Pg 1 Sc 1

H1 Numerical Limitations

1.1 Basic concepts

Absolute and relative error:

Absolute = $AF(a) = \text{approx.}(a) - \text{true}(a)$

Relative = $RF(a) = AF(a) / \text{true}(a)$

precision: the number of digits

Accuracy: the number of correct significant digits

Truncation error: Difference between true result and result given by exact arithmetic.

Rounding error: Difference between true result and result by rounded arithmetic

1.2 floating-point number system

floating point is characterized by:

- β : Base or radix
- p : precision
- $[L, U]$: exponent range

$$\Rightarrow \text{float} = \pm\left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \dots + \frac{d_{p-1}}{\beta^{p-1}}\right)\beta^E \quad \text{with: } 0 \leq d_i \leq \beta-1, \; L \leq E \leq U$$

normalized if: $d_0 = 1$

(Binary system has $\beta = 2$)

1.3 properties

float is finite and discrete.

smallest positive normalized number (underflow level):
β^L

largest (overflow): $\beta^{U+1}(1-\beta^{-p})$

Real number representable in given floating point system: Machine numbers.

⇒ otherwise round to nearest float: rounding error

accuracy of floating-point system is machine precision.

1.4 Good practice for computer arithmetic

Cancellation:

- avoid subtracting 2 almost identical numbers

Addition:

- avoid adding small and large numbers
- Perform a sequence of additions ordered from the smallest to the largest.

2. Introduction and notation

linear system: $Ax = b$

$$= \begin{cases} a_{11}x_1 + a_{12}x_2 \cdots = b_1 \\ a_{21}x_1 + a_{22}x_2 \cdots = b_2 \\ \vdots \\ a_{m1}x_1 + \cdots a_{mn}x_n = b_n \end{cases}$$

An $n \times n$ matrix is nonsingular if:
- A has inverse $A^{-1} \Rightarrow AA^{-1} = I$
- $\det(A) \neq 0$
- $\text{rank}(A) = n$
- $x \neq 0 \Rightarrow Ax \neq 0$

2.1 Solving linear systems

To solve we transform $Ax = b$ in another one whose solution is easy to compute:
We can multiply by a nonsingular matrix without effecting the solution.

Easy to calculate? $\Rightarrow$ triangular linear system.

A matrix L is Lower triangular if $l_{ij} = 0$ if $i < j$
$\Rightarrow$ solved by forward substitution:

$$Lx = b \Rightarrow x_1 = b_1/l_{11}, \; x_i = (b_i - \sum_{j=1}^{i-1} l_{ij}x_j)/l_{ii}$$

A matrix U is upper triangular if $u_{ij} = 0$ if $i > j$
$\Rightarrow$ solved by backward substitution:

$$Ux = b \Rightarrow x_n = b_n/u_{nn}, \; x_i = (b_i - \sum_{j=1}^{n} u_{ij}x_j)/u_{ii}$$

Elementary elimination matrices.
A matrix (nonsingular) that transforms a system into a triangular linear system.

Gaussian elimination = LU factorization / decomposition.
⇒ • Use elementary elimination matrices so that the system transforms into upper triangular.
• $M = M_{n-1} \cdots M_1 \Rightarrow MAx = Mb$ upper triangular:
$L = M^{-1}$ lower triangular.
⇒ $U = MA$, $L = M^{-1}$
⇒ $LUx = b$ solved by first: $Ly = b$
then: $Ux = y \Rightarrow \otimes$

Partial pivoting

Gauss-Jordan elimination
⇒ variation of gaussian elimination
⇒ eliminates entries below and above diagonal entries
⇒ 50% more computationally expensive than Gauss.-el.

2.3 special types of linear system

- symmetric $A = A^T$
- positive definite $x^TAx > 0$ for all $x \neq 0$
- Banded: $a_{ij} = 0$ for all $|i-j| > \beta$, β = bandwidth of A
- sparse: most entries are zero.

Symmetric positive definite system:
Cholesky factorization.

if A is symmetric and positive definite: $U = L^T \Rightarrow A = LL^T$

⇒ Cholesky factorization: compute LL^T ⇒ easily computable

Computational complexity
- LU factorization takes about $\frac{n^3}{3}$ floating point operations
- A complete matrix about n^3
- Solving LU about n^2

2.4 Sensitivity and conditioning

Vector norms:

p-norms ⇒ $\|x\|_p = \left(\sum_{i=1}^{n} \|x_i\|^p\right)^{1/p}$

⇒
- 1-norm: $\|x\|_1 = \sum_{i=1}^{n} \|x_i\|$
- 2-norm: $\|x\|_2 = \left(\sum_{i=1}^{n} \|x_i\|^2\right)^{\frac{1}{2}}$
- ∞-norm: $\|x\|_\infty = \max_{1 \le i \le n} \|x_i\|$

In general:
- $\|x\|_1 \ge \|x\|_2 \ge \|x\|_\infty$
- $\|x\|_1 \le \sqrt{n}\,\|x\|_2$
- $\|x\|_2 \le \sqrt{n}\,\|x\|_\infty$

For all p-norms:
- $\|x\| > 0$ if $x \neq 0$
- $\|ax\| = \|a\| \cdot \|x\|$
- $\|x+y\| \le \|x\| + \|y\|$

Matrix norms:

$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$

$\|A\|_1 = \max_j \sum_{i=1}^{m} \|a_{ij}\|$ & $\|A\|_\infty = \max_i \sum_{j=1}^{n} \|a_{ij}\|$

properties:
- $\|A\| \ge 0$ if $A \neq 0$
- $\|aA\| = \|a\| \cdot \|A\|$
- $\|A+B\| \le \|A\| + \|B\|$
- $\|AB\| \le \|A\| \cdot \|B\|$
- $\|Ax\| \le \|A\| \cdot \|x\|$

Matrix condition number:

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

Error estimation.

$Ax = b$ and $Ax' = b + \Delta b$

$A\Delta x = \Delta b$

$$\Rightarrow \frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\Delta b\|}{\|b\|}$$

Residual

residual r of approx. solution x' of $Ax = b$

$\Rightarrow r = b - Ax'$

In theory for nonsingular A: $r = 0$ and $\Delta x = \|x - x'\| = 0$

$\Rightarrow$ In practice not!

$$\Rightarrow \frac{\|\Delta x\|}{\|x'\|} \leq \text{cond}(A) \frac{\|r\|}{\|A\| \cdot \|x'\|}$$

2.5 Software

scipy.linalg.solve, linalg.lu, linalg.cholsky

2.1 Introduction

An overdetermined problem $Ax = b$ with $A: m \times n$ with $m > n$

minimize the residual: $r = b - Ax$

2.2 Normal equations

To minimize $r = b - Ax$

We have: $A^T Ax = A^T b$ if $\text{rank}(A) = n$

2.3 Problem transformations

Transform it into a square linear system.

Orthogonal transformations

Q is orthogonal if $Q^T Q = I$, it preserves the Euclidean norm (2-norm) of a vector

Triangular least squares problems.

$$\begin{bmatrix} R \\ 0 \end{bmatrix} x \cong [c_1 c_2] \Rightarrow \|r\|_2^2 = \|c_1 - Rx\|_2^2 + \|c_2\|_2^2$$

if $Rx = c_1 \Rightarrow \|r\|_2^2 = \|c_2\|_2^2$

QR-factorization

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix} \; ; \; Q^T b = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

$$\Rightarrow \|r\|_2^2 = \|c_1 - Rx\|_2^2 + \|c_2\|_2^2$$

if $Rx = c_1 \Rightarrow \|r\|_2^2 = \|c_2\|_2^2$

Householder transformations

$$H = I - 2\frac{vv^T}{v^Tv}$$

$$H = H^{-1} = H^T$$

The product of Householder transformations is always an orthogonal matrix:

$$Q^T = H_n \cdots H_1 \quad \text{or} \quad Q = H_n^T \cdots H_1^T$$

2.4 rank deficiency

if $\text{rank}(A) < n \Rightarrow$ will perform QR factorization but Upper triangular matrix will be singular.

$\Rightarrow$ multiple x vectors have the same minimal norm

2.5 Singular value Decomposition

Diagonal linear leastsquares even easier to solve.

SVD of A: $A = U\Sigma V^T$

Σ: $\sigma_{ij} = \begin{cases} 0 & i \neq j \\ \sigma_i \geq 0 & i = j \end{cases}$

σ_i are the singular values of A

collums u_i of U and v_i of V are singular vectors

Useful for ill-conditioned or nearly rank-deficient problems.

Other applications:

Euclidean matrix norm: $\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sigma_{max}$

Euclidean condition number: $\text{cond}_2(A) = \frac{\sigma_{max}}{\sigma_{min}}$

rank determination: = number of singular values.

Pseudoinverse

- of a scalar: $\sigma \Rightarrow \frac{1}{\sigma}$
- of a Diagonal matrix: Transpose and pseudo-inverse of each entry

$\Rightarrow$ A pseudo inverse:

$$A^+ = V \Sigma^+ U^T$$

if A is square and nonsingular: $A^{-1} = A^+$

$Ax \cong b \Rightarrow x \triangleq A^+ b$

$A^+ = (A^T A)^{-1} A^T$

2.6 Positivity and conditionality

matrix with $\text{rank}(A) = n$

$$\text{cond}(A) = \|A\|_2 \cdot \|A^+\|_2$$

if $\text{rank}(A) < n \Rightarrow \text{cond}(A) = \infty$

2.7 What method to use?

- Easiest $\Rightarrow$ normal equations, however computationally quite expensive and error is proportional to $(\text{cond}(A))^2$
- Most efficient and accurate orthogonalization method $\Rightarrow$ Householder method.
- SVD most expensive but offers superb robustness

#3 Eigenvalue problems

3.1 Introduction

$Ax = \lambda x \Rightarrow \lambda =$ eigenvalue
$x =$ eigenvector

Characteristic polynomial:

$$\det(A - \lambda I) = 0$$

$\Rightarrow$ Not a good numerical way of computing the eigenvalues.

Properties:
- If A is symmetric/Hermitian, all are real
- $Ax = \lambda x \Rightarrow (A - \sigma I)x = (\lambda - \sigma)x$, x remains unchanged
- A^{-1} and A have same eigenvectors but $\frac{1}{\lambda}$
- A^k and A have same eigenvectors but λ^k
- polynomials: $p(t) \Rightarrow p(A) = p(\lambda)x$
- Similarity if there is a $B = T^{-1}AT$, it has the same eigenvalues but eigenvectors Ty

3.2 Calculating eigenvalues and eigenvectors

Power Iteration

$\Rightarrow$ Estimates the dominant eigenvalue and vector.

Works by multiplying a nonzero vector repeatedly by the matrix.

Usually works well but might fail:
- More than 1 eigenvalue with max. modulus $\Rightarrow$ converge to a linear combination of corresponding eigenvectors
- Real matrix and starting vector can't converge to a complex vector.

Inverse iteration.

⇒ Search the smallest eigenvalue

Using power iteration with LU-factorization.

Rayleigh quotient iteration.

finding λ can be considered a linear least squares problem.

$$x\lambda \cong Ax \qquad \text{for an approximate eig.-vector } x.$$

$$\Rightarrow \quad \lambda = \frac{x^T A x}{x^T x}$$

Deflation.

remove a known eigenvalue from a matrix

so that $v_1 \Rightarrow v_1^T x = \lambda_1 \Rightarrow A - x_1 v_1^T$ has eigenvalues: $\lambda = \lambda_2 \dots \lambda_n$

⇒ Not a good way of calculating it.

QR-iteration:

⇒ fastest and most used method.

$$A_m = Q_m R_m$$

$$A_{m+1} = R_m Q_m$$

⇒ converges to a matrix with the eigenvalues on its diagonal.

3.3 Software

Scipy.linalg.eig.

4 Nonlinear equations

4.1 Introduction

$f(x) = y$ or $f(x) = 0$
$\Rightarrow$ find the roots.

4.2 number of solutions

possible to have degenerate solutions
$\Rightarrow$ multiple roots.
if $f(x) = f'(x) = f''(x) = \dots = f^{(m-1)}(x) = 0$ and $f^{(m)}(x) \neq 0$
x has multiplicity m
if $m = 1 \Rightarrow$ simple root.

4.3 Sensitivity

condition number of f for root x^* is $\frac{1}{\|f'(x^*)\|}$
for a multiple root is the condi. number infinite

Multiple dimensions: cond. number: $\|J_f^{-1}(x)\|$
with J = jacobian

4.4 convergence rate and stopping criteria

convergence rate = effectiveness of a certain algorithm
$e_k = x_k - x^*$ error at iteration k
converge with rate r if: $\lim_{k \to \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = C$
for $C > 0$

- $r = 1$ and $C < 1$: linear convergence
- $r > 1$: superlinear convergence
- $r = 2$: quadratic convergence
- $r = 3$: cubic convergence

Stopping criterion is if

$\|x_{k+1} - x_k\| / \|x_k\| < \varepsilon$ for error tolerance ε

4.1 Nonlinear in 1D

Bisection Method

Begin with a bracket and iteratively reduce the bracket.

convergence $r = 1$ and $c = 0.5$

Always discards half of the bracket at each iteration

for a starter interval $[a, b]$, after k iterations is the length equal to $(b-a)/2^k$

for tolerance ε it requires $\log_2\left(\frac{b-a}{\varepsilon}\right)$ iterations

Fixed-point iteration

$x = g(x)$ is a fixed point

$$x_{k+1} = g(x_k)$$

pairs solutions are $f(x) = 0$.

if $\|g'(x^*)\| < 1$ it is locally convergent

if $\|g'(x^*)\| > 1$ then the scheme diverges for every initial value different from x^*

ideally $\|g'(x^*)\| = 0 \Rightarrow \lim_{k \to \infty} \frac{\|e_{k+1}\|}{\|e_k\|} = \frac{g''(x^*)}{2}$

Newton's method

$f(x+h) \approx f(x) + h f'(x)$, $h = -\frac{f(x)}{f'(x)}$ if $f'(x) \neq 0$

$\Rightarrow$ iterative.

- for simple roots it has asymptotic quadratic convergence
- for multiple roots with multiplicity m (linear convergence) with $C = 1 - \frac{1}{m}$

Secant Method

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

Use Newton with this as derivative.

Inverse interpolation

fit a polynomial p to the values of x_k as function of $f(x_k)$, next solution is $p(0)$.

inverse quadratic interpolation ⇒ fit a parabola.

Root finding in Scipy

Brent method ⇒ is a safeguarded method.

Roots of polynomial functions

find all roots:

- Use one of the method per root.
- Use a routine.

⇒ !. Form the companion matrix and compute eigen.

~~Systems of nonlinear equations~~

Define Jacobian:

$J_f(x)\delta = -f(x)$ then $x+\delta$ is an approximate zero of f.

Able to use Newton and secant.

4.5 optimization

5.1 introduction

some constraints that need to be fulfilled.

concepts and notation

function $f: \mathbb{R}^n \to \mathbb{R}$ and a set $S \subseteq \mathbb{R}^n$ we seek $x^* \in S$ such that f attains a minimum at x^*

x^* is called a minimizer of the objective function

S is the feasible set, if $S = \mathbb{R}^n \Rightarrow$ unconstrained

Use local minimums instead of global minimums

5.2 optimality conditions

Unconstrained optimality conditions

- gradient: $\nabla f(x^*) = 0$
- Hessian matrix:
  - Positive definite: x^* is a minimum
  - Negative definite: x^* is a maximum
  - indefinite: x^* is a saddle point.

5.3 optimization in 1Dimension

a function f is unimodal on an interval $[a,b]$ if there is a x^* that minimizes the function.

Golden section search

compare function values to exclude parts of the interval, the relative distance is the golden ratio, repeat until given error tolerance.

Successive parabolic interpolation

Use 3 points and fits a parabola, minimum of parabola is used as approximate of the minimum

Newton's method

$$f(x+h) \approx f(x) + f'(x)h + \frac{1}{2} f''(x)h^2$$

5.4 multidimensional unconstrained optimization

Direct search

Nelder and mead: $f: \mathbb{R}^n \to \mathbb{R}$ evaluated at $n+1$ starting points.

A new point replaces the worst point, the point is generated on the line of worst point - centroid of the others.

Useful for nonsmooth objective functions.

Steepest decent

$$\phi(\alpha) = f(x + \alpha s) \quad \text{with } s = -\nabla f(x)$$

$\Rightarrow$ 1 dimensional problem, solvable.

Method is reliable but can have slow progress.

Newton's method

$$f(x+s) \approx f(x) + \nabla f(x)^T s + \frac{1}{2} s^T H_f(x) s$$

$$H_f(x)s = -\nabla f(x)$$

Quasi-Newton Methods

$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f(x_k)$$

α_k : line search parameter
B_k : approximation of the Hessian matrix

Secant updating Methods

a factorization of B_k updating instead of B_k

Conjugate gradient method

suitable for large problems.
modifies the gradient at each step to remove components in previous directions.
conjugate product: $(x, y) = x^T H y$

5.5 Nonlinear Least Squares:

residual: $r_i(x) = y_i - f(t_i; x)$
to minimize $\phi(x) = \frac{1}{2} r(x)^T r(x)$

Gauss-Newton method

$$\left(J^T(x_k) J(x_k)\right) p_k = -J^T(x_k) r(x_k)$$

$$\Rightarrow J(x_k) p_k \cong -r(x_k)$$

Levenberg-Marquardt Method

When Gauss-New yields an ill-conditioned or rank-deficient linear least squares.

$$\left(J^T(x_k) J(x_k) + \mu_k I\right) p_k = -J^T(x_k) r(x_k)$$

$$\begin{bmatrix} J(x_k) \\ \sqrt{\mu_k} I \end{bmatrix} \cong \begin{bmatrix} -r(x_k) \\ 0 \end{bmatrix}$$

5.6 Constrained optimization

search local minima

- Equality constraints: $g(x) = 0$
- InEquality constraints: $h(x) \leq 0$

6. interpolation

6.1 introduction

interpolation means fitting some function to give data so that the function has the same values as the give data.

for give data $(t_i, y_i) \Rightarrow f(t_i) = y_i$ f is the interpolant

6.2 polynomial interpolation of discrete data

Monomial basis

n monomials: $\phi_j(t) = t^{j-1}$

equation we want to solve:

$$Ax = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} = y$$

↳ vandermonde matrix

polynomial can be evaluated with Horner's rule, Endunos Nested evaluation or synthetic division.

$$P_{n-1}(t) = x_1 + t(x_2 + t(x_3 + t(\cdots(x_{n-1} + x_n t)\cdots)))$$

Lagrange interpolation

for give data (t_i, y_i), Lagrange basis function:

$$\ell_j(t) = \frac{\prod_{k=1, k\neq j}^{n} (t - t_k)}{\prod_{k=1, k\neq j}^{n} (t_j - t_k)}$$

polynomial is give by: $P_{n-1}(t) = y_1 \ell_1(t) + y_2 \ell_2(t) + \cdots + y_n \ell_n(t)$

Newton interpolation

Newton basis functions:

$$\pi_j(t) = \prod_{k=1}^{j-1} (t - t_k)$$

$$p_{n-1}(t) = x_1 + x_2(t-t_1) + x_3(t-t_1)(t-t_2) \dots$$

$$p_{j+1}(t) = p_j(t) + x_{j+1}\pi_{j+1}(t)$$

Polynomial interpolation of a continuous function

Chebyshev points defined on interval $[-1,1]$:

$$t_i = \cos\left(\frac{(2i-1)\pi}{2k}\right)$$

6.3 piecewise polynomial interpolation

interval divided into subintervals
and lower degree polynomials are fitted
Points at which the interpolant changes are knots.

Cubic spline interpolation

A spline with n knots $\Rightarrow$ Cubic splines has $4(n-1)$ free parameters.

Used in software.

7. Numerical Differentiation

7.1 integration

based on Riemann sums:

$$R_n = \sum_{i=1}^{n} (x_{i+1} - x_i) f(\xi_i)$$

if R_n is finite, then the function is Riemann integrable.

7.2 uniqueness, existence, conditioning

- Integrable if it is bounded
- uniqueness is build into the definition
- well behaved with a small condition number.

7.3 Numerical Quadrature

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

Numerical approximation is numerical quadrature.

quadrature rule: $Q_n(f) = \sum_{i=1}^{n} w_i f(x_i)$

x_i: nodes, w_i: weights

to find weights => method of undetermined coefficients

general system of moment equations:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & & & \vdots \\ x_n^{n-1} & & & \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} b-a \\ (b^2-a^2)/2 \\ \vdots \\ (b^n-a^n)/n \end{bmatrix}$$

accuracy and useful concepts
degree 'd' if it is exact for polynomial of degree d but not d+1

degree gives accuracy
Rough error bound:

$$\|I(f) - Q_n(f)\| \leq \frac{1}{4} h^{n+1} \|f^{n}\|_{\infty}$$

$h = \max \{ x_{i+1} - x_i \}$ $i = 1 \dots n-1$

Newton-Cotes quadrature

open Newton-Cotes rule has nodes:

$$x_i = a + i(b-a)/(n+1)$$

closed:

$$x_i = a + (i+1)(b-a)/(n+1)$$

- Midpoint rule:

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

- Trapezoid rule

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

- Simpson's rule

$$S(f) = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Error Analysis

- Midpoint rule:

$$I(f) = f(m)(b-a) + \frac{f''(m)}{24}(b-a)^3 + \frac{f^{4}(m)}{1920}(b-a)^5$$
$$= M(f) + E(f) + F(f)$$

- Trapezoid rule:

$$E(f) \approx \frac{T(f) - M(f)}{3}$$

- Simpson's rule:

$$I(f) = \frac{2}{3} M(f) + \frac{1}{3} T(f) - \frac{2}{3} F(f) + \dots = S(f) - \frac{2}{3} F(f) + \dots$$

⇒ Newton-Cotes quadrature are relatively easy to derive and apply but has some serious drawbacks

Clenshaw-Curtis quadrature

quadrature based on chebyscev points.

Gaussian quadrature

nodes and weights are chosen to maximize the degree of the resulting quadrature rule

- nodes are symmetrically placed about the midpoint
- nodes are usually irrational numbers
- more difficult than Newton-Cotes

Composite quadrature

divide interval in subintervals with length: $h = \frac{b-a}{n}$ and taking a simple quadrature in each subinterval

Adaptive quadrature

the interval of integration is selectively refined to reflect the behavior of any particular integrand function.

7.4 other integration problems

Tabular Data

Integrable by piecewise interpolation

Improper integrals

- Integrating an integral with unbounded interval of integration:
  - Replace infinites with finites
  - Transform variable so that new interval is finite
  - Use a quadrature rule
- Integrating an integral with singularities
  - Remove by transforming the variable or by dividing out or subtracting the singularity.

Double integrals

Multiple integrals

=> Monte carlo

7.5 numerical differentiation

Finite Difference Approximations.

- First derivative:
  - forward difference formula: $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
  - backwards difference formula: $f'(x) \approx \frac{f(x)-f(x-h)}{h}$
  - centered difference formula: $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$

- Second derivative:
  - centered: $f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$

7.6 Richardson extrapolation

Based on values for nonzero stepsizes, estimate what the value would be for a stepsize of zero

examples: Romberg integration.

7. Solving ODEs and PVP

7.1 Introduction

Differential equations has derivatives in it

Only one independent variable : ordinary differential equation.

7.2 Numerically Solving ODEs

Euler forward method

Euler's method: $t_{k+1} = t_k + h_k$:

$$y_{k+1} = y_k + h_k f(t_k, y_k)$$

=> single step method.

- finite difference approximation

$$y'(t) = f(t, y)$$

- Taylor series.

Accuracy and Stability

- Rounding error
- Truncation error (method error)
- global error => $e_k = y_k - y(t_k)$
- local error, error made in one step of the method.

Accuracy is said to be of order p if

$$I_k = O(h_k^{p+1})$$

Stability

Implicit methods

Euler backward method:

$$y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1})$$

Stability of euler backward method to stable:

$$\left| \frac{1}{1+h\lambda} \right| \leq 1 \quad \text{with } y' = \lambda y$$

Stiffness

a stable ODE is stiff if its Jacbian has eigenvalues that differ greatly in magnitude.

Runge-Kutta methods

Taylor series methods take higher derivatives in to account

RK-method uses approximations for the higher derivatives
Butcher tableau's are a way to write down all necessary information in an elegant way.

Adaptive stepsize

$$h_{opt} = h_{current} \left(\frac{\varepsilon}{h_{current} \tau} \right)^{\frac{1}{N}}$$

classical RK-method uses a fixed step.
Heun's method uses adaptive stepsize as well as Dormand-Prince method
Bogacki-Shampine has the first-same-as-last property

Extrapolation method

extrapolating a function to fit the computed data.

example is Richardson extrapolation.

Multistep methods

uses information at more than one previous point to estimate the solution.

example: Adams-Bashforth method.

properties:

- Use single-step in the beginning
- changing step size is complicated
- local error estimation: predictor - corrector
- Implicit methods have higher stability than explicit
- Implicit can be effective to compute diff problems.

Multivalue methods

based on polynomial interpolation

2.3 BVPs for ODEs

Introduction

if the side conditions are specified at more than one point => boundary value problem

Shooting Method

for 2 point boundary value problem

replaces BVP by a sequence of initial value problems

7.1 Introduction

partial differential equation involves partial derivatives of an unknown function with respect to more than one independent variable.

7.2 Classification and examples

some important:
- Heat equation
- Wave equation
- Laplace equation

general form:

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0$$

if: $\cdot\ b^2 - 4ac > 0$: Hyperbolic
$\cdot\ b^2 - 4ac = 0$: parabolic
$\cdot\ b^2 - 4ac < 0$: elliptic

7.3 Solving time-dependent problems

2 methods:
- Semidiscrete methods: discretize in space but leave the time variable continuous, this then forms an ODE
- Fully discrete methods: all variables are discretized

Heat equation

$$u_t = c u_{xx} \quad 0 \le x \le L,\ t \ge 0$$

$\cdot\ u(0,x) = f(x);\ 0 \le x \le L$
$\cdot\ u(t,0) = \alpha,\ u(t,L) = \beta;\ t \ge 0$

solve with a semidiscrete method

Wave equation

$$u_{tt} = c u_{xx} \quad 0 \le x \le L, \; t \le 0$$

- $u(0,x) = f(x)$, $u_t(0,x) = g(x)$, $0 \le x \le L$
- $u(t,0) = \alpha$, $u(t,L) = \beta$, $t \ge 0$

solve with a fully discrete method

Implicit methods

Some methods have no stability restriction on the relative sizes of Δt and Δx.

Crank-Nicolson Method (Trapezoid method) is unconditionally stable.

7.4 Solving time-independent problems

Laplace equation

special case of poisson equation:

$$u_{xx} + u_{yy} = f(x,y)$$

Laplace $\Leftrightarrow f = 0$

We have:

- Dirichlet boundary conditions: solution u is specified.
- Neumann boundary conditions: u_x or u_y is specified.
- Robin boundary conditions: a combination of the above.

Laplace equation = potential equation

solve with a finite difference method

8 Fast Fourier Transform

8.1 Notation

euler's identity: $e^{i\theta} = \cos\theta + i\sin\theta$

$$\Rightarrow \cos(2\pi f t) = \frac{e^{2\pi i f t} + e^{-2\pi i f t}}{2}$$

$$\sin(2\pi f t) = \frac{e^{2\pi i f t} - e^{-2\pi i f t}}{2i}$$

$$\omega_n = \cos\left(\frac{2\pi}{n}\right) - i\sin\left(\frac{2\pi}{n}\right) = e^{-\frac{2\pi i}{n}}$$

8.2 Discrete Fourier transform

Definition

a sequence $x = [x_0 \dots x_{n-1}]^T$, discret fourier transform

$$\Rightarrow y = [y_0 \cdots y_{n-1}]^T \Rightarrow y_m = \sum_{k=0}^{n-1} \omega_n^{mk} x_k$$

$$\Rightarrow y = F_n x$$

$$F_n^{-1} = \frac{1}{n} F_n^H$$

Frequencies

$$t_k = t_0 + \frac{k}{f_s} \Rightarrow k = (t_k - t_0) f_s$$

$$\omega_n^{-mk} = \cos\left(\frac{2\pi m k}{n}\right) + i\sin\left(\frac{2\pi m k}{n}\right)$$

$$= \cos\left(\frac{2\pi m}{n}(t_k - t_0) f_s\right) + i\sin\left(\frac{2\pi m}{n}(t_k - t_0) f_s\right)$$

with f_s the sampling rate

Negative frequencies

$W_n^{-mk} = W_n^{(n-mk)}$ (arbitrary integer because $W_n^n = 1$)

$$\Rightarrow x_k = \frac{1}{n} \sum_{m=0}^{n-1} y_m W_n^{(n-m)k}$$

or

$$x_k = \frac{1}{n} \sum_{m=0}^{n-1} y_m W_n^{(-n-m)k}$$

Highest frequency is $\frac{f_s}{2}$ => Nyquist frequency

DC component (zero frequency)

Lowest frequency = 0

=> corresponding coefficient is sum of x_k (the DC component)

DFT of a real sequence

DFT of a real sequence is in general complex

- y_0 is real
- n is even: $y_{n/2}$ is real
- n is odd: $y_{(n+1)/2}$ may be complex

2.3 FFT - algorithm

$$D_{n/2} = \text{diag}\left(1, W_n \cdots W_n^{(n/2)-1}\right)$$

to apply F_n => apply $F_{n/2}$ to its even and odd subsequences and scale the result by $\pm D_{n/2}$

=> Fast Fourier transform

Limitations and extensions

- input sequence needs to be evenly spaced
- input is assumed to be periodic.
- sequence is assumed to be power of 2 in length.

Performance demo

7.1 Applications

Signal processing

=> · to filter out unwanted high-freq. (take the FFT, set high freq to zero and take inverse)

· processing of data which contains multiple periodicities.

Efficient calculation of convolutions

2 periodic sequences:

$$\{u * v\}_n = \sum_{k=0}^{m-1} v_k u_{m-k}$$

=> equivalent to multiplication by a circular matrix

=> more efficient to use FFT algorithm to transform the inputs to the frequency domain, compute one pair wise multiplication and transform the result back to the domain

Autocorrelation

expresses the similarity between a sequence and a delayed copy of itself.

=> convolution of the sequence with a reversed copy of itself.

Fast polynomial multiplication

polynomial product as a convolution:

$$f = f_1 * f_2$$

12 Random Deviates

12.1 Pseudo-random number generation

Any computational result is not random.

Don't use LCG for serious applications

Key ingredients of a PRNG

- A seed
- Algorithm has an internal state
- A recurrence relation.
- Fixed parameters

Properties of pseudo-random sequences

1. sequence is deterministic
2. a limited number of pseudo random values
3. pseudo-random sequences must be periodic

Desirable characteristics of a PRNG

1. uniformly distributed.
2. No apparent statistical correlations between subsequent values

Modern PRNG algorithms

- Xorshift

Transformations of univariate continuous distributions.

- Inverse transform sampling
- The Box-muller transform

References
- Mersenne Twister
- XorShift
- XorShift 256**

Built-in, Numpy and Scipy implementations
Built-in random ⇒ Don't use
Numpy uses permuted Congruential Generator
Scipy has a broad Selection

9.2 Basics of the Monte Carlo

Relies on this identity:

$$\langle g(X) \rangle = \int g(x) p_X(x) dx$$

Error estimation

$$V\left[\frac{1}{N}\sum_{i=1}^{N} g(X_i)\right] = \frac{1}{N} V(g(X_i))$$

$$\sigma = \sqrt{\frac{V(g(X_i))}{N}}$$

$$V(g(X_i)) \approx \frac{1}{N-1}\sum_{i=1}^{N} \left(g(X_i) - \overline{g(X_i)}\right)^2$$

Application to error propagation

Common pitfall
For some choices of g, it becomes infeasible to convey the MC estimate

Monte Carlo integration

$$I = \int_\Omega g(x) dx$$

$$\rightarrow I = V_\Omega \int_\Omega g(x) p_{X \sim U(a,b)} dx$$

7.3 Discrete Markov chain Monte Carlo methods

Definition of a discrete Markov chain

A discrete Markov chain is a stochastic process
⇒ the probability of observing x_{i+1} is determined by: • previous state x_i
• index i

$$\Rightarrow p_{x_{i+1}}(x_{i+1}) = \int T_{i\to i+1}(x_{i+1}|x)\, p_{x_i}(x_i)\, dx$$

- function $T_{i\to i+1}$ is not explicitly dependent on x_{i-m}
- if $T_{i\to i+1}$ same for all i ⇒ time-homogeneous chain

important property: $\int T(x_{i+1}|x_i)\, dx_{i+1} = 1$

The stationary distribution of a discrete Markov chain

$$\int T(x_2|x_1)\, p(x_1)\, dx = p(x_2)$$

the chain as a random number generator for its corresponding stationary distribution.

statistical technicalities:
- Stationary distribution does not always exist
- It is not necessarily unique

Metropolis-Hastings algorithm

- A stationary distribution satisfies the global balance condition: after convolution with the transition probability, the same probability is recovered.

- Detailed balance:

$$T(x_2|x_1)p(x_1) = T(x_1|x_2)p(x_2)$$

Metropolis-Hasting defines a Markov chain that satisfies detailed balance for any given stationary distribution

constructed:
- Generation step
- Acceptance or rejection step

Algorithm:

1. Start with an initial state for which the probability is non-zero
2. Generate a step according to the proposal distribution $g(x_2|x_1)$
3. compute the acceptance ratio:

$$AR = \frac{p(x_2)}{p(x_1)} \frac{g(x_1|x_2)}{g(x_2|x_1)}$$

4. repeat steps 2 & 3 until sample size is sufficient