PYU33C01 Computer Simulations I
Numerical and Statistical Methods

(13 hours lectures + 2 hours computer laboratory)

S Hutzler

October 5, 2023

Contents

0 Prerequisite: Taylor Expansion **1**

1 Numerical Differentiation **3**

1.1 Difference schemes 3

1.2 Example 4

2 Richardson Extrapolation with Application to Numerical Derivatives **7**

3 Data Handling **9**

3.1 Lagrange Interpolation 9

3.2 Least-square fitting 11

3.2.1 Linear regression 11

3.2.2 Fitting to polynomials of degree m 12

3.2.3 Fitting to non-linear functions 13

4 Numerical Integration **14**

4.1 Introduction 14

4.2 Trapezoid, Simpson's and Boole's rule 15

4.3 Romberg Integration Scheme 17

4.4 Improper Integrals 18

5 Ordinary Differential Equations (ODE) **21**

5.1 First order differential equations with initial values 21

5.2 Second order ODEs 23

5.3 Constants of motion 25

5.4 Adaptive step sizes 25

6 Random numbers and Monte Carlo Integration **27**
6.1 Generation of (pseudo) random numbers 28
6.2 Tests of Random number generators 29
6.3 Creating non-uniform distributions of random numbers 30
6.4 Monte Carlo Integration 33
6.4.1 The statistics of Monte Carlo integration 33
6.4.2 Importance Sampling 35

7 Monte Carlo Simulations **36**
7.1 Canonical ensemble, Boltzmann distribution, partition sum 36
7.2 Metropolis algorithm 37
7.3 Modelling the Ising Ferromagnet 39
7.3.1 The model 40
7.3.2 Implementation of the Metropolis algorithm 40
7.3.3 Properties of the Ising model for an infinite square lattice . . . 42
7.4 Finite size scaling 43
7.5 The q-state Potts model 43
7.6 Simulated annealing and the travelling salesman problem 44
7.6.1 The model Hamiltonian 45
7.6.2 Notes on the algorithm 46
7.6.3 Results 46

8 Genetic Algorithms **48**
8.1 Introduction 48
8.2 Application to spin glasses 49
8.3 Important features of genetic algorithms 51
8.4 Example: finding maximum of a function $f(x)$ 51

9 Neural networks **53**
9.1 Introduction 53
9.2 A simple model: the Perceptron (1958) 53
9.3 The Hopfield model 56

A References **57**

Chapter 0

Prerequisite: Taylor Expansion

- Expansion of a function in a Taylor series is a most fundamental technique in physics, used extensively in the derivation of numerical methods.
- Aim is to find an approximation to a function $f(x)$ for values near some particular value $x = a$ where one knows $f(x = a)$.
- A crude approximation of $f(x)$ is a straight line that passes through $f(a)$ and has the correct slope at $x = a$. Remembering that the local slope of a function corresponds to its first derivative at that point this gives

$$f(x) \simeq f(a) + (x - a)\frac{df}{dx}|_{x=a} = f(a) + (x - a)f'(a). \tag{1}$$

- An approved approximation takes into account the curvature of the function, i.e. $\frac{d^2 f(a)}{dx^2} = f''(a)$. This corresponds to the *change* in slope at $x = a$. So let's approximate $f(x)$ by a quadratic function $c(x) = c_0 + c_1(x - a) + c_2(x - a)^2$ and fix the constants c_0, c_1 and c_2 by demanding that the derivatives with respect to x at $x = a$ are correct.

$$\frac{d^0 f(x = a)}{dx^0} = f(a) \equiv c(a) = c_0$$
$$\frac{df(a)}{dx} = f'(a) \equiv c'(a) = c_1 + 2c_2(x - a)|_{x=a} = c_1$$
$$\frac{d^2 f(a)}{dx^2} = f''(a) \equiv c''(a) = 2c_2$$

Thus the approximation for $f(x)$ is given by

$$f(x) \simeq c(x) = f(a) + (x - a)f'(a) + \frac{(x - a)^2}{2} f''(a). \tag{2}$$

- A continuation of this scheme leads to the **Taylor expansion** (1715):

$$f(x) = f(a)+(x-a)f'(a)+\frac{(x-a)^2}{2!}f''(a)+\frac{(x-a)^3}{3!}f'''(a)+\frac{(x-a)^4}{4!}f^{(4)}(a)+\cdots+rest \tag{3}$$

  Provided all the derivatives exist we can write (**Taylor series**)

$$f(x) = \sum_{n=0}^{\infty} \frac{(x-a)^n}{n!} f^{(n)}(a) \tag{4}$$

  where $f^{(n)}(a) = \frac{df^n(x)}{dx^n}|_{x=a}$.

  For $a = 0$ this is also called **Mac Laurin series**.

- Examples for expansion around $a = 0$:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$
$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

- But note that series does not converge for all functions, e.g.

$$f(x) = \begin{cases} \exp(-1/x^2) & \text{for } x \neq 0 \\ 0 & \text{for } x = 0 \end{cases}$$

  where all derivatives are zero at $x = 0$.

Chapter 1

Numerical Differentiation

1.1 Difference schemes

- The first derivative of a function is defined as

$$f'(x) = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{1.1}$$

.

- Approximation for finite h:

$$f'_h(x) = \frac{f(x+h) - f(x)}{h} \tag{1.2}$$

How good is this approximation?

- Taylor expansion:

$$f(x+h) = f(x) + hf'(x) + h^2/2!f''(x) + \cdots \tag{1.3}$$

thus

$$f'(x) = \frac{1}{h}[f(x+h) - f(x) - h^2/2!f''(x) - \cdots] \tag{1.4}$$

- Thus we obtain the **forward difference approximation**

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h) \tag{1.5}$$

with the error $O(h) = -h/2!f''(x) - h^2/3!f'''(x) - \cdots$ explicitly given. Here the notation $O(h)$ simply refers to the **leading order in** h of the error, i.e. here linear in h.

- Let us perform a Taylor expansion of

$$f(x-h) = f(x) - hf'(x) + h^2/2!f''(x) - h^3/3!f'''(x) + \cdots \tag{1.6}$$

We can then write down the **backward difference approximation** as

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h). \tag{1.7}$$

The error of both approximations are of the same order, BUT the LEADING terms have OPPOSITE sign.

- Evaluating $f(x+h) - f(x-h)$ we find the **central difference approximation** which is now quadratic in h.

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \tag{1.8}$$

where now $O(h^2) = -h^2/3!f'''(x) - h^4 f^{(5)}(x) - \cdots$, i.e. all terms involving odd powers of h have disappeared.

- To compute **second derivatives** compute $f(x+h) + f(x-h)$ to obtain

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2) \tag{1.9}$$

where $O(h^2) = -h^2/12f^{(4)}(x) - h^4/360f^{(6)}(x) + \cdots$, i.e. there are **no odd powers** of h.

- The **forward difference method for the 2nd derivative** reads

$$f''(x) = \frac{f(x) - 2f(x+h) + f(x+2h)}{h^2} + O(h) \tag{1.10}$$

where now the error is linear in h, $O(h) = -hf'''(x) - \frac{7}{12}h^2 f^{(4)}(x) \cdots$.

1.2 Example

- Consider $f(x) = x\exp(x)$, thus $f'(x) = \exp(x) + x\exp(x) = \exp(x)(1+x)$. Example calculations for different numerical differentiation schemes are shown below.

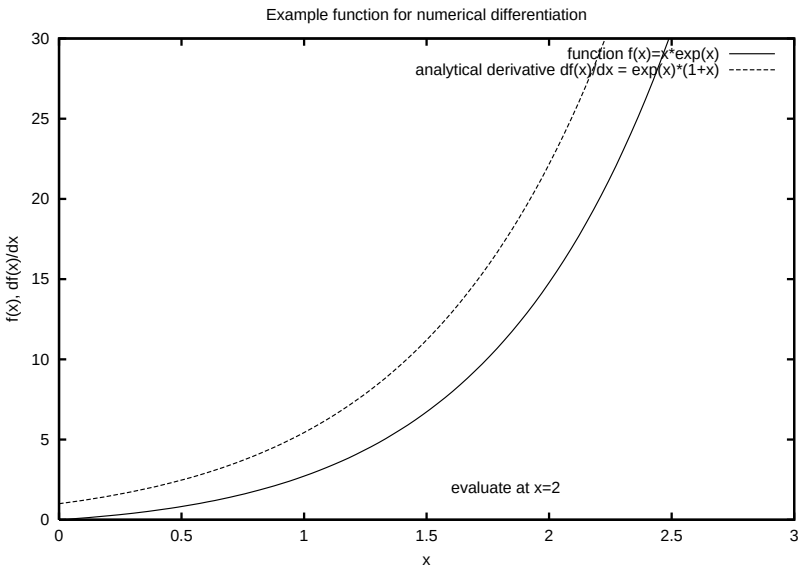

Figure 1.1: Example function $f(x) = x\exp(x)$ and its analytical first derivative $f'(x) = \exp(x)(1+x)$.

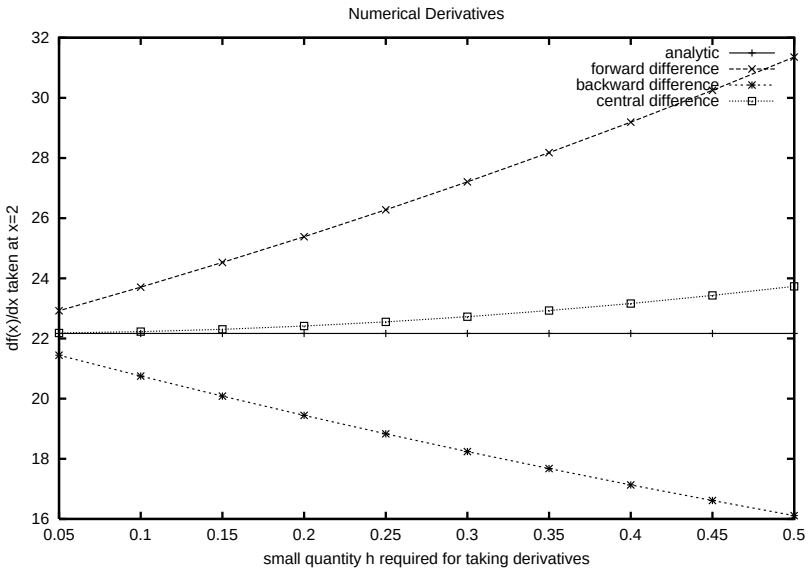

Figure 1.2: Analytical and numerical evaluation of the first derivative **at x=2.**

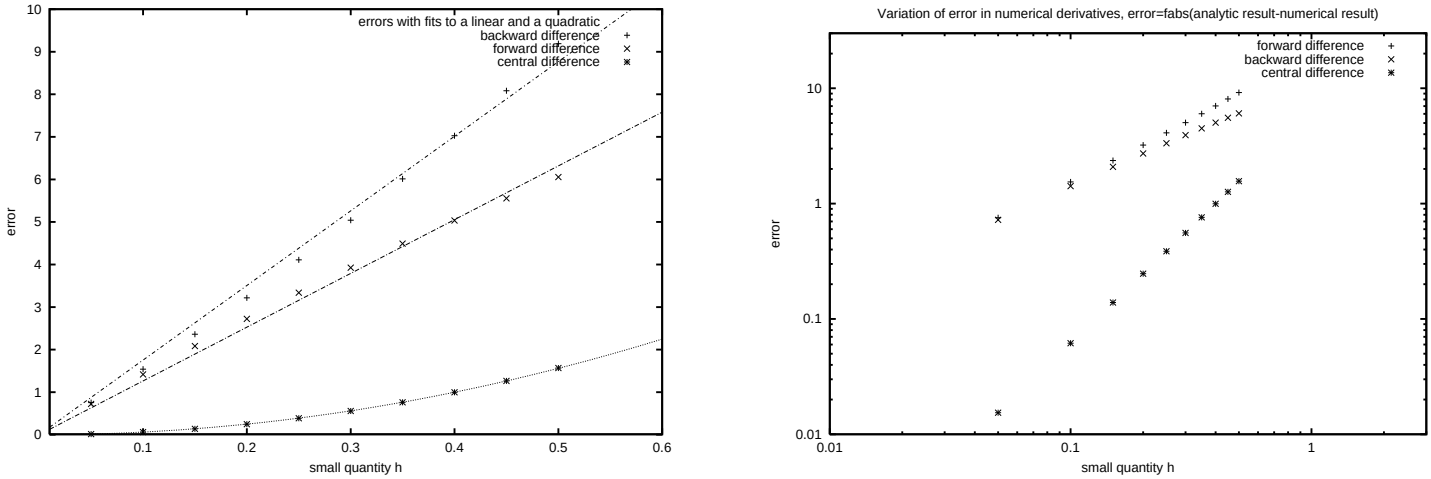

Figure 1.3: Absolute error as a function of step size h for the different differentiation schemes. *Left:* Fits to a linear and a quadratic function respectively. *Right:* Plot on a double logarithmic scale, showing lines of slope one and two, approximately.

Analyze accuracy of numerical results by computing absolute and relative error.

$$\text{absolute error} = |\text{true value - approximative value}|$$
$$\text{relative error} = \left|\frac{\text{true value - approximative value}}{\text{true value}}\right|$$

The *order* of the leading term of the error can be obtained from a plot of the error $E(h)$ as a function of h: for $E(h) \simeq h^n$ we obtain $\ln E(h) \simeq n \ln h$. The order corresponds to the slope of such a plot.

Chapter 2

Richardson Extrapolation with Application to Numerical Derivatives

- Invented by the meteorologist L.F. Richardson ("'Weather prediction by numerical process", 1922), this is a **general technique for the elimination of leading error terms**. Here it will be introduced for deriving numerical differentiation schemes.

- central difference approximation:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{6}h^2 f'''(x) + \cdots \tag{2.1}$$

- now use step size **twice as large**, $2h$,

$$f'(x) = \frac{f(x+2h) - f(x-2h)}{4h} - \frac{4}{6}h^2 f'''(x) + \cdots \tag{2.2}$$

  this results in a leading error which is **four times** as large as before.

- We can eliminate this error in h^2 by computing eq.(2.1) - eq.(2.2)/4 $= 3/4 f'(x)$. We finally arrive at

$$f'(x) = \frac{f(x-2h) - f(x+2h) + 8f(x+h) - 8f(x-h)}{12h} + \frac{1}{30}h^4 f^{(5)}(x) + \cdots \tag{2.3}$$

  which is called the **five point central difference method**.

- Let us now **generalise this procedure** and develop a computational scheme.

- **Short-hand notation:** $D_1(h)$: approximation of first derivative using the 3 point central difference method (CDM) with step size h. $D_1(2h)$: CDM with step size $2h$.

$$D_1(h) = \frac{f(x+h) - f(x-h)}{2h}, \tag{2.4}$$

$$D_1(2h) = \frac{f(x+2h) - f(x-2h)}{4h}. \tag{2.5}$$

- $D_1(2h)$ contains 4 times the leading error of $D_1(h)$ (c.f. eqn.(2.1)), thus the difference between the two is 3 times the error of $D_1(h)$.

- Let's compute this error and then remove from $D_1(h)$ to get an improved estimate, called $D_2(h)$, of the real derivative.

$$D_2(h) := D_1(h) - \left(\frac{D_1(2h) - D_1(h)}{2^2 - 1}\right) = \frac{4D_1(h) - D_1(2h)}{3} \tag{2.6}$$

  Since D_1 has only *even* powers of h in the error, the leading error in $D_2(h)$ is of order $O(h^4)$.

- next step: $D_2(2h)$ contains error that is 2^4 times as large as error in $D_2(h)$. Let's remove it to obtain

$$D_3(h) := D_2(h) - \left(\frac{D_2(2h) - D_2(h)}{2^4 - 1}\right) = \frac{16D_2(h) - D_2(2h)}{15}. \tag{2.7}$$

- The general expression is given by

$$D_{n+1}(h) := D_n(h) - \left(\frac{D_n(2h) - D_n(h)}{2^{2n} - 1}\right) = \frac{2^{2n}D_n(h) - D_n(2h)}{2^{2n} - 1}. \tag{2.8}$$

- The implementation of this scheme into a computer code is as follows.

h	D_1	D_2	D_3	$\cdots$
$0.4 \rightarrow$	$D_1(0.4) \searrow$			
$0.2 \rightarrow$	$D_1(0.2) \rightarrow\searrow$	$D_2(0.2) \searrow$		
$0.1 \rightarrow$	$D_1(0.1) \rightarrow\searrow$	$D_2(0.1) \rightarrow\searrow$	$D_3(0.1) \searrow$	
$0.05 \rightarrow$	$D_1(0.05) \rightarrow\searrow$	$D_2(0.05) \rightarrow\searrow$	$D_3(0.05) \rightarrow\searrow$	$D_4(0.05) \searrow$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

  - each new row is computed with only one newly evaluated derivative and entries of previous row
  - four or five columns are usually enough as rounding errors will creep in

Chapter 3

Data Handling

3.1 Lagrange Interpolation

Given a discrete data set (x_i, y_i) the aim is to find a polynomial $p(x)$ that passes through all these points.

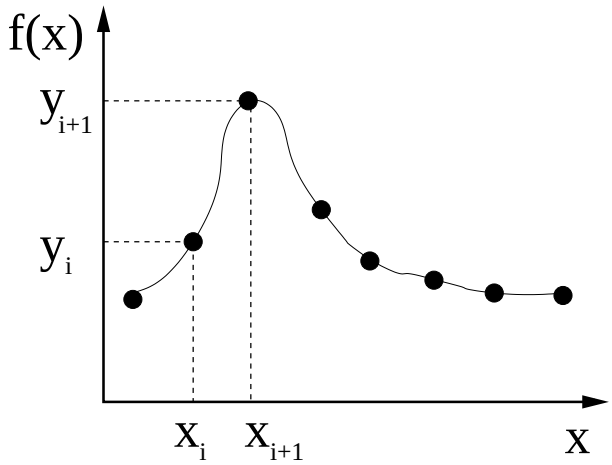

Figure 3.1: Example of a Lagrange interpolation. 8 data points are interpolated by a 7th order polynomial.

- Alternative formulation of this problem: aim is to approximate a function $f(x)$ by a polynomial $p(x)$. All one knows about $f(x)$ are its values $y_i = f(x_i)$ at points x_i.
- For two points the connecting straight line is given by $p(x) = y_i + \frac{y_{i+1}-y_i}{x_{i+1}-x_i}(x - x_i)$, for $x_i \leq x \leq x_{i+1}$.

- The generalisation ("n data points may be interpolated by an $(n-1)$th order polynomial") is called the method of **Lagrange interpolation**, given by

$$p(x) = \sum_{j=1}^{n} l_{j,n}(x) f(x_j),$$

$$\text{with } \textbf{weighting factors } l_{j,n}(x) = \Pi_{i \neq j}^{n} \frac{x - x_i}{x_j - x_i}. \tag{3.1}$$

  The following can be shown,

$$\sum_{j=1}^{n} l_{j,n}(x) = 1.$$

- Lagrange interpolation for $n = 2$:

$$p(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2), \tag{3.2}$$

  i.e. an equation for a straight line $p(x)$ passing through $(x_1, f(x_1))$ and $(x_2, f(x_2))$.

- Lagrange interpolation for $n = 3$:

$$p(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3). \tag{3.3}$$

  This is a *quadratic* in x passing through the three points $(x_1, f(x_1)), (x_2, f(x_2)), (x_3, f(x_3))$.

Comment: global vs local interpolation

global interpolation using high order polynomial may lead to "artificial extrema" that do not represent the data. Preferable: *local* quadratic/cubic interpolation of consecutive sets of 3 or 4 data points

Comment: Cubic splines

- a sophisticated version of $n = 4$ interpolation: sequence of third order polynomial interpolation of 4 points, so that the first and second derivatives (slope and curvature) match at the endpoints of each local interpolation

- used by many graphics packages

- further required input: derivatives at the endpoints

3.2 Least-square fitting

- experimental (and numerical!) data contains "scatter" due to experimental errors, numerical accuracy etc. Data should be described by a "best fit" without requiring the approximation of the data to actually coincide with it at any of the specified data points.

- procedure: evaluate difference between "functional guess" $f(x)$ and data $(x_1, y_1), \cdots, (x_N, y_N)$ by computing the *sum of the squared residuals*,

$$S = \sum_{i=1}^{N} (f(x_i)_{p_1,\cdots,p_k} - y_i)^2, \tag{3.4}$$

  and then choose *fit parameters* $p_1, \cdots, p_k$ contained in $f(x)$ so that S is minimised.

3.2.1 Linear regression

- example: moving object, measure position x_i as function of time t_i, N measurements, we guess (based on some experience) that velocity is a constant, aim is to determine it.

- functional guess

$$x(t) = a + bt \tag{3.5}$$

- sum of squared residuals given by $S = \sum_{i=1}^{N} (x(t_i) - x_i)^2$

- minimisation of S with respect to fit parameters a and b

$$\frac{\partial S}{\partial a} := 0; \quad \frac{\partial S}{\partial b} := 0; \tag{3.6}$$

$$\frac{\partial S}{\partial a} = 2 \sum_{i=1}^{N} (x(t_i) - x_i) \frac{\partial x}{\partial a}|_{t=t_i} = 2 \sum_{i=1}^{N} (a + bt_i - x_i) := 0 \tag{3.7}$$

$$\frac{\partial^2 S}{\partial a^2} = 2N > 0, \text{ minimum}$$

$$\frac{\partial S}{\partial b}|_{t=t_i} = 2 \sum_{i=1}^{N} (a + bt_i - x_i) t_i := 0 \tag{3.8}$$

$$\frac{\partial^2 S}{\partial b^2} = 2 \sum_{i=1}^{N} t_i^2 > 0, \text{ minimum}$$

- thus solve the two resulting *linear* equations for the fit parameters a and b:

$$aN + b\sum_{i=1}^{N} t_i = \sum_{i=1}^{N} x_i \tag{3.9}$$

$$a\sum_{i=1}^{N} t_i + b\sum_{i=1}^{N} t_i^2 = \sum_{i=1}^{N} x_i t_i \tag{3.10}$$

3.2.2 Fitting to polynomials of degree m

$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_m x^m$, *linear* in coefficients c_i

- sum of the squared residuals: $S = \sum_{i=1}^{N}(p(x_i) - y_i)^2$, where we have N data points (x_i, y_i)

- minimisation of S with respect to the c_i gives $m+1$ equations

$$\begin{aligned}
\frac{\partial S}{\partial c_0} &= \sum_{i=1}^{N} 2x_i^0(c_0 + c_1 x_i + \cdots + c_m x_i^m - y_i) = 0 \\
\frac{\partial S}{\partial c_1} &= \sum_{i=1}^{N} 2x_i^1(c_0 + c_1 x_i + \cdots + c_m x_i^m - y_i) = 0 \\
&\vdots \\
\frac{\partial S}{\partial c_m} &= \sum_{i=1}^{N} 2x_i^m(c_0 + c_1 x_i + \cdots + c_m x_i^m - y_i) = 0
\end{aligned} \tag{3.11}$$

- set of $(m+1)$ linear equations for $m+1$ unknowns c_i

- rewrite as

$$\begin{aligned}
c_0 N + c_1\sum_{i=1}^{N} x_i + \cdots + c_m\sum_{i=1}^{N} x_i^m - \sum_{i=1}^{N} y_i &= 0 \\
&\vdots \\
c_0\sum_{i=1}^{N} x_i^m + c_1\sum_{i=1}^{N} x_i^{m+1} + \cdots + c_m\sum_{i=1}^{N} x_i^{m+m} - \sum_{i=1}^{N} x_i^m y_i &= 0
\end{aligned} \tag{3.12}$$

These are called *normal equations* which can be solved for example by Gaussian elimination or the method of *LU* matrix decomposition.

- **remarks**
  - higher order polynomials do not necessarily result in better fits
  - check whether residual error $y(x_i) - p(x_i)$ is scattered about 0
  - more meaningful: use of a functional form resulting from physical theory

3.2.3 Fitting to non-linear functions

example: $I(\lambda) = \frac{I_0}{1+4(\lambda-\lambda_0)^2/\Gamma^2}$, (Cauchy distribution/Lorentzian function: broadening of spectral lines)

- here $I(\lambda)$ is **non-linear** function of fit parameters λ_0, Γ, I_0
- normal equations are also non-linear!
- **general approach:** N data-points (x_k, y_k), proposed function $Y(x_k; a_1, \cdots, a_m)$ with m fit-parameters (coefficients) a_i (in the above example $m = 3$)
- **procedure:** minimise sum of squared residuals S by varying a_i in turn, until a minimum is found
  - choose initial guesses $a_1^0, a_2^0, \cdots, a_m^0$ to give S close to a minimum (e.g. get estimates by plotting data and function Y); thus S will be approximately quadratic
  - evaluate S at $a_1^0 + h_1, a_1^0, a_1^0 - h_1$, holding all other parameters fixed
  - find minimum of a quadratic interpolating polynomial through points. Call the value of the first fit parameter at which S has a minimum a_1^1.
  - SKETCH (see lecture)
  - $a_1^1 = a_1^0 - \frac{h_1}{2} \frac{S(a_1^0+h_1,...)-S(a_1^0-h_1,...)}{S(a_1^0+h_1,...)-2S(a_1^0,...)+S(a_1^0-h_1,...)}$
  - repeat procedure for $a_2, a_3, ..., a_m$ with $h_2, h_3, ..., h_m$ in turn
  - this completes **one** iteration, resulting in parameters $a_1^1, a_2^1, a_3^1, ..., a_m^1$
  - repeat procedure until S no longer decreases by factor of 2 or similar
  - decrease step sizes h_i in every cycle (e.g. half the step size)
  - method can be slow to converge but works quite well, improvement would require more information on the "shape" of S using Hessian matrix (second derivatives)

Chapter 4

Numerical Integration

4.1 Introduction

- definite integral of continuous function f(x):

$$\int_a^b f(x)dx = \lim_{N\to\infty}\left(h\sum_{i=1}^{N} f(x_i)\right) \tag{4.1}$$

  where $h = (b-a)/N$ and $x_i = a + (i-1)h$. *[Comment: Riemann's definition of an integral is more general and does not require equal spacing.]*

- Numerical quadrature: draw function on graph paper and count number of boxes or *quadrilaterals* lying below the curve of the function. (Anaxagoras of Clazomenae, 5th century BC)

- Numerical approximation of eqn. 4.1:

$$\int_a^b f(x)dx \simeq \sum_{i=1}^{N} f(x_i)w_i \tag{4.2}$$

  where w_i are appropriately chosen *weights*.

- In the general case this approximation is exact only in the limit $N \to \infty$.

  - Simple integration algorithms (section 4.2): choice of appropriate weights w_i for equally spaced points x_i.
  - Gaussian quadrature: use of orthogonal functions, spacing of evaluation points depends on integration range

4.2 Trapezoid, Simpson's and Boole's rule

Trapezoid rule

- $N+1$ points, N intervals of lenght $h = (b-a)/N$. Consider interval i and approximate function $f(x)$ by a straight line passing through points $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$.
- Thus $\int_{x_i}^{x_i+h} f(x)dx \simeq h\frac{(f_i+f_{i+1})}{2} = h/2f_i + h/2f_{i+1}$.
- For the entire integration region this results in the **composite trapezoid rule**

$$\int_a^b f(x)dx \simeq h/2f_1 + hf_2 + hf_3 + \cdots hf_N + h/2f_{N+1} \tag{4.3}$$

  where $f_{N+1} = f(x_{N+1}) = f(b)$. Thus the weights w_i are given by $w_i = \{h/2, h, \cdots, h, h/2\}$.

Approximation or algorithmic error

Express $f(x)$ as a Taylor expansion and then integrate term by term.

$$\begin{aligned} I_{exact} = \int_{x_i}^{x_i+h} \left(f_i + f_i'(x-x_i) + f_i''(x-x_i)^2/2 + f'''(x-x_i)^3/6\cdots\right) dx \\ = f_ih + f_i'h^2/2 + f_i''h^3/6 + \cdots \end{aligned}$$

Then compare this with the terms in the Trapezoid rule.

$$\begin{aligned} I_{trap} = h/2(f_i + f_{i+1}) = h/2(f_i + f_i + hf_i' + h^2/2f_i'' + h^3/6f_i''' + \cdots \\ = f_ih + f_i'h^2/2 + f_i''h^3/4 + \cdots \end{aligned}$$

Thus the leading term of the absolute error in this interval is given by $|I_{exact} - I_{trap}| = |-\frac{h^3}{12}f_i''|$.

The total error when integrating from a to b, i.e. for $N = (b-a)/h$ intervals, is thus $\propto h^2$.

Simpson's rule

- In interval i and $i+1$ approximate function $f(x)$ by a parabola through the points $(x_i, f(x_i))$, $(x_{i+1}, f(x_{i+1}))$ and $(x_{i+2}, f(x_{i+2}))$. This parabola may be obtained

using Lagrange interpolation. Thus we have

$$\int_{x_i}^{x_i+2h} f(x)dx \simeq \int_{x_i}^{x_i+2h} [\frac{(x-x_{i+1})(x-x_{i+2})}{(x_i-x_{i+1})(x_i-x_{i+2})} f_i + \frac{(x-x_i)(x-x_{i+2})}{(x_{i+1}-x_i)(x_{i+1}-x_{i+2})} f_{i+1} + \frac{(x-x_i)(x-x_{i+1})}{(x_{i+2}-x_i)(x_{i+2}-x_{i+1})} f_{i+2}]dx = \cdots$$
$$\cdots = h/3\left[f_i + 4f_{i+1} + f_{i+2}\right],$$

where $h = x_{i+1} - x_i$.

- The **composite Simpson rule** requires an even number N of intervals. It is given by

$$\int_a^b f(x)dx = \int_{x_1}^{x_{N+1}} f(x)dx \simeq$$
$$\simeq h/3\left(f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + \cdots + 4f_N + f_{N+1}\right).$$

  Thus the weights w_i are given by $w_i = \{h/3, 4h/3, 2h/3, 4h/3, \cdots, 4h/3, h/3\}$.

- Here the leading error term in an interval of width $2h$, computed in the same way as for the trapezoid rule, is given by $|-\frac{h^5}{90} f_i^{(4)}|$. Simpson's rule is thus more accurate than the Trapezoid rule; the total error over the entire range from a to b is $\propto h^4$.

Higher order rules

Using similar procedures higher order integration schemes can be worked out.

- Simpson's three-eights rule, based on cubic approximation of $f(x)$

$$\int_{x_1}^{x_4} f(x)dx = 3h/8(f_1 + 3f_2 + 3f_3 + f_4) + O(h^5) \tag{4.4}$$

- Boole's rule, based on quartic approximation of $f(x)$

$$\int_{x_1}^{x_5} f(x)dx = 2h/45(7f_1 + 32f_2 + 12f_3 + 32f_4 + 7f_5) + O(h^7) \tag{4.5}$$

4.3 Romberg Integration Scheme

- Romberg integration is a numerical scheme for integration to a high order of accuracy, limited only by numerical rounding errors. It is based on the systematic removal of integration errors using *Richardson extrapolation*.

- The integral of a function $f(x)$ may be conveniently expressed by the Euler-McLaurin Rule (derived for example in deVries, pp. 153-155).

$$\int_{x_1}^{x_{N+1}} f(x)dx = h\left(f_1/2 + f_2 + f_3 + \cdots + f_N + f_{N+1}/2\right) + \frac{h^2}{12}(f_1' - f_{N+1}') - \frac{h^4}{720}(f_1''' - f_{N+1}''') + \cdots \tag{4.6}$$

  Note that the first part is simply the composite trapezoid rule for N intervals of width h. Higher order terms in h involve evaluation of function derivatives only at the endpoints x_1 and x_{N+1}.

- Introduce new notation: $T_{m,0}$: result of integration obtained using *trapezoid rule* with $N = 2^m$ intervals for width h; leading error term is $O(h^2)$.

- $T_{m+1,0}$: trapezoid rule integration with $N = 2^{m+1}$ intervals of width $h/2 \Rightarrow$ leading error $O(h^2/4)$

- combine $T_{m,0}$ and $T_{m+1,0}$ to eliminate leading error term to give

$$T_{m+1,1} = \frac{4T_{m+1,0} - T_{m,0}}{3}.$$

  This corresponds to (composite) Simpson's rule!

- $T_{m+1,1}$ has error $O(h^4)$; halving interval $\Rightarrow$ $O(h^4/16)$ compute $T_{m+2,1}$; then eliminate leading error term in $O(h^4)$

$$T_{m+2,2} = \frac{16T_{m+2,1} - T_{m+1,1}}{15}.$$

  This corresponds to (composite) Boole's rule.

- **Romberg Integration Scheme**

$$T_{m+k,k} = \frac{4^k T_{m+k,k-1} - T_{m+k-1,k-1}}{4^k - 1}. \tag{4.7}$$

- computational scheme

$$
\begin{array}{lclcl}
T_{m,0} & & & & \\
 & \searrow & & & \\
T_{m+1,0} & \rightarrow & T_{m+1,1} & & \\
 & \searrow & & \searrow & \\
T_{m+2,0} & \rightarrow & T_{m+2,1} & \rightarrow & T_{m+2,2} \\
 & \searrow & & \searrow & \\
T_{m+3,0} & \rightarrow & \ldots & & \\
\ldots & & & &
\end{array}
$$

- in practice $k \leq 5$; use the evaluation points of $T_{m,0}$ for computation of $T_{m+1,0}$
- Romberg integration is based on evaluation of errors using derivatives *at the end-points* of the integration region, it will fail if these derivatives diverge.
- Example: $\int_{-1}^{+1} \sqrt{1-x^2}dx$
- convergence check should be included in a Romberg computational routine: evaluation of

$$R_m = \frac{T_{m-1,0} - T_{m,0}}{T_{m,0} - T_{m+1,0}} \tag{4.8}$$

  $R_m \simeq 4$ implies convergence
- the example above requires more evaluation points near $x = \pm 1$ and less in the middle; it can be integrated by change of variables $x = \cos\theta; dx = -sin\theta d\theta$
- $\int_{-1}^{+1} \sqrt{1-x^2}dx = \int_0^{\pi} \sin^2\theta d\theta$; Romberg integration is then possible (alternatively, and better here: analytical integration, $\int_0^{\pi} \sin^2\theta d\theta = [\theta/2 - \sin(2\theta)/4]_0^{\pi} = \pi/2$, i.e. the area of a semi-circle with radius 1.).

4.4 Improper Integrals

(Faires and Burden, Numerical Methods, Brooks/Cole Publishing Company 1998, 2nd)

- function unbounded somewhere in the interval of integration OR one or more integration endpoints at infinity

- here: treat case where integrand is unbounded at left endpoint, then show that all other cases can be reduced to this case
- singularity at the left boundary; from Analysis: $\int_a^b \frac{1}{(x-a)^p} dx$ converges if and only if $0 < p < 1$. $\int_a^b \frac{dx}{(x-a)^p} = \frac{(b-a)^{1-p}}{1-p}$
- if $f(x)$ is of the form $f(x) = \frac{g(x)}{(x-a)^p}$ with $0 < p < 1$ and $g(x)$ continuous in [a,b], then $\int_a^b f(x)dx$ also exists.

Numerical approximation using Composite Simpson rule

- approximate $g(x)$ by a polynomial, using the first few terms of the Taylor series of $g(x)$
- as an example let's use the first four terms of the series and call this polynomial $P_4(x)$;
- we thus have $g(x) \simeq P_4(x)$,
  with $P_4(x) = g(a) + g'(a)(x-a) + \frac{g''(a)}{2!}(x-a)^2 + \frac{g'''(a)}{3!}(x-a)^3 + \frac{g^{(4)}(a)}{4!}(x-a)^4$,
  or in shorter notation $P_4(x) = \sum_{k=0}^{4} \frac{g^{(k)}(a)}{k!}(x-a)^k$;
- write $\int_a^b f(x)dx = \int_a^b \frac{g(x)-P_4(x)}{(x-a)^p} dx + \int_a^b \frac{P_4(x)}{(x-a)^p} dx = I_1 + I_2$
- integrate I_2 analytically (this is generally the dominant term):
  $I_2 = \sum_{k=0}^{4} \int_a^b \frac{g^{(k)}(a)}{k!}(x-a)^{k-p} dx = \sum_{k=0}^{4} \frac{g^{(k)}(a)}{k!(k+1-p)}(b-a)^{k+1-p}$
- define
  $$G(x) = \begin{cases} \frac{g(x)-P_4(x)}{(x-a)^p} & \text{if } a < x \leq b \\ 0 & \text{if } x = a \end{cases}$$
  and integrate $G(x)$ numerically using Composite Simpson to give $I_1 = \int_a^b G(x)dx$;
  (Note that there is no problem with the endpoint $x = a$, since $\frac{g(x)-P_4(x)}{(x-a)^p}$ is of order $(x-a)^{5-p}$, i.e. $\to 0$ for $x \to a$)

Example: $\int_0^1 \frac{\exp(x)}{\sqrt{x}} dx$, i.e. $p = 1/2, a = 0$ and $g(x) = \exp(x)$

- $P_4(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$
- $I_2 = \int_0^1 \frac{P_4(x)}{\sqrt{x}} dx = ... = 2 + 2/3 + 1/5 + 1/21 + 1/108 \simeq 2.9235450$; this is the dominant term, see below

- $G(x)$ is given by

$$G(x) = \begin{cases} \frac{\exp(x) - P_4(x)}{\sqrt{x}} & \text{if } 0 < x \leq 1 \\ 0 & \text{if } x = 0 \end{cases}$$

- apply Composite Simpson's rule (5 evaluation points, $h = 0.25$): $I_1 = \int_0^1 G(x) \simeq \frac{0.25}{3}[0 + 4(0.0000170) + 2(0.0004013) + 4(0.0026026) + 0.0099485] = 0.0017691$
- hence $\int_0^1 \frac{\exp(x)}{\sqrt{x}} dx \simeq 2.9235450 + 0.0017691 = 2.9253141$
- *Mathematica*©:

```
In[1]:=Integrate[Exp[x]/Sqrt[x],x]
Out[1]:=Sqrt[Pi] Erfi[Sqrt[x]]
```

  with Erfi[x] = erf(x) $= \frac{2}{\sqrt{\Pi}} \int_0^x \exp^{-t^2} dt$, called *error function*

```
In[2]:= N[Integrate[Exp[x]/Sqrt[x], {x, 0, 1}],8]
Out[2]:= Sqrt[Pi] Erfi[1] = 2.9253035
```

- absolute error: 0.0000106
- **error analysis:** $E = -h^4 \frac{(b-a)}{180} G^{(4)}(\mu)$ (see error evaluation for Simpson's rule: $2\frac{h^5}{90} G^{(4)}(\mu)$; composite rule, thus $h = (b-a)/4$) here: $|G^{(4)}(x)| < 1$ on [0,1], thus $E = \frac{1-0}{180} 0.25^4 \times 1 = 0.0000217$ (upper bound!)

Other cases

- singularity at right endpoint, $x = b$: change of variables, $z = -x \Rightarrow \int_{-b}^{-a} f(z)dz$ has singularity at left (see above).
- singularity at c with $a < c < b$: write integral as sum of two improper integrals, to be integrated separately, $\int_a^b f(x)dx = \int_a^c f(x)dx + \int_c^b f(x)dx$
- convergent integral $I = \int_a^\infty \frac{1}{x^p} dx$ for $p > 1$. substitution: $t = x^{-1}$, thus $I = \int_0^{1/a} \frac{1}{t^{2-p}} dt$ with left-hand singularity
- generally, if $I = \int_a^\infty f(x)dx$ convergent, variable change leads to singularity at left, $I = \int_0^{1/a} t^{-2} f(t^{-1})dt$
- $\int_0^\infty f(x)dx = \int_0^1 f(x)dx + \int_1^\infty f(x)dx$, then proceed as above (case $a = 1$)

Chapter 5

Ordinary Differential Equations (ODE)

5.1 First order differential equations with initial values

$$\frac{d}{dt}x(t) = f(x,t) \tag{5.1}$$

- Aim is to compute $x(t)$ for given starting value, i.e. $x(t = t_0) = x(0) = x_0$ is known. (The value of the derivative may in general depend on both x and t.)
- ODE's are very important in physics, often they are *second order differential equations*, such as Newton's equation $m\frac{d^2x}{dt^2} = F(x, \frac{dx}{dt}, t)$. These can be written as a system of two first order ODE's (section 5.2).
- Taylor series expansion of x around t gives

$$x(t + \Delta t) = x(t) + \frac{dx}{dt}|_t \Delta t + \frac{1}{2}\frac{d^2x}{dt^2}|_t (\Delta t)^2 + \cdots \tag{5.2}$$

Euler methods

- crudest approximation: simply drop terms of order $(\Delta t)^2$ and higher

$$x_{i+1} = x_i + f(x_i, t_i)\Delta t \tag{5.3}$$

  with $t_i = i\Delta t$ and notation $x_i = x(t_i)$

  (see sketch in lecture!)

- define *local* order by the order in Δt to which approximation agrees with exact solution, so simple Euler method is locally a first order approximation
- aim: compute x at $t = b$, starting from $t = 0$; requires $N = b/\Delta t$ steps; since error is quadratic in each step the total error will be $N\Delta t^2 \propto \Delta t$
- convention: simple Euler method is an approximation of *zeroth* order *globally*
- smaller Δt gives smaller errors, but leads to longer run time and eventually also to round-off errors
- retain higher order derivatives?, i.e.

$$x_{i+1} \simeq x_i + f_i \Delta t + \frac{1}{2}\left[f_i \frac{\partial f}{\partial x}|_i + \frac{\partial f}{\partial t}|_i\right](\Delta t)^2, \tag{5.4}$$

  where we have used $\frac{d^2x}{dt^2} = \frac{d}{dt}f(x,t) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial t}\frac{dt}{dt} = f\frac{\partial f}{\partial x} + \frac{\partial f}{\partial t}$. This approximation is locally of second order, globally of first order BUT requires partial derivatives of $f(x,t)$: **not practical**. However, eqn.(5.4) is useful to evaluate approximation error in other integration schemes

Alternative view: integration of eqn.(5.1)

$$x_{i+1} = x_i + \int_{t_i}^{t_{i+1}} f(x,t)dt \tag{5.5}$$

- problem: unknown x is also in the integrand!
- aim is thus to *approximate* this integral
- mean value theorem: there is a value t_m in interval $[t_i, t_{i+1}]$ so that eqn.(5.5) can be re-written as

$$x_{i+1} = x_i + f(x_m, t_m)\Delta t = x_i + \frac{dx}{dt}|_{t_m}\Delta t. \tag{5.6}$$

  (however, t_m is not known)

- The simple Euler rule is produced by replacing t_m with t_i, and thus x_m with x_i.
- Using the *midpoint rule* for the integral evaluation results in the **modified Euler method**:

$$x_{i+1} = x_i + f(x_{mid}, t_i + \frac{\Delta t}{2})\Delta t \tag{5.7}$$

  where we *approximate* $x_{mid} \simeq x_i + f(x_i, t_i)\frac{\Delta t}{2}$.

- the **improved Euler method**, is based on using the trapezoid rule for the integral evaluation, i.e. an evaluation of $f(x,t)$ at the beginning (x_i, t_i) and at an *approximation* of the end point $(x_i + f(x_i,t_i)\Delta t, t_{i+1})$ of the integration region.

$$x_{i+1} = x_i + \left(f(x_i,t_i) + f(x_i + f(x_i,t_i)\Delta t, t_{i+1})\right)\frac{\Delta t}{2} \tag{5.8}$$

- Modified and improved Euler method are also called **second-order Runge-Kutta approximations**, locally of second order, and globally of first order in the sense defined above. (Proof by comparison with Taylor expansion of $x(t_i+\Delta t)$.)

  see sketches

- **fourth-order Runge-Kutta method**

$$x_{i+1} = x_i + \frac{\Delta t}{6}\left(f(x_1', t_1') + 2f(x_2', t_2') + 2f(x_3', t_3') + f(x_4', t_4')\right) \tag{5.9}$$

  where

$$x_1' = x_i, \qquad t_1' = t_i \tag{5.10}$$
$$x_2' = x_i + \frac{1}{2}f(x_1', t_1')\Delta t \qquad t_2' = t_i + \frac{\Delta t}{2} \tag{5.11}$$
$$x_3' = x_i + \frac{1}{2}f(x_2', t_2')\Delta t \qquad t_3' = t_i + \frac{\Delta t}{2} \tag{5.12}$$
$$x_4' = x_i + f(x_3', t_3')\Delta t \qquad t_4' = t_i + \Delta t \tag{5.13}$$

- this is a method which is very commonly used

- for special case of $f = f(t)$ only, it can be derived by evaluation of eqn.5.5 using Simpson's rule

5.2 Second order ODEs

$$\frac{d^2y}{dx^2} = y'' = f(x, y, y') \tag{5.14}$$

- introduce two new variables:

$$y_1 := y \tag{5.15}$$
$$y_2 := y' = \frac{dy}{dx} \tag{5.16}$$

thus eqn. 5.14 is reduced to a set of two first order ODEs

$$\frac{dy_1}{dx} = y_1' = y_2 \tag{5.17}$$

$$\frac{dy_2}{dx} = y_2' = f(x, y_1, y_2) \tag{5.18}$$

write as vectors and use methods for 1st order ODEs for the two components

- set $f_1 = y_2$; $f_2 = f(x, y_1, y_2)$.

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ f(x, y_1, y_2) \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \vec{f}$$

writing

$$\vec{\hat{y}} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

we thus want to solve $\vec{\hat{y}}' = \vec{f}$.

example: object falling from a height

(here, for illustrative purposes: use simple Euler)

- governing equation (Newton):

$$m\ddot{y} = -gm + \gamma\dot{y} \tag{5.19}$$

with γ coefficient of drag. Rewrite as $\ddot{y} = -g + \frac{\gamma}{m}\dot{y}$ and set:

$$\begin{aligned} \dot{y} &= v_y = f_1 \\ \dot{v_y} &= -g + \frac{\gamma}{m} v_y = f_2 \end{aligned} \tag{5.20}$$

- simple Euler:

$$v_y(t_{n+1}) = v_y(t_n) + \dot{v_y}(t_n)\Delta t = v_y(t_n) + (\frac{\gamma}{m} v_y(t_n) - g)\Delta t \tag{5.21}$$

$$y(t_{n+1}) = y(t_n) + \dot{y}(t_n)\Delta t = y(t_n) + v_y(t_n)\Delta t \tag{5.22}$$

- procedure: move along in pairs $(v_y(t_0), y(t_0)) \to (v_y(t_1), y(t_1)) \to (v_y(t_2), y(t_2))$...

5.3 Constants of motion

- some quantities might be conserved for physical reasons
- Example: mass attached to a spring in frictionless horizontal motion ("air table")
$$\ddot{x} = \frac{dv}{dt} = a = F/m = -\frac{kx}{m} \tag{5.23}$$
with velocity v, force F and spring constant k
- two first order ODEs:
$$\frac{dv}{dt} = -\frac{k}{m}x$$
$$\frac{dx}{dt} = v$$
simple Euler rule for time-step Δt:
$$v(t_0 + \Delta t) = v(t_0) + \Delta t \frac{dv}{dt}|_{t_0} = v_0 - \frac{k}{m}x_0 \Delta t$$
$$x(t_0 + \Delta t) = x(t_0) + \Delta t \frac{dx}{dt}|_{t_0} = x(t_0) + v(t_0)\Delta t$$
- total energy of the system, given by $E = \frac{1}{2}mv^2 + \frac{k}{2}x^2$, is a constant, **BUT** is found to increase proportional to $(\Delta t)^2$ according to the Euler scheme. Better: improved Euler method

application: construct of energy conserving algorithm

- use simple Euler for position: $x(t_0 + \Delta t) = x(t_0) + v(t_0)\Delta t$
- determine **magnitude** of velocity from energy conservation $E = \frac{1}{2}mv^2 + \frac{k}{2}x^2$
- get **sign** of v from simple Euler for velocity: $v(t_0 + \Delta t) = v(t_0) - \frac{k}{m}x_0 \Delta t$

5.4 Adaptive step sizes

- what is the optimal choice of step size Δh? too large $\rightarrow$ introduce approximation errors; too small $\rightarrow$ time consuming + increased importance of rounding errors
- idea: check for accuracy of computed solution (over entire integration range) by carrying out numerical integration using two different step sizes, e.g. $\Delta h, 2\Delta h$. Take the difference in the numerical results as an *estimate* of the error made in the integration; if error larger than tolerance, reduce step size.

use of the two integration results to reduce error

using Runge-Kutta method, estimate error *at every step* by performing in parallel *two* steps of width h and *one* step of $2h$:

$$\begin{aligned} y(x+2h) &= y_{2X1} + 2h^5\Phi + O(h^6) \\ y(x+2h) &= y_{1X2} + (2h)^5\Phi + O(h^6) \end{aligned} \tag{5.24}$$

here $y(x+2h)$ denotes the unknown *exact* solution, and y_{2X1} and y_{1X2} the numerical approximations, Φ is proportional to $\frac{d^5y}{dx^5}$

Richardson interpolation can improve on the approximation by eliminating the error proportional to h^5:

$$y_{improved} = \frac{16y_{2X1} - y_{1X2}}{15}$$

determine required step width h_0 to achieve the acceptable error Δ_0 by proceeding as follows:

- compute, as a measure of accuracy, the difference between the approximations, $\Delta = y_{1X2} - y_{2X1} \propto h^5$
- scaling for two different step widths h_0 and h_1 thus given by

$$h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{1/5} \tag{5.25}$$

- since Δ_1 and Δ_0 are known, h_0 can be computed
- this can be done automatically during the integration of the ODE; the step size is thus not kept constant

Chapter 6

Random numbers and Monte Carlo Integration

The generation of random numbers is too important to be left to chance.
Robert R. Coveyou (1969)

- used for simulations of random processes (e.g. thermal motion, radioactive decay) and for Monte Carlo integration.
- random number generators are based on "deterministic randomness", i.e. some deterministic iterative algorithm that produces a series of numbers that appear random (see also deterministic chaos).
- call a sequence of numbers $r_1, r_2, \cdots$ *random* if there are *no correlations* in it (see section 6.2.

probability density function $p(x)$: the probability of finding r in interval $[x, x + dx]$ is given by $p(x)dx$.

If all numbers in a sequence occur equally likely the sequence is called **uniform**. In this case the probability density is a constant.

example: uniform density in interval $[0 : 1]$

$$p(x)dx = \begin{cases} dx & \text{for } 0 < x < 1 \\ 0 & \text{else} \end{cases} \tag{6.1}$$

normalisation:

$$\int_{-\infty}^{+\infty} p(x)dx = 1 \tag{6.2}$$

Tables of *real* random numbers, obtained using experimental data (e.g. radioactive decay, or some electronic devices) exist, but also these may contain bias.

6.1 Generation of (pseudo) random numbers

Linear congruent or power residue method

obtain sequence $\{r_1, r_2, r_3, \cdots, r_k\}$ over interval $[0, M-1]$ using the following *iterative* procedure:

$$r_i := (ar_{i-1} + c)\text{mod}M = \text{remainder}\left(\frac{ar_{i-1} + c}{M}\right), \tag{6.3}$$

where a, c, M are integer constants; M is the **length** of the **sequence** and r_1 is called the **seed**.

- trivial example: $c = 1, a = 4, M = 9; r_1 = 3$, leading to $r_{2-10} = 4, 8, 6, 7, 2, 0, 1, 5, 3$.
- The choice of a, c, M is crucial!
- www.gnu.org: a = 1103515245, c = 12345 and M = 2^{31}
- www.gnu.org: defined on 48-bit unsigned integers: a = 25214903917, c = 11 and m = 2^{48} (not straightforward to program)

  [int 16 bits (32); longint: 32 bits; float 32 bits; double 64 bits]

Multiplicative congruential algorithm

A good set is ($a = 7^5, c = 0, M = 2^{31} - 1$), i.e.

$$r_i := (ar_{i-1})\text{mod}M \tag{6.4}$$

Comments

- $0 \leq r_i \leq M - 1$; $(a, c > 0)$
- in order to obtain random number x_i with $0 \leq x_i \leq 1$ divide r_i by $(M - 1)$.
- starting from the same seed gives the same sequence; seed can be chosen as the current time, taken from the computer that is used to perform the calculation

- **DON'T write your own random number generators but use PUBLISHED EXAMPLES**

6.2 Tests of Random number generators

visual test: correlations lead to pattern

plot x_{n+i} versus x_n where i is kept fixed (correlation between n^{th} and $(n+i)^{th}$ random number

evaluate k^{th} moment

$$\langle x^k \rangle = \int_0^1 x^k p(x) dx \simeq \frac{1}{N} \sum_{i=1}^{N} {x_i}^k + O(N^{-1/2}) \tag{6.5}$$

for $p(x)$ uniform (eqn.(6.1):

$$\langle x^k \rangle = \int_0^1 x^k dx = \frac{1}{k+1} \tag{6.6}$$

thus

$$\frac{1}{N} \sum_{i=1}^{N} {x_i}^k \simeq \frac{1}{k+1} + O(N^{-1/2}), \tag{6.7}$$

here the term $O(N^{-1/2})$ indicates randomness (see section 6.4) and the term $\frac{1}{k+1}$ indicates uniformity

near-neighbour correlations

$$C(k) = \frac{1}{N} \sum_{i=1}^{N} x_i x_{i+k} \tag{6.8}$$

where $k = 1, 2, \cdots$

$$C(k) \simeq \int_0^1 dx \int_0^1 dy \, x \, y P(x, y) \tag{6.9}$$

with *joint probability density* $P(x, y)$

for random numbers that are independent and uniformly distributed: $P(x, y) = p(x)p(y) = 1$, thus $C(k) = 1/4$ for all k.

further tests

- run simulation with $r_1, r_2, r_3, \cdots$ then with $1 - r_1, 1 - r_2, 1 - r_3, \cdots$; both sequences are random $\rightarrow$ results should not differ beyond statistics
- try several different random number generators: do they lead to consistent results?

6.3 Creating non-uniform distributions of random numbers

- Aim: Based on a *uniform* distribution $p(x)$ of random numbers (values between 0 and 1) create random numbers that are distributed with a user defined *non-uniform* distribution $q(y)$
- Fundamental transformation law of probabilities:

$$|q(y)dy| = |p(x)dx| \tag{6.10}$$

  or

$$q(y) = p(x)\left|\frac{dx}{dy}\right|.$$

  Using property of uniformity of $p(x)$, eqn.(6.1), we obtain

$$q(y)dy = \left|\frac{dx}{dy}\right| dy \tag{6.11}$$

  Based on this find the mapping between x and y.

$$x = \int_{-\infty}^{y} q(y')dy := Q(y) \tag{6.12}$$

- Thus $y = Q^{-1}(x)$. But this inverse can only be computed analytically for a few special cases!

  (see figure in lecture for an illustration of the above equation)

- Example:

$$q(y) = \begin{cases} 1/\lambda \exp(-y/\lambda) & \text{for } y > 0 \\ 0 & \text{for } y < 0 \end{cases} \tag{6.13}$$

$$x = Q(y) = \int_{-\infty}^{y} q(y')dy = 1/\lambda \int_{0}^{y} exp(-y/\lambda)dy' = -\exp(-y'/\lambda)|_0^y = 1-\exp(-y/\lambda)$$

Thus $-y/\lambda = \ln(1-x) \rightarrow y = -\lambda \ln(1-x)$

x are uniformly distributed, y are exponentially distributed between 0 and ∞

Gaussian (Normal) distribution

- aim: create random numbers that follow a Gaussian distribution

$$q(y)dy = \frac{1}{\sqrt{2\pi}} \exp(-y^2/2)dy \tag{6.14}$$

- here: need to consider transformation in 2 dimensions

$$q(y_1, y_2)dy_1dy_2 = p(x_1, x_2) \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} dy_1dy_2 \tag{6.15}$$

  where $|\cdots|$ is the determinant of the Jacobian matrix.

- Ansatz:

$$\begin{aligned} y_1 &= \sqrt{-2\ln x_1}\cos(2\pi x_2) \\ y_2 &= \sqrt{-2\ln x_1}\sin(2\pi x_2). \end{aligned} \tag{6.16}$$

- Let's now show that these y_1, y_2 are indeed Gaussian distributed. Re-write in terms of x_1, x_2,

$$\begin{aligned} x_1 &= \exp[-\frac{1}{2}(y_1^2 + y_2^2)] \\ x_2 &= \frac{1}{2\pi}\arctan\frac{y_2}{y_1}. \end{aligned} \tag{6.17}$$

- The (absolute value of the) determinant of the Jacobian matrix is given by

$$\begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = \left(\frac{1}{\sqrt{2\pi}}\exp(-y_1^2/2)\right)\left(\frac{1}{\sqrt{2\pi}}\exp(-y_2^2/2)\right)$$

- thus each y_i is independently distributed according to a normal distribution

- implementation of the computation of y_1, y_2 in eqn (6.16) (from Numerical Recipes in C, Press *et al.*):

  choose point (v_1, v_2) in unit circle around zero by picking random numbers v_1 and v_2 in interval [-1:+1] and accepting choice only if $R^2 = v_1^2 + v_2^2 \leq 1$;

  use $R^2 =: x_1$ as a random number, it is uniformly distributed in interval $[0, 1]$;

  angle θ that (v_1, v_2) defines with respect to v_1 axis serves as random angle $2\pi x_2$, i.e. $\theta := 2\pi x_2$;

  sin and cosine terms required for evaluation of y_1 and y_2 in Eqn(6.16) are now given by $\sin 2\pi x_2 = \frac{v_2}{\sqrt{R^2}}$ and $\cos 2\pi x_2 = \frac{v_1}{\sqrt{R^2}}$;

  this finally leads to

$$y_1 = \sqrt{-2 \ln R^2} \frac{v_1}{\sqrt{R^2}} \tag{6.18}$$

$$y_2 = \sqrt{-2 \ln R^2} \frac{v_2}{\sqrt{R^2}} \tag{6.19}$$

  Note that there is thus no need to compute sine and cos functions to obtain the Gaussian distributed random numbers y_1, y_2 (eqn.(6.16)), a choice of a point (v_1, v_2) in a unit circle is sufficient.

Von-Neumann Rejection

method to create random numbers x_i that are distributed according to defined distribution $W(x)$ (see also figure in lecture)

- choose $W_0 \geq \max W(x)$ (W_0 serves as a scale factor)
- choose points $(x_i, W_i) := (r_{2i-1}, W_0 r_{2i})$ where r_i are random numbers from a uniform distribution
- if $W_i < W(x_i)$ accept x_i
- if $W_i > W(x_i)$ reject x_i
- the accepted values of x_i will be distributed with $W(x)$.

6.4 Monte Carlo Integration

based on mean value theorem:

$$I = \int_a^b f(x)dx = (b-a)\Big\langle f \Big\rangle \tag{6.20}$$

MC integration is based on evaluating the mean $\Big\langle f \Big\rangle$ by use of a sequence of N random numbers x_i, distributed uniformly in $[a, b]$. The function $f(x)$ is *sampled* at these points.

$$\Big\langle f \Big\rangle \simeq \frac{1}{N}\sum_{i=1}^{N} f(x_i) =: \Big\langle f \Big\rangle_N \tag{6.21}$$

thus

$$\int_a^b f(x)dx \simeq (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{6.22}$$

Example computation of known integrals, such as $\int_0^1 \exp(x)dx$ show that convergence is very slow. In order to see the merits of MC integration let's have a closer look at the origin of this.

6.4.1 The statistics of Monte Carlo integration

Consider computation of integral $\int_a^b f(x)dx$.

Use of m different sets of random numbers $x_1, ..., x_N, x_{N+1}, ..., x_{2N}, ..., x_{(m-1)N} ... x_{mN}$ results in m different values I_m (distribution of I_m).

Central Limit Theorem: for large values of m distribution converges to a Gaussian of width $\propto \frac{1}{\sqrt{N}}$

Error in Monte-Carlo integration: standard deviation from the mean, here *estimated* from the sampled points.

$$\sigma_m \simeq \frac{\sigma}{\sqrt{N}}, \tag{6.23}$$

where σ is the *standard deviation of the function* $f(x)$ to be integrated. The standard deviation is *estimated* as

$$\sigma \simeq \sqrt{\langle f^2 \rangle_N - \langle f \rangle^2}_N \text{ with } \langle f \rangle_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i) \text{ and } \langle f^2 \rangle_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i)^2. \tag{6.24}$$

Thus

$$\int_a^b f(x)dx \approx (b-a)(\left\langle f \right\rangle_N \pm \sigma_m) \text{with 68.3 \% confidence} \tag{6.25}$$

$\pm 2\sigma_m$ with 95.4% confidence

$\pm 3\sigma_m$ with 99.7% confidence

Increasing N reduces standard deviation, but only $\propto \frac{1}{\sqrt{N}}$, why use MC integration at all?

- MC integration always gives *estimates* of integrals, where other methods fail
- advantage of MC integration for multi-dimensional integrals

example: multi-dimensional integration

- consider use of trapezoid rule for integration (N evaluation points, interval width h): error due to inadequacy of linear approximation of function $f(x)$ to be integrated; error $E \propto h^2$

  decrease h by $\sqrt{2}$ (increase N by $\sqrt{2}$), error $\rightarrow E/2$

  in 2d-integration: need to increase N by 2

  in d-dimensions: increase N by $2^{d/2}$

- error in MC Integration is of stochastic/probabilistic nature

  1d: N function evalulations

  d-dimensions: N function evaluations (but d random coordinates are now required for each point)

  convergence slow but independent of dimensionality of integral

example: aim is to reduce error by 2
d=4, trapezoid rule, $N \rightarrow 4N$ MC: $N \rightarrow 4N$
d=5, trapezoid $N \rightarrow 5.65N$ MC $N \rightarrow 4N$
thus MC converges faster than trapezoid rule for $d > 4$

6.4.2 Importance Sampling

Enhance accuracy of MC integration by using a mathematical transformation that decreases the variance σ^2 of the function to be integrated.

example: say $g(x) \simeq f(x)$ and $g(x)$ can be integrated analytically

$$\int_a^b f(x)dx = \int_a^b \frac{f(x)}{g(x)} g(x)dx = \int_{y(a)}^{y(b)} \frac{f(y)}{g(y)} dy, \tag{6.26}$$

where we have used a variable transformation via $dy = g(x)dx$, i.e. $y(x) = \int_0^x g(x)dx$. y can be integrated analytically.

We then sample y uniformly and integrate $f(y)/g(y)$ using MC integration.

Note that $f(y)/g(y) \simeq 1$, and thus has smaller variance than $f(x)$

example:

$$I = \int_0^1 \exp(x)dx = \int_0^1 \frac{\exp(x)}{1+x}(1+x)dx \tag{6.27}$$

where (1+x) are the first two terms of a Taylor expansion

thus

$$y = \int_0^x (1+x)dx = x + x^2/2 \tag{6.28}$$

$x_{1,2} = -1 \pm \sqrt{1+2y}$. Since $x, y > 0$ use positive sign.

finally:

$$I = \int_0^{3/2} \frac{\exp(\sqrt{1+2y}-1)}{\sqrt{1+2y}} dy \tag{6.29}$$

which will be sampled uniformly from 0 to 3/2

alternative interpretation:

$$I = \int_a^b \frac{f(x)}{g(x)} g(x)dx = \left\langle f/g \right\rangle_g \simeq \frac{1}{N} \sum_{x_i, non-uniform} \frac{f(x_i)}{g(x_i)} \tag{6.30}$$

where $g(x)$ acts as a **weighting function** or probability distribution: random numbers x_i are no longer distributed uniformly, but they are distributed according to $g(x)$

Chapter 7

Monte Carlo Simulations

7.1 Canonical ensemble, Boltzmann distribution, partition sum

In statistical physics one often considers the **canonical ensemble**. It is represented by a system (i.e. a box containing a gas) in a heat bath.

The system is characterised by its *micro-states* s (i.e. specified by position and velocities of the gas molecules) with their corresponding energy values E_s. The system takes on the temperature T of the heat bath (environment).

In thermal equillibrium the probability of finding a microstate with energy E_s is

$$p_s = \frac{1}{Z} \exp(-\beta E_s) \tag{7.1}$$

with $\beta = 1/(k_B T)$ and $\exp(-\beta E_s)$ is called the Boltzmann factor.

Z is a normalisation. In the case of discrete micro-states (as in the examples in this chapter) it is given by

$$Z = \sum_{s=1}^{\Omega} e^{-\beta E_s} \tag{7.2}$$

and is called **partition sum** (in equilibrium there is a partition $p(s)$ into possible states) or **Zustandssumme** (sum over all Ω accessible microstates of the system, **here: classical discrete system**). The temperature T is measured in Kelvin, the **Boltzmann constant** is $k_B = 1.38 \times 10^{-23} J/K$

Some important relationships

mean energy given by:

$$\left\langle E \right\rangle = \sum_s E_s p_s = \frac{1}{Z} \sum_s E_s e^{-\beta E_s} \tag{7.3}$$

$$\frac{\partial \ln Z}{\partial \beta} = \frac{1}{Z} \sum_s (-E_s) e^{-\beta E_s} = -\left\langle E \right\rangle \tag{7.4}$$

The **variance** in the energy is given by:

$$\left\langle (E - \langle E \rangle)^2 \right\rangle = \left\langle E^2 \right\rangle - \left\langle E \right\rangle^2 = \frac{\partial^2 \ln Z}{\partial \beta^2} \tag{7.5}$$

heat capacity at constant volume C_V:

$$C_V = \frac{\partial \left\langle E \right\rangle}{\partial T}|_{V,N} = \frac{1}{k_B T^2} \left(\left\langle E^2 \right\rangle - \left\langle E \right\rangle^2 \right) \tag{7.6}$$

magnetic systems: magnetisation M, field H, susceptibility χ

$$\chi = \lim_{H \to 0} \frac{\partial \left\langle M \right\rangle}{\partial H} \tag{7.7}$$

find

$$\chi = \frac{1}{k_B T} \left(\left\langle M^2 \right\rangle - \left\langle M \right\rangle^2 \right) \tag{7.8}$$

Thus C_V and χ are determined by **fluctuations**.

7.2 Metropolis algorithm

How do we compute these averages? There are too many microstates to compute them all, which ones should one choose?

Consider the partition sum

$$Z = \sum_s e^{-\beta E_s} \tag{7.9}$$

where the sum is over all accessible microstates.

example: 100 hours computing time = $3.6 \times 10^5 s$

10^{-6} s per computation (10^6 flops = 1Mflop; FLOP: FLoating point Operations Per Second) $\Rightarrow 3.6 \times 10^{11}$ computations available;

consider a spin system with N spins: 2^N combinations/states;

computation of all possible values of energy E_s requires $2N \times 2^N$ steps (see eqn.(7.12);

thus we can only compute a (very small) system of $N = 32$ (in practice: $3 \times 3 \times 3$);

for N=64 we need 10^{21} flops = 2 million petaflops;

- 2007: IBM's top supercomputer Blue Gene/P: up to three peta-flops
- 2013: Tianhe-2 (Milky Way), 34-55 peta-flops
- 2016: Sunway TaihuLight, 93 peta-flops
- 2018: IBM SUMMIT, 143.5 peta-flops
- 202: Frontier, Oak Ridge National lab USA, 1194 peta-flop0
- real system: $N = 10^7 \times 10^7 \times 10^7$;

$\Rightarrow$ **sample** over a **finite** set of states;

random choice? **no**, physically important states might not be picked (randomly picked states might be very unlikely states)

Idea of Metropolis N, Rosenbluth A, Rosenbluth M, Teller A and Teller E (1953), "Equations of State Calculations by Fast Computing Machines" *J. Chem. Phys.* **21** 1087-1092:

Develop a procedure by which a sequence of states (configurations) can be generated in a way that they relax into thermal equilibrium (and thus are governed by eqn. (7.1)**).**

Metropolis algorithm:

start with state 1 (of energy E_1), choose state 2 (of energy E_2) randomly,

for energy $E_2 < E_1$ accept new state, $1 \Rightarrow 2$, else ($E_2 \geq E_1$) accept it with probability $w = \exp(\frac{E_1 - E_2}{k_B T})$ ($0 < w \leq 1$)

[*procedure:* compute random number r, $0 \leq r \leq 1$ and accept state 2 if $r \leq w$, else retain previous state and choose a different random state 2]

Expressed in terms of **transition rate** (transition probability per time-step), $W(s_1 \to s_2)$:

$$W(s_1 \to s_2) = \begin{cases} 1 & E_2 < E_1 \\ e^{\frac{(E_1 - E_2)}{k_B T}} & E_2 \geq E_1. \end{cases}$$

Why does this transition rule model thermal equilibrium?

Say $E_1 > E_2$:
Metropolis algorithm gives $W(s_1 \to s_2) = 1$ and $W(s_2 \to s_1) = \exp \frac{(E_2 - E_1)}{k_B T}$.

In thermal equilibrium the probability of finding system in any particular state will (on average) be independent of time, i.e. the number of transitions from s_1 to s_2 must equal the number of transitions from s_2 to s_1:

"Detailed Balance"

$$p_1 W(s_1 \to s_2) = p_2 W(s_2 \to s_1), \tag{7.10}$$

where p_1 is the probability of being in state s_1 and p_2 is the probability of being in state s_2. (detailed balance: this condition holds for all states i and j)

Thus

$$\frac{p_1}{p_2} = \exp\left(\frac{E_2 - E_1}{k_B T}\right) = \frac{\exp(\frac{-E_1}{kT})}{\exp(\frac{-E_2}{k_B T})}, \tag{7.11}$$

in agreement with thermal equillibrium, eqn. (7.1).

This stochastic process leads to a sequence of physically relevant states which can be used to compute mean values of some physical quantity. [1]

7.3 Modelling the Ising Ferromagnet

Ising model: simple model of ferro-magnetism; it shows a (2nd order) **phase transition** (for two and three dimensions) at a critical temperature T_c: when cooled down from $T > T_c$, net magnetisation M goes from zero to a maximum value M_0

examples of phase transitions: fluid-solid (first order), super-conductivity (second order)

[1]For ongoing mathematical research concerning the convergence of the Metropolis algorithm see Diaconis, P. and Saloff-Coste, L. (1998). What do we know about the Metropolis Algorithm? *Jour. Computer and System Sciences* **57** 20-35.

questions:

- how do we simulate thermal equilibrium?
- how slowly does one need to cool?
- phase transitions in finite system?

7.3.1 The model

square lattice, lattice sites $i = 1, \cdots, N$ with lattice variables $S_i \in \{+1, -1\}$ corresponding to permanent magnetic spins up/down

micro-state given by $\vec{S} = (S_1, S_2, \cdots, S_N)$
total energy H of state $\vec{S}$:

$$H(\vec{S}) = -J \sum_{(i,j)_{nn}} S_i S_j - h \sum_i S_i \tag{7.12}$$

where J is the **exchange energy**, h is an externally applied magnetic field, $(i,j)_{nn}$ indicates that the sum is over all nearest neighbours j of spin S_i.
ferromagnet: $J > 0$
anti-ferromagnet $J < 0$

7.3.2 Implementation of the Metropolis algorithm

we need to compute $H(S) - H(S')$ where S is the old configuration, and S' is a new configuration $\Rightarrow \exp(\Delta H / k_b T)$. Let's rewrite eqn.(7.12):

$$H(S) = -JS_i \sum_{j \in N(i)} S_j - hS_i + \text{rest} \tag{7.13}$$

where the rest is everything in eqn.(7.12) that does not involve the spin S_i; $N(i)$ denotes all neighbours of spin i
$\vec{S'}$ is identical to $\vec{S}$ apart from $S'_i = -S_i$, so is also has the same rest!
thus:

$$\Delta H = H(S) - H(S') = -2S_i(J \sum_{j \in N(i)} S_j + h) = -2S_i h_i \tag{7.14}$$

where we have defined the **inner field** h_i **of spin** S_i as

$$h_i = J \sum_{j \in N(i)} S_j + h \tag{7.15}$$

finally we get for the **Metropolis algorithm**:

1. choose starting configuration $\vec{S(0)} = (S_1, S_2, \cdots, S_N)$
2. chose randomly/sequentially index i and compute $\Delta H = -2S_i h_i$
3. for $\Delta H \geq 0$ flip spin: $S_i \Rightarrow -S_i$
   for $\Delta H < 0$ pick random $r \in [0, 1]$; if $r < exp(\Delta H / k_B T)$: $S_i \Rightarrow -S_i$, else keep the old configuration
4. iterate steps 2 and 3

Use **periodic boundary conditions** (lattice size L). This may be done copying spins at the sides: $S[i][L] \Rightarrow S[i][0]$, $S[i][1] \Rightarrow S[i][L+1]$, $S[1][i] \Rightarrow S[L+1][i]$, $S[L][i] \Rightarrow S[0][i]$.

(See simulation shown in lecture.)

Of interest is the macroscopic magnetisation M and its dependence on temperature T. $M(T)$ is given by

$$M(T) = \sum_i \langle S_i \rangle \tag{7.16}$$

where the sum is over the total number of spins N and $\langle ... \rangle$ denotes a (time/ensemble) average.

results:

- at high $T > T_c$ the net magnetisation M fluctuates around zero
- clusters of parallel spins appear at $T \simeq T_c$ resulting in a non-zero net magnetisation
- quick cooling from $T = 1.5T_c$ to $0.4T_c$ leads to formation of domains of positive and negative magnetisation, domain walls diffuse and interact, finally resulting in one single domain

7.3.3 Properties of the Ising model for an infinite square lattice

mean magnetisation per spin: $m(T) = \lim_{N\to\infty} \frac{\left\langle M(T)\right\rangle}{N}$

2nd order phase transition (i.e. continuous in entropy) at $T = T_c$

near T_c : $m(T) \propto (T_c - T)^\beta$, β is called **critical exponent**

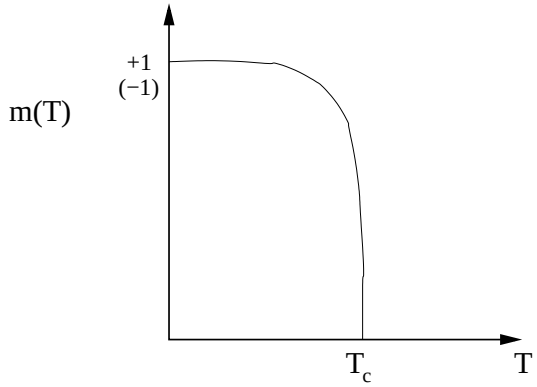

Figure 7.1: Temperature dependence of the mean magnetisation per spin.

susceptibility: $\chi = \frac{1}{k_B T}\left(\left\langle M^2\right\rangle - \left\langle M\right\rangle^2\right)$

$$\text{near } T_c: \ \chi \propto |T - T_c|^{-\gamma} \tag{7.17}$$

heat capacity: $C_V = \frac{1}{k_B T^2}\left(\left\langle E^2\right\rangle - \left\langle E\right\rangle^2\right)$

$$\text{near } T_c: \ C \propto |T - T_c|^{-\alpha} \tag{7.18}$$

correlation length: linear dimension $\xi(T)$ of a typical magnetic domain

$$\text{near } T_c: \ \xi \propto |T - T_c|^{-\nu} \tag{7.19}$$

Onsager 1944: $T_c = \frac{J}{k_B}\frac{2}{\ln(\sqrt{2}+1)} \simeq 2.269\frac{J}{k_B}$

$\beta = 1/8, \ \nu = 1, \ \gamma = 7/4, \ \alpha = 0$

($\alpha = 0$, there is a divergence, but weaker than a power law (logarithmic))

Simulations will suffer from **finite size effects** which makes it harder to determine the critical exponents.

key problem: what happens for $\xi \simeq L$?

7.4 Finite size scaling

Fig. 17.2 of Gould/Tobochnik shows results of simulations for lattice sizes L=8 and L=16 and the theoretical curve for the specific heat of the Ising ferromagnet.

Deviations from the scaling behavior for infinite system occur at $\xi(T) \simeq L$. As $\xi(T) \propto |T - T_c|^{-\nu}$ it follows

$$|T - T_c| \propto L^{-1/\nu} \tag{7.20}$$

Using $C(T) \propto |T - T_c|^{-\alpha}$ one obtains

$$C(T_c) \propto L^{\alpha/\nu} \tag{7.21}$$

Similarly:

$$m(T_c) \propto L^{\beta/\nu} \tag{7.22}$$

$$\chi(T_c) \propto L^{\gamma/\nu} \tag{7.23}$$

Repeat calculations for various lattice sizes L and plot $\ln C$ as a function of L at $T = T_c$. Note that in general it might be hard to determine T_c (Ising Model: use the Onsager result).

7.5 The q-state Potts model

model similar to Ising model, one (constant) coupling constant J but now **several possible states per lattice site**.

Hamiltonian:

$$H = -J \sum_{i,j=nn(i)} \delta_{S_i,S_j} \tag{7.24}$$

S_i at site i can have the values $1, 2, \cdots, q$.

$$\delta_{a,b} = \begin{cases} 1 & \text{for } a = b \\ 0 & \text{else} \end{cases}$$

$q = 2$: Ising model

initialise with random values of q at the lattice sites (disordered states); cooling down leads to domain formation (clusters of equal values of q will appear)

small values of q: continuous phase transitions (2nd order), larger values for q: first-order (discontinuous) phase transitions

applications: modelling grain growth in metals, modelling diffusion of gas in foams (grains or bubbles are represented by sites with same values of q)

7.6 Simulated annealing and the travelling salesman problem

generalised Ising model:

$$E = \sum_{i,j} J_{i,j} S_i S_j \tag{7.25}$$

where the coupling constants $J_{i,j}$ are **random variables**, $S_i \in \{+1, -1\}$

this may result in a very complicated energy landscape,

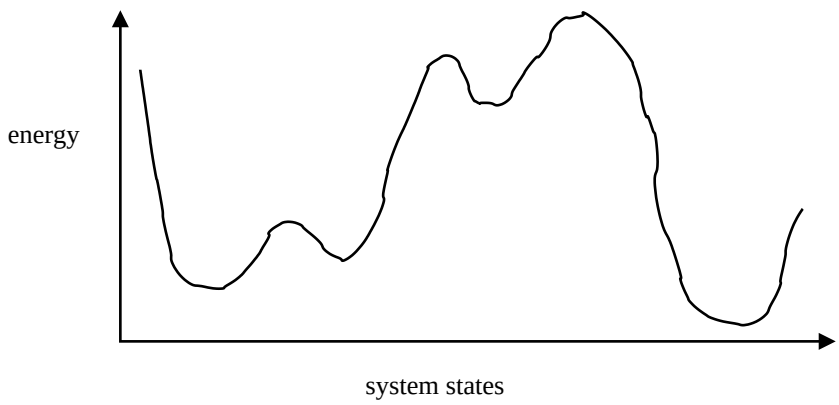

Figure 7.2: Sketch of ragged energy landscape. How can one find the (lowest) minimum?

question: can one find the global minimum, or an energy value which is close to it?

answer: method of **simulated annealing** can deal with such a problem, here we will introduce it in

The Travelling Salesman Problem:

"given a (random) distribution of cities in a plane, find the shortest *circuit* that visits every city only once"

N cities, route described by **order** of cities: $N!$ possibilities; N starting points and 2 directions give the same circuit, therefore there are $\frac{(N-1)!}{2}$ possible circuits

take $N = 100$: this results in 10^{155} circuits!

one can show: all **exact** methods to determine the optimal route require a computing time $\propto e^N$ (travelling salesman problem is a *non-polynomial problem*, **NP**)

exact solutions can only be found for small N (see demonstration in lecture)

7.6.1 The model Hamiltonian

x_{ij}: distance between cities i and j

define *connectivity* $c_{ij} = 1$ if i directly connected to j, else $c_{ij} = 0$

$\vec{S} = (c_{1,2}, c_{1,3}, \cdots c_{1,N}, c_{2,3}, c_{2,4}, \cdots c_{N-1,N}) = (1, 0, 0, 1, ..., 0, 0, 1)$ is *one* of the $\frac{(N-1)!}{2}$ possible circuits, its length/energy is given by

$$H(\vec{S}) = \sum_{i<j} c_{ij} x_{ij} \tag{7.26}$$

idea from solid state physics (difference between glass and crystal): allow a hot system to cool down slowly, so that it stays in thermal equilibrium during the cooling process; as temperature $T \to 0$ the state of the lowest energy E_0 should be reached

here T is a *formal temperature parameter*, for every value of T we model thermal equilibrium using a stochastic process based on detailed balance, eqn.(7.10). At $T = 0$ the transition probability is only finite for $\Delta H = 0$.

how slow shall we cool? system needs to have enough time to relax into thermal equilibrium

the time τ required to 'climb an energy mountain' $\Delta H > 0$ can be estimated using the **Arrhenius** law:

$$\tau = \tau_0 e^{\Delta H/T} \tag{7.27}$$

[Comment: this law was formulated by Arrhenius (1889) to describe the dependence of the speed of a chemical reaction on temperature and activation energy; in statistical mechanics $e^{-\Delta H/T}$ corresponds to the fraction of molecules that have enough kinetic energy to overcome an energy barrier of ΔH]

$\Rightarrow$ cooling schedule: $T(\tau) \geq \frac{\Delta H}{\ln(\tau/\tau_0)}$, ($\tau > \tau_0$, some natural time-scale)

$$T(\tau) \propto \frac{1}{\ln(\tau/\tau_0)} \tag{7.28}$$

Thus temperature must decrease at a slower and slower rate as the (computing) time progresses. In praxis there will not be enough computing time available to always operate in thermal equilibrium, thus the actual minimum (minimal path) will be different from the one that is found in the simulation, the path *length*, however, will be *nearly minimal*.

alternative computational method: genetic algorithms (see chapter 8)

7.6.2 Notes on the algorithm

- state $\vec{S}$ gives the **order** of the cities $\vec{S} = (i_1, i_2, \cdots, i_N)$.
- choose position p within the string S and length $l \leq N/2$ at random and create $\vec{S'} = (i_1, i_2, \cdots, i_{p-1}, i_{p+l}, i_{p+l-1}, \ldots, i_p, i_{p+l+1}, \cdots, i_N)$. (see fig. 7.3)
- compute new length of the circuit and perform the Metropolis algorithm

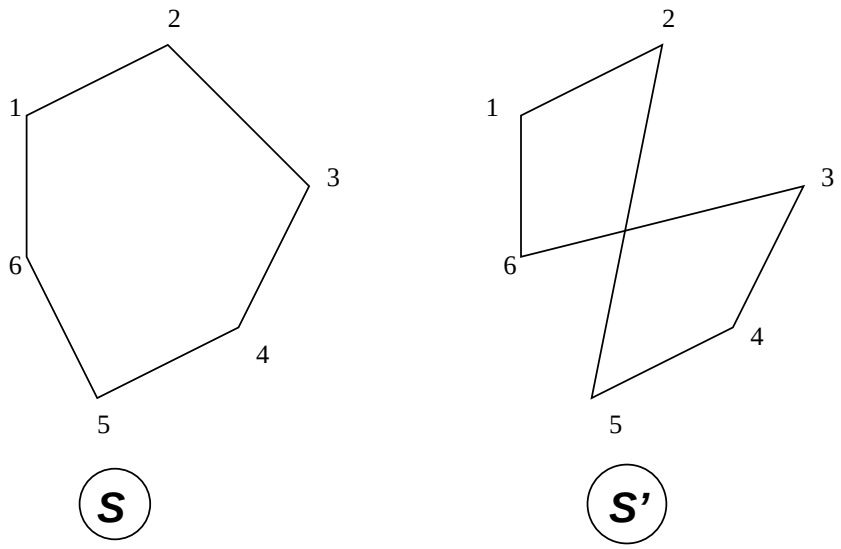

Figure 7.3: Two different circuits. Here $l = 2$ and $p = 3$.

alternative to Metropolis algorithm: replace Boltzmann factor by Heaviside step function, $\Theta[T - (H(\vec{S'}) - H(\vec{S}))]$: $\vec{S} \to \vec{S'}$ always accepted if $\vec{H}(S') < \vec{H}(S) + T$ **(threshold accepting)**

(both methods may be examined in the sample program by Kinzel and Reents, see lecture)

7.6.3 Results

analytical estimates for shortest circuit C_0 of N (uncorrelated) cities located in area $L \times L$:

- regarding only nearest neighbours: $C_0 \geq \frac{1}{2} L \sqrt{N}$

- regarding next nearest neighbours: $C_0 \geq 0.599L\sqrt{N}$

for numerical purposes the following *scaled length* is useful: $l = \frac{C_0}{L\sqrt{N}}$

numerical result: $l = 0.7120 \pm 0.0002$ for $N \to \infty$ (A. G. Percus and O. C. Martin, "Dimensional Dependence in the Euclidean Travelling Salesman Problem", *Phys. Rev. Lett.* **76**, 1188 (1996))

sample computation: $N = 100$, $l_{initial} \simeq 4.8$, $l_{final} \simeq 0.721$

suggestions:

- check the scaling of l with $\sqrt{N}$ numerically
- examine threshold sampling (faster **and** better?)
- set $T = 0$: algorithm only allows downward movement in the energy landscape; it is fast, but $l \simeq 0.909$

Chapter 8

Genetic Algorithms

8.1 Introduction

evolution in nature: how can random changes in genetic code lead to ever increasing complexity in nature?

genotype: DNA, strings of 1 and 0 $\Rightarrow$ **phenotype** (red or brown hair? fast or slow runner? high or low energy ? etc.)

simple model of natural selection:

1. random changes in genetic code:

   (a) mutation ($0 \rightarrow 1$, $1 \rightarrow 0$) at randomly chosen sites,

   (b) sexual reproduction

   genetic changes may effect the phenotype, and thus the "fitness" of the individual, i.e.. how it copes with its surroundings

2. selection of phenotype according to a fitness criteria

modelling sexual reproduction:

A = 0011001010
B = 0001110001
exchange bits 4 to 7 (randomly chosen locations)
A' = 0011110010 B' = 0001001001

then enlarge the total population by adding A' and B', then apply a **selection criterion**

selection criteria:

- nature: interaction with environment and other species
- solid state physics: construct a fitness function, based on energy considerations

8.2 Application to spin glasses

phenotype: square lattice occupied by $L \times L$ spins s_i with values $s_i = \pm 1$

$$E = -\sum_{i,j=nn(i)} T_{ij} s_i s_j \tag{8.1}$$

with random coupling constants T_{ij}. For $|T_{ij}| \leq 1$ the minimum energy possible is given by $E_{min} = -2L^2$, the maximum energy possible is $E_{max} = +2L^2$.

as a measure of **fitness** we define:

$$F = 2L^2 - E \geq 0 \tag{8.2}$$

translation of genotype into phenotype:

let's map a (genetic) string of length L^2 onto a square lattice (see figure 8.1)

site (i, j) corresponds to n^{th} bit on a string with $n = (j-1)L+i$, with $1 \leq i, j \leq L$.

if a bit at position n of the string is 1 then spin is up at site (i, j)
if bit is 0 then spin is down at site (i, j)

algorithm to find ground state of spin glass:

1. input N binary strings of length L^2
2. perform mutations and recombinations (see section 8.1)
3. compute the corresponding lattice energies (periodic b.c.), and compute fitness functions

(1,1) 1	(1,2) 4	(1,3) 7
(2,1) 2	(2,2) 5	(2,3) 8
(3,1) 3	(3,2) 6	(3,3) 9

Figure 8.1: Translating lattice sites into string positions.

4. select N strings according to probability as given by fitness of every string
5. back to 2

input parameters

string/lattice size L, population number N, number of recombinations per generation (50% ?), number of mutations per generations (20% ?), number of generations (end of program)
computational check: compute ground-state of the ferromagnet ($J_{ij} = 1$ for all i, j)

selection according to fitness

see figure 8.2

$$F_{tot} = \sum_{i=0}^{N} F_i \tag{8.3}$$

F_i are the fitness values of the individual genotype strings

Choose a random number r, distributed uniformly between 0 and F_{tot}. Find index i for which

$$\sum_{j=0}^{i-1} F_j \leq r \leq \sum_{j=0}^{i} F_j \tag{8.4}$$

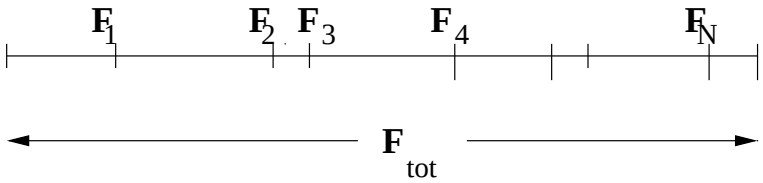

Figure 8.2: Sampling according to a non-uniform probability distribution.

8.3 Important features of genetic algorithms

- the evolution of the genetic code is not driven directly from the make-up of the genotype, but from its corresponding phenotype
- the way the strings are changed is not closely related to the 2d lattice (same procedure may be applied for 3d lattice)

The genetic coding does not need to mimic the phenotype expression, "a certain gene cannot be tied to a particular trait".

8.4 Example: finding maximum of a function $f(x)$

program *Holland* from Bialynicki-Birula, "Modelling Reality", Oxford University Press 2004, see lecture demonstration

- create a string P, defined by its *genotype*, which is made up of 32 genes of value 0 or 1, randomly selected; interpret this string as a binary number, i.e. $0 \times 2^{-1} + 1 \times 2^{-2} + \cdots + 1 \times 2^{-32}$, which results in a real number x between 0 and 1. This real number, x, is the phenotype corresponding to the string P.

  Evaluate the function to be maximised at x and interpret $f(x)$ as the fitness of P

- repeat in loop
  - create individual N with same genotype as P
  - mutate N's genotype by reversing each bit with probability m, good choice $m = 1/32$
  - calculate phenotypes of N and P
  - if $f(x_N) > f(x_P)$ then discard P and rename N as P, otherwise discard N

- works reasonably well but can get stuck when a small change in the phenotype requires a large number of changes in the genotype. The largest number (i.e. phenotype) smaller than 0.5 (i.e. of genotype 01111111111111111111111111111111), differs from 0.5 (i.e. genotype 10000000000000000000000000000000), in 32 positions!

- this effect is called *Hamming cliff*: similar genotypes do not always correspond to similar phenotypes

- remedy: use *Gray code* instead of *binary code*; nearby numbers always only differ in one bit!

number	binary notation	gray code
0	0000	0000
1	0001	0001
2	00<u>10</u>	00<u>1</u>1
3	0011	0010
4	0<u>100</u>	0<u>1</u>10
5	0101	0111
6	01<u>10</u>	01<u>0</u>1
7	0111	0100
8	<u>1000</u>	<u>1</u>100
9	1001	1101

Program *Holland*, named after founder of genetic algorithms, John Holland works in both modes, and is particularly effective when based on the Gray code (see lecture).

Chapter 9

Neural networks

9.1 Introduction

Neural networks: algorithms based on very simplified models of how the brain works

Features of the brain:

- associative memory: reconstruct incomplete patterns from stored patterns
- learning capability (training)
- 10^{11} brain cells (**neurons**), interconnected by 10^{15} links (**synapses**) (massively parallel)
- **strength** of these connections are **updated** as memory is stored (learning)
- electro-chemical processes in neurons, a neuron can fire if its potential due to other firing connected neurons exceeds a certain **threshold**

9.2 A simple model: the Perceptron (1958)

- N input neurons S_i, characterised by binary state: $S_i = +1$: neuron i is active ("fires"), $S_i = -1$: neuron i at rest
- one output neuron S_0, linked to input neurons via N synaptic weights w_i

- these weights w_i model how strong the signal arriving from S_i will be transformed into an electric potential at S_0; exciting synaptic weight: $w_i > 0$; inhibiting synaptic weight $w_i < 0$:

- if the sum of the potentials acting at S_0 exceed the threshold value of zero, then the neuron S_0 is firing ($S_0 = +1$), else it is quiet ($S_0 = -1$)

  sum of potentials:

$$h = \sum_{j=1}^{N} w_j S_j \tag{9.1}$$

  reaction of neuron S_0

$$S_0 = sgn(h), \tag{9.2}$$

  where $sgn(x)$ is the signum function which extracts the sign of a given real number

- **learning:** the neuronal network is trained by a number ν of *examples*, <u>not by rules</u>. Training patterns are input/output pairs $(\vec{x}_\nu, y_\nu)$ with input $\vec{x}_\nu = (x_{\nu 1}, x_{\nu 2} \cdots x_{\nu N})$ and output $y_\nu \in \{+1, -1\}$

- training corresponds to adaption of the weights; <u>for a perfectly trained neuron</u> one obtains $y_\nu = sgn(\vec{w} \cdot \vec{x}_\nu) = sgn\left(\sum_{j=1}^{N} w_j x_{\nu j}\right)$

- Hebb rule (1949)

$$\Delta\vec{w}(t) = \frac{1}{N} y_\nu \vec{x}_\nu \text{ if } (\vec{w} \cdot \vec{x}_\nu) y_\nu \leq 0 \text{(i.e. when the prediction is wrong); else } \Delta\vec{w} = 0 \tag{9.3}$$

  when the prediction is right there is thus no update of the weight required

  (not proven here (Rosenblatt, 1960))

In the example shown in the lecture (Kinzel/Reents, nn.c) input are a succession of left and right mouse clicks. Based on sample input of a test person the neural network will make a prediction on whether the next input will be a left or right mouse click. The network is able to detect/learn patterns in the input, its prediction rate will exceed the 50% corresponding to a random guess.

$$x_{\nu_i}, y_\nu \in \{+1, -1\} = \{\text{left button, right button}\} \tag{9.4}$$

Percepton has as input a moving window of N mouse-clicks, resulting in e.g. $\vec{x}_\nu = (1, -1, -1, 1, 1, -1, 1, 1, 1, \cdots, 1)$ for a particular ν.

The prediction is then $sgn(\vec{w} \cdot \vec{x}_\nu)$, see eqn. (9.2), followed by an update of the weights according to the Hebb rule, eqn. (9.3).

Algorithm

```
while (1) {
```

1. input:

   ```
   if (getch() == '1')input=1; else input= -1; runs++;
   ```

2. compute the potential:

   ```
   h=0;
   for (i=0;i<N;i++) h+=weight[i] * neuron [i];
   ```

3. count the correct predictions:

   ```
   if (h*input >0) correct++;
   ```

4. learning (Hebb rule):

   ```
   if (h*input <0)
   for (i=0;i<N;i++)
   weight[i] += input*neuron[i]/(float)N;
   ```

5. move input window:

   ```
   for (i=N-1; i>0; i--) neuron[i] = neuron[i-1];
   neuron[0]=input;
   } /* end of while */
   ```

more advanced version where all neurons are interlinked: Hopfield model

9.3 The Hopfield model

- more advanced model where all neurons are interlinked
- neuron i is linked to other neurons via synapses $J_{ij} \in \mathbf{R}$, the value of J_{ij} indicates the strength of the coupling, cells can inhibit or excite each other; $J_{ij} > 0$: exciting, $J_{ij} < 0$: damping
- potential acting on a neuron is given by $h_i(t) = \sum_j J_{ij}(t) S_j(t)$
- **learning phase:** p training patterns characterised by $\xi_i^\mu = \pm 1 (\mu = 1, 2, \cdots, p)$ are stored by setting up the synaptic matrix J_{ij}:

$$J_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu \qquad \text{(Hebb rule)} \tag{9.5}$$

  the learned patterns correspond to p local minima, where the energy is computed from $E = -\frac{1}{2} \sum_i \sum_j J_{ij} S_i S_j$.

- **test phase:** take configuration $S_i(0)$ that is different from the learned pattern ξ_i^μ (say $S_i(0)$ is an incomplete or noisy pattern): network is not in an energy minimum; aim is to reconstruct the pattern and thus find the right energy minimum (associative memory)

Different patterns correspond to different locally stable states

in practice: maximum number of patterns that can be stored with N neurons: $p_{max} \simeq 0.14N$

Appendix A

References

1. Nicholas J. Giordano and Hisao Nakanishi, Computational Physics, Pearson Prentice Hall, 2nd ed., 2006, [numerical methods are summarised in the appendices]

2. Rubin H Landau and Manuel J Páez, Computational Physics – Problem solving with computers John Wiley, 1997

3. Rubin H Landau and Manuel J Páez, Computational Problems for Physics – With Guided Solutions Using Python CRC Press, 2018

4. Paul L DeVries, A First Course in Computational Physics, John Wiley, 1994

5. Harvey Gould, Jan Tobochnik, An Introduction to Computer Simulation Methods – Application to Physical Systems, Addison-Wesley, 1996 (Chapters 15,16,17)

6. J Douglas Faires and Richard Burden, Numerical Methods, Brooks/Cole Publishing Company, 1998

7. Iwo Bialynicki-Birula and Iwona Bialynicka-Birula, Modeling Reality – How computers mirror life Oxford University Press, 2004

8. Wolfgang Kinzel, Georg Reents, Physik per Computer, Spektrum Akademischer Verlag, Heidelberg 1996 (in German) Springer, Heidelberg 1997 (in English) (Chapters 3 and 5)