Chapter 1: Floating Point Numbers

topics:

- floating point numbers and their representation
- rounding errors and errors of arithmetic operations and simple functions
- error analysis of expressions

practice:

- representation of special numbers like 1/10, 17, 1/3, 0.25 etc
- understand what the rounding error is, examples above, what is a floating point number
- when are operations exact, how large are the errors in general
- analyse error in expressions
- what is a relative error, what is an absolute error
- how does error in the data affect the result
- what is backward error analysis, give an example
- what is the condition number of a function and a matrix
- what is the condition of solving a linear system of equations

number representation

- B is base, mostly $B = 10$ (decimal system for human computers) or $B = 2$ (binary system for digital computers)
- how can you write any integer as a polynomial in B , how to compute the coefficients, what are their properties
- a floating point number is a rational number, what is the numerator and what is the denominator

floating point numbers

- representation $x = \pm 0.d_1 d_2 \cdots d_t \cdot B^e$ where digits $d_k \in \{0, \ldots, B-1\}$ and integer exponent e (variable)
- normalisation $d_1 \neq 0$
- most common: IEEE 754 double precision with $t = 53$
- advantages:
  - constant relative error
- disadvantage:
  - more complicated to implement than the earlier fixed point numbers

rounding function ϕ

- best approximation of any real number in number system: $|\phi(x) - x| \leq |y - x|$ for all elements y of the number system
- make ϕ unique in some way, common choice: even least significant digit

properties of rounding function for floating point system

- reduction to integer rounding (ϕ_0):

$$\phi(x) = \phi_0(B^{t-e}x)/B^{t-e}$$

- error bound $|\phi(x) - x| \leq 0.5B^{-t+e}$
- relative error bound

$$\frac{|\phi(x) - x|}{|x|} \leq 0.5B^{-t+1}$$

- what is the ϵ where $|\phi(x) - x| \leq \epsilon|x|$ and $\phi(x)$ is a normalised floating point number

arithmetic and simple functions

- implemented to give best possible results
- result of computer addition is the rounded value of the exact sum: $x + y$ is then computed to $\phi(x + y)$
- same for simple functions
- why is this a good choice?

properties of floating point arithmetic

- most major laws of arithmetic do not hold – except for commutative law

error analysis

- steps:
    1. parse expression and represent as a sequence of simple steps containing either one arithmetic operation or the evaluation of one simple function (like $\cos$, $\exp$ etc)
    2. for each simple step there should be one rounding error, bound according to number system used
    3. do linear propagation of previous errors in the simple step
- this gives
    1. after parsing, we get sequence of statements which are evaluated in order

$$x_k = f_k(x)$$

    2. rounding of input data and statements
    3. this gives statements for the error

Chapter 2: Equations

Some questions to consider

- What is the mathematical problem to be solved?
- Do we know if the solution exists, if there is only one?
- What is the method?
- How accurate are the computed results?
- How long does it take? (arithmetic operations, function evaluations)
- Is the method optimal? What would an optimal method look like?
- What maths are we using to get error bounds?
- For which problems does the method not work?
- What are direct and iterative solvers, which are better?

Gaussian elimination

topics:

- LU factorisation, elementary matrices and elimination
- number of operations, main steps of Gaussian elimination
- partial pivoting

practice:

- compute LU factorisation of small matrices (2 by 2 etc)
- check examples when factorisation exists, when it does not exist
- understand the factorisation related to partial pivoting, do again simple examples

problem

- find vector x such that $Ax = b$ for given matrix A and vector b
- assume A is invertible (non-singular), i.e. exists matrix B such that $BA = I$

solution method

- direct solver: find exact solution x in a finite number of arithmetic steps
- Gaussian elimination is a direct solver which generates LU factorisation $A = LU$ during the elimination
  - L is lower triangular and defined by elementary matrices E_k
  - U is upper triangular

elementary matrix E_k

- for any given vector a with components a_j and integer k let m be the array with components $m_1 = \cdots m_k = 0$ and $m_j = a_j/a_k$ for $j > k$
- m_j defines an elementary matrix by $E_k = I - me_k^T$
- matrix E_k annihilates components $a_{k+1}, \ldots, a_n$ when multiplied with a :

$$E_k a = (a_1, \ldots, a_k, 0, \ldots, 0)^T$$

- inverse of E_k :

$$E_k^{-1} = I + me_k^T$$

Gaussian elimination is a method to compute the LU factorisation

- choose $E_1, \ldots, E_{n-1}$ such that
  - $E_k = I - m_k e_k^T$
  - E_1 anihilates all elements after the first one of the first column of A and more generally
  - E_k anihilates all elements after the k -th one of the k -th column of $E_{k-1} \cdots E_1 A$
- we then get a triangular matrix U with

$$E_{n-1} \cdots E_1 \, A = U$$

- with

$$L := E_1^{-1} \cdots E_{n-1}^{-1} = I + \sum_{k=1}^{n-1} m_k e_k^T$$

one gets the LU factorisation

$$A = LU$$

- the LU factorisation does not always exist and the Gaussian algorithm will then break down as one of the diagonal elements required for the computation of m_k may be zero
- if the factorisation exists, it can be used for the solution of $Ax = b$
- the factorisation requires $O(n^3)$ floating point operations

solving $Ax = b$ with $A = LU$

use two steps to solve the system of equations:

- first solve

  $$Ly = b$$

  for a vector y by forward elimination
- the compute x from

  $$Ux = y$$

  by back substitution

as the two matrices L and U are triangular only $O(n^2)$ operations are required to solve them

Gaussian elimination with partial pivoting

- not every matrix has LU factorisation, Gaussian elimination may break down
- even if there is no break-down, factors L and U can be inaccurate
- LU factorisation exists for symmetric positive definite A

- cure for general case: (partial) pivoting
- swap rows during elimination to get pivot with largest absolute value
- can show factorisation

$$PA = LU$$

  where all the (nonzero) elements of L satisfy $|l_{ij}| \leq 1$
- in contrast to the Gaussian elimination this factorisation always exists
- even when the ordinary LU factorisation exists, partial pivoting is still preferable as it is more stable

Fourier transforms

topics:

- Fourier matrix, inverse, convolution theorem
- Factorisation of Fourier matrix F_{2m} and FFT
- fast matrix vector multiplication for circulant matrices
- the 2D case, deconvolution

practice:

- consider examples of Fourier matrix for small n (2,3,4)
- compute the butterfly matrix and factorisation for the case of $n = 4$, maybe 6

problem and notation

- evaluate the discrete Fourier transform (DFT)

$$\hat{x}_k = \sum_{j=0}^{N-1} \omega_n^{jk}\, x_j$$

  for $k = 0, \ldots, N-1$ and $\omega_N = \exp(-2\pi i/N)$ (note $\omega_N^N = 1$)
- DFT is a matrix vector product

$$\hat{x} = \mathrm{DFT}(x) = F_N x$$

- DFT matrix

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \cdots & \omega_N^{N-1} \\ \vdots & \vdots & & \vdots \\ 1 & \omega_N^{N-1} & \cdots & \omega_N^{(N-1)^2} \end{bmatrix}$$

properties

- inverse $\mathrm{IDFT}(x) = F_N^{-1} x = \frac{1}{N} F_N^* x$
- periodicity $\hat{x}_{k+N} = \hat{x}_k$
- linearity $\mathrm{DFT}(\alpha x + \beta y) = \alpha\,\mathrm{DFT}(x) + \beta\,\mathrm{DFT}(y)$
- shifting $\mathrm{DFT}(\{x_{n-m}\})_k = \omega_N^{-km} \hat{x}_k$
- Parseval

$$\sum_{n=0}^{N-1} x_n y_n^* = \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}_k \hat{y}_k^* \qquad \text{and thus} \qquad \sum_{n=0}^{N-1} |x_n|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |\hat{x}_k|^2$$

- convolution theorem $\mathrm{DFT}(x * y) = \mathrm{DFT}(x) \cdot \mathrm{DFT}(y)$ where the convolution is defined by $(x * y)_k = \sum_{j=0}^{N-1} x_j\, y_{(k-j \mod N)}$ and the Hadamard product by $(\hat{x} \cdot \hat{y})_k = \hat{x}_k \hat{y}_k$

FFT

splitting

- select even and odd components $y_n = x_{2n}$ and $z_n = x_{2n+1}$
- rewrite Fourier transform of x in terms of (shorter) Fourier transforms of y and z

$$\begin{aligned} \hat{x}_k &= \hat{y}_k + \omega_N^k \hat{z}_k \\ \hat{x}_{k+N/2} &= \hat{y}_k - \omega_N^k \hat{z}_k \end{aligned}$$

- use this approach recursively for y and z gives fast Fourier transform (FFT)

- complexity $O(N \log_2 N)$ compared to $O(N^2)$ of matrix vector product

2D DFT

$$\hat{X}_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{mn}\, \omega_M^{mk}\, \omega_N^{nl}$$

- notation and matrix products

$$\hat{X} = \mathrm{DFT2}(X) = F_M X F_N = \mathrm{DFT}(\mathrm{DFT}(X)^T)^T$$

denoising and deblurring

- 1D denoising (similar for 2D) of data $b_k = x_k + \epsilon_k$ for some noise ϵ_k

$$\hat{y}_k = \frac{\hat{b}_k}{1 + 4\lambda \sin^2(\pi k / N)}$$

  where $\lambda > 0$ regularisation or smoothing parameter
- 2D deblurring of data $Y = B * X$ (2D convolution)

$$\hat{X}_{kl} = \frac{\hat{Y}_{kl}}{\hat{B}_{kl}}$$

Iterative solvers for linear systems

topics:

- Splitting, Jacobi, GS, SOR
- Fixed point operations, matrix norm, spectral radius, convergence theory
- Iterative refinement / improvement
- Steepest descent (gradient descent) / CG

practice:

- consider examples with small matrices by hand, do a couple of iterates
- look at a lot of special examples of matrices (with special structure …)
- compute the errors and follow how they drop for various methods
- compute eigenvalues for 2x2 matrices and appy to convergence theory and matrix norms
- compare various matrix norms and how they explain convergence

Problem

- find x^* which solves $Ax = b$
- in contrast to direct solvers, iterative solvers compute successive approximations $x^{(k)}$ to x^*
- stationary iterative solver: generate $x^{(k)}$ with recursion of the form

$$x^{(k+1)} = Ex^{(k)} + g$$

simple iterative solution idea

- find matrix M with
  - $M - A$ small
  - $Mx = b$ is solved quickly for any b
- rewrite equation equation in fixed point form

$$Mx = b - (A - M)x$$

- iteration

$$x^{(k+1)} = M^{-1}b - M^{-1}(A - M)x^{(k)}$$

- interpretations:
  - error correction form $x^{(k+1)} = x^{(k)} - M^{-1}(Ax^{(k)} - b)$ (add approximate error)
  - fix point iteration $x^{(k+1)} = Ex^{(k)} + g$ where $E = I - M^{-1}A$ (error matrix) and $g = M^{-1}b$

matrix splitting – how to get M

$$A = L + D + U$$

where L, D, U are lower triangular, diagonal and upper triangular parts (NOT THE SAME as in Gaussian elimination!)

- Jacobi ($M = D$ and $E_J = -D^{-1}(L + U)$)

$$x^{(k+1)} = -D^{-1}(L + U)x^{(k)} + D^{-1}b$$

- Gauss-Seidel ($M = L + D$ and $E_{GS} = -(L + D)^{-1}U$)

$$x^{(k+1)} = -(L + D)^{-1}Ux^{(k)} + (L + D)^{-1}b$$

convergence

- error $e^{(k)} = x^{(k)} - x^*$
- as $x^{(k+1)} = Ex^{(k)} + g$ one gets

$$e^{(k+1)} = Ee^{(k)}$$

- main tool: spectral radius

$$\rho(E) = \max_i |\lambda_i(E)| \leq \|E\|$$

- error bound

$$\|e^{(k)}\| \leq \rho(E)^k \|e^{(0)}\|$$

convergence of Jacobi and Gauss-Seidel methods

- important definition: A diagonally dominant if at least one of the two following conditions hold:

$$\sum_{j \neq i} |\quad| < |\quad| \quad \text{or}$$
$$\sum_{i \neq j} a_{ij} |a_{ij}| < a_{ii} |a_{jj}|$$

- if A is diagonally dominant one has convergence of Jacobi and Gauss-Seidel methods as

$$\rho(E_J) < 1 \quad \text{and} \quad \rho(E_{GS}) < 1$$

optimisation based approach

mathematics

- for symmetric positive definite matrices A
- target function

$$u(y) = \frac{1}{2} y \cdot Ay - b \cdot y$$

  where $x \cdot y$ denotes scalar product
- solution of linear system is minimum of quadratic function f :

$$Ax = b \quad \Leftrightarrow \quad x = \text{argmax}_y \, u(y)$$

- we will use “energy” scalar product and norm defined by

$$\langle x, y \rangle = x \cdot Ay, \quad \|x\|_A = \sqrt{\langle x, x \rangle}$$

general optimisation method with optimal step size

- iteration defined by search direction d^k and step size α_k as

$$x^{k+1} = x^k + \alpha_k d^k$$

- choose step size such that $u(x^k + \alpha_k d^k)$ is minimial gives

$$\alpha_k = \frac{r^k \cdot d^k}{\|d^k\|_A^2}$$

two different search directions:

- steepest descent along gradient:

$$d^k = r^k := b - Ax^k$$

- conjugate gradient with an orthogonalisation to avoid going down same direction again:

$$d^k = r^k - \beta_{k-1} d^{k-1}$$

where

$$\beta_{k-1} = \frac{\langle r^k, d^{k-1} \rangle}{\|d^{k-1}\|_A^2}$$

Nonlinear Equations

topics:

- bisection, existence of solution, Bolzano theorem, convergence rate
- fixed point iteration, convergence, uniqueness
- Newton's and secant methods, convergence rates

practice:

- what happens for f being a linear function, quadratic polynomial
- do a couple of steps for a function with known zero – eg take any function $g(x)$ then $f(x) = g(x) - g(x^*)$ is certain to have zero at x^*
- search for problems where the methods will not work and try it out

problem

- solve equation $f(x) = 0$ for some real function f

existence of a solution (Bolzano's theorem)

- if $f(x)$ is continuous real valued function on interval $[a, b]$ and if $f(a)f(b) \leq 0$
- then $f(x) = 0$ has a solution x^*

numerical methods

- define a sequence x_n with $x_{n+1} = F_n(x_0, \ldots, x_n)$ such that $x_n \to x^*$ for $n \to \infty$

bisection

- initiate with $x_0 < x_1$ such that $f(x_0)f(x_1) < 0$
- for $n = 1, 2, \ldots$
  - let
    $a_n = \text{argmax}\{x_k \mid \text{sign } f(x_k) = \text{sign } f(x_0),\ k = 0, \ldots, n\}$
  - let
    $b_n = \text{argmin}\{x_k \mid \text{sign } f(x_k) = \text{sign } f(x_1),\ k = 0, \ldots, n\}$
  - let $x_{n+1} = (a_n + b_n)/2$
  - stop if $(b_n - a_n) \leq \epsilon$

remark: in practice you can update the values of a_n and b_n

accuracy and convergence

- error $e_n = x_n - x^*$
- bound on the error

$$|\quad| \leq \frac{b_k - a_k}{2} \leq \qquad (\quad - \quad)$$

- use small error bound as stopping criterion $\qquad 2^{k+1} \qquad x_1 \qquad x_0$

Fixed point method

Definition

- recursion $x_{n+1} = F(x_n)$
- necessary condition: $x^* = F(x^*)$

Construction

- correct with error $x^* = x_n - e_n$
- for good x_n : $e_n \approx \alpha f(x_n)$ for some α
- simple algorithm $F(x) = x - \alpha f(x)$

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

Convergence

- error $e_n = x_n - x^*$:

$$|e_n| \leq \lambda^n |e_0|$$

- here F is contractive: $|F(x) - F(y)| \leq \lambda |x - y|$ for some $0 < \lambda < 1$

Newton's method

constructs sequence x_k using approximate error $\hat{e}_k$

- $x_{k+1} = x_k - \hat{e}_k$
- $\hat{e}_k = f(x_k)/f'(x_k)$

- requires that $f \in C^1$ where a, b from existence theorem
- requires that $x_0 \in [a, b]$ is chosen sufficiently close to the desired x^*

motivation: Talyor's remainder theorem gives

- $x^* = x_k - e_k$
- $e_k = f(x_k)/f'(\xi_k)$
- for some ξ_k with $|\xi_k - x_k| \leq |e_k|$

accuracy and local convergence

- for small errors e_k the value $\hat{e}_k$ approximates e_k
    - use $\hat{e}_k$ as error indicator
- Taylor's theorem gives for some (not the same as above) ξ_k close to x_k

$$e_{k+1} = -\frac{f''(\xi_k)}{2f'(x_k)}\, e_k^2$$

- for sufficiently small e_k Newton's method reduces the error (hence condition on x_0)
- Newton's method is locally second order convergent

- this analysis requires $f \in C^2[a, b]$

secant method

constructs sequence x_k using approximate error $\hat{e}_k$

- $x_{k+1} = x_k - \hat{e}_k$
- $\hat{e}_k = f(x_k) * (x_{k-1} - x_k)/(f(x_{k-1}) - f(x_k))$
- requires that x_0 is chosen sufficiently close to the desired x^*

motivation: simple algebra gives

- $x^* = x_k - e_k$
- $e_k = f(x_k) * (x^* - x_k)/(f(x^*) - f(x_k))$

similar to Newton's method but uses difference quotient instead of derivative

accuracy and local convergence

- for small errors e_k the value $\hat{e}_k$ approximates e_k
    - use $\hat{e}_k$ as error indicator
- Taylor's theorem gives for some (not the same as above) ξ_k close to x_k

$$e_{k+1} = -\frac{f''(\xi_k)}{2f'(x_k)}\, e_k e_{k-1}$$

- for sufficiently small e_k secant method reduces the error (hence condition on x_0)
- can show that approximately $|e_{k+1}| = C|e_k|^\phi$ for $\phi = (1+\sqrt{5})/2$
- secant method is order ϕ convergent (ϕ between 1 and 2, about 1.62)

- this analysis requires $f \in C^2[a,b]$

Chapter 3: Approximation

Interpolation

topics:

- Weierstrass and Taylor theorems, Rolle
- existence and uniqueness
- Lagrange (cardinal polynomials) and Newton forms, monomial basis
- interpolation error
- Chebyshev points, Runge example

practice:

- compute examples with 2 or three points, consider any scenario (data points) – what works what does not
- consult your calculus text if necessary!

problem

given values y_k of some (unknown) function $u(x)$ at some given points x_k for $k = 0, \ldots, n$ find an approximation of $u(x)$

approach: polynomial collocation

find polynomial $p(x)$ of degree n such that $p(x_k) = y_k$ for all k

- the equations $p(x_k) = y_k$ are the interpolation equations and are the collocation equations for the interpolation problem

existence and uniquenes

use numerical algorithms below to show existence and fundamental theorem of algebra to show uniquenes

numerical methods

- we consider three different methods which use three different sets of basis functions
- as polynomial interpolation function is unique we obtain three different representations of the same polynomial of degree n

interpolation using the monomial basis

- representation of polynomial:

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

- basis functions: monomials independent of the interpolation points x_k
- interpolation equations

$$x_j$$

$$Xa = y$$

where vectors a and y have components a_k and y_k respectively and X is the Vandermonde matrix with elements

$$X_{kj} = x_k^j, \quad k, j = 0, \ldots, n$$

unique solution of interpolation equations

- one can prove that if all the interpolation points x_k are distinct ($x_k \neq x_j$ for $k \neq j$) then the Vandermonde matrix X is regular or invertible
- thus the interpolation problem has exactly one solution – which proves existence (and also uniquenes)

fast algorithm for the evaluation of the monomial base representation

Horner's method:

- start with $b_0 = a_n$
- iterate $b_{k+1} = x * b_k + a_{n-k+1}$

then $p(x) = b_n$

- prove by applying distributive law (and also associative and commutative laws) of real arithmetic
- thus evaluation of polynomial in this basis requires n multiplications and $n + 1$ additions

Lagrangian interpolation

cardinal or Lagrangian polynomials

$$l_k(x) = \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

- $l_k(x)$ is a polynomial of degree n
- $l_k(x_k) = 1$ and $l_k(x_j) = 0$ for $j \neq k$

Lagrangian interpolation function

$$p(x) = y_0 l_0(x) + \cdots + y_n l_n(x)$$

- this proves again existence of the interpolation polynomial if x_k distinct
- no solution of a linear system of equations is required to determine the coefficients which are just the values at the interpolation points y_k
- evaluation more costly than with monomials at single points

Newton's interpolation

- determines recursively all polynomials $p_k(x)$ of degree $k = 0, \ldots, n$ such that

$$p_k(x_j) = y_j, \quad j = 0, \ldots, k$$

Newton's basis functions

$$n_k(x) = (x - x_0)(x - x_1) \cdots (x - x_{k-1})$$

recursive formulation of Newton's interpolation polynomials p_k :

-
- $p_{k+1}(x) = p_k(x) + c_{k+1} n_k(x)$

p_0 c_0

where coefficients c_k determined recursively from interpolation conditions

$$p_k(x_k) = y_k, \quad k = 0, \ldots, n$$

- unique solution if all x_k distinct

Newtonian interpolation function

$$p(x) = c_0 + c_1(x - x_0) + \cdots + c_m(x - x_0) \cdots (x - x_{m-1})$$

interpolation error

let f be (possibly unknown) function and $y_k = u(x)$ and $p(x)$ interpolation polynomial with $p(x_k) = y_k, k = 0, \ldots, n$

- interpolation error function $e(x) = p(x) - u(x)$
- interpolation conditions give $e(x_k) = 0$ for $k = 0, \ldots, n$

- Taylor's formula gives

$$e(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} w(x)$$

where

$$w(x) = (x - x_0) \cdots (x - x_n)$$

- here x is arbitrary and ξ_x typically unknown are in an intervall which contains all points x_k
- note that $f^{(n+1)}(x) = e^{(n+1)}(x)$ as $p^{(n+1)}(x) = 0$

- the theory requires $f \in C^{(n+1)}$

properties of $w(x)$

- for uniformly spaced x_k (e.g. $x_k = k/n$) the function $w(x)$ is large at close to the extreme x_k
- this can lead to large interpolation errors even for smooth $u(x)$ as the Runge example $u(x) = 1/(1 + 25x^2)$ demonstrates
- a better choice are the Chebyshev points for an intervall $[a, b]$

$$x_k = \frac{a+b}{2} + \frac{b-a}{2} cos(\zeta_k)$$

  where $\zeta_k = \pi * (2 * k + 1)/(2 * n + 2)$
- in this case one gets for $[a, b] = [-1, 1]$

$$w(x) = T_n(x) = \cos(n \arccos(x))$$

  which is of uniform size over the intervall

Quadrature

topics

- trapezoidal rule, Newton-Cotes
- Gaussian quadrature
- Romberg

practice

- simple examples with up to around 4 quadrature points
- what happens with various (specific like x^3 etc) polynomials for the simple examples

problem

determine $I = \int_a^b u(x)\, dx$ for some real function f

transformation to standard interval $[-1, 1]$

$$I = \int_a^b u(x)\,dx = \frac{b-a}{2} \int_{-1}^1 u((b+a)/2 + x(b-a)/2)\,dx$$

mathematics: Riemann sums

$$S_n = \sum_{i=0}^{n-1} hu(\overline{x}_i)$$

where $\overline{x}_i \in [x_i, x_{i+1}]$ for $i = 0, \ldots, n-1$ and $x_i = ih$ for $h = (b-a)/n$

- if $f \in C[a, b]$ then S_n converges to I if $n \to \infty$
- Riemannian sums are actually a numerical method but will not be further discussed here as such

the main method: trapezoidal rule

$$T(f, h) = \sum_{i=0}^{n-1} h_i \frac{u(x_i) + u(x_{i+1})}{2}$$

- for general x_i and $h_i = x_{i+1} - x_i$ but here we choose $h_i = h$ constant
- like the Riemann sum, the trapezoidal rule is a composite rule where the integral is decomposed in integrals over small subintervals which are then approximated
- the approximation error originates from the interpolation error with a linear function on each of the subintervals

error of the trapezoidal rule

- error $e_h := I - T(f,h)$
- use Taylor remainder formula to get

$$e_h = -\frac{1}{12}(b-a)\, f''(\xi)\, h^2$$

- Trapezoidal rule is second order accurate in h
- for $f \in C^{2r+2}[a,b]$ one gets the Euler-MacLaurin expansion

$$e_h = \sum_{k=1}^{r} C_{2k} h^{2k} + O(h^{2r+2})$$

  where the last term contains higher order derivatives of f

- coefficients of the Euler-MacLaurin expansion:

$$C_2 = (f'(b) - f'(a))/12, \quad C_{2m} = c_{2m}[f^{2m-1}(b) - f^{2m-1}(a)]$$

  for explicit formulas see slides of the course

Romberg = Richardson extrapolation for the trapezoidal rule

- use extrapolation to cancel terms of Euler-MacLaurin expansion
- in contrast to differentiation we start with largest h and reduce by dividing by 2
- Romberg scheme: start with $R_{n,0} = T(f, (b-a)/2^n)$ for $n = 0, 1, 2, \ldots$ and determine for $m = 1, 2, \ldots$

$$R_{n,m} = \frac{4^m R_{n,m-1} - R_{n-1,m-1}}{4^m - 1}, \quad m \le n$$

- Euler-MacLaurin formula provides error

$$e_{n,m} := I - R_{n,m} = O(h_n^{2m})$$

where for sufficiently smooth f

$= (b-a)/$

- rounding error is not an issue as integration well-posed

h_n 2_n

Gaussian quadrature

- many quadrature rules are of the form

$$Q_n = \sum_{i=0}^{n} A_i \, u(x_i)$$

- the weights A_i are chosen for non-composite rules such that the rules are exact for polynomials of degree n – this can be done by interpolation and in practice by the method of undetermined coefficients
- methods include the (simple) trapezoidal rule, Simpson's method, Newton-Cotes rules and Gaussian rules

- Gaussian quadrature is special as both the A_i and x_i are chosen to guarantee that the rules are exact for polynomials of degree $2n+1$

the midpoint rule – Gaussian quadrature with one point

$$Q_0(f) = (b-a)\, f\left(\frac{a+b}{2}\right)$$

- applied as the base of a composite rule this leads to a Riemann sum with the same accuracy $O(h^2)$ as the trapezoidal rule – the other Riemann sume are typically only $O(h)$

general Gaussian quadrature rules $Q_n(f)$

- the method of undetermined coefficients is not feasible for large n as one needs to solve large nonlinear systems of equations
- the rule $Q_n(f)$ uses x_i to be the $n+1$ roots of the Legendre polynomial $q_{n+1}(x)$ of degree $n+1$

- the weights are then chosen as always to guarantee that the rule is exact for polynomials of degree up to n
- the rules of general intervals $[a, b]$ are obtained by transformation to the interval $[-1, 1]$ and applying the Gaussian rule on that interval

Legendre polynomials on $[-1, 1]$

- $q_n(x) = x^n + \alpha_{n-1} x^{n-1} + \cdots \alpha_0$ where α_k are chosen such that the q_n are orthogonal, i.e.,

$$\int_{-1}^{1} q_n(x) q_m(x)\, dx = 0, \quad n \neq m$$

- q_n has $n - 1$ distinct roots
- if x_i of Q_n are the zeros of q_{n+1} then $Q_n(p(x) q_{n+1}(x)) = 0$ for any polynomial p , and in particular any polynomial of degree less than $n + 1$
- for the polynomials of degree less than $n + 1$ one also has $\int_{-1}^{1} p(x) q_{n+1}(x)\, dx = 0$ by orthogonality
- as any polynomial or degree up to $2n + 1$ can be written as $p_1(x) + p_2(x) q_{n+1}(x)$ where $p_i(x)$ are polynomials of degree n or less one then sees that the Gaussian rule is exact for this choice

error of Gaussian quadrature

- error formula for Gaussian quadrature obtained from Taylor series expansion

$$e_h := I - \sum_{i=0}^{n} A_i\, u(x_i) = \frac{f^{2n+2}(\xi)}{(2n+2)!} \int_a^b w(x)\, dx$$

  for some $\xi \in [a, b]$ and

$$w(x) = \prod_{i=0}^{n} (x - x_i)^2$$

Finite differences

topics:

- just a few examples to illustrate approximating derivatives for backward differentiation formula in ODE solvers

practice:

- try this for some more examples and apply to ODE solvers and quadrature

problem

for given $u(x_0), \ldots, u(x_m)$ find approximation to $u'(z)$

- equidistant points $x_j = x + jh$

one finite difference quotient provides 3 approximations

$$d_h(x) = \frac{u(x+h) - u(x)}{h}$$

- gives three approximations for $z = x$, for $z = x + h$ and for $z = x + h/2$
  - forward difference $D_h^+(x) = d_h(x)$ approximates $u'(x)$
  - backward difference $D_h^-(x + h) = d_h(x)$ approximates $u'(x + h)$
  - central difference $D_h^c(x + h/2) = d_h(x)$ approximates $u'(x + h/2)$
- all these approximations are used

errors from the Taylor remainder theorem

- forward difference for $u \in C^2$

$$D_h^+ u(x) = u'(x) + \frac{h}{2} u''(\xi_+)$$

- backward difference for $u \in C^2$

$$D_h^- u(x+h) = u'(x+h) - \frac{h}{2} u''(\xi_-)$$

- central difference for $u \in C^3$

$$D_h^c u(x+h/2) = u'(x+h/2) + \frac{h^2}{24} u'''(\xi_c)$$

a central difference approximation for the second derivative

$$D^2 u(x,h) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}$$

- the Taylor remainder theorem with $k = 2$ gives for $f \in C^2$ and some $\xi \in [x-h, x+h]$:

$$D^2 u(x,h) = f''(\xi)$$

- thus the approximation converges to the second derivative for all $f \in C^2$

- in the case where $f \in C^{(4)}$ one gets an error formula again with some $\xi \in [x-h, x+h]$:

$$e_h(x) := f^{(2)}(x) - D^2 u(x,h) = -\frac{h^2}{12} f^{(4)}(\xi)$$

The method of undetermined coefficients

- problem: given $u(x+kh)$ for $k=0,\dots,m$ determine $u'(z)$
- method: approximation $Du(z) \approx u(z)$ with

$$Du(z) = \frac{1}{h}\sum_{k=0}^{m} \gamma_k u(x+kh)$$

- property: γ_k do not depend on x or h

method of undetermined coefficients

- compute γ_k such that for all polynomials $p(x)$ to degree m one has

$$p'(z) = \sum_{k=0}^{m} \gamma_k p(k)$$

- choose $p(x) = x^s$ for $s = 0,\dots,m$ gives system of $m+1$ linear equations

$$\sum_{k=0}^{m} \gamma_k k^s = k z^{k-1}$$

Chapter 4: Ordinary Differential Equations

topics:

- one-step methods, Runge-Kutta, convergence theorem
- A stability
- step-size control
- multi-step methods, method of unknown coefficients

practice:

- try to compute the coefficients for all sorts of known and unknown methods
- compute Lipschitz constants (relation to derivative!) for simple functions
- recognise simple shapes for region of A-stability (ellipses)

the problem

- find real vector valued function $x(t)$ such that

$$\frac{dx(t)}{dt} = f(x(t), t)$$

  for $t \in [a, b]$ and $x(a) = x_a$.

mathematics

- $x(t)$ exists and is unique if $f(y, t)$ is continuous in t and Lipschitz continuous in y :

$$\|f(y, t) - f(z, t)\| \leq L\|y - x\|$$

- reformulation as 2nd kind Volterra integral equation

$$x(t) = x_a + \int_a^t f(s, x(s))\, ds$$

numerical methods

we consider 2 classes of numerical techniques for the solution which determine approximations x_k for $x(t_k)$ for $a = t_0 < t_1 < \cdots < t_n = b$:

- one step methods

$$x_{k+1} = x_k + \varphi(x_k, t_k)$$

- linear multistep methods

$$\sum_{j=0}^{m} \alpha_j x_{k+1-j} = h \sum_{j=0}^{m} \beta_j f_{k+1-j}$$

where $t_k = a + kh$ and $f_k := f(x_k, t_k)$

- compared to one-step methods, multistep methods require less function evaluations as earlier ones are reused. However, they do require a start-up method initially and they also suffer under spurious solutions.

convergence theory for one-step methods

- convergence means that the $x_k \to x(t_k)$ for $h \to 0$
- one step methods are convergent if they are stable and consistent
- a one-step method is stable if $\varphi(y, t)$ is Lipschitz-continuous in y
- the local discretisation error at t of a one step method is

$$\eta(t, h) = \frac{x(t+h) - x(t)}{h} - \varphi(x(t), t)$$

  where $x(t)$ is the exact solution
- a one-step method is consistent if for $h \to 0$ one has

$$\sup_{t \in [a,b]} \eta(t, h) \to 0$$

- a one-step method has consistency order p if for $h \to 0$ one has

$$\eta(t, h) = O(h^p)$$

- Convergence theorem for one step method
  - if consistency order is $p \geq 1$ and $|\varphi(x, t) - \varphi(y, t)| \leq L|x - y|$ for all $0 \leq t \leq T$ and $0 \leq h \leq T - t$ and all x and y then

$$|x(t_n) - x_n| \leq Ch^p$$

    i.e. method has convergence order p

examples of one-step methods

- Euler's method (first order method)

$$x_{k+1} = x_k + hf(x_k, t_k)$$

- Heun's method (second order method)

$$x_{k+1} = x_k + \frac{h}{2}(f(x_k, t_k) + f(x_k + hf(x_k, t_k), t_{k+1}))$$

- midpoint method (second order method)

$$x_{k+1} = x_k + hf(x_k + hf(x_k, t_k)/2, t_k + h/2,)$$

- implicit Euler's method (first order)

$$x_{k+1} - hf(x_{k+1}, t_{k+1}) = x_k$$

  implicit methods are popular due to their higher stability but they require a nonlinear solver at every step to solve for x_{k+1}
- these simpler rules are derived using a quadrature rule for the integral in

$$x(t_{k+1}) = x(t_k) + \int_{t_k}^{t_{k+1}} f(s, x(s))\, ds$$

Runge-Kutta methods

a large and popular class of one-step methods of the form

$$x_{k+1} = x_k + h(b_1 k_1 + \cdots + b_s k_s)$$

where

$$k_j = f(x_k + h(a_{j,1} k_1 + \cdots + a_{j,j-1} k_{j-1}),\ t_k + c_j h)$$

- the constants are chosen such that $c_j = \sum_i a_{i,j}$ and $\sum_i b_i = 1$
- the constants are chosen such that one gets an optimal local discretisation error
- the most popular method is the 4th order one where

$$= \quad + \frac{h}{6}(\quad + 2 \quad + 2 \quad + \quad)$$

$$x_{k+1} \quad k_1 \quad x_k \qquad = k f(x_k, k_2) \qquad k_3 \qquad k_4$$

$$k_2 = f(x_k + hk_1/2,\, t_k + h/2)$$

$$k_3 = f(x_k + hk_2/2, t_k + h/2)$$

$$k_4 \qquad = f(x_k + hk_3, t_{k+1})$$

examples and derivation of linear multi-step methods

Adams-Bashforth methods

$$x_{k+1} = x_k + h \sum_{i=1}^{n} \beta_i f_{k+1-i}$$

- obtained from

$$x(t_{k+1}) = x(t_k) + \int_{t_k}^{t_{k+1}} f(s, x(s))\, ds$$

  by interpolating $f(s, x(s))$ with a polynomial which goes through (x_j, f_j) for $j = k + 1 - n, \ldots, k$
- example fourth order method

$$x_{k+1} = x_k + \frac{h}{24}(55 f_k - 59 f_{k-1} + 37 f_{k-2} - 9 f_{k-3})$$

Adams-Moulton methods

$$x_{k+1} = x_k + h \sum_{i=0}^{n} \beta_i f_{k+1-i}$$

- implicit method
- obtained from

$$x(\quad\quad) = x(\quad) + \int_{t_k}^{t_{k+1}} f(s, x(s))\, ds$$

by interpolating $f(s, x(s))$ with a polynomial which goes through (x_j, f_j) for $j = k+1-n, \ldots, k+1$

- example fourth order method

$$x_{k+1} = x_k + \frac{h}{24}(9f_{k+1} + 19f_k - 5f_{k-1} + f_{k-2})$$

backward differentiation formula

$$x_{k+1} = \sum_{i=1}^{m} \alpha_i x_{k+1-i} + h\beta_0 f_{k+1}$$

obtained by replacing the derivative in $dx/dt = f(x(t), t)$ by a finite difference approximation

derivation of general multistep methods by method of unknown coefficients

choosing the coefficients such that the equations are exact when $x_k = (kh)^j$ and $f_k = j(kh)^{j-1}$ (derivative of x) for $j = 0, \ldots, n$ for some chosen n

systems and higher order ODEs

- One and multistep methods can be used for systems of ODEs, see lecture notes and slides for details.
- Higher order ODEs are first reduced to systems of first order ODEs, see lecture notes and slides.

A-stability

- one would like to choose large step sizes for slowly varying solutions $x(t)$
- rounding errors might introduce more quickly varying but decaying components which are amplified in many numerical solvers unless very small stepsizes are chosen
- this does not happen for A-stable methods where decaying solutions lead to decaying approximations
- a decaying solution is one where $x(t) \to 0$ for $t \to \infty$

the model problem

$$\frac{dx}{dt} = \lambda x$$

for some complex λ

- solution: $x(t) = x_0 \exp(\lambda t)$ which is decaying (stable) for λ with real part $Re(\lambda) < 0$
- applying these results to more general linear ODEs using the eigenvalue decomposition
- applying these results to nonlinear ODEs using local linearisation

numerical solutions of the model problem

$$x_{k+1} = \rho(\lambda h)\, x_k$$

- decays to zero if $|\rho(\lambda h)| < 1$
- if this happens we call the method A-stable for this λ and h
- if method is A-stable for all h and λ with $Re(\lambda) < 0$ then the method is unconditionally A-stable

examples

- Euler: $\rho(\lambda h) = 1 + h\lambda$, is A-stable in the circle defined by $|1 + h\lambda| < 1$

- implicit Euler: $\rho(\lambda h) = \frac{1}{1-h\lambda}$ is unconditionally A-stable