(nk^2)$ where n is the dimension of the vectors and k is the number of vectors.

3.5 Newtons Method $\mathbf{R^n}$

$$x_{n+1} = x_n - DF^{-1}(x_n)F(x_n) \tag{40}$$

The method is an iterative method giving an approximation. In the equation DF is the **Jacobian Matrix**.

$$\begin{bmatrix} \frac{\delta f_1}{\delta x_1} & \frac{\delta f_1}{\delta x_2} & \cdots & \frac{\delta f_1}{\delta x_n} \\ \frac{\delta f_2}{\delta x_1} & \frac{\delta f_2}{\delta x_2} & \cdots & \frac{\delta f_2}{\delta x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\delta f_n}{\delta x_1} & \frac{\delta f_n}{\delta x_2} & \cdots & \frac{\delta f_n}{\delta x_n} \end{bmatrix} \tag{41}$$

There are some problems with the Jacobian matrix

1. It might be hard to calculate or even impossible. Solution to this is to approximate the Jacobian
2. We always avoid calculating the inverse by rewriting it $Y = DF^{-1} \iff DF \cdot Y = F$ and solve for Y

The convergence rate of the multidimensional Newton Method is also **quadratic** near a root. This is also shown by Taylor expanding near the root as in the 1D case.

3.6 More Iterative Methods for $Ax = b$

Let $A = L + D + U$, where A is the matrix, L the lower triangular elements, D the diagonal element and U the upper triangular elements. Then a fixed point iteration can be performed

$$x_{n+1} = Gx_n + C \tag{42}$$

where

$$G = I - Q^{-1}A \tag{43}$$
$$C = Q^{-1}b \tag{44}$$

We still need to know what Q is. How we choose the Q matrix depends on which method we want to use. We have the following methods:

1. **Jacobi** - $Q = D$
2. **Richardson** - $Q = I$
3. **Gauss-Seidel** - $Q = (D + L)$

Iterative Theorem 1:

If A is strictly **diagonally dominant**, i.e

$$|a_{kk}| > \sum_{j=1}^{n} |a_{kj}| \text{ , where } j \neq k \tag{45}$$

for all rows, then all the iterative methods converge for any initial choice of x_0

Iterative Theorem 2:

All the iterative methods converge for any initial choice of x_0 if and only if

$$\rho(G) < 1 \tag{46}$$

where $\rho(G)$ is the spectral radius.

General information about the iterative methods:

- Gauss-Seidel is faster than Jacobi
- Gauss-Seidel and Jacobi have a cost of $\mathcal{O}(n^2)$ operations per iteration
- One iterative method may converge to solution while the others may not. Check which converge with the spectral radius!

Gaussian elimination may be better than iteration depending on size of system. If

$$\frac{\ln tol}{\ln \rho(G)} < \frac{n}{3} \tag{47}$$

then iterative methods are better than Gaussian elimination (n is the size of A)

3.7 Gaussian elimination

The normal method when faced with a linear system $Ax = b$ and gives an exact solution. You simply perform the elimination and a back substitution.

The **cost** of Gaussian **elimination** is

$$\frac{2n^3}{3} + \frac{n^2}{2} - \frac{7n}{6} \tag{48}$$

and for **back substitution** is n^2.

3.8 LU factorization

Solves $Ax = b$ by factorization of A. The power of this method is solving the same system multiple times for different b_i.

$$PA = LU \implies Ax = b \iff LUx = Pb \tag{49}$$

Where L is a lower triangular matrix with ones in the diagonal, U is a upper triangular matrix and P is the **permutation matrix**. A permutation matrix keeps track of row changes while performing Gaussian elimination on the original A matrix. This needs to be done to because of a phenomenon called **swamping**. To prevent swamping we check that each **partial pivoting** element are the largest in it's column, i.e $|a_{p1}| \geq |a_{i1}|$.

The **cost** of LU factorization is approximately the same as Gaussian

$$\approx \frac{2n^3}{3} + 2n^2 \tag{50}$$

However, this cost is for finding L, U, P and back substitution. After the matrices are found, you can solve the same system for a different b_i for the cost of $2n^2$ as L and U are triangular and therefore only needs a back substitution.

3.9 QR Factorization

We factorise our matrix to $A = QR$ where Q is an orthogonal matrix and R is an upper triangular matrix. This is done by the following method:

1. Create Q matrix out of all vectors q_j from Gram-Schimdt with A's column vectors as input
2. R is then simply $Q^T A$

We like this method because orthogonal matrix $\implies Q^{-1} = Q^T$. To solve $Ax = b$ we do

$$Ax = b \iff QRx = b \iff Rx = Q^T b \tag{51}$$

Because this is upper triangular this is easy to solve. If no solution exists we get the best least square approximation of the solution.

3.10 Power Method

Eigenvalues are extremely important and this method finds the greatest absolute eigenvalue for a given matrix A by iterating this to find the eigenvector:

$$x_{k+1} = \frac{Ax_k}{||Ax_k||} \tag{52}$$

After finding the eigenvector for the largest eigenvalue, we retrieve the eigenvalue with the **Rayleigh quotient**:

$$\lambda = \frac{x^T A x}{x^T x} \tag{53}$$

Assuming $A : n \times n$ with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ satisfying $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ the Power Method **converges** rate of convergence is given by

$$\frac{|\lambda_{k+1}|}{|\lambda_k|} \tag{54}$$

and is **linear**. This for almost any initial vector.

3.11 Rayleigh Quotient Iteration

To find all eigenvalues given a matrix A we can use the Power Method but in iterations. Assume the matrix is of size n, then we have n eigenvalues. To find all we simply run the Power Method to find the largest eigenvalue. Now that we now this value, we can shift it out of the matrix, i.e $A_{new} = A - \lambda I$. Applying the Power Method to A_{new} will now give back the second largest eigenvalue of the original matrix A. Repeating this will give all eigenvalues.

The **convergence** is quadratic but can be cubic if A symmetric.

To run the algorithm we need an initial vector x_0 and a square matrix A.

for *j = 1, 2, 3, ...* **do**
  $u_{j-1} = \frac{x_{j-1}}{||x_{j-1}||}$ % normalize
  $\lambda_{j-1} = u_{j-1}^T A u_{j-1}$ % Rayleigh
  $Solve(A - \lambda_{j-1} I) x_j = u_{j-1}$ % Inverse Power Method (could just use Power Method?)
end

---

4 Interpolation (L12, L13, L14, L15, L16, L17)

In real life we do not have continuous functions. Instead, we use data sampled at certain times. These data points $(x_0, y_0, z_0, \dots), (x_1, y_1, z_1 \dots), \dots$ can be of several dimensions. We use this to represent a discrete function. From these discrete points, we can interpolate a continuous function, i.e our model consisting of elementary functions. Another motivation is that real life measurements come with errors and noise. Many times the given data **over determine** the problem, i.e more equations than variables, making a solution non-existent.

4.1 Least Square

As in statistics when we try to approximate parameters for a distribution, the Least Squares method tries to find the solution which minimizes the error from the "real value". The underlying model can both be **linear** and **non-linear**. The general idea is that after gathering data points you assume a model and insert the measured points into the model:

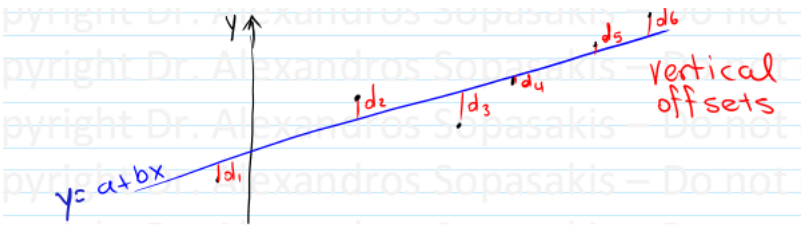
$$y_0 = f(x_0) \tag{55}$$
$$y_1 = f(x_1) \tag{56}$$
$$y_2 = f(x_2) \tag{57}$$
$$\vdots \tag{58}$$
$$\iff y = Ac \tag{59}$$

We now want a solution with the smallest possible squared error

$$R^2 = d_1^2 + d_2^2 + d_1^2 + \cdots + d_n^2 \tag{60}$$

where n is the number of data points, R is the residual and d_j is the Euclidean distance from model to the measured value.

If **linear** we have the model $y = a + bx$



and therefore

$$d_j = (y_j - (a + bx_j)) \tag{61}$$

We are interested in the Least Square which minimizes R which depends on the parameters a, b. This is achieved by

1. Take the derivative of R and set it to zero

2. Solve for the parameters a and b

This can also be done by using the QR-method above and should give same solution.

For a **non-linear** model, for example exponential model $y = C_1 e^{kx}$, we do the same procedure but now $f(x)$ is $C_1 e^{kx}$ instead of $a+bx$ as in the linear case. Inserting the data points we get a system $Ac = y$ which we can solve using the QR-method to get the least square approximation. We are once again here trying to find the coefficients of our model.

4.2 Polynomial Interpolation

Given data points a natural model for us to use is a polynomial. These are easy to use, compute derivatives, primitives etc.

Interpolating Polynomial Definition:

A polynomial $p \in \mathbf{P_n}$ interpolates the points (x_i, y_i) for $i = 0, \ldots, n$ if

$$p(x_k) = y_k, \, \forall k \in \{0, \ldots, n\} \tag{62}$$

Werestrass Approximation Theorem:

Suppose $f(x)$ is a continuous function on $[a, b]$. Then $\forall \epsilon > 0$ there exists polynomial $p(x)$ defined on $[a, b]$ such that

$$|f(x) - p(x)| \leq \epsilon, \, \forall x[a, b] \tag{63}$$

In simple English - we can find a polynomial as close as we like to any given function $f(x)$.

4.3 Vandermonde Method

This is the easiest interpolating approach. Given $n + 1$ distinct points use an n degree polynomial $p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ and evaluate it for each data point (x_i, y_i).

$$y_0 = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n \tag{64}$$
$$y_1 = a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n \tag{65}$$
$$\vdots \tag{66}$$
$$y_n = a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n \tag{67}$$

This gives us a linear system $Va = y$

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & & \ddots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{68}$$

Some properties of the Vandermonde matrix V are:

1. If all a_i are distinct then $\det(V) \neq 0$
2. A square Vandermonde matrix is invertible if and only if the a_i are distinct

The advantage of this method is the simplicity. However, the **disadvantage** is that the Vandermonde matrix will have a **large condition number**.

4.4 Lagrange Interpolation

Given data points (x_i, y_i) the goal of this method is interpolate these $n+1$ points with a polynomial of degree n:

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \tag{69}$$

You find the Lagrange polynomial by doing the following. Let

$$p_n(x) = \sum_{i=0}^{n} y_i L_i(x), \text{ where} \tag{70}$$

$$L_i(x) = \prod_{j=0, i\neq j}^{n} \frac{x - x_j}{x_i - x_j} \iff \tag{71}$$

$$\text{(written out ...) } \frac{(x-x_0)(x-x_1)\dots(x-x_n)}{(x_i-x_0)(x_i-x_1)\dots(x_i-x_n)} \tag{72}$$

The **error estimate** of Lagrange interpolation can be proven to be

$$|f(x) - p_n(x)| < \frac{f^{(n+1)}(\zeta)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \tag{73}$$

The **advantages** of Lagrange are:

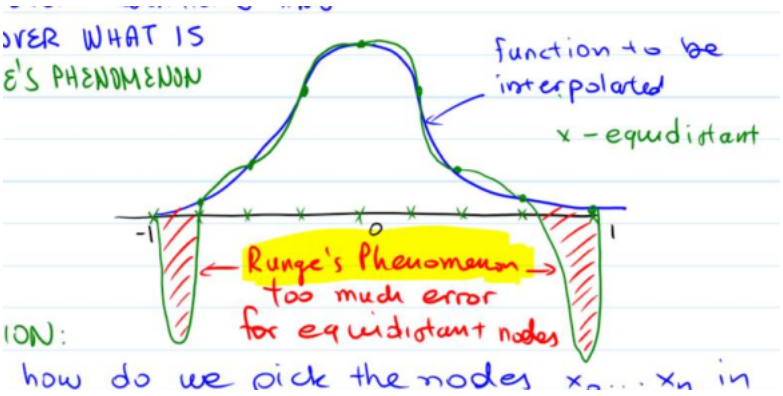
- Error estimate is provided
- Does not restrict nodes to be evenly spaced at x

The **disadvantages** of Lagrange are:

- The error is very difficult to know in advance since normally we do not know the actual function $f(x)$
- We must recompute everything from the beginning if an extra data point is added to existing data set

4.5 Chebyshev

If Lagrange is used with equally spaced data points we run into **Runge's Phenomenon**. The phenomenon is that the error increase drastically at the end points.



To remedy this, and minimize the error estimate of Lagrange we instead pick points that are not equally spaced, but given by the roots of **Chebyshev polynomials**. The Chebyshev polynomials are given by

$$T_n(x) = \cos\left(n \arccos\left(x\right)\right), \ -1 \leq x \leq 1 \tag{74}$$

or the recursive formula

$$T_{n+1} = 2x \cdot T_n(x) - T_{n-1}(x) \ , \ n = 1, 2 \ldots \tag{75}$$
$$T_0 = 1 \tag{76}$$
$$T_1 = x \tag{77}$$

The x_i are the roots of the Chebyshev polynomials on the interval $[-1, 1]$. To transform the new points to any interval we use

$$\hat{x_i} = \frac{1}{2}[(b-a)x_i + a + b] \tag{78}$$

where $\hat{x_i}$ is the new x-point of the interval $[a, b]$.

4.6 Cubic Splines

Having a n-degree polynomial for $n+1$ data points is over kill and some times bad. Another idea is to split all the data points into intervals and use several interconnecting cubic polynomials to interpolate.

Cubic Spline Definition: The polynomial $\mathbf{S} \in \mathbf{C}^2[a, b]$ is called a cubic spline on $[a, b]$ if it is a third degree polynomial in each interval between the given nodes (x_j, y_j) for $j = 0, \ldots, n$, i.e

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{79}$$
$$\text{for } i = 0, 1, \ldots, n-1 \tag{80}$$

with the following properties:

1. $S_i(x_i) = y_i$ for $i = 0, \ldots, n-1$
2. $S_{i-1}(x_i) = y_i$ for $i = 1, \ldots, n$
3. $S'_{i-1}(x_i) = S'_i(x_i)$ for $i = 1, \ldots, n-1$
4. $S''_{i-1}(x_i) = S''_i(x_i)$ for $i = 1, \ldots, n-1$

The properties gives us $4n-2$ conditions. There are $4n$ parameters a_i, b_i, c_i, d_i. We need two more conditions to solve this system and therefore we make assumptions about the end points:

- **Natural Spline** - $S''_0(x_0) = 0$ and $S''_{n-1}(x_n) = 0$
- **Clamped Spline** - $S'_0(x_0) = v_1$ and $S'_{n-1}(x_n) = v_2$
- **Curvature Adjusted Spline** - $S''_0(x_0) = v_1$ and $S''_{n-1}(x_n) = v_2$
- **Not-a-Knot Spline** - $S'''_0(x_1) = S'''_1(x_1)$ and $S'''_{n-2}(x_{n-1}) = S'''_{n-1}(x_{n-1})$

The **error** for a **Natural Cubic Spline** is given by

$$|S(x) - f(x)| \leq \frac{h^4}{180} |f^{(5)}(\zeta)| \tag{81}$$

where h is the equal distance between points.

To find the **coefficients** a_i, b_i, c_i, d_i we first define a new variable $\sigma_i = S''_i(x_i)$ and $\sigma_n = S''_{n-1}(x_n)$. From this variable we can describe all other:

$$a_i = \frac{\sigma_{i+1} - \sigma_i}{6h} \tag{82}$$

$$b_i = \frac{\sigma_i}{2} \tag{83}$$

$$c_i = -h\frac{2\sigma_i + \sigma_{i+1}}{6} + \frac{y_{i+1} - y_i}{h} \tag{84}$$

$$d_i = y_i \tag{85}$$

We still need do not know our σ_i. However these can be find using

$$\sigma_{i-1} + 4\sigma_i + \sigma_{i+1} = \frac{6}{h^2}(y_{i-1} - 2y_i + y_{i+1}) \tag{86}$$

The equation above forms the following system

$$\begin{bmatrix} 4 & 1 & 0 & \cdots & 0 \\ 1 & 4 & 1 & \cdots & 0 \\ 0 & 1 & 4 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 4 & 1 & 0 \\ 0 & \cdots & 1 & 4 & 1 \\ 0 & \cdots & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \vdots \\ \vdots \\ \sigma_{n-1} \end{bmatrix} = \frac{6}{h^2} \begin{bmatrix} y_0 - 2y_1 + y_2 - \sigma_0 \\ y_0 - 2y_1 + y_2 \\ \vdots \\ \vdots \\ \vdots \\ y_{n-2} - 2y_{n-1} + y_n - \sigma_n \end{bmatrix} \tag{87}$$

This a $Ax = b$ system which is always solvable because A is diagonally dominant. To solve this system, the **cost** is $\mathcal{O}(k^2 n)$ where k is the number of diagonals, in our case above $k = 3$, and n is the dimension of A.

4.7 Bezier Curves

Bezier curves are used widely in computer graphics. Instead of focusing on interpolation we are now interested in **smoothness** and **functionality**.

Bezier Curve Definition:

Given a set of points $\{P_i = (x_i, y_i)\}$, $i = 0, 1, \ldots, n$ then the following is a parametric Bezier curve of degree n:

$$P(t) = (x(t), y(t)) = \sum_{i=0}^{n} P_i B_i^n(t) \text{ , where explicitly} \tag{88}$$

$$x(t) = x_o B_o^n(t) + x_1 B_1^n(t) + \cdots + x_n B_n^n(t) \tag{89}$$

$$y(t) = y_o B_o^n(t) + y_1 B_1^n(t) + \cdots + y_n B_n^n(t) \tag{90}$$

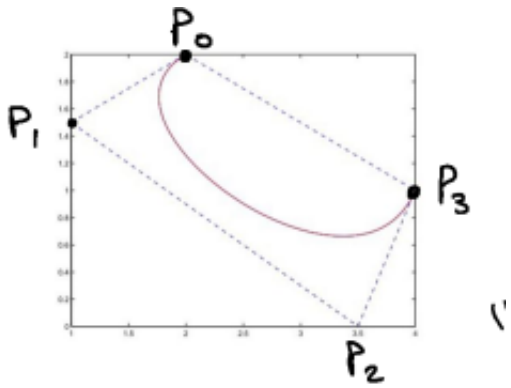
$$\text{where } t \in [0, 1] \tag{91}$$

The $B_i^n(t)$ are the so called **Bernstein Polynomials** defined as

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} \cdot t^i \tag{92}$$

A short example for a **third degree** Bezier Curve. We need 4 points for a curve of degree 3, i.e $(x_0, y_0), (x_1, y_1), \ldots, (x_3, y_3)$. The first and last points are called **end points** and the middle points are called **control points**. Properties of this curve is

1. $P(t)$ is continuous and has derivatives of all orders
2. $P(0) = P_0$ and $P(1) = P_3$ lie on the curve i.e the end points are interpolated
3. The curve is always tangent to $P_1 - P_0$ and $P_3 - P_2$



You can create **Composite Bezier Curves** from using repeated third degree Bezier Curves. When connecting these curves, we ensure that they are continuous but not necessarily smooth.

To get the derivatives to agree we need to pick the middle points next to the shared end points on a line.

The de Casteljau Algorithm:

To compute a point $t = t_0$, on the Bezier Curve, $B(t)$, with control points $P_0, P_1, \ldots, P_n$:

1. Define $P_i^{(0)} = P_i$ for $i = 0, 1, \ldots, n$

2. Execute

for *j = 1:n* **do**
  for *i = 0:n-j* **do**
    $P_i^j = P_i^{j-1}(1 - t_0) + P_{i+1}^{j-1} \cdot t_0$
  end
end

3. $B(t_0) = P_0^n$

---

5 Fourier Transforms (L17, L18, L19)

The Fourier transforms of interest is the **Discrete Fourier Transform (DFT)** and the **Fast Fourier Transform (FFT)**. These remarkable transforms have been endless applications and have been crucial in for example sending signals. The transforms rely on complex numbers so we start with a short review.

5.1 Complex Arithmetic

A number is complex if it contains the imaginary number $i = \sqrt{-1}$ and a complex number is further determined by it's real and imaginary parts $z = a + ib$. A complex number can also be represented on polar from

$$z = a + ib \iff \tag{93}$$
$$re^{i\theta} \text{ ,where} \tag{94}$$
$$e^{i\theta} = \cos\theta + i\sin\theta \tag{95}$$
$$r = \sqrt{a^2 + b^2} \tag{96}$$

A complex conjugate $\bar{z} = a - ib$ is defined as flipping the sign of the imaginary part.

Roots of Unity

An nth root of unity, $n \in \mathbf{N}^+$, is a number z satisfying the equation $z^n = 1$. The nth roots of unity are given

$$z = e^{\frac{2\pi k}{n}i}, \text{ for } k = 0, 1, \ldots, n-1 \tag{97}$$

Primitive Root of Unity:

An nth root of unity is primitive if $z^k \neq 1$ for $k = 1, 2, \ldots, n-1$.

Important **properties** of primitive nth roots of unity are:

Let $w = e^{-\frac{2\pi l}{n}i}$, where $l < n$

1. $1 + w^k + w^{2k} + \cdots + w^{(n-1)k} = 0$ for $1 \leq k \leq n-1$
2. $1 + w^n + w^{2n} + \cdots + w^{(n-1)n} = n$
3. $w^{-1} = w^{n-1}$
4. If n is even, $w^{\frac{n}{2}} = 1$

5.2 The Discrete Fourier Transform

In this transform we have as input evenly spaced data points (t_i, x_i). From these inputs we try to find the modes/frequencies for the sinus, cosinus waves that can describe the input signal as a sum of these waves. The transform is given by

$$y_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j w^{jk} \text{ , for } k = 0, 1, \ldots, n-1 \tag{98}$$
$$w = e^{-\frac{2\pi}{n}i} \tag{99}$$

or as a system of $Fx = y$ where F is the **Fourier Matrix**

$$\frac{1}{\sqrt{n}}\begin{bmatrix} w^0 & w^0 & w^0 & \cdots & w^0 \\ w^0 & w^1 & w^2 & \cdots & w^{n-1} \\ w^0 & w^2 & w^4 & \cdots & w^{2(n-1)} \\ \vdots & & \ddots & & \vdots \\ w^0 & w^{n-1} & w^{2(n-1)} & \cdots & w^{(n-1)(n-1)} \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} \tag{100}$$

The Fourier Matrix is both **orthogonal** and **symmetric**. Hence the inverse is given by the conjugate transpose of F

$$F^{-1} = \bar{F}^T \tag{101}$$

Interpolating with DFT

Given data points $x_0, x_1, \ldots, x_{n-1}$ occuring at evenly spaced points on the interval $[c, d]$ denoted by $t_j = c + j\frac{(d-c)}{n}$ for $j = 0, 1, \ldots, n-1$. Then

$$P(t) = \frac{1}{\sqrt{n}}\sum_{k=0}^{n-1}(a_k \cos\frac{2\pi k(t-c)}{d-c} - b_k \sin\frac{2\pi k(t-c)}{d-c}) \tag{102}$$

where $P(t_j) = x_j$, i.e interpolating, and $Fx_j = a_j + ib_j = y_j$. This can be simplified further to

$$P(t) = \frac{a_0}{\sqrt{n}} + \frac{2}{\sqrt{n}}\sum_{k=1}^{\frac{n}{2}-1}(a_k \cos\frac{2\pi k(t-c)}{d-c} - b_k \sin\frac{2\pi k(t-c)}{d-c}) + \frac{a_{n/2}}{\sqrt{n}}\cos\frac{n\pi(t-c)}{d-c} \tag{103}$$

Here we assume n is even. If n is not even you can still use this formula as the effect is negligible.

A small trick you can do to avoid calculating sinus and cosinus functions is for $x_0, x_1, \ldots, x_{n-1}$ data points assume that the modes $y_n, y_{n+1}, \ldots$ i.e higher nodes than you can calculate are all equal to zero. Then when you inverse transform with $\bar{F}^T \cdot y = x$ we get more data points out. These points are on the original interpolating curve. This can be seen as extrapolating in the frequency plane.

Compression with DFT

You do not need to use all the nodes y_j in the polynomial $P(t)$. Leaving out some of the modes, $m < n$, leads to the best **Least Square Approximation** and therefore compressing the signal to only it's crucial frequencies.

$$P_{lq}(t) = \frac{a_0}{\sqrt{n}} + \frac{2}{\sqrt{n}}\sum_{k=1}^{\frac{m}{2}-1}(a_k \cos\frac{2\pi k(t-c)}{d-c} - b_k \sin\frac{2\pi k(t-c)}{d-c}) + \frac{a_{m/2}}{\sqrt{n}}\cos\frac{n\pi(t-c)}{d-c} \tag{104}$$

Filtering with DFT

Similar to compression, we can choose which modes we want in our resulting signal. In real life, a signal often have high frequency noise. This can be filtered out by only taking the first modes which is equivalent to the lower frequencies. Note! We can also filter our lower frequencies and keep the higher, or do band of frequencies. The sky is the limit.

5.3 The Fast Fourier Transform

FFT is built upon the principal of **divide and conquer**. Instead of computing the modes y_k of a signal of size N at a cost of $\mathcal{O}(N^2)$ we compute modes of 2 signals of size $N/2$ at a cost $2\frac{N^2}{4} = N^2/2$ which is cheaper. If N is a power of 2, e.g $N = 2^q$, you can repeat this process q times, each time reducing a factor of 2 in cost. The signals will then have length $\frac{N}{2^q} = 1$ and the cost is $\frac{N^2}{2^q} = \frac{N^2}{N} = N$ and we do this q times. The **total cost** of FFT will then be $qN = N \log_2 N$.

In a strict proof you show that the recurrence relation can be described as

$$T(N) = 2 * T(N/2) + N. \tag{105}$$

This gives us $\mathcal{O}(N)$ at each level and we have $\log_2(N) = q$ levels.

---

6 Numerical Integration (L20, L21)

Normally we cannot calculate an integration and therefore we approximate it numerically with a sum:

$$\int_a^b f(x)\,dx \approx \sum_{k=0}^{n} w_k\, f(x_k) \tag{106}$$

where w_k is the a weight function and $f(x_k)$ is a function evaluation. There are many formulas that do this but we are mainly interested in

- **Newton-Cotes**
- **Composite Newton-Cotes**
- **Gaussian Quadrature**

Algebraic Degree of Accuracy (ADA)

The algebraic degree of accuracy of a quadrature formula is given by the power of the polynomial $p_n(x)$ for which the quadrature is exact, i.e no error. This is also sometimes called *degree of precision.*

This measurement only works for polynomials and is of interest as we normally do not know $f(x)$ and approximate it with polynomials.

6.1 Newton-Cotes

This is the simplest of the methods. Given a number of data points we try to approximate the area under them. The main idea is to first approximate $f(x)$ given the data points and then estimate the area. To estimate $f(x)$ we can use interpolating methods such as Lagrange, DFT, Cubic Spline etc. To find the error of our approximation we Taylor expand and see that our equation, e.g for the Trapezoid rule (2 points) we have

$$\int_{x_1}^{x_2} f(x)\,dx = \int_{x_1}^{x_2} p(x)\,dx + \int_{x_1}^{x_2} e(x)\,dx \tag{107}$$

$$\int_{x_1}^{x_2} e(x)\,dx = \frac{1}{2} f''(\zeta) \int_{x_1}^{x_2} (x - x_0)(x - x_1)\,dx \tag{108}$$

This can be generalized to more points. If a integration formula (quadrature) includes end points of interval then it is called a **closed** Newton-Cotes Quadrature and other wise **open** Newton-Cotes Quadrature.

ADA of Newton-Cotes Theorem:

Suppose the following quadrature for $n + 1$ points

$$\int_a^b f(x)\,dx \approx \sum_{k=0}^{n} w_k\, f(x_k) \tag{109}$$

Then the formulas are exact up to polynomials of degree $n+1$ if **n is even** or exact up to polynomials of degree n if **n is odd**.

The commonly used formulas are listed below. Assume n is derived from data points $x_0, \ldots, x_n$.

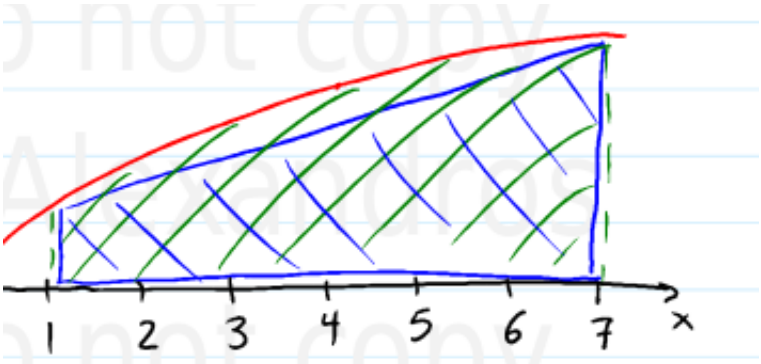
Midpoint

The input is one point x_0 and h is $(x_1 - x_{-1})/2$

$$\int_{x_{-1}}^{x_1} f(x)\,dx = 2hf(x_0) + \frac{h^3}{3}f''(\zeta) \tag{110}$$

The ADA of midpoint is 1.

Trapezoidal

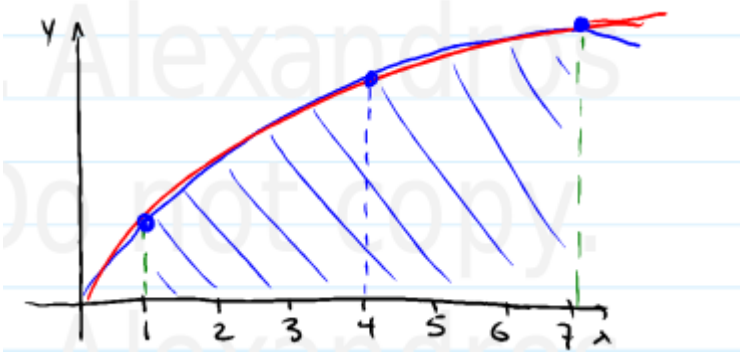


The input is two points x_0, x_1 and h is $(x_1 - x_0)$

$$\int_{x_0}^{x_1} f(x)\,dx = \frac{h}{2}[f(x_0) + f(x_1)] - \frac{h^3}{12}f''(\zeta) \tag{111}$$

The ADA of midpoint is 1.

Simpsons's



The input is three points x_0, x_1, x_2 and h is the equidistant between the points, e.g $(x_1 - x_0)$

$$\int_{x_0}^{x_2} f(x)\,dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{90}f^{(4)}(\zeta) \tag{112}$$

The ADA of midpoint is 3.

Simpsons's 3/8 Rule

The input is four points x_0, x_1, x_2, x_3 and h is the equidistant between the points, e.g $(x_1 - x_0)$

$$\int_{x_0}^{x_3} f(x)\,dx = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] - \frac{3h^5}{80}f^{(4)}(\zeta) \tag{113}$$

The ADA of midpoint is 3.

6.2 Composite Newton-Cotes

The Newton-Cotes formulas are good as long as the interval in not wide. If the interval is wide, the simplistic nature of the formulas such as assuming a line does not differ much from the real $f(x)$ leads to big errors. Therefore, if we split up the interval and use composites of the simplistic formulas, we get good accuracy.

Composite Midpoint Rule

Suppose $f \in \mathbf{C}^2[a, b]$. Then the composite midpoint rule for, $a = x_0, x_1, \ldots, x_n = b$, is given by

$$\int_a^b f(x)\,dx = 2h\sum_{j=0}^{n/2} f(x_{2j}) - \frac{(b-a)h^2}{6}f''(\zeta) \tag{114}$$

where $h = \frac{b-a}{n+2}$, $x_j = a + (j+1)h$ for $j = -1, 0, \ldots, n, n+1$ and $\zeta \in (a, b)$.

Composite Trapezoidal Rule

Suppose $f \in \mathbf{C}^2[a, b]$. Then the composite trapezoidal rule for $n+1$ points, $a = x_0, x_1, \ldots, x_n = b$, is given by

$$\int_a^b f(x)\,dx = \frac{h}{2}[f(a) + f(b) + 2\sum_{j=1}^{n-1} f(x_j)] - \frac{(b-a)h^2}{12}f''(\zeta) \tag{115}$$

where $h = \frac{b-a}{n}$, $x_j = a + jh$ for $j = 0, \ldots, n$ and $\zeta \in (a, b)$.

Composite Simpson's Rule

Suppose $f \in \mathbf{C}^{(4)}[a, b]$. Then the composite Simpson's rule for $n+1$ points, $a = x_0, x_1, \ldots, x_{2m} = b$, is given by

$$\int_a^b f(x)\,dx = \frac{h}{3}[f(a) + f(b) + 2\sum_{j=1}^{m-1} f(x_{2j}) + 4\sum_{j=1}^{m} f(x_{2j-1})] - \frac{(b-a)h^4}{180} f''(\zeta) \tag{116}$$

where $h = \frac{b-a}{2m}$, $x_j = a + jh$ for $j = 0, \ldots, n$ and $\zeta \in (a, b)$.

General Formula

To create a general quadrature formula given a interval $[a, b]$ and data points $x_0, x_1, \ldots, x_n$ on the interval you simply do

$$\int_a^b f(x)\,dx \approx w_0 f(x_0) + w_1 f(x_1) + \cdots + w_n f(x_n) \tag{117}$$

Then just assuming $f(x)$ as growing degree of polynomials we get a system of equations where we can find all coefficients w_i

$$\int_a^b 1\,dx \approx w_0 f(x_0) + w_1 f(x_1) + \cdots + w_n f(x_n) \tag{118}$$

$$\int_a^b x\,dx \approx w_0 f(x_0) + w_1 f(x_1) + \cdots + w_n f(x_n) \tag{119}$$

$$\int_a^b x^2\,dx \approx w_0 f(x_0) + w_1 f(x_1) + \cdots + w_n f(x_n) \tag{120}$$

$$\vdots \tag{121}$$

You stop assuming higher degree polynomials when you have the same amout of equations as w_i, i.e n equations. The ADA of this quadrature is n or $n + 1$.

6.3 Gaussian Quadratures

Just like interpolation, having equally distanced data points is sub optimal. A smarter choice of the data points can be necessary because of existing data and also produce better approximations.

Legendre Polynomials

This is *orthogonal* polynomials, similar to Chebyshev, defined at the interval $x \in [-1, 1]$ and have the following properties:

$$LE_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n \text{ , for } x \in [-1, 1] \tag{122}$$

$$LE_n(x) \text{ is polynomial of degree n for } x \in [-1, 1] \tag{123}$$

$$\int_{-1}^{1} Q(x) LE_n(x)\,dx = 0 \text{ if } Q(x) \text{ is of degree n or less} \tag{124}$$

Legendre polynomials are also given recursively

$$(n+1)LE_{n+1}(x) = (2n+1)xLE_n(x) - nLE_{n-1}(x) \tag{125}$$
$$LE_0 = 1 \tag{126}$$
$$LE_1 = x \tag{127}$$

When using Legendre polynomials to calculate a Gaussian quadrature you first need to find the n-degree Legendre polynomial $LE_n(x)$. The roots of $LE_n(x)$ are the x_i you pick on the interval $[-1, 1]$. After you have found n points x_i you find the coefficients by assuming test functions, e.g $f(x) = 1$, or by integrating over a Lagrange polynomial $L_i(x)$

$$w_i = \int_{-1}^{1} L_i(x)\, dx \tag{128}$$

When all x_i and w_i have been found for the interval $[-1, 1]$ you convert these to an arbitrary interval $[a, b]$ by the following formulas

$$x_{[a,b]} = \frac{b-a}{2} x_{[-1,1]} + \frac{b+a}{2} \tag{129}$$
$$w_{[a,b]} = \frac{b-a}{2} w_{[-1,1]} \tag{130}$$

After this you have all your w_i and x_i, and the next step is to approximate $f(x)$ on your given interval $[a, b]$.

Hence, a Gaussian quadrature of a function is the linear combination of n function evaluations at the Legendre roots and weights shifted to $[a, b]$.

Different from Newton-Cotes, Gaussian quadratures have a **ADA of** $2n - 1$ which is a big improvement.

Proof of ADA Gaussian Quadratures:

Let $P(x)$ be of degree $2n - 1$ and let us compute the Gaussian quadrature of it. Using long division we can express $P_{2n-1}(x)$ we get

$$P_{2n-1}(x) = Q_{n-1}(x)LE_n(x) + R_{n-1} \implies \tag{131}$$
$$\int_{-1}^{1} P_{2n-1}(x)\, dx = \int_{-1}^{1} Q_{n-1}(x)LE_n(x)\, dx + \int_{-1}^{1} R_{n-1}(x)\, dx \tag{132}$$

As we know that the rest term $R(x)$ can be calculated exactly with for example Newton-Cotes. Still, for the answer to be exact, i.e ADA $= 2n - 1$, we need the term

$$\int_{-1}^{1} Q_{n-1}(x)LE_n(x)\, dx \tag{133}$$

to be $= 0$. If $LE_n(x)$ is **Legendre Polynomials** this is true and thus ADA $= 2n - 1$.

7 Differential Equations (L22)

A differential equation is an equation involving derivatives. Here the goal is commonly to find the function $f(x)$ describing a phenomenon in nature, instead of a numerical answer. If the equation contains derivatives of only **one** variable they are ordinary (ODE). If the equation contains derivatives of **more than one** variable they are partial (PDE).

7.1 Ordinary Differential Equations (ODE)

Commonly we have ODEs with **Initial Value Problem, (IVP)**, for example

$$y' = f(t, y) \tag{134}$$
$$y(0) = y_0 \tag{135}$$

This system may be solvable analytically, but in some cases it must be numerically approximated. ODEs of higher order can be reduced to these simple IVP systems. For example a third degree system

$$y''' - 3y' + 4y = 0 \tag{136}$$
$$y(0) = 1 \tag{137}$$
$$y'(0) = 0 \tag{138}$$
$$y''(0) = 1 \tag{139}$$

can be reduced by firstly introducing variables

$$u_1 = y \implies u_1' = u_2 \tag{140}$$
$$u_2 = y' \implies u_2' = u_3 \tag{141}$$
$$u_3 = y'' \implies u_3' = \text{ solve } y''' \text{ in initial equation} \tag{142}$$

This becomes a linear system $Au = u'$. The initial conditions for u_i is given from original system.

7.2 Euler's Method

This method produces a numerical approximation of the IVP by iterating the **Explicit** or **Implicit** variation. The explicit tends to be unstable, i.e wanting to blow up, while the implicit tends to be stable. This stability analysis is based on the **Linear Test Equation**

$$y' = \lambda y \tag{143}$$

Here the value of λ and the step size h play a crucial roll in stability. This is most easily seen by pictures. Depending on the $f(t, y)$ we should choose one or the other method. If $f(t, y)$ wants to blow up, then explicit generally is the best choice. If $f(t, y)$ wants to be stable, then implicit generally is the best choice. However, you can do the stability analysis for $f(t, y)$ or create a vector field to be sure and not guess.

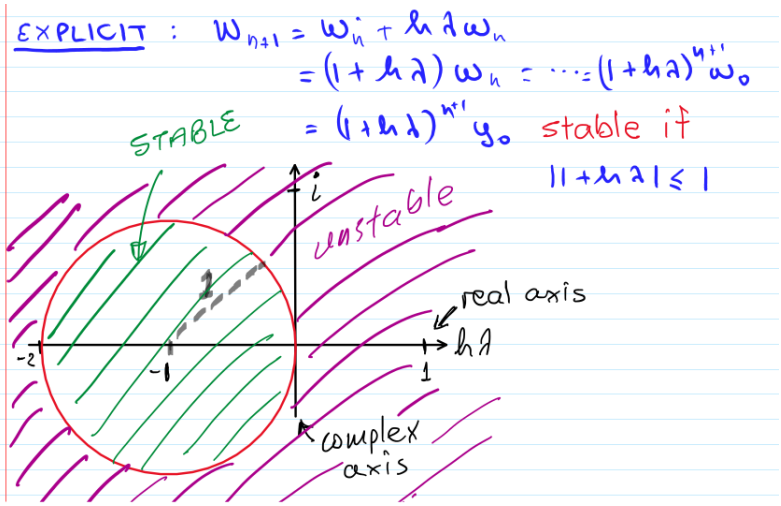
Explicit

$$w_{i+1} = w_i + hf(t_i, w_i) \tag{144}$$

where

$$i = 0, 1, 2, \ldots \tag{145}$$
$$w_0 = y_0 \tag{146}$$
$$h = t_{i+1} - t_i \tag{147}$$

The iteration starts at t_0. Stability is given by:



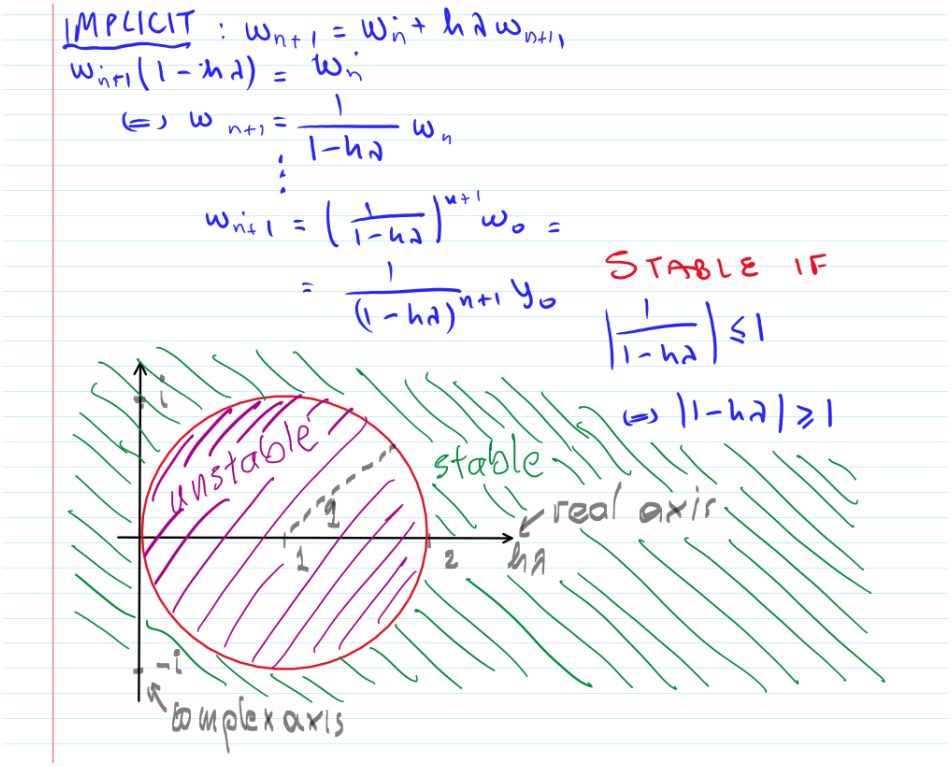
Convergence rate is linear.

Implicit

$$w_{i+1} = w_i + hf(t_{i+1}, w_{i+1}) \tag{148}$$

The difference here is $f(t_{i+1}, w_{i+1})$. Otherwise it is the same. To iterate the system you first need to factor out all w_{i+1} on one side.

Stability is given by:



Convergence rate is linear.

The implicit method is different from explicit in the way that it creates a equation system to solve, instead of just needing to plug in the initial values as in explicit to generate next t_i. This is seen most easily in an example of a non-linear two dimensional system:

$$u_1'(t,y) = sin(u_2(t)) \tag{149}$$
$$u_2'(t,y) = -u_1(t) + t^2 \tag{150}$$

A variable change $(u,v) = (u_1, u_2)$ gives us

$$u' = sin(v) \tag{151}$$
$$v' = -u + t^2 \tag{152}$$

and the formula for implicit Euler gives the following general equations to solve

$$u_{i+1} = u_i + h \cdot \sin(v_{i+1}) \tag{153}$$
$$v_{i+1} = v_i - h \cdot u_{i+1} + h \cdot t_{i+1}^2 \tag{154}$$

Assume now that $h = 1$, $t_0 = 0$ and $(u_0, v_0) = (\pi/2, 0)$. This gives us the following system

$$u_1 = u_0 + h \cdot \sin(v_1) = \pi/2 + \sin(v_1) \tag{155}$$
$$v_1 = v_i - h \cdot u_1 + h \cdot t_1^2 = -u_1 + 1 \tag{156}$$

Because we have the term $\sin(v_{i+1})$ the system is difficult to solve - solution need to be approximated. This is done by rewriting the problem so we can use multi dimension Newton.

$$u_1 - \pi/2 - \sin(v_1) = 0 \tag{157}$$
$$v_1 + u_1 - 1 = 0 \tag{158}$$

The iteration of Newton will then give an approximation of (u_1, v_1). This can be repeated to next (u_2, v_2) and so on.

Theory

Consistent (Method)

How well a numerical method agrees with ODE - studies truncation error.

Convergent (Solution)

A numerical solution is said to be convergent if the numerical solution approaches the exact solution as step size $h \to 0$. All methods presented are convergent.

Stable (Method and Solution)

How well can the method handle small rounding and truncation errors. Can such small errors swamp the solution and make it blow up?

---

8 Optional extra material

8.1 Horner's Method

The most effective way in terms of NOO to evaluate a polynomial $P_n(x)$ of degree n is the following:

$$P_n(x) = ((...(a_n x + a_{n-1})x + a_{n-2})x + ... + a_2)x + a_1)x + a_0 \tag{159}$$
