FYTN03
HT08

Lecture Notes, FYTN03 Computational Physics

1 Introduction

Numerical methods in physics is simple. Just take your formula and replace everything looking like δx with Δx. This also implies $\int \to \sum$, ie.

$$\int_a^b f(x)dx \approx \sum_{n=0}^{N} \Delta x f(a + n\Delta x),$$

with $N\Delta x = b - a$.

When we do this we introduce an error. The smaller Δx the smaller the (truncation) error, but the required computational time also increases since we have to do a larger number of function evaluations.

To understand the truncation error we typically simply consider the Taylor expansion:

$$f(x + \Delta x) = \sum_{n=0}^{\infty} \frac{\Delta x^n}{n!} f^{(n)}(x).$$

This is all you need to know.

Almost.

To reliably do numerical physics computations you need to have lots of experience and learn lots of tricks. Especialy important is to understand tricks involving pseudo random numbers. You also need to learn how to write programs. In this course you will learn a lot of tricks. You will gain a beginning of experience. You will, however not learn programming.

2 Errors. Interpolation and Extrapolation.

[NR: 1.3, 3.0, 3.1, 5.7 (3.2, 3.3)]

2.1 Errors

Numerical calculations are done in integer or floating-point arithmetic. Integer arithmetic is exact, whereas floating-point arithmetic has roundoff errors.

The floating-point representation looks like

$$s \cdot M \cdot 2^p \qquad \begin{cases} s & \text{sign} \\ M & \text{mantissa} \\ p & \text{integer exponent} \end{cases}$$

Any real number can be represented this way, and the representation becomes unique if we require eg. that $\frac{1}{2} \leq M < 1$ for any non-zero number.

On the computer, each floating-point number becomes a string of bits, each bit being 0 or 1. Suppose the number of bits per floating-point number (the "wordlength") is 32, with 1 sign bit, 8 bits for p and 23 bits for M. p and M may then (for example) be represented as

$$\begin{aligned} p &= \underbrace{x_7 \cdot 2^7 + x_6 \cdot 2^6 + ... + x_0 \cdot 2^0 - 2^7}_{-2^7, -2^7+1, ..., 2^7-1} \qquad x_i \in \{0, 1\} \\ M &= y_1 \cdot 2^{-1} + y_2 \cdot 2^{-2} + ... + y_{23} \cdot 2^{-23} \qquad y_i \in \{0, 1\} \end{aligned}$$

Roundoff errors arise because the number of y_i's is finite.

How large are the roundoff errors? The number 1 can be represented exactly ($x_7 = x_0 = y_1 = 1$, the other bits zero). The smallest number > 1 that can be represented exactly is $1 + 2^{-22}$ ($x_7 = x_0 = y_1 = y_{23} = 1$, the other bits zero). This gives us an estimate of the typical *relative precision* ε of 32-bit floating-point numbers: $\varepsilon \sim 2^{-22} \sim 10^{-7}$.

C has two data types for floating-point numbers, `float` and `double`. The size of a float or a double is not specified by the language. To find out the size, one can use the function `sizeof` which returns the number of bytes for a given type (1 byte = 8 bits).

In a C program you also have access to the precision, in you `#include` the standard header file `float.h`: `FLT_EPSILON` gives the precision for `float` and `DBL_EPSILON` gives the precision for `double`

Roundoff errors are especially troublesome when subtracting two numbers with small relative difference. Consider eg. one of the solutions of the equation $ax^2 + bx + c = 0$:

$$x = -\frac{b - \sqrt{b^2 - 4ac}}{2a}$$

for $\sqrt{ac} \ll b$. Here the trick is simple. Just multiply with $b + \sqrt{b^2 - 4ac}$ in both denominator and nominator:

$$x = -\frac{b - \sqrt{b^2 - 4ac}}{2a} \times \frac{b + \sqrt{b^2 - 4ac}}{b + \sqrt{b^2 - 4ac}} = -\frac{2c}{b + \sqrt{b^2 - 4ac}}.$$

In addition to roundoff errors, most numerical calculations contain errors due to approximations, such as discretization or truncation of an infinite series. Such errors are called *truncation errors*. Truncation errors are errors that would persist even on a hypothetical computer with infinite precision.

Example: Numerical derivation.

Suppose a derivative $f'(x)$ is calculated by using

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2} f''(x) + ... \\ f'(x) &= \frac{f(x+h) - f(x)}{h} + O(h) \end{aligned}$$

- The truncation error is $\varepsilon_t \sim h\,|f''(x)|$.
- The roundoff errors in $f(x+h)$ and $f(x)$ are $\gtrsim \varepsilon |f(x)|$, where ε is the relative floating-point precision. Therefore, a lower bound on the total roundoff error ε_r is $\varepsilon_r \gtrsim \varepsilon |f(x)|/h$.

Note that h large $\Rightarrow \varepsilon_t$ large, whereas h small $\Rightarrow \varepsilon_r$ large.

For the total relative error, we get the estimate

$$\frac{\varepsilon_r + \varepsilon_t}{|f'|} \gtrsim \frac{h\,|f''| + \varepsilon\,|f|\,/h}{|f'|} = \frac{1}{|f'|} \left[\left(\sqrt{h\,|f''|} - \sqrt{\varepsilon\,|f|\,/h} \right)^2 + 2\sqrt{\varepsilon\,|f''f|} \right]$$

With an optimal choice of h (so that the square vanishes), this becomes

$$\frac{\varepsilon_r + \varepsilon_t}{|f'|} \gtrsim \sqrt{\varepsilon}\sqrt{\frac{|f''f|}{f'^2}}$$

So, the error is $\mathcal{O}(\varepsilon^{1/2})$ rather than $\mathcal{O}(\varepsilon)$, which does make a difference: $\varepsilon = 10^{-7} \Rightarrow \varepsilon^{1/2} \approx 3 \cdot 10^{-4}$.

How can we improve on this?

1. One way is to derive a better approximation of the derivate by direct manipulation of the Taylor expansion.

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3!}f'''(3) + ... \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{3!}f'''(3) + ... \\ \frac{f(x+h) - f(x-h)}{2h} &= f'(x) + \frac{h^2}{3!}f'''(x) + \frac{h^4}{5!}f^{(5)}(x) + ... \end{aligned}$$

   The truncation error is $\mathcal{O}(h^2)$ for this "central difference", which is one order better than for the "forward difference" in the example above (as a result, the total error scales as $\varepsilon^{2/3}$ for the central difference, as can be easily verified).

2. Another possibility is to use Richardson extrapolation.

2.2 Richardson Extrapolation

Suppose we want to determine a quantity a_0 that satisfies

$$a_0 = \lim_{h \to 0} a(h)$$

where h may be thought of as a step-size parameter. Suppose further that

1. the functional form of $a(h)$ is known to be $a(h) = a_0 + a_1h^2 + a_2h^4 + ...$
2. the value $a(h)$ is known for $h = h_0, 2h_0, 2^2h_0, \ldots$

(a_0 could be a derivate and $a(h)$ the central-difference approximation).

We can then obtain an improved estimator $\hat{a}(h)$ of a_0 in the following way:

$$\begin{cases} a(h) = a_0 + a_1 h^2 + a_2 h^4 + \mathcal{O}(h^6) \\ a(2h) = a_0 + 4a_1 h^2 + 16 a_2 h^4 + O\left(h^6\right) \end{cases} \quad \Rightarrow \quad 4a\left(h\right) - a\left(2h\right) = 3a_0 - 12 a_2 h^4 + O\left(h^6\right)$$

$$\Rightarrow \hat{a}\left(h\right) \equiv \frac{4a\left(h\right) - a\left(2h\right)}{3} = a_0 - 4a_2 h^4 + O\left(h^6\right)$$

$\hat{a}(h)$ is an improved estimator of a_0 because the truncation error is $\mathcal{O}(h^4)$ instead of $\mathcal{O}(h^2)$.

We can now go on to eliminate the h^4 term by forming

$$\hat{\hat{a}}\left(h\right) \equiv \frac{16\hat{a}\left(h\right) - \hat{a}\left(2h\right)}{15} = a_0 + O\left(h^6\right) \qquad \text{(verify this!)}$$

and so on. To calculate $\hat{\hat{a}}(h)$ for one value $h = h_0$, we need to know $a(h)$ for $h = h_0$, $2h_0$ and $4h_0$.

$$\begin{array}{ccccc} a(4h_0) & & & & \\ & \searrow & & & \\ a(2h_0) & \rightarrow & \hat{a}(2h_0) & & \\ & \searrow & & \searrow & \\ a(h_0) & \rightarrow & \hat{a}(h_0) & \rightarrow & \hat{\hat{a}}(h_0) \end{array}$$

Comments

The same procedure can be applied for more general versions of the assumptions 1 and 2. The precise form of the expressions for $\hat{a}$ and $\hat{\hat{a}}$ will then change.

Examples of methods that use this technique are Romberg's integration method and the Burlirsh-Stoer method for ordinary differential equations.

2.3 Interpolation and Extrapolation

Suppose we are given a set of data points (x_i, y_i), $i = 1, 2, \ldots, N$, and want to find an approximating or interpolating function $f(x; \{c_j\})$, where the c_j's are parameters. For a given functional form of f, the task then is to find optimal parameters c_j.

Suppose f is taken to be a polynomial of degree $M - 1$,

$$f\left(x; \{c_j\}\right) = c_1 + c_2 x + ... + c_M x^{M-1}$$

Whether or not it is possible to find c_j's such that

$$f\left(x_i;\{c_j\}\right)=y_i\,,\ i=1,2,...,N \Leftrightarrow \begin{cases} c_1+c_2x_1+...+c_Mx_1^{M-1}=y_1 \\ \qquad\qquad \vdots \\ c_1+c_2x_N+...+c_Mx_N^{M-1}=y_N \end{cases}$$

depends on N and M — we have a linear system of N equations for the M unknown parameters c_j.

- $M > N$
  This case is the least interesting one — there are too many parameters.

- $M = N$
  In this case, there is a unique solution which is given by the Lagrange interpolation formula

$$\begin{aligned} f\left(x\right) &= \frac{\left(x-x_2\right)\left(x-x_3\right)\cdots\left(x-x_N\right)}{\left(x_1-x_2\right)\left(x_1-x_3\right)\cdots\left(x_1-x_N\right)}\cdot y_1+...+\frac{\left(x-x_1\right)\cdots\left(x-x_{N-1}\right)}{\left(x_N-x_1\right)\cdots\left(x_N-x_{N-1}\right)}\cdot y_N \\ &= \sum_{i=1}^{N}\frac{\prod_{j\neq i}(x-x_j)}{\prod_{j\neq i}(x_i-x_j)}y_i \end{aligned}$$

  provided that $x_i \neq x_j$ whenever $i \neq j$ (geometrically: through any two points there is a unique straight line or first-degree polynomial; through any three points there is unique second-degree polynomial; etc.).

- $M < N$
  In general, there is no solution in this case, so one must approximate rather than interpolate. This is often done by using the method of least squares; that is, the c_j's are determined by minimizing

$$\sum_{i=1}^{N}[f(x_i;\{c_j\})-y_i]^2$$

Taking $f(x,\{c_j\})$ to be a polynomial is a convenient choice, for example, in numerical integration. However, it is far from always the best choice when it comes to approximating functions (consider, for example, the function $(1-x)^{-1}=1+x+x^2+\ldots$). Two alternatives to polynomials are rational functions and cubic splines.

Rational functions

- Interpolation.
  Example: Assume that there are three data points (x_i, y_i) and that $f(x;\{c_j\}) = (x - c_1)/(c_2 x + c_3)$. Interpolation then amounts to solving a linear system of three equations for the c_j's:

$$f(x_i;\{c_j\}) = y_i \;\Leftrightarrow\; c_1 + c_2 x_i y_i + c_3 y_i = x_i \qquad i = 1,2,3$$

  In general we can use any rational polynomial

$$R_{\mu\nu}(x) = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{1 + \sum_1^\mu p_i x^i}{\sum_0^\nu q_j x^j},$$

  where we choose $\mu + \nu + 1$ equal to the number of points N and arrive at the equation system

$$1 + \sum_1^\mu p_i x_k^i - y_k \sum_0^\nu q_j x_k^j = 0$$

- Padé approximation. Suppose we want to approximate a function $g(x)$ and know $g(x)$ as well as a number of derivatives $g'(x)$, $g''(x)$, ... for $x = 0$. In the Padé method, the approximating function $f(x;\{c_j\})$ is rational, and the parameters c_j are determined so that

$$f\left(x = 0;\{c_j\}\right) = g\left(0\right), \quad \frac{d^k}{dx^k} f\left(x = 0;\{c_j\}\right) = \frac{d^k g\left(0\right)}{dx^k}$$

  up to highest possible order k.

Cubic spline-interpolation

Let $a = x_1 < x_2 < ... < x_N = b$. The cubic spline function is obtained by using one cubic polynomial for each subinterval $[x_i, x_{i+1}]$. These polynomials are put together in such a way that the resulting function as well as its first two derivatives become continuous over the whole interval $a \leqslant x \leqslant b$.

This will give us $N - 1$ functions with four parameters each, ie. $4(N - 1)$ unknown. We have $2(N - 1)$ equations forcing each function to go through the two connecting points and $2(N - 2)$ equations for the continuity of the first and second derivatives. This means we will have to specify two more conditions, eg. the third derivatives for the last and first point.

Note that splines may severely misrepresent functions which contain kinks, or if the points have some statistical fluctuations around function values. In the latter case one should use function fitting rather than interpolation (see appendix).

3 Numerical integration (quadrature)

[NR: 4.0, 4.1, 4.2, 4.3, 4.5 (4.4, 4.6)]

We will discuss several methods for calculating integrals. For an integral

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

all these methods can be written in the form

$$I \approx \sum_i A_i f(x_i) \qquad \begin{cases} A_i & \text{weights} \\ x_i & \text{abscissas} \end{cases}$$

As mentioned in the introduction, the more points we use, the better the precision and the longer the computing time. So let's try to choose the points, x_i, as efficiently as possible, and let's see if we can find some tricks to allow us to reduce the number of points without reducing the precision. Here we will investigate three different possibilities:

1. Equidistant x_i's (section 3.1).
2. Gaussian quadrature, where the x_i's are zeros of polynomials (section 3.2).
3. Monte Carlo, where the x_i's are randomly chosen (section 4.4).

Another possibility is to use methods for ordinary differential equations. This can be done because $I = y(b)$, where $y(x)$ denotes the solution to the initial value problem

$$\frac{dy}{dx} = f(x) \qquad y(a) = 0$$

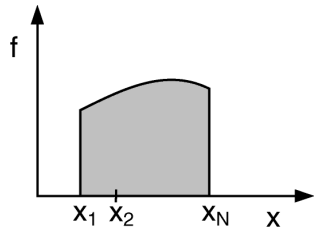
This can be a good approach if the integral is dominated by a few sharp peaks.

3.1 Equidistant x_i

3.1.1 Trapezoidal Rule, Simpson's Rule,...

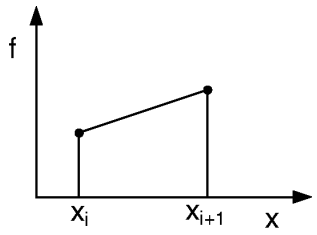
Consider

$$I = \int_{x_1}^{x_N} f(x)\,dx \qquad f_i \equiv f(x_i) \qquad x_{i+1} = x_i + h,\ i = 1, \ldots, N-1$$



- The trapezoidal rule.
  Use a linear approximation of $f(x)$ on each subinterval $[x_i, x_{i+1}]$:

$$f(x) \approx \frac{x_{i+1}-x}{h} f_i + \frac{x-x_i}{h} f_{i+1} \;\Rightarrow\; \int_{x_i}^{x_{i+1}} f(x)\,dx \approx \frac{h}{2}\left(f_i + f_{i+1}\right)$$



  From the expansion

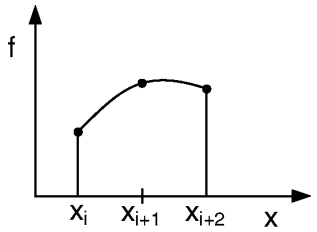
$$f(x) = f_i + (x-x_i) f'(x_i) + O\left(h^2\right) = f_i + (x-x_i)\frac{f_{i+1}-f_i}{h} + O\left(h^2\right)$$

  it follows that the truncation error in $f(x)$ is $\mathcal{O}(h^2)$. This in turn suggests that the integral has an error of $\mathcal{O}(h^3)$ (the length of the interval is h). Adding up the intervals to a fixed integration region will give us $\mathcal{O}(1/h)$ intervals and the total error of the integral will be $\mathcal{O}(h^2)$

- The Simpson rule.
  This method is similar, but uses second-degree instead of first-degree polynomials. This means that three points are needed for the interpolation, which makes it necessary to look at intervals $[x_i, x_{i+2}]$. Lagrange's interpolation formula and integration give:

$$\int_{x_i}^{x_{i+2}} f(x)\,dx \approx \frac{h}{3}\left(f_i + 4f_{i+1} + f_{i+2}\right)$$

  The same kind of error estimate as above indicates that the truncation error is $\mathcal{O}(h^4)$ in this case. It turns out, however, that the actual error is smaller than that, $\mathcal{O}(h^5)$, thanks to cancellation.



So far, we looked at subintervals. The full integral is obtained by adding the results for the subintervals. Consider, for simplicity, the trapezoidal rule.

1. Closed formula. Apply the result above to each of the subintervals, which in particular means that the end points x_1 and x_N are used. This gives

$$\int_{x_1}^{x_N} f(x)\,dx \approx h\left(\tfrac{1}{2}f_1 + f_2 + ... + f_{N-1} + \tfrac{1}{2}f_N\right)$$

   The truncation error should not be worse than $\sim Nh^3 \sim h^2$ (the total length $x_N - x_1 = (N-1)h$ is fixed).

2. Open formula. Here, to avoid using the end points, the two end intervals are treated differently. This is necessary if f has an end-point singularity (eg. $\int_0^1 dx/\sqrt{x}$), or has a limiting value at the endpoint (eg. $\sin(x)/x$ at 0). For the end intervals, the following estimates can be used:

$$\int_{x_1}^{x_2} f(x)\,dx \approx hf_2 + O\left(h^2\right) \qquad \int_{x_{N-1}}^{x_N} f(x)\,dx \approx hf_{N-1} + O\left(h^2\right)$$

   The fact that the estimates for these two subintervals are poorer does not affect the scaling of the total truncation error, which remains $\mathcal{O}(h^2)$ when adding two $\mathcal{O}(h^2)$ terms.

3.1.2 Romberg's Method = Trapezoidal Rule + Richardson Extrapolation

Romberg's method is based on the Euler-Maclaurin summation formula, which we state without proof. Let

$$T(h) = h(\tfrac{1}{2}f_1 + f_2 + ... + f_{N-1} + \tfrac{1}{2}f_N)$$

be the closed trapezoidal rule. The Euler-Maclaurin summation formula says that

$$T(h) - \int_{x_1}^{x_N} f(x)dx = c_1 h^2 + c_2 h^4 + \dots$$

where

$$\begin{cases} c_k = \dfrac{B_{2k}}{(2k)!}\left(f^{(2k-1)}(x_N) - f^{(2k-1)}(x_1)\right) \\ B_n \text{ are Bernoulli numbers, defined by} \dfrac{t}{e^t - 1} = \displaystyle\sum_{n=0}^{\infty} B_n \frac{t^n}{n!} \\ \text{ie. } B_n = \frac{d^n}{dt^n}\left(\frac{t}{e^t - 1}\right)|_{t=0} \end{cases}$$

In particular, this result shows that the truncation error of the trapezoidal rule contains only even powers of h. If we use Richardson extrapolation, we will therefore gain two powers of h at each step. The situation is exactly the same as for the central-difference approximation of a derivative, so from the results in section 2.2, it immediately follows that

$$\begin{aligned} \hat{T}(h) &= \frac{4T(h) - T(2h)}{3} = \int_{x_1}^{x_N} f(x)\,dx + O\left(h^4\right) \\ \hat{\hat{T}}(h) &= \frac{16\hat{T}(h) - \hat{T}(2h)}{15} = \int_{x_1}^{x_N} f(x)\,dx + O\left(h^6\right) \end{aligned}$$

Comments

- The Euler-Maclaurin formula
  - For fixed h, the sum on the RHS does not necessarily approach the LHS as the number of terms tends to infinity (it is an asymptotic series).
  - The formula is sometimes useful in the opposite direction — to calculate a sum by replacing it with an integral.
  - There is an analogous formula for the open trapezoidal rule.
- Trapezoidal rule + 1 Richardson step = Simpson's rule
  To see this, consider an arbitrary subinterval $[x_i, x_{i+2}]$:

$$\begin{cases} T(2h) = h\left(f_i + f_{i+2}\right) \\ T(h) = h\left(\frac{1}{2}f_i + f_{i+1} + \frac{1}{2}f_{i+2}\right) \end{cases}$$

$$\Rightarrow\ \hat{T}(h) = \tfrac{4}{3}T\left(h\right) - \tfrac{1}{3}T\left(2h\right) = \ldots = \underbrace{\frac{h}{3}\left(f_i + 4f_{i+1} + f_{i+2}\right)}_{\text{Simpson's formula}}$$

3.2 Gaussian Quadrature

Let f be a polynomial of degree $< N$. Lagrange's interpolation formula tells us that

$$f(x) = \sum_{i=1}^{N} L_i(x) f(x_i) \qquad L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

for all possible choices of $x_1, \ldots, x_N$ such that $x_i \neq x_j$ if $i \neq j$. This implies that, for all such f, the integration formula

$$\int_a^b f\left(x\right) w\left(x\right) dx \approx \sum_{i=1}^{N} f\left(x_i\right) \underbrace{\int_a^b L_i\left(x\right) w\left(x\right) dx}_{\equiv A_i}$$

is exact. Here, $w(x) \geq 0$ is an arbitrary "weight" function and it can, of course, be put to 1, but it can also be set to some other function. Eg. consider a function which has an integrable singularity which is known to behave like $1/\sqrt{x}$ as $x \to 0$, then we can use $w(x) = 1/\sqrt{x}$ and replace $f(x)$ by $\sqrt{x}f(x)$ which may be better approximated by a polynomial.

In Gaussian quadrature, this formula becomes exact for all polynomials with degree $< 2N$, by a careful choice of the x_i's.

To see how this works, we need to define orthogonality. Using the same a, b and $w(x)$ as in the integration formula, we define the scalar product of two arbitrary functions f and g by

$$\langle f|g\rangle = \int_a^b f\left(x\right) g\left(x\right) w\left(x\right) dx\,,$$

and we say that f and g are orthogonal if $\langle f|g\rangle = 0$. For any given scalar product, it is possible to construct a sequence of orthogonal polynomials $\varphi_0, \varphi_1, \ldots$ with degrees $0, 1, \ldots$ (φ_n has degree n) by using the Gram-Schmidt method:

- $\varphi_0\left(x\right) = 1$

- $\varphi_1(x) = x + a\varphi_0(x)$ $\qquad\qquad \langle\varphi_1|\varphi_0\rangle = 0 \;\Rightarrow\; a = -\dfrac{\langle\varphi_0|x\rangle}{\langle\varphi_0|\varphi_0\rangle}$
- $\varphi_2(x) = x^2 + b\varphi_1(x) + c\varphi_0(x)$
  $\langle\varphi_2|\varphi_1\rangle = \langle\varphi_2|\varphi_0\rangle = 0 \;\Rightarrow\; b = -\dfrac{\langle\varphi_1|x^2\rangle}{\langle\varphi_1|\varphi_1\rangle}$ and $c = -\dfrac{\langle\varphi_0|x^2\rangle}{\langle\varphi_0|\varphi_0\rangle}$
- ...
- $\varphi_{i+1}(x) = (x - a_i)\varphi_i(x) - b_i\varphi_{i-1}(x)$

  $\Rightarrow\; a_i = \dfrac{\langle x\varphi_i|\varphi_i\rangle}{\langle\varphi_i|\varphi_i\rangle}$ and $b_i = \dfrac{\langle\varphi_i|\varphi_i\rangle}{\langle\varphi_{i-1}|\varphi_{i-1}\rangle}$,
- ...

By construction, φ_{i+1} is orthogonal to φ_i and φ_{i-1}. With induction we can then also show that if φ_i is orthogonal to *all* φ_j with $j < i$, so is φ_{i+1}. We have for $n > 1$

$$\langle\varphi_{i+1}|\varphi_{i-n}\rangle = \langle(x - a_i)\varphi_i - b_i\varphi_{i-1}|\varphi_{i-n}\rangle = \langle x\varphi_i|\varphi_{i-n}\rangle = \langle\varphi_i|x\varphi_{i-n}\rangle$$

which is zero since $x\varphi_{i-n}$ can be written as a linear combination of polynomials of degree $j < i$.

In NR 4.5 you will find formulas for the orthogonal polynomials corresponding to some useful weight functions.

Suppose that φ_N is the Nth-degree polynomial obtained by the Gram-Schmidt procedure, for the particular scalar product defined above. The *Gaussian quadrature*-theorem says that if $x_1, ..., x_N$ are taken to be the zeros of φ_N, then the integration formula

$$\int_a^b f(x)\, w(x)\, dx \approx \sum_{i=1}^{N} A_i f(x_i) \qquad A_i = \int_a^b L_i(x)\, w(x)\, dx$$

is exact for all polynomials with degree $< 2N$ (it can be shown that $a < x_i < b$ for all i).

"Proof": Let f be an arbitrary polynomial with degree $< 2N$. We can then write $f = q\varphi_N + r$ for some polynomials q and r with degree $< N$. It follows that (in simplified notation)

$$\int fw = \int q\varphi_N w + \int rw$$

where $\int q\varphi_N w = 0$ because q is a linear combination of $\varphi_0, ..., \varphi_{N-1}$, all of which are orthogonal to φ_N. The second term on the RHS can be written as

$$\int rw = \sum_{i=1}^{N} A_i r(x_i)$$

because we know that this formula is exact for polynomials with degree $< N$. But

$$\sum_{i=1}^{N} A_i r(x_i) = \sum_{i=1}^{N} A_i f(x_i) - \sum_{i=1}^{N} A_i q(x_i) \underbrace{\varphi_N(x_i)}_{0},$$

where $\varphi_N(x_i) = 0$ because of the choice of x_i's. This completes the proof.

Example: Determine x_1, x_2, A_1 and A_2 so that the formula

$$\int_{-1}^{1} f(x)\,dx \approx A_1 f(x_1) + A_2 f(x_2)$$

becomes exact for all polynomials with degree < 4.

Solution: Use Gaussian quadrature (with $N = 2$). The relevant scalar product is

$$\langle f|g\rangle = \int_{-1}^{1} f(x)\, g(x)\, dx \qquad (w(x) \equiv 1)$$

The orthogonal polynomials are known in this case, so we do not have to carry out the Gram-Schmidt procedure. They are called Legendre polynomials and the first three are given by

$$P_0(x) = 1;\ P_1(x) = x;\ P_2(x) = \tfrac{1}{2}\left(3x^2 - 1\right);$$

and the following are given by

$$(i+1)P_{i+1}(x) = (2i+1)xP_i(x) - iP_{i-1}(x)$$

The desired abscissas x_1 and x_2 are the zeros of $P_2(x)$,

$$x_1 = -\frac{1}{\sqrt{3}} \qquad \text{and} \qquad x_2 = \frac{1}{\sqrt{3}}.$$

The weights A_1 and A_2 can be determined by simply performing the integrals in the definition above. Another, slightly simpler, way is to use that the integration formula must be exact for the functions $f(x) = 1$ and $f(x) = x$, which gives

$$\begin{cases} A_1 + A_2 = \int_{-1}^{1} dx = 2 \\ -\frac{1}{\sqrt{3}} A_1 + \frac{1}{\sqrt{3}} A_2 = \int_{-1}^{1} x dx = 0 \end{cases} \quad \Rightarrow \quad A_1 = A_2 = 1$$

With these x_1, x_2, A_1 and A_2, the integration formula becomes exact for all polynomials with degree < 4.

4 Random Numbers and Monte Carlo

[NR: 7.0, 7.1, 7.2, 7.3, 7.6, 7.8 (only importance sampling in 7.8)]

Monte Carlo calculation is a widely used term that can mean different things. Common to such calculations is that random numbers are involved.

- Monte Carlo calculation can mean simulation of a process that indeed is stochastic in nature (for example, scattering processes).

- But it can also be a calculation of an integral or a sum, in which the random numbers serve just as a computational tool. This type of Monte Carlo calculation is common, for example, in statistical physics, where it can be used to calculate ensemble averages.

The plan of this section is as follows. We begin with some basics on random variables and probability theory. We then discuss methods for generating random numbers on the computer. Finally, we discuss Monte Carlo integration and summation.

4.1 Random variables

4.1.1 Some Definitions

A random variable X is defined by its probability distribution (or frequency function) $p(x) \geq 0$ which has the following properties[1]

$$p(x)dx = P\{x < X < x + dx\} = \text{the probability that } x < X < x + dx$$

$$\int_a^b p\,(x)\,dx = P\{a < X < b\}$$

$$\int_{-\infty}^{\infty} p\,(x)\,dx = 1 \qquad \text{(normalization)}$$

The average $\langle f(X)\rangle$ of an arbitrary function $f(X)$ is defined as

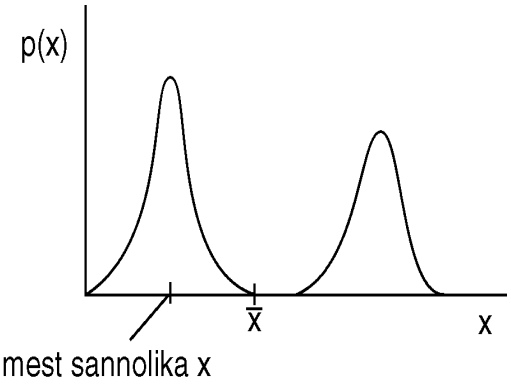
$$\langle f\,(X)\rangle = \int_{-\infty}^{\infty} f\,(x)\,p\,(x)\,dx$$

Some standard quantities for characterizing the distribution of a random variable X are:

- The *mean*

$$\langle X\rangle = \int_{-\infty}^{\infty} xp\,(x)\,dx$$

  which in general is not the same as the most probable value.



[1]Throughout the text, $P\{\cdots\}$ denotes the probability that the statement within curly brackets is true.

- The *variance*

$$\sigma_X^2 = \langle (X - \langle X \rangle)^2 \rangle = \langle X^2 \rangle - 2\langle X \rangle^2 + \langle X \rangle^2 = \langle X^2 \rangle - \langle X \rangle^2$$

  which is a measure of the fluctuations in X. The square root of the variance, σ_X, is the *standard deviation*.

- Higher-order *moments* $\langle X^n \rangle$ $(n = 1, 2, 3, \ldots)$.

Example: The *normal distribution* with mean μ and variance σ^2 is given by

$$p\,(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

(verify that the mean and variance indeed are μ and σ^2, respectively)

The simultaneous distribution $p(x, y)$ of two random variables X and Y is called the *joint distribution* of these variables, and is defined by

$$p\,(x, y)\, dxdy = P\{x < X < x + dx \;\text{ and }\; y < Y < y + dy\}\,.$$

Some useful one-dimensional distributions that can be obtained from $p(x, y)$ are:

- The *marginal distributions* in x and y:

$$p\,(x) = \int_{-\infty}^{\infty} p\,(x, y)\, dy \qquad\qquad p\,(y) = \int_{-\infty}^{\infty} p\,(x, y)\, dx$$

- The *conditional probabilities* $p(x|y)$ and $p(y|x)$.

$p(x|y)$ is the probability of x given y. For each y, $p(x|y)$ is a normalized distribution in x. The function $x \longmapsto p(x, y)$ is proportional to $p(x|y)$ but is not normalized. The precise relation between these two functions of x is given by *Bayes' rule*, which says that

$$p\,(x, y) = p\,(x|y)\, p\,(y) \qquad (= p(y|x)p(x)\,)\;.$$

Two random variables X and Y are *independent* if

$$p\,(x, y) = p\,(x)\, p\,(y)\;.$$

If so, then $p(x|y) = p(x, y)/p(y) = p(x)$ for all y.

Two random variables X and Y are *uncorrelated* if

$$\langle (X - \langle X \rangle)(Y - \langle Y \rangle) \rangle = \langle XY \rangle - \langle X \rangle \langle Y \rangle = 0 \,.$$

Independence is stronger than uncorrelatedness; if X and Y are independent, it immediately follows that

$$\langle XY \rangle = \langle X \rangle \langle Y \rangle$$

so X and Y are then uncorrelated, too.

The converse is not true, as the following example shows.

Example of uncorrelated but dependent random variables.
Let X and Y be independent binary random numbers both with possible values 0 and 1, and assume that $p_X(0) = p_X(1) = p_Y(0) = p_Y(1) = 1/2$. Put $U = X + Y$ and $V = |X - Y|$. There are four equally probable states of this system:

X	Y	probability	U	V
0	0	1/4	0	0
0	1	1/4	1	1
1	0	1/4	1	1
1	1	1/4	2	0

The marginal distributions of U and V are:

$$\begin{pmatrix} p_U(0) \\ p_U(1) \\ p_U(2) \end{pmatrix} = \begin{pmatrix} 1/4 \\ 1/2 \\ 1/4 \end{pmatrix} \qquad \begin{pmatrix} p_V(0) \\ p_V(1) \end{pmatrix} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$$

U and V are dependent, because

$$p(U = 0, V = 1) = 0 \neq p_U(0)\, p_V(1) \,.$$

U and V are nevertheless uncorrelated, because

$$\begin{cases} \langle UV \rangle = \frac{1}{4} \cdot 0 + \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 0 = \frac{1}{2} \\ \langle U \rangle = \frac{1}{4} \cdot 0 + \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 = 1 \\ \langle V \rangle = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2} \end{cases} \quad \Rightarrow \quad \langle UV \rangle = \langle U \rangle \langle V \rangle$$

4.1.2 Sums of Independent and Identically Distributed Random Variables

Suppose $X_1, \ldots, X_N$ are independent and identically distributed (iid) random variables with mean μ and variance σ^2. What is then the distribution of their sum? This question has a surprisingly simple answer in the limit $N \to \infty$.

This result, called the *central limit theorem*, says that the random variable

$$\tilde{S}_N = \frac{X_1 + ... + X_N - N\mu}{\sigma\sqrt{N}}$$

becomes normally distributed with zero mean and unit variance as $N \to \infty$.

"Proof": Let us sketch how to prove this. For this purpose, it is convenient to introduce the so-called characteristic function $\Phi(k)$ for the random variable

$$S_N = \frac{1}{N}(X_1 + \ldots + X_N) - \mu\,,$$

defined as

$$\Phi(k) = \langle \mathrm{e}^{ikS_N} \rangle\,.$$

$\Phi(k)$ is the Fourier transform of the probability distribution $p_N(s)$ of S_N, which means that $p_N(s)$ can be obtained as the inverse Fourier transform of $\Phi(k)$,

$$\Phi\,(k) = \int_{-\infty}^{\infty} \mathrm{e}^{iks} p_N(s) ds \;\Rightarrow\; p_N(s) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{e}^{-iks}\Phi\,(k)\,dk$$

For large N, $\ln\Phi(k)$ can be estimated in the following way:

$$\begin{aligned}
\ln\Phi\,(k) &= \ln\langle \mathrm{e}^{\frac{ik}{N}(X_1-\mu)} \cdots \mathrm{e}^{\frac{ik}{N}(X_N-\mu)} \rangle \\
&= N \ln\langle e^{\frac{ik}{N}(X-\mu)} \rangle \qquad\qquad \text{(the } X_i\text{'s are iid)} \\
&= N \ln\langle 1 + \frac{ik}{N}\,(X-\mu) + \frac{1}{2}\left(\frac{ik}{N}\right)^2 (X-\mu)^2 + O\left(N^{-3}\right)\rangle \\
&= N \ln\left[1 - \frac{k^2}{2N^2}\sigma^2 + O\left(N^{-3}\right)\right] \sim -\frac{k^2\sigma^2}{2N} \qquad N \to \infty
\end{aligned}$$

So, $\Phi(k) \sim \mathrm{e}^{-\frac{k^2\sigma^2}{2N}}$ as $N \to \infty$, which gives

$$\begin{aligned}
p_N(s) &\sim \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{e}^{-iks}\mathrm{e}^{-\frac{k^2\sigma^2}{2N}}\,dk = \left\{ u = k\frac{\sigma}{\sqrt{2N}} \right\} = \\
&= \frac{1}{2\pi}\frac{\sqrt{2N}}{\sigma}\int_{-\infty}^{\infty} \mathrm{e}^{-(u^2 + iu\sqrt{2N}s/\sigma)} du = \\
&= \frac{1}{2\pi}\frac{\sqrt{2N}}{\sigma}\underbrace{\int_{-\infty}^{\infty} \mathrm{e}^{-(u+i\frac{s}{\sigma}\sqrt{\frac{N}{2}})^2} du}_{\sqrt{\pi}} \cdot \mathrm{e}^{-\frac{s^2 N}{2\sigma^2}}
\end{aligned}$$

$$\boxed{p_N(s) \sim \frac{1}{\sqrt{2\pi\sigma^2/N}} \mathrm{e}^{-\frac{s^2}{2\sigma^2/N}} \qquad N \to \infty}$$

This implies that the rescaled variable $\tilde{S}_N$ indeed is normally distributed with zero mean and unit variance for large N.

It is very important to note that in this derivation we never looked at the precise form of the probability distribution of the individual X_i's — the result holds irrespective of the precise form of this distribution. This makes the central limit theorem extremely useful.

4.1.3 Confidence intervals

Suppose we make N measurements of some quantity μ and that these can be viewed as independent and identically distributed random variables X_i with

$$\begin{aligned} \langle X_i \rangle &= \mu \\ \langle X_i^2 \rangle - \langle X_i \rangle^2 &= \sigma^2 \end{aligned}$$

for $i = 1, \ldots, N$. The average

$$M_N = \frac{1}{N}\left(X_1 + ... + X_N\right)$$

provides an unbiased estimator of μ, since $\langle M_N \rangle = \mu$. A not unreasonable error bar on this estimate is $M_N \pm \sigma_{M_N}$, where σ_{M_N} is given by

$$\sigma_{M_N}^2 = \langle (M_N - \langle M_N \rangle)^2 \rangle = \frac{1}{N^2} \sum_i (\langle X_i^2 \rangle - \langle X_i \rangle^2) = \frac{\sigma^2}{N} \,.$$

However, there are two problems with this error estimate.

The first one is how to estimate the unknown quantity σ_{M_N}. This is relatively easy. If we put

$$Y_N = \frac{1}{N} \sum_i \left[X_i - \frac{1}{N} \sum_j X_j \right]^2 = \frac{1}{N} \sum_i X_i^2 - \left(\frac{1}{N} \sum_i X_i \right)^2$$

then

$$\begin{aligned} \langle Y_N \rangle &= \frac{1}{N} \sum_i \langle X_i^2 \rangle - \frac{1}{N^2} \sum_{i,j} \langle X_i X_j \rangle = \\ &= \left(\frac{1}{N} - \frac{1}{N^2} \right) \sum_i \underbrace{\langle X_i^2 \rangle}_{\sigma^2 + \mu^2} - \frac{1}{N^2} \sum_{i \neq j} \underbrace{\langle X_i X_j \rangle}_{\mu^2} = \frac{N-1}{N} \sigma^2 = (N-1)\sigma_{M_N}^2 \end{aligned}$$

This implies that

$$\frac{1}{N-1}Y_N = \frac{1}{N(N-1)}\sum_i \left[X_i - \frac{1}{N}\sum_j X_j\right]^2$$

provides an unbiased estimator of $\sigma^2_{M_N}$; that is, $\langle Y_N\rangle/(N-1) = \sigma^2_{M_N}$.

The second and more fundamental problem is how to assign a probabilistic meaning to the error bar. This can be done by making use of the central limit theorem, which says that $\tilde{S}_N = (M_N - \mu)/\sigma_{M_N}$ becomes normally distributed with zero mean and unit variance for large N. This means that

$$P\left\{|M_N - \mu| < \sigma_{M_N}\right\} = P\left\{-1 < \frac{M_N - \mu}{\sigma_{M_N}} < 1\right\} \approx \frac{1}{\sqrt{2\pi}}\int_{-1}^{1} \mathrm{e}^{-t^2/2}dx \approx 0.683$$

for large N. Hence, the probability is 68% that μ is in the interval $M_N \pm \sigma_{M_N}$. In other words, if we assign M_N a statistical error of σ_{M_N}, then we have chosen a confidence level is 68%. An error of $2\sigma_{M_N}$ corresponds to a confidence level of 95%.

4.1.4 Moments and Cumulants

Let $\Phi(k) = \langle \mathrm{e}^{ikX}\rangle$ be the characteristic function for a random variable X. A Taylor expansion of the exponential function yields

$$\Phi\left(k\right) = \sum_{n=0}^{\infty} \frac{(ik)^n}{n!}\langle X^n\rangle$$

where $\langle X^n\rangle$ is the nth moment of X.

Closely related to the moments are the so-called *cumulants*, which are denoted by $\langle X^n\rangle_c$, $n = 1, 2, \ldots$. The cumulants are defined by the Taylor expansion of $\ln\Phi(k)$:

$$\ln\Phi\left(k\right) = \sum_{n=1}^{\infty} \frac{(ik)^n}{n!}\langle X^n\rangle_c$$

By expanding the LHS of this equation in terms of the moments, it is possible to see the connection between cumulants are moments. It turns out that the nth cumulant $\langle X^n\rangle_c$ can be expressed in terms of the first n moments. For the first two cumulants, one finds (verify this)

$$\begin{aligned}\langle X\rangle_c &= \langle X\rangle \\ \langle X^2\rangle_c &= \langle X^2\rangle - \langle X\rangle^2\end{aligned}$$

It may seem unnecessary to introduce these quantities that anyhow can be expressed in terms of the moments. However, the cumulants have two important properties that make them quite useful:

- The cumulants provide a simple measure of how close a given distribution is to the normal distribution, because all cumulants of order three and higher vanish for the normal distribution.
- Cumulants of independent variables are additive; if X and Y are independent and $Z = X + Y$, then
$$\langle Z^n \rangle_c = \langle X^n \rangle_c + \langle Y^n \rangle_c$$
for all n.

4.2 Transforming Random Numbers

On the computer, one typically has access to some random number generator that delivers (pseudo) random numbers R uniformly distributed between 0 and 1,

$$p(r) = \begin{cases} 1 & 0 < r < 1 \\ 0 & \text{otherwise} \end{cases}$$

Such random numbers are sometimes called rectangularly distributed.

In this subsection, we look at how to obtain random numbers with other distributions, assuming that we have uniformly distributed random numbers at our disposal. The question of how to generate uniformly distributed random numbers is briefly discussed in the next subsection.

4.2.1 The Transformation Method

Let X be an arbitrary random variable with distribution $p_X(x)$ and suppose $Y = f(X)$, where f is a monotonous function. What is then the distribution $p_Y(y)$ of Y?

Let $P_X(x)$ and $P_Y(y)$ denote the *cumulative distributions* of X and Y, respectively; that is,

$$P_X(x) = P\{X \leq x\} = \int_{-\infty}^{x} p_X(t)\, dt \qquad \text{and} \qquad P_Y(y) = P\{Y \leq y\} = \int_{-\infty}^{y} p_Y(t)\, dt\,.$$

If $y = f(x)$ is an increasing function of x, then

$$P_X(x) = P_Y(y) \;\Rightarrow\; p_X(x) = \frac{dP_X}{dx} = \frac{dP_Y}{dy}\frac{dy}{dx} = p_Y(y)\frac{dy}{dx}$$

If, on the other hand, $y = f(x)$ is a decreasing function of x, then

$$P_X(x) = 1 - P_Y(y) \;\Rightarrow\; p_X(x) = -p_Y(y)\frac{dy}{dx}$$

This shows that the relation between $p_X(x)$ and $p_Y(y)$ can be written as

$$p_X(x) = p_Y(y)\left|\frac{dy}{dx}\right|$$

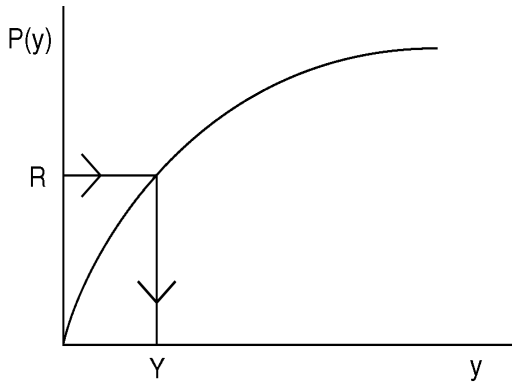
for all monotonous f (and the expression remains valid in higher dimensions if $|dy/dx|$ is thought of as the Jacobian).

Assume now that $X = R$ is uniformly distributed between 0 and 1. What should then the transformation function f look like in order for $Y = f(R)$ to have a certain distribution $p_Y(y)$? Look for an increasing f. The cumulative distributions must then satisfy

$$P_Y(y) = P_R(r) = \int_{-\infty}^{r} p_R(t)dt = r \qquad \Rightarrow \qquad y = P_Y^{-1}(r) \qquad (0 \le r \le 1)$$

which means that $f = P_Y^{-1}$. Since $1 - R$ and R have the same distribution, we can equally well take $Y = P_Y^{-1}(1 - R)$. This makes Y a decreasing function of R.

This way of generating random numbers with different distributions will be called the *transformation method*.



Example 1: The exponential distribution.

Suppose we have access to random numbers R uniformly distributed between 0 and 1 and want random numbers with the distribution

$$p(y) = \begin{cases} \lambda \mathrm{e}^{-\lambda y} & y \geq 0 \\ 0 & y < 0 \end{cases} \qquad (\lambda > 0)$$

Use the transformation method. The first step is to calculate the cumulative distribution

$$P(y) = \int_{-\infty}^{y} p\,(t)\,dt = \left[-\mathrm{e}^{-\lambda t}\right]_0^y = 1 - \mathrm{e}^{-\lambda y}$$

The transformation $r \to y$ is then obtained by solving

$$P(y) = P_R(r) = r \qquad \Rightarrow \qquad 1 - \mathrm{e}^{-\lambda y} = r \qquad \Rightarrow \qquad y = -\frac{1}{\lambda}\ln(1-r)$$

So, $Y = -\lambda^{-1}\ln(1-R)$ has the desired distribution.

Example 2: The Box-Muller method for normally distributed random numbers.

The transformation method cannot be directly applied to the distribution

$$p(y) = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{-y^2/2} \qquad \text{(assume, for simplicity, zero mean and unit variance)}$$

because we cannot get a closed expression for $P(y)$. It turns out, however, that this problem can be circumvented by considering two independent variables Y_1 and Y_2 with the same distribution $p(y)$,

$$p\,(y_1, y_2) = \frac{1}{2\pi}\mathrm{e}^{-(y_1^2+y_2^2)/2}\,.$$

In polar coordinates (ρ, θ), this distribution becomes

$$p\,(\rho,\theta) = p\,(y_1,y_2)\underbrace{\left|\frac{\partial\,(y_1,y_2)}{\partial\,(\rho,\theta)}\right|}_{\rho} = \underbrace{\rho\,\mathrm{e}^{-\rho^2/2}}_{p(\rho)} \cdot \underbrace{\frac{1}{2\pi}}_{p(\theta)}\,.$$

The fact that this distribution factorizes, $p(\rho,\theta) = p(\rho)p(\theta)$, implies that ρ and θ can be generated independently.

- The θ distribution is uniform. Take $\theta = 2\pi R_1$, where R_1 is uniformly distributed between 0 and 1.

- Use the transformation method for ρ.

$$P(\rho) = \int_0^\rho t\mathrm{e}^{-t^2/2}dt = 1 - \mathrm{e}^{-\rho^2/2} = r \;\Rightarrow\; \rho = \sqrt{-2\ln(1-r)}$$

  So, ρ can be obtained as $\rho = \sqrt{-2\ln(1-R_2)}$, where R_2 is another uniformly distributed random number between 0 and 1.

A transformation back to Cartesian coordinates gives us two independent random numbers with the desired distribution:

$$\begin{cases} Y_1 = \sqrt{-2\ln(1-R_2)}\cos 2\pi R_1 \\ Y_2 = \sqrt{-2\ln(1-R_2)}\sin 2\pi R_1 \end{cases}$$

4.2.2 The Accept/Reject Method

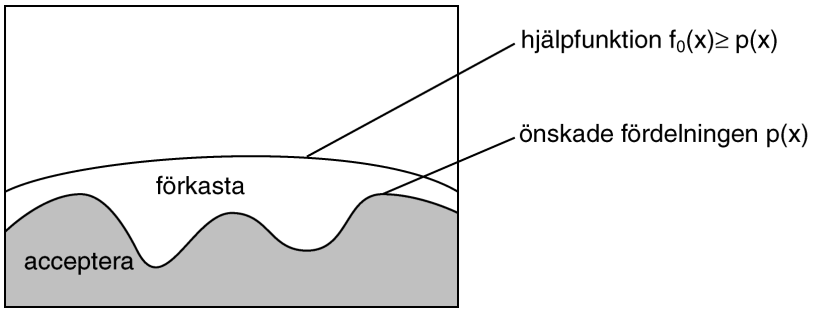
The transformation method is simple and convenient provided that a closed expression for P^{-1} can be obtained, but this is far from always the case. A more general approach, which does not require knowledge of P^{-1}, is the *accept/reject method*. To generate random numbers with a given distribution $p(x)$, this method makes use of an auxiliary function $f_0(x)$, which must satisfy $f_0(x) \geq p(x)$. f_0 should be chosen so that it is easy to obtain random numbers with the distribution $p_0(x) \propto f_0(x)$ (f_0 itself is not a normalized distribution). For a given f_0, the method is as follows (see figure).

1. Draw a point (X, Y) from the uniform distribution for the area under f_0. This can be done in two steps:

   (a) Draw X from the distribution $p_0(x) \propto f_0(x)$.

   (b) Take $Y = f_0(X)R$, where R is uniformly distributed between 0 and 1.

2. Accept (X, Y) if $Y < p(X)$, and reject otherwise.



The accepted points (X, Y) obtained this way will be uniformly distributed in the region under p. This implies that the X component of the accepted points has the distribution $p(x)$.

Example: Kahn's method for normally distributed random numbers.

Consider the distribution

$$p(x) = \begin{cases} 2\dfrac{1}{\sqrt{2\pi}}\mathrm{e}^{-x^2/2} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

(if a random number with this distribution is given a random sign, a normally distributed random number is obtained).

Use the accept/reject method with

$$f_0\,(x) = \begin{cases} \sqrt{\dfrac{2}{\pi}}\mathrm{e}^{-x+1/2} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

This choice of f_0 is OK since

$$\frac{p(x)}{f_0(x)} = \mathrm{e}^{-(x-1)^2/2} \leq 1\,,$$

and the corresponding probability distribution is

$$p_0(x) = \frac{f_0(x)}{\int_{-\infty}^{\infty} f_0(x)dx} = \begin{cases} e^{-x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

We now follow the steps above:

1. Draw (X, Y) in the area under f_0. If R_1 and R_2 are uniformly distributed between 0 and 1, we can put
   (a) $X = -\ln R_1$ (exponential distribution; see previous example).
   (b) $Y = f_0(X)R_2$.
2. Accept if
$$Y < p(X) \qquad \Leftrightarrow \qquad R_2 < \mathrm{e}^{-(X-1)^2/2}$$

This gives us a simple two-step "algorithm":

1. Put $X = -\ln R_1$.
2. Accept X if $R_2 < \mathrm{e}^{-(X-1)^2/2}$.

How efficient is this method? This depends crucially on the acceptance rate, which is the area under p (which is 1) divided by the area under f_0. In this particular example, we find an acceptance probability of

$$\frac{1}{\sqrt{\frac{2e}{\pi}} \int_0^\infty \mathrm{e}^{-x} dx} \approx 0.76\,.$$

4.2.3 The Veto Algorithm

The 'radioactive decay' type of problems is very common. In this kind of problems there is one variable t, which may be thought of as giving a kind of time axis along which different events are ordered. The probability that 'something will happen' (a nucleus decay) at time t is described by a function $f(t)$, which is non-negative in the range of t values to be studied. However, this naïve probability is modified by the additional requirement that something can only happen at time t if it did not happen at earlier times $t' < t$. (The original nucleus cannot decay once again if it already did decay; possibly the decay products may decay in their turn, but that is another question.)

The probability that nothing has happened by time t is expressed by the function $\mathcal{N}(t)$ and the differential probability that something happens at time t by $\mathcal{P}(t)$. The basic equation then is

$$\mathcal{P}(t) = -\frac{d\mathcal{N}}{dt} = f(t)\,\mathcal{N}(t)\ .$$

For simplicity, we shall assume that the process starts at time $t = 0$, with $\mathcal{N}(0) = 1$.

The above equation can be solved easily if one notes that $d\mathcal{N}/\mathcal{N} = d\ln\mathcal{N}$:

$$\mathcal{N}(t) = \mathcal{N}(0)\exp\left\{-\int_0^t f(t')\,dt'\right\} = \exp\left\{-\int_0^t f(t')\,dt'\right\} \ ,$$

and thus

$$\mathcal{P}(t) = f(t)\exp\left\{-\int_0^t f(t')\,dt'\right\} \ .$$

With $f(t) = c$ this is nothing but the textbook formula for radioactive decay. In particular, at small times the correct decay probability, $\mathcal{P}(t)$, agrees well with the input one, $f(t)$, since the exponential factor is close to unity there. At larger t, the exponential gives a dampening which ensures that the integral of $\mathcal{P}(t)$ never can exceed unity, even if the integral of $f(t)$ does. The exponential can be seen as the probability that nothing happens between the original time 0 and the final time t. In the parton-shower language of Quantum Chromo Dynamics, this corresponds to the so-called Sudakov form factor.

If $f(t)$ has a primitive function with a known inverse, it is easy to select t values correctly:

$$\int_0^t \mathcal{P}(t')\,dt' = \mathcal{N}(0) - \mathcal{N}(t) = 1 - \exp\left\{-\int_0^t f(t')\,dt'\right\} = 1 - R \ ,$$

which has the solution

$$F(0) - F(t) = \ln R \quad \Longrightarrow \quad t = F^{-1}(F(0) - \ln R) \ .$$

If $f(t)$ is not sufficiently nice, one may again try to find a better function $g(t)$, with $f(t) \leq g(t)$ for all $t \geq 0$. However to use the normal accept/reject method with this $g(t)$ would not work, since the method would not correctly take into account the effects of the exponential term in $\mathcal{P}(t)$. Instead one may use the so-called veto algorithm:

1. start with $t_0 = 0$;
2. select $t_i = G^{-1}(G(t_{i-1}) - \ln R)$, i.e. according to $g(t)$, but with the constraint that $t_i > t_{i-1}$,
3. compare a (new) R with the ratio $f(t_i)/g(t_i)$; if $f(t_i)/g(t_i) \leq R$, then return to point 2 for a new try;

4. otherwise t_i is retained as final answer.

It may not be apparent why this works. Consider, however, the various ways in which one can select a specific time t. The probability that the first try works, $t = t_1$, i.e. that no intermediate t values need be rejected, is given by

$$\mathcal{P}_0(t) = \exp\left\{-\int_0^t g(t')\,dt'\right\}\, g(t)\,\frac{f(t)}{g(t)} = f(t)\exp\left\{-\int_0^t g(t')\,dt'\right\}\ ,$$

where the ratio $f(t)/g(t)$ is the probability that t is accepted. Now consider the case where one intermediate time t_1 is rejected and $t = t_2$ is only accepted in the second step. This gives

$$\mathcal{P}_1(t) = \int_0^t dt_1 \exp\left\{-\int_0^{t_1} g(t')\,dt'\right\} g(t_1)\left[1-\frac{f(t_1)}{g(t_1)}\right]\exp\left\{-\int_{t_1}^t g(t')\,dt'\right\} g(t)\,\frac{f(t)}{g(t)}\,,$$

where the first exponential times $g(t_1)$ gives the probability that t_1 is first selected, the square brackets the probability that t_1 is subsequently rejected, the following piece the probability that $t = t_2$ is selected when starting from t_1, and the final factor that t is retained. The whole is to be integrated over all possible intermediate times t_1. The exponentials together give an integral over the range from 0 to t, just as in $\mathcal{P}_0$, and the factor for the final step being accepted is also the same, so therefore one finds that

$$\mathcal{P}_1(t) = \mathcal{P}_0(t)\int_0^t dt_1\,[g(t_1) - f(t_1)]\ .$$

This generalizes. In $\mathcal{P}_2$ one has to consider two intermediate times, $0 \le t_1 \le t_2 \le t_3 = t$, and so

$$\begin{aligned}\mathcal{P}_2(t) &= \mathcal{P}_0(t)\int_0^t dt_1\,[g(t_1)-f(t_1)]\int_{t_1}^t dt_2\,[g(t_2)-f(t_2)]\\ &= \mathcal{P}_0(t)\frac{1}{2}\left(\int_0^t [g(t')-f(t')]\,dt'\right)^2\ .\end{aligned}$$

The last equality is most easily seen if one also considers the alternative region $0 \le t_2 \le t_1 \le t$, where the rôles of t_1 and t_2 have just been interchanged, and the integral therefore has the same value as in the region considered. Adding the two regions, however, the integrals over t_1 and t_2 decouple, and become equal. In general, for $\mathcal{P}_i$, the i intermediate times can be ordered in $i!$ different ways. Therefore the total probability to accept t, in any step, is

$$\mathcal{P}(t) = \sum_{i=0}^{\infty}\mathcal{P}_i(t) = \mathcal{P}_0(t)\sum_{i=0}^{\infty}\frac{1}{i!}\left(\int_0^t [g(t')-f(t')]\,dt'\right)^i$$

$$= f(t) \exp\left\{ - \int_0^t g(t')\, dt' \right\} \exp\left\{ \int_0^t [g(t') - f(t')]\, dt' \right\}$$
$$= f(t) \exp\left\{ - \int_0^t f(t')\, dt' \right\} , \qquad (1)$$

which is the desired answer.

If the process is to be stopped at some scale $t_{\max}$, i.e. if one would like to remain with a fraction $\mathcal{N}(t_{\max})$ of events where nothing happens at all, this is easy to include in the veto algorithm: just iterate upwards in t at usual, but stop the process if no allowed branching is found before $t_{\max}$.

Usually $f(t)$ is a function also of additional variables $\vec{x}$. The methods of the preceding section are easy to generalize if one can find a suitable function $g(t, \vec{x})$ with $f(t, \vec{x}) \leq g(t, \vec{x})$. The $g(t)$ used in the veto algorithm is the integral of $g(t, \vec{x})$ over $\vec{x}$. Each time a t_i has been selected also an $\vec{x}_i$ is picked, according to the conditional probability $g(\vec{x}|t_i)$, and the $(t, \vec{x})$ point is accepted with probability $f(t_i, \vec{x}_i)/g(t_i, \vec{x}_i)$.

4.3 Random Number Generators

Random numbers are generally pseudo random numbers on the computer, obtained by a deterministic algorithm. Whether or not these random numbers are "random enough" depends on both the algorithm used and the problem at hand. The use of deterministic algorithms may seem strange, but has two major advantages, speed and reproducibility. Reproducibility can be a valuable property when debugging a program.

Most programming languages provide some built-in random number generator, but there are sometimes good reasons not to use these. One possible reason is portability. Another possible reason is that built-in random number generators are not seldom of quite poor quality.

4.3.1 The Linear Congruential Method

A widely used deterministic algorithm for generating random numbers is the linear congruential method; many random number generators rely on this algorithm or variants of it.

In its simplest version, this method uses a recursion formula of the form

$$i_{j+1} = a i_j + c \qquad \mod m$$

to generate a sequence $i_1, i_2, \ldots$ of integers. Putting $x_j = i_j/m$, we obtain a normalized sequence $x_1, x_2, \ldots$ of numbers between 0 and 1. The hope is that these numbers, for a suitable choice of the integer parameters a, c and m (see table in NR), will behave as approximately independent and uniformly distributed random numbers.

Clearly, these x_j's are not independent, and therefore it is important to test how strong the correlations between the x_j's are. One way to do this is as follows.

- x_1, $x_2, \ldots$ should be uniformly distributed on the unit interval from 0 to 1.
- (x_1, x_2), $(x_3, x_4), \ldots$ should be uniformly distributed on the unit square.

$\vdots$

- $(x_1, \ldots, x_n)$, $(x_{n+1}, \ldots, x_{2n}), \ldots$ should be uniformly distributed on the unit cube in n dimensions.

For the linear congruential method with $c = 0$, it can be shown that all possible n-tuples $(x_{j+1}, \ldots, x_{j+n})$ fall onto one of at most $(n!\, m)^{1/n}$ different hyperplanes. This number is not very large; if we, for example, take $m = 2^{32}$ and $n = 10$, then $(n!\, m)^{1/n} \approx 42$. This shows that this method exhibits correlations that definitely may cause problems in applications where many random numbers are needed.

It is also worth noting that the recursion formula above is periodic, with a period of m or smaller, so m should be very large.

There are methods that are better than this one; for a discussion, see NR. Nevertheless, it should be kept in mind that random number generators are not perfect, and they should be chosen with care in applications where lots of random numbers are needed.

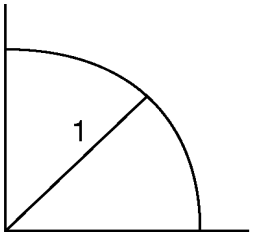
4.4 Monte Carlo Integration and Summation

Example: Monte Carlo calculation of π.

Consider the first quadrant of the unit circle. Its area ($= \pi/4$) can be written as

$$I = \int_{\text{first quadrant}} dxdy$$

Let us see how this integral can be estimated by using random numbers.



Suppose we have N random points drawn from the uniform distribution on the unit square. For each point i, we introduce a binary random variable χ_i such that $\chi_i = 1$ if point i is in the first quadrant of the unit circle, and $\chi_i = 0$ otherwise. The mean of each χ_i is I, the area of the first quadrant (since the area of the square is 1). It follows that

$$\frac{1}{N}\sum_{i=1}^{N}\chi_i = \frac{1}{N}\{\text{no. of points in the first quadrant}\} \to I \qquad N \to \infty$$

which gives us a method for estimating I (and thereby π).

Let us now generalize this by considering

$$I = \int f(x)p(x)dx$$

where $f(x)$ is an arbitrary function and $p(x)$ is some probability distribution. Assume that $X_1, \ldots, X_N$ are independent random variables, all with the distribution $p(x)$. $f(X_1), \ldots, f(X_N)$ are then independent and identically distributed random variables. This means, according to the central limit theorem, that

$$I_N = \frac{1}{N}\sum_{i=1}^{N} f(X_i)$$

is approximately normally distributed for large N, with

- mean $\langle I_N \rangle = \int f(x)p(x)dx = I$
- variance $\sigma_N^2 = (\langle f(X)^2 \rangle - \langle f(X) \rangle^2)/N \to 0$ as $N \to \infty$

Hence, I_N may be used as an estimator of I for large N. The method can be immediately generalized to higher dimensions.

Sums can be dealt with in a similar way. Consider

$$S = \sum_i f(i)p(i)$$

where $f(i)$ is an arbitrary function and $p(i)$ is some discrete probability distribution. If $I_1, \ldots, I_N$ are independent random numbers with the distribution $p(i)$, we can estimate S by using

$$S \approx S_N = \frac{1}{N} \sum_{k=1}^{N} f(I_k)$$

for large N.

4.4.1 Convergence Rate

What about the efficiency of Monte Carlo integration? Let us compare the efficency of this method in one dimension, $D = 1$, with that of the Simpson rule.

Consider an integral over an interval of length L, and let T_ε denote the amount of computer time needed to achieve an accuracy of $\mathcal{O}(\varepsilon)$.

- Simpson's rule

$$\begin{cases} \varepsilon \sim h^4 & (h \text{ step size}) \\ T_\varepsilon \propto \{\text{no. of function values}\} \sim L/h \end{cases} \qquad \Rightarrow \qquad T_\varepsilon \sim \varepsilon^{-1/4}$$

- Monte Carlo

$$\begin{cases} \varepsilon \sim N^{-1/2} & (N \text{ no. of points}) \\ T_\varepsilon \propto N \end{cases} \qquad \Rightarrow \qquad T_\varepsilon \sim \varepsilon^{-2}$$

This comparison shows that the convergence of the Monte Carlo method is typically much slower than that of the Simpson rule for $D = 1$. The strength of the Monte Carlo method is its generality. If, for example, D is large or the boundary of the integration region is complex, there often are few alternatives to Monte Carlo.

4.4.2 Importance Sampling

A Monte Carlo calculation of a given integral

$$I = \int f(x)dx$$

may use random numbers drawn from any probability distribution $p(x) > 0$. In fact, with $X_1, \ldots, X_N$ drawn from any given $p(x) > 0$, we may estimate I as

$$I = \int \frac{f(x)}{p(x)} p(x)dx \approx I_N^p = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$$

The mean of the estimator I_N^p is, of course, independent of p, $\langle I_N^p \rangle = I$. The variance $\langle (I_N^p)^2 \rangle - \langle I_N^p \rangle^2$ depends, by contrast, strongly on p. In order to have a reasonable performance, it is therefore crucial to make a careful choice of p.

In principle, it is known what the optimal choice of p is, namely $p(x) \propto |f(x)|$ (see NR 7.8). In practice, this is of little help because finding the proportionality constant in this relation is as difficult as finding the integral we want to compute. However, it is often possible to make an educated guess of p that is useful, although not perfect.

Example: The Ising model.

The Ising model is a simple model for ferromagnetism. The system consists of N binary spin variables $\sigma_i = \pm 1$ that live on a lattice. In the absence of an external magnetic field, the energy E of a configuration $\sigma = (\sigma_1, \ldots, \sigma_N)$ is given by

$$E = -J \sum_{\langle ij \rangle} \sigma_i \sigma_j$$

where the sum runs over all nearest-neighbor pairs on the lattice. The thermodynamic behavior of the system is governed by the Boltzmann weight

$$p(\sigma) \propto \mathrm{e}^{-E(\sigma)/kT}$$

where k is Boltzmann's constant and T the temperature. The average of an observable O, the total magnetization say, at temperature T is given by

$$\langle O \rangle = \sum_{\sigma} O(\sigma) p(\sigma) = \frac{\sum_{\sigma} O(\sigma) \mathrm{e}^{-E(\sigma)/kT}}{\sum_{\sigma} \mathrm{e}^{-E(\sigma)/kT}}$$

How can we calculate such an average?

- Exactly, by exhaustive enumeration of all possible states? No, because the number of states, 2^N, is too large; already for $N = 100$ (which is a very small system), there are $2^{100} \sim (10^3)^{10} = 10^{30}$ possible states.

- "Naive" Monte Carlo? Here, we would draw configurations $\sigma^{(1)}, \ldots, \sigma^{(J)}$ from the uniform distribution $p_0(\sigma) = 1/2^N = \text{constant}$, and estimate

$$\langle O \rangle \approx \left(\frac{1}{J} \sum_{j=1}^{J} O(\sigma^{(j)}) \mathrm{e}^{-E(\sigma^{(j)})/kT} \right) \times \left(\frac{1}{J} \sum_{j=1}^{J} \mathrm{e}^{-E(\sigma^{(j)})/kT} \right)^{-1}$$

  But E is an extensive quantity, which means that there will be huge fluctuations in $\mathrm{e}^{-E/kT}$. As a result, the variance is very large and the convergence very slow.

- Importance sampling. If instead we draw $\sigma^{(1)}, \ldots, \sigma^{(J)}$ from the Boltzmann distribution $p(\sigma)$, then we can use an estimate

$$\langle O \rangle \approx \frac{1}{J} \sum_{j=1}^{J} O(\sigma^{(j)})$$

  that does not contain any Boltzmann factor $\mathrm{e}^{-E/kT}$. This is typically an enormous improvement. The next question then is how to generate Boltzmann distributed configurations. A widely used method for this is the Metropolis algorithm.

4.5 The Metropolis Algorithm

Consider a system with state or configuration space T, and suppose we want to sample some distribution $\tilde{p}(\sigma)$, $\sigma \in T$. For simplicity, we assume T to be discrete (as in the Ising model).

The Metropolis algorithm can be thought of as a guided random walk in the state space T,

$$\sigma_1 \to \sigma_2 \to \sigma_3 \to \ldots$$

Here, σ_n denotes the state of the system at "time" n. The guidance is such that the probability distribution p_n of σ_n approaches $\tilde{p}$ for large n; that is,

$$\lim_{n \to \infty} p_n(\sigma) = \tilde{p}(\sigma)\,. \tag{2}$$

This is meant to hold irrespective of what the initial distribution p_1 is.

A fundamental property of the Metropolis algorithm is that p_{n+1} is entirely determined by p_n; in order to determine p_{n+1}, we do not have to know where the system was at time $n-1$, $n-2, \ldots$. A stochastic process with this property is called a *Markov chain*. For a Markov chain, the time evolution can be described in terms of a transition matrix $W(\sigma, \sigma')$; $W(\sigma, \sigma')$ being the conditional probability of finding the system in state σ at time $n+1$, given that it was in state σ' at time n. Another important property of the Metropolis algorithm is that the transition matrix $W(\sigma, \sigma')$ does not change with time n.

These two properties imply that the time evolution of p_n is given by a simple vector-matrix equation,

$$p_{n+1}(\sigma) = \sum_{\sigma'} W(\sigma, \sigma') p_n(\sigma') \tag{3}$$

with a constant (n independent) matrix W.

The key question now is how to ensure that $p_n \to \tilde{p}$ as $n \to \infty$, equation (1). Useful information about this can be obtained from the theory for general Markov chains with constant transition matrices ("stationary" Markov chains). For a general process of this type, it can be shown that equation (1) does hold independent of the initial distribution if the following two conditions are met:

1. The distribution $\tilde{p}$ is stationary. This means that $p_n = \tilde{p} \Rightarrow p_{n+1} = \tilde{p}$. Another way to say this is that $\tilde{p}$ should be an eigenvector of the matrix W with eigenvalue 1; that is,

$$\tilde{p}(\sigma) = \sum_{\sigma'} W(\sigma, \sigma') \tilde{p}(\sigma') \qquad \text{(for all } \sigma\text{)}$$

2. The process is ergodic. Loosely speaking, this means that each state can be reached from each other state. A somewhat more precise formulation of this condition can be found in the hand-out for the first computer exercise.

We are not going to prove that these two requirements are sufficient to ensure that equation (1) holds, but to give an idea of how it works, we will prove two weaker statements. For this purpose, we need to define the distance between two arbitrary distributions p_a and p_b, which can be taken as

$$\|p_a - p_b\| = \sum_{\sigma} |p_a(\sigma) - p_b(\sigma)|$$

Statement 1: Suppose $\tilde{p}$ is stationary. Then the distance $\|p_n - \tilde{p}\|$ is a non-increasing function of n.

Proof: Using equation (2) and that $\tilde{p}$ is stationary, we obtain

$$\begin{aligned}
\|p_{n+1} - \tilde{p}\| &= \sum_\sigma |\, p_{n+1}(\sigma) - \tilde{p}(\sigma)| \\
&= \sum_\sigma \Big| \sum_{\sigma'} W(\sigma, \sigma')(p_n(\sigma') - \tilde{p}(\sigma')) \Big| \\
&\leqslant \sum_\sigma \sum_{\sigma'} W(\sigma, \sigma') |p_n(\sigma') - \tilde{p}(\sigma')| \\
&= \|\, p_n - \tilde{p}\| \qquad\qquad \Big(\sum_\sigma W(\sigma, \sigma') = 1 \Big)
\end{aligned}$$

Statement 2: Suppose, in addition to the stationarity of $\tilde{p}$, that $W(\sigma, \sigma') > 0$ for all σ, σ' and that $p_n \neq \tilde{p}$. Then the inequality above is strict,

$$\|\, p_{n+1} - \tilde{p}\| < \|\, p_n - \tilde{p}\|$$

Proof: That $p_n \neq \tilde{p}$ implies that $p_n(\sigma') - \tilde{p}(\sigma')$ takes on both positive and negative values, since p_n and $\tilde{p}$ both are normalized distributions. The same must then be true for $W(\sigma, \sigma')(p_n(\sigma') - \tilde{p}(\sigma'))$, since $W(\sigma, \sigma') > 0$ for all σ, σ'. From this follows that the inequality must be strict.

The Metropolis algorithm provides a simple and general way to ensure that condition 1 above is met. This is achieved by designing the basic update of the system in such a way that detailed balance is fulfilled, a condition that is stronger than the condition 1 above. How this is done is discussed in the hand-out for the first computer exercise.

Comments

For simplicity, we have here considered discrete systems. The Metropolis algorithm can be easily applied to continuous systems, too.

The fact that the states generated by the Metropolis algorithm are not independent makes this method fundamentally different from methods such as the simple transformation and accept/reject methods. Methods like these two are sometimes called *static*, and methods like the Metropolis algorithm are then called *dynamic*.

5 Optimization (minimization/maximization)

[NR: 9.6, 10.0, 10.1, 10.4, 10.5, 10.6, 10.9]

Optimization is a wide field. Sometimes there exists a well-established method that can be used in a black-box manner, but many optimization problems are true challenges.

Suppose we want to minimize some function $f(\mathbf{x})$ in D dimensions, $\mathbf{x} \in R^D$. How to proceed depends on a number of things, such as

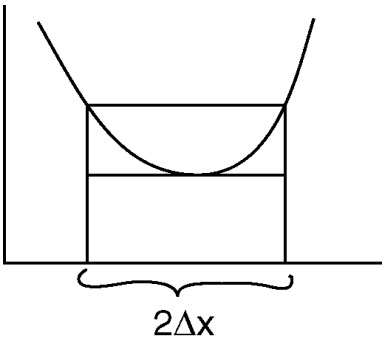
- What is the dimensionality D?
- Are there constraints on $\mathbf{x}$?
- Is the problem linear or non-linear?
- Is f such that minimization is a smooth downhill process, or are there traps in the form of local minima?
- Do we want the global minimum of f, or is it sufficient to make a local minimization?
- Do we have access to derivatives of f?

In this section, we first look at a simple and general scheme called simulated annealing that can be tried if the aim is to make a global optimization (section 5.1). We then discuss a few different methods for local optimization; first, the downhill simplex method without derivatives (section 5.2), and then the conjugate-gradient and quasi-Newton methods that do use derivatives (sections 5.3.2 and 5.3.3). We end with a few words on optimization in the presence of constraints (section 5.4).

But first of all, a general remark on precision. Suppose we want to minimize a function $f(x)$ (assume, for simplicity, $D = 1$). A Taylor expansion about the minimum $x_{\min}$ gives

$$f(x) \approx f(x_{\min}) + \tfrac{1}{2}\,(x - x_{\min})^2\, f''(x_{\min}) \qquad\qquad (x \text{ near } x_{\min})$$

since $f'(x_{\min}) = 0$. Near $x_{\min}$, the roundoff error in $f(x)$ is $\gtrsim \varepsilon |f(x_{\min})|$, where ε is the relative floating-point precision. If now $|x - x_{\min}|$ is so small that $|f(x) - f(x_{\min})|$ is comparable to the roundoff error, then we cannot expect to be able to come closer to $x_{\min}$, irrespective of what search method we use. This gives an estimate of the smallest possible error, Δx, in $x_{\min}$:



$$\tfrac{1}{2}\left(\Delta x\right)^2 f''\left(x_{\min}\right) \gtrsim \varepsilon \left|f\left(x_{\min}\right)\right| \;\Rightarrow\; \Delta x \gtrsim \sqrt{\varepsilon}\sqrt{\frac{2f\left(x_{\min}\right)}{f''\left(x_{\min}\right)}}$$

This shows that the error in $x_{\min}$ ($\gtrsim \varepsilon^{1/2}$) is typically much larger than that in $f(x_{\min})$ ($\gtrsim \varepsilon$).

5.1 Global Optimization. Simulated annealing.

For functions with many local minima, global minimization is generally very hard, because the system tends to get trapped in local minima. The simulated-annealing method is an attempt to circumvent this problem by using Metropolis dynamics. The function we want to minimize is then thought of as an energy E. With Metropolis dynamics, steps upwards in E do occur (probability $\mathrm{e}^{-\Delta E/kT}$), which is needed in order for the system to be able to escape from local minima. Note also that the Boltzmann factor $\propto \mathrm{e}^{-E/kT}$ gives a high statistical weight to low-E states at low temperature.

In a simulated-annealing run, the "temperature" T serves as a control parameter. The calculations are started at a high temperature where the mobility of the system is high. The temperature is then gradually decreased till the system freezes. The hope is that the final frozen state will be the global energy minimum.

Of course, this will not always be the case, so it is essential to repeat the experiment for many different initial conditions. A key parameter in the simulations is the rate of cooling. If the cooling is too rapid, it is likely that the system gets stuck in a local minimum with non-minimal E.

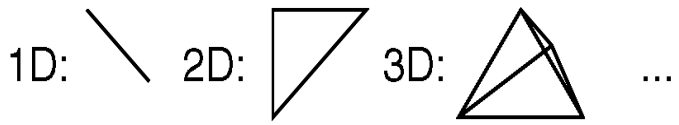
The simulated-annealing method is young compared to the Metropolis method (proposed in 1983 and 1953, respectively) but has been applied to a wide range of physical and non-physical problems. Non-physical applications include various combinatorial

optimization problems, such as the traveling salesman problem. In this problem, there are N cities to be visited, and each city is to be visited precisely once. The task is to minimize the distance that the salesman has to travel. Solving this problem exactly is possible only for small N, because the number of alternative routes, $(N-1)!$, grows rapidly with N. With simulated annealing it becomes possible to study larger N. A simulated-annealing program for this problem can be found in NR 10.9.

5.2 Local Optimization without Derivatives. The Downhill Simplex Method

When searching for the minimum of a function $f(\mathbf{x})$ in D dimensions, it is of help to know the gradient $\nabla f(\mathbf{x})$, because $-\nabla f(\mathbf{x})$ is the direction in which the decrease of $f(\mathbf{x})$ is fastest (locally). In what direction should one search if the gradient $\nabla f(\mathbf{x})$ is not available?

A simple and general scheme that does not rely on information on the gradient is the downhill simplex method. A simplex is a geometrical object with one more vertex than dimension: a line segment for $D = 1$, a triangle for $D = 2$, a tetrahedron for $D = 3$, and so on. In the downhill simplex method, the simplex is a dynamic object that can grow and shrink. When it reaches a minimum, it gives up and shrinks down around it.



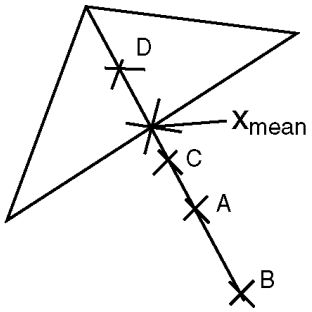
The method proceeds iteratively, starting from some simplex with vertices $\mathbf{x}_1, \ldots, \mathbf{x}_{D+1}$ which we assume ordered so that $f(\mathbf{x}_{D+1}) \leq f(\mathbf{x}_D) \leq \ldots \leq f(\mathbf{x}_1)$. The elementary move can be seen as an attempt to improve the worst point, $\mathbf{x}_1$, and is as follows.

Calculate the center of the face defined by the points $\mathbf{x}_2, \ldots, \mathbf{x}_{D+1}$,

$$\mathbf{x}_{\text{mean}} \equiv \frac{1}{D} \sum_{i=2}^{D+1} \mathbf{x}_i$$

Since all these D points are better than $\mathbf{x}_1$, it makes sense to try to move $\mathbf{x}_1$ in the direction of $\mathbf{x}_{\text{mean}}$. Therefore, the next step is to reflect $\mathbf{x}_1$ across this face to

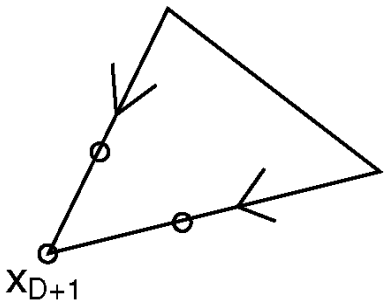
$$\mathbf{x}_a = \mathbf{x}_{\text{mean}} + (\mathbf{x}_{\text{mean}} - \mathbf{x}_1)$$



Whether this point is accepted or not depends on the value of $f(\mathbf{x}_a)$:

- If $f(\mathbf{x}_{D+1}) < f(\mathbf{x}_a) < f(\mathbf{x}_2)$, replace $\mathbf{x}_1$ by $\mathbf{x}_a$.
- If $f(\mathbf{x}_a) < f(\mathbf{x}_{D+1})$, try a larger step (the direction seems good) to
$$\mathbf{x}_b = \mathbf{x}_{\text{mean}} + 2(\mathbf{x}_{\text{mean}} - \mathbf{x}_1)$$
$\mathbf{x}_1$ is then replaced by the best of $\mathbf{x}_a$ and $\mathbf{x}_b$.
- If $f(\mathbf{x}_a) > f(\mathbf{x}_2)$, try instead a smaller step to
$$\mathbf{x}_c = \mathbf{x}_{\text{mean}} + \tfrac{1}{2}(\mathbf{x}_{\text{mean}} - \mathbf{x}_1)$$
If $f(\mathbf{x}_c) < f(\mathbf{x}_2)$, replace $\mathbf{x}_1$ by $\mathbf{x}_c$. Otherwise, try an even smaller step to
$$\mathbf{x}_d = \mathbf{x}_{\text{mean}} - \tfrac{1}{2}(\mathbf{x}_{\text{mean}} - \mathbf{x}_1)$$
If $f(\mathbf{x}_d) < f(\mathbf{x}_2)$, replace $\mathbf{x}_1$ by $\mathbf{x}_d$. If this attempt also fails, the simplex is too large to give a good idea of what direction to choose. Therefore, we now give up and shrink all the vertices towards the best one
$$\mathbf{x}_i \to \mathbf{x}_i + \tfrac{1}{2}(\mathbf{x}_{D+1} - \mathbf{x}_i) \qquad i = 1, \ldots, D$$

This update is iterated untill the values are no longer improving, according to some stopping criterion.



5.3 Local Optimization with Derivatives

5.3.1 Successive Line Minimizations

Many methods for multidimensional minimization are based on successive line minimizations. Usually, the gradient of the function is used to decide on what lines to be considered. The actual minimization along each of these lines may or may not use derivatives.

For one-dimensional minimization, there exist methods that are relatively robust and fast. Assume that we have such a method at our disposal and want to minimize a function $f(\mathbf{x})$, $\mathbf{x} \in R^D$. We may then proceed as follows.

1. Pick a starting point $\mathbf{x}_0$ and a direction $\mathbf{h}_0$.
2. Determine λ_0 by minimization of f along the line $\lambda \mapsto \mathbf{x}_0 + \lambda \mathbf{h}_0$.
3. Put $\mathbf{x}_1 = \mathbf{x}_0 + \lambda_0 \mathbf{h}_0$ and select a new direction $\mathbf{h}_1$.
4. Determine λ_1 by minimization of f along the line $\lambda \mapsto \mathbf{x}_1 + \lambda \mathbf{h}_1$.
5. ...

But what directions $\mathbf{h}_i$ should we use?

- The D basis vectors $\hat{\mathbf{e}}_i$?
  No, we don't want to be restricted to these directions (see figure in NR 10.5).
- $\mathbf{h}_i = -\nabla f(\mathbf{x}_i)$?
  This is called *steepest descent* and may seem like a natural choice. However, this method has an unwanted property that tends to make it inefficient (see figure in NR 10.6): consecutive directions $\mathbf{h}_i$ and $\mathbf{h}_{i+1}$ are, by construction, orthogonal. To see this, note that $\mathbf{x}_{i+1}$ is obtained by minimizing f along the line $\lambda \mapsto \mathbf{x}_i + \lambda \mathbf{h}_i$, which implies that

$$0 = \frac{df(\mathbf{x}_i + \lambda \mathbf{h}_i)}{d\lambda}\Big|_{\lambda=\lambda_i} = \nabla f(\mathbf{x}_{i+1}) \cdot \frac{d(\mathbf{x}_i + \lambda \mathbf{h}_i)}{d\lambda} = (-\mathbf{h}_{i+1}) \cdot \mathbf{h}_i$$

Conjugate directions

Near minima, quadratic approximations are generally good. Let us therefore consider a quadratic function

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}\cdot\mathbf{A}\mathbf{x} - \mathbf{b}\cdot\mathbf{x} + c \qquad \begin{cases} \mathbf{x} \text{ and } \mathbf{b} \in R^D \\ \mathbf{A} \text{ symmetric, positive definite } D\times D \text{ matrix} \\ c \text{ number} \end{cases}$$

($\mathbf{A}$ is assumed positive definite so that f has a minimum). If we were to minimize this f by successive line minimizations, how should we then choose the directions $\mathbf{h}_i$?

When minimizing f, we are searching for an $\mathbf{x}$ at which all the components of the vector $\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$ vanish. By construction, one component of $\nabla f(\mathbf{x}_{i+1})$ has to be zero, $\nabla f(\mathbf{x}_{i+1})\cdot\mathbf{h}_i = 0$. Can we by a suitable choice of the $\mathbf{h}_i$'s see to that $\nabla f(\mathbf{x}_{i+1})\cdot\mathbf{h}_{i-1} = 0$, too?

$$\begin{aligned} \mathbf{h}_{i-1}\cdot\nabla f(\mathbf{x}_{i+1}) &= \mathbf{h}_{i-1}\cdot[\mathbf{A}(\mathbf{x}_i + \lambda_i\mathbf{h}_i) - \mathbf{b}] = \\ &= \mathbf{h}_{i-1}\cdot[\nabla f(\mathbf{x}_i) + \lambda_i\mathbf{A}\mathbf{h}_i] \\ &= \lambda_i\mathbf{h}_{i-1}\cdot\mathbf{A}\mathbf{h}_i \end{aligned}$$

So,

$$\mathbf{h}_{i-1}\cdot\nabla f(\mathbf{x}_{i+1}) = 0 \qquad \text{if} \qquad \boxed{\mathbf{h}_{i-1}\cdot\mathbf{A}\mathbf{h}_i = 0}$$

$\mathbf{h}_{i-1}$ and $\mathbf{h}_i$ are called *conjugate, or* $\mathbf{A}$ *orthogonal*, if this condition is fulfilled.

5.3.2 The Conjugate Gradient Method

This is a method based on successive line minimizations in which the directions $\mathbf{h}_i$ are conjugate if the function is quadratic. This makes the method efficient for such functions. For a general function, the method is expected to work well if we are sufficiently close to the minimum.

The algorithm is simple to formulate. For convenience, put $\mathbf{g}_i = -\nabla f(\mathbf{x}_i)$, where f is the function to be minimized. Let $\mathbf{x}_0$ be an arbitrary starting point and take $\mathbf{h}_0 = \mathbf{g}_0$. Pairs ($\mathbf{x}_1$, $\mathbf{h}_1$), ($\mathbf{x}_2$, $\mathbf{h}_2$),... are then generated by using the recursion formulas:

$$\begin{cases} \mathbf{x}_{i+1} = \mathbf{x}_i + \lambda_i\mathbf{h}_i & \text{where } \lambda_i \text{ is determined by line minimization} \\ \mathbf{h}_{i+1} = \mathbf{g}_{i+1} + \gamma_i\mathbf{h}_i & \text{where } \gamma_i = \dfrac{\mathbf{g}_{i+1}\cdot\mathbf{g}_{i+1}}{\mathbf{g}_i\cdot\mathbf{g}_i} \end{cases}$$

(with $\gamma_i = 0$ this would be the steepest descent method).

The choice of the parameter γ_i is such that if

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}\cdot\mathbf{A}\mathbf{x} - \mathbf{b}\cdot\mathbf{x} + c$$

(as above), then $\mathbf{h}_i \cdot \mathbf{A}\mathbf{h}_{i+1} = 0$ and also $\mathbf{h}_{i+1} \cdot \mathbf{A}\mathbf{h}_i = 0$, since $\mathbf{A}$ is symmetric.

In fact we will show something more general, namely that for such f, the following orthogonality relationships hold

$$\begin{array}{ll} \mathbf{h}_i \cdot \mathbf{A}\mathbf{h}_j = 0 & (i \neq j) \\ \mathbf{g}_i \cdot \mathbf{g}_j = 0 & (i \neq j) \\ \mathbf{g}_i \cdot \mathbf{h}_j = 0 & (i > j) \end{array}$$

and that we can write

$$\lambda_i = \frac{\mathbf{g}_i \cdot \mathbf{h}_i}{\mathbf{h}_i \cdot \mathbf{A}\mathbf{h}_i} = \frac{\mathbf{g}_i \cdot \mathbf{g}_i}{\mathbf{h}_i \cdot \mathbf{A}\mathbf{h}_i} \qquad (i = 0,\, 1, \ldots)$$

We will prove this using induction, i.e. assuming it holds for all $i, j \leq n$, we will show that it holds also for $i, j \leq n + 1$.

First we look at some general relationships, noting that $\mathbf{h}_i \cdot \mathbf{g}_{i+1} = 0$ by construction.

So, we have

$$0 = \mathbf{h}_i \cdot \mathbf{g}_{i+1} = \mathbf{h}_i \cdot (\mathbf{b} - \mathbf{A}\mathbf{x}_{i+1}) = \mathbf{h}_i \cdot (\mathbf{b} - \mathbf{A}\mathbf{x}_i - \lambda_i \mathbf{A}\mathbf{h}_i) = \mathbf{h}_i \cdot (\mathbf{g}_i - \lambda_i \mathbf{A}\mathbf{h}_i) = \mathbf{h}_i \cdot \mathbf{g}_i - \lambda_i \mathbf{h}_i \cdot \mathbf{A}\mathbf{h}_i$$

and

$$\lambda_i = \frac{\mathbf{g}_i \cdot \mathbf{h}_i}{\mathbf{h}_i \cdot \mathbf{A}\mathbf{h}_i}.$$

Also,

$$\mathbf{g}_i \cdot \mathbf{h}_i = \mathbf{g}_i \cdot \mathbf{g}_i + \gamma_{i-1} \mathbf{g}_i \cdot \mathbf{h}_{i-1} = \mathbf{g}_i \cdot \mathbf{g}_i$$

so we can write

$$0 = \mathbf{h}_{i+1} \cdot \mathbf{A}\mathbf{h}_i = (\mathbf{g}_{i+1} + \gamma_i \mathbf{h}_i) \cdot \frac{\mathbf{g}_i - \mathbf{g}_{i+1}}{\lambda_i}$$

where we know that $\mathbf{g}_{i+1} \cdot \mathbf{g}_i = 0$, $\mathbf{h}_i \cdot \mathbf{g}_{i+1} = 0$ and $\mathbf{h}_i \cdot \mathbf{g}_i = \mathbf{g}_i \cdot \mathbf{g}_i$, so

$$0 = \frac{1}{\lambda_i}(\gamma_i \mathbf{g}_i \cdot \mathbf{g}_i - \mathbf{g}_{i+1} \cdot \mathbf{g}_{i+1}) \Rightarrow \gamma_i = \frac{\mathbf{g}_{i+1} \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i \cdot \mathbf{g}_i}$$

Now, assuming that $\mathbf{g}_i \cdot \mathbf{g}_j = \mathbf{h}_i \cdot \mathbf{A}\mathbf{h}_j = 0$ for all $i \neq j$ with $i, j \leq n$ (this is certainly true for $n = 1$) we need to show that it is true for all $i \neq j$ with $i, j \leq n + 1$:

$$\mathbf{g}_{n+1} \cdot \mathbf{g}_i = \mathbf{g}_n \cdot \mathbf{g}_i - \lambda_n \mathbf{h}_n \cdot \mathbf{A}\mathbf{g}_i = -\lambda_n \mathbf{h}_n \cdot \mathbf{A}\mathbf{g}_i.$$

For $i = 0$ and $i = n$ this is zero, otherwise we have

$$= -\lambda_n \mathbf{h}_n \cdot \mathbf{A}(\mathbf{h}_i - \gamma_{i-1} \mathbf{h}_{i-1}) = 0.$$

Similarly we have for $\mathbf{h}_{n+1} \cdot \mathbf{A}\mathbf{h}_i$, which is zero for $i = n$, otherwise

$$\mathbf{h}_{n+1} \cdot \mathbf{A}\mathbf{h}_i = \mathbf{g}_{n+1} \cdot \mathbf{A}\mathbf{h}_i + \gamma_n \mathbf{h}_n \cdot \mathbf{A}\mathbf{h}_i = \mathbf{g}_{n+1} \cdot \frac{\mathbf{g}_i - \mathbf{g}_{i+1}}{\lambda_i} = 0.$$

We also need to show that everything holds for $n = 1$:

Clearly $\mathbf{g}_1 \cdot \mathbf{h}_0 = \mathbf{g}_1 \cdot \mathbf{g}_0 = 0$. Also

$$\mathbf{h}_0 \cdot \mathbf{A}\mathbf{h}_1 = \mathbf{h}_1 \cdot \mathbf{A}\mathbf{h}_0 = (\mathbf{g}_1 + \gamma_0 \mathbf{g}_0) \cdot \frac{\mathbf{g}_0 - \mathbf{g}_1}{\lambda_0} = \frac{1}{\lambda_0}(\gamma_0 \mathbf{g}_0 \cdot \mathbf{g}_0 - \mathbf{g}_1 \cdot \mathbf{g}_1) = 0\,.$$

In particular, this shows that the vectors $\mathbf{g}_i$ are pairwise orthogonal. This can hold for at most D different $\mathbf{g}_i \neq \mathbf{0}$ in D dimensions. Hence, $\mathbf{g}_i = \mathbf{0}$ for some $i \leq D$. But if $\mathbf{g}_i = \mathbf{0}$, then $\mathbf{x}_i$ is the solution we want. This means that the method needs at most D steps to find the minimum. This holds when the function is quadratic. For a general f the orthogonality relations are at best approximate, and there is no guarantee of convergence within a finite number of steps. In this case, the recursion formulas are iterated untill some stopping criterion is fulfilled. Note that the algorithm is written in such a way that it can be directly applied to general functions f.

Solution of linear equations by the conjugate gradient method

Suppose we want to solve a linear equation system

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\mathbf{A}$ is a symmetric, positive definite matrix. One approach to this problem is to make use of the fact that the solution must be the minimum $\mathbf{x}_{\min}$ of the function

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x} \cdot \mathbf{A}\mathbf{x} - \mathbf{b} \cdot \mathbf{x}\,,$$

because $\mathbf{x}_{\min}$ satisfies

$$\nabla f(\mathbf{x}_{\min}) = \mathbf{A}\mathbf{x}_{\min} - \mathbf{b} = 0$$

So, in principle, we can solve our linear algebra problem by using a minimization method such as the conjugate-gradient method. Is this a good approach? It can be computationally convenient if the matrix $\mathbf{A}$ is large but sparse (that is, $\mathbf{A}$ has many elements but relatively few of them are nonzero). The reason that the conjugate gradient method is interesting in this case is that the matrix $\mathbf{A}$ enters the algorithm only through expressions of the type $\mathbf{A} \times$ (vector), which makes it relatively easy to take advantage of the sparseness of the matrix.

5.3.3 The Quasi-Newton Method

Minimizing $f(\mathbf{x})$ can be viewed as solving $\nabla f(\mathbf{x}) = 0$, which is a system of D generally non-linear equations,

$$\begin{cases} \partial_1 f(\mathbf{x}) = 0 \\ \vdots \\ \partial_D f(\mathbf{x}) = 0 \end{cases}$$

Suppose we are at some point $\mathbf{x}$. A Taylor expansion about this point gives

$$f(\mathbf{x}') = f(\mathbf{x}) + (\mathbf{x}' - \mathbf{x}) \cdot \nabla f(\mathbf{x}) + \tfrac{1}{2}(\mathbf{x}' - \mathbf{x}) \cdot \mathbf{A}(\mathbf{x})(\mathbf{x}' - \mathbf{x}) + \ldots$$

where $\mathbf{A}(\mathbf{x})$ is the so-called Hessian matrix with elements $A_{ij}(\mathbf{x}) = \partial_i \partial_j f(\mathbf{x})$. By taking the gradient with respect to the primed variables of both sides of this equation, we obtain

$$\nabla f\,(\mathbf{x}') = 0 + \nabla f(\mathbf{x}) + \mathbf{A}(\mathbf{x})(\mathbf{x}' - \mathbf{x}) + \ldots$$

From this we see that if the omitted higher-order terms can be neglected, and if $\mathbf{x}'$ is given by

$$\mathbf{x}' - \mathbf{x} = -\mathbf{A}^{-1}(\mathbf{x}) \cdot \nabla f(\mathbf{x})\,,$$

then we have $\nabla f(\mathbf{x}') = 0$. The Newton method would be to iterate this equation till some stopping criterion is fulfilled.[2]

This method has, however, two disadvantages: first, the second derivatives $\partial_i \partial_j f$ are needed; and second, the equation system $\mathbf{A}(\mathbf{x})\mathbf{y} = \nabla f(\mathbf{x})$ must be solved for $\mathbf{y} = \mathbf{A}^{-1}(\mathbf{x}) \cdot \nabla f(\mathbf{x})$ *at each step*. Usually, this makes the method impractical, but there are exceptions, like the Levenberg-Marquardt method for χ^2 minimization.

The so-called quasi-Newton method circumvents these two problems. This method is based on successive line minimizations and can be written as

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda_i \mathbf{H}_i \cdot \nabla f(\mathbf{x}_i)\,,$$

where the parameter λ_i is determined by line minimization, and $\mathbf{H}_i$ is a $D \times D$ matrix constructed so that $\mathbf{H}_i \to \mathbf{A}^{-1}$ as $i \to \infty$. For details, see NR 10.7.

The quasi-Newton method is often comparable in efficiency to the conjugate-gradient method, but requires more memory if D is large; the memory requirement scales as D^2 for the quasi-Newton method and as D for the conjugate-gradient method.

[2]The "standard" Newton method for a one-dimensional problem $g(x) = 0$ is given by the recursion formula $x' - x = -g(x)/g'(x)$.

5.4 Constrained minimization

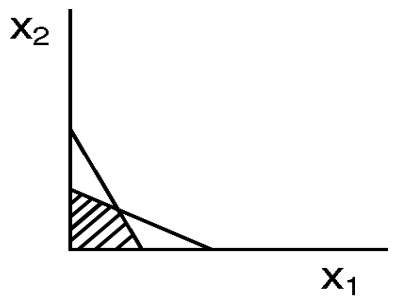
5.4.1 Linear optimization/linear programming

Constrained minimization with a linear cost function and linear constraints is referred to as linear optimization or linear programming. For such problems there is a well-established method called the Simplex method, which is described in NR 10.8.

For a simple example of such a problem, consider the cost function $f(x_1, x_2) = x_1 + x_2$ with the constraints

$$2x_1 + x_2 \leq 2 \qquad x_1 + 2x_2 \leq 2 \qquad x_1, x_2 \geq 0$$

Since the gradient $\nabla f = (1, 1)$ is constant, the desired minimum must be somewhere on the boundary, and it is easy to see that the solution is $(x_1, x_2) = (0, 0)$.



5.4.2 Lagrange multiplier

Suppose we want to minimize a function $f(\mathbf{x})$ subject to the constraint $u(\mathbf{x}) = 0$, $\mathbf{x} \in R^D$. One possible approach is to look for stationary points of the auxiliary function $\tilde{f}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda u(\mathbf{x})$, where λ is called a *Lagrange multiplier*. A stationary point of this function is a solution of the $D + 1$ equations

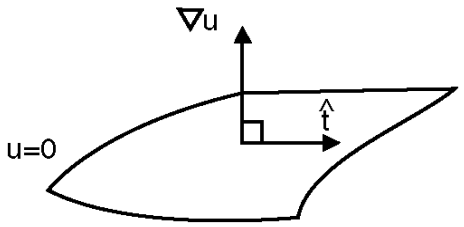
$$\begin{aligned} 0 &= \nabla_x \tilde{f}(\mathbf{x}, \lambda) = \nabla f(\mathbf{x}) + \lambda \nabla u(\mathbf{x}) \\ 0 &= \frac{\partial \tilde{f}}{\partial \lambda} = u(\mathbf{x}) \end{aligned}$$

The last equation is just the constraint, which says that $\mathbf{x}$ must fall onto the hypersurface defined by $u(\mathbf{x}) = 0$. The gradient $\nabla u(\mathbf{x})$ is normal to this surface. The first D equations imply that $\nabla f(\mathbf{x})$ is parallel (or anti-parallel) to $\nabla u(\mathbf{x})$. If $\hat{\mathbf{t}}$ is an

arbitrary unit vector in the tangential plane of the surface $u(\mathbf{x}) = 0$, it follows that

$$\frac{\partial f}{\partial t} = \hat{\mathbf{t}} \cdot \nabla f = 0\,.$$

This means that the derivative of f is zero in every "allowed" direction, which is precisely what we want.



This method is widely used in analytical calculations. As a numerical method, it has the disadvantage that a system of generally non-linear equations must be solved, which can be tricky.

5.4.3 Soft constraints

Consider the same task once more; minimize $f(\mathbf{x})$ subject to the constraint $u(\mathbf{x}) = 0$. A method that often works better numerically than the previous one is to introduce the constraint in a soft manner, by forming the auxiliary function

$$\tilde{f}(\mathbf{x}, \Lambda) = f(\mathbf{x}) + \Lambda u(\mathbf{x})^2$$

This function is to be minimized. Numerically, this is generally more convenient than solving a system of equations. The desired solution is obtained by extrapolating results for large Λ, $\mathbf{x}_\Lambda$, to $\Lambda = \infty$, $\mathbf{x} = \lim_{\Lambda\to\infty} \mathbf{x}_\Lambda$. Note that the penalty term $\Lambda u(\mathbf{x})^2$ grows large as $\Lambda \to \infty$ unless $\mathbf{x}$ satisfies $u(\mathbf{x}) = 0$.

6 Ordinary Differential Equations (ODE)

[NR: 16.0, 16.1, 16.3, 16.6 (16.2, 16.4, 16.7)]

6.1 Introduction

This section deals with numerical integration of ODEs. We will discuss methods for solving systems of first-order ODEs

$$\begin{cases} \dfrac{dy_1}{dx} = f_1\left(x, y_1, \ldots, y_n\right) \\ \quad \vdots \\ \dfrac{dy_n}{dx} = f_n\left(x, y_1, \ldots, y_n\right) \end{cases}$$

for given initial values of all the components y_i at some "time" x_0 (for a discussion of boundary value problems, see chapter 17 of NR). In vector notation, this initial value problem can be written as

$$\frac{d\mathbf{y}}{dx} = \mathbf{f}(x, \mathbf{y}) \qquad \mathbf{y}(x_0) = \mathbf{y}_0 \,.$$

Looking only at first-order ODEs is not a strong limitation, because any n^{th} order ODE of the form

$$\frac{d^n y}{dx^n} = f\left(x, y, \frac{dy}{dx}, \ldots, \frac{d^{n-1}y}{dx^{n-1}}\right)$$

can be written as a system of first-order ODEs by the transformation

$$\begin{array}{l} y_1 = y \\ y_2 = \dfrac{dy}{dx} \\ \vdots \\ y_n = \dfrac{d^{n-1}y}{dx^{n-1}} \end{array} \qquad \Rightarrow \qquad \begin{cases} \dfrac{dy_1}{dx} = y_2 \\ \dfrac{dy_2}{dx} = y_3 \\ \vdots \\ \dfrac{dy_n}{dx} = f\left(x, y_1, \ldots, y_n\right) \end{cases}$$

For convenience, we will often assume that $n = 1$. This does not mean that these methods work only for $n = 1$; on the contrary, generalizing to $n > 1$ is typically easy.

Example: Consider the one-dimensional Newton equation

$$m\frac{d^2x}{dt^2} = -V'(x)$$

Putting $x_1 = x$ and $x_2 = dx/dt$, we obtain the first-order ODEs

$$\begin{cases} dx_1/dt = x_2 \\ dx_2/dt = -V'(x_1)/m \end{cases}$$

which are the corresponding Hamilton equations.

(Hamilton's equations of motion are given by $dx/dt = \partial H/\partial p$ and $dp/dt = -\partial H/\partial x$, where $p = m\,dx/dt$ and $H = p^2/2m + V(x)$.)

6.2 Euler's Method

Consider the first-order ODE

$$\frac{dy}{dx} = f(x, y)\,.$$

To solve this equation numerically, we discretize x using a step size h:

$$\begin{aligned} x_n &= x_0 + nh \quad n = 0, 1, \ldots \\ y_n &= y\,(x_n) \end{aligned}$$

Approximating the derivative with a simple forward difference,

$$\left.\frac{dy}{dx}\right|_{x=x_n} = \frac{y_{n+1} - y_n}{h} + O\,(h)\ ,$$

we obtain the recursion formula

$$y_{n+1} = y_n + hf(x_n, y_n)\,.$$

This can be used to calculate y_n for arbitrary n, starting from a given y_0.

This is called the Euler method. Its local truncation error is $\sim h^2$ (one step), which makes the global truncation error $\sim h$ (this is how the error scales for a fixed total length in x).

A method whose global error scales as h^n is called an n^{th}-order method, so the Euler method is a first-order method.

Low-order methods require small step sizes h, which makes the required number of steps large. Therefore, the computational cost tends to be high.

6.3 Taylor Expansion

A straightforward way to reduce the truncation error is to make use of the Taylor expansion

$$y\,(x_n + h) = y_n + h\left.\frac{dy}{dx}\right|_{x=x_n} + \frac{h^2}{2}\left.\frac{d^2y}{dx^2}\right|_{x=x_n} + \ldots$$

where $dy/dx = f$ and

$$\frac{d^2y}{dx^2} = \frac{df}{dx} = \partial_x f + \partial_y f \cdot \frac{dy}{dx} = \partial_x f + \partial_y f \cdot f \,.$$

Keeping the first three terms we obtain

$$y\left(x_n + h\right) = y_n + hf\left(x_n, y_n\right) + \frac{h^2}{2}\left[\partial_x f\left(x_n, y_n\right) + f\left(x_n, y_n\right)\partial_y f\left(x_n, y_n\right)\right] + O\left(h^3\right)$$

which is a second-order method, or one order better than the Euler method. Clearly, the same procedure can be used to obtain higher-order methods. This approach has, however, the disadvantage that higher and higher derivatives of the function f will be needed.

6.4 The Runge-Kutta Method

The Runge-Kutta method avoids this problem — no derivatives of f are needed. Instead, the estimator y_{n+1} is constructed using values of f itself at carefully chosen points between x_n and x_{n+1}. By increasing the number of points, the order of the method can be increased.

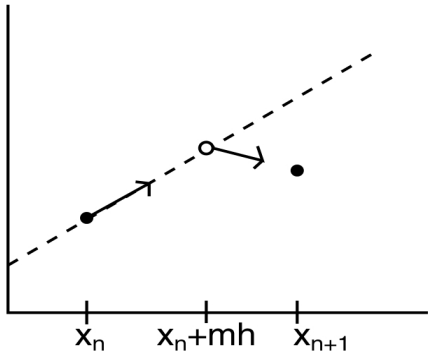
6.4.1 Second-Order Runge-Kutta (local error $\sim h^3$)

Consider the ansatz

$$\begin{aligned} k_1 &= hf\left(x_n, y_n\right) \\ k_2 &= hf\left(x_n + mh, y_n + mk_1\right) \\ y_{n+1} &= y_n + ak_1 + bk_2 \end{aligned}$$

which combines two different f values and has three parameters: a, b and m.

The idea is to determine these three parameters so that the error becomes as small as possible. For this purpose, we make a Taylor expansion of y_{n+1} about $x = x_n$:



$$\begin{aligned} k_2 &= hf(x_n, y_n) + mh^2 \partial_x f(x_n, y_n) + mh^2 f(x_n, y_n) \partial_y f(x_n, y_n) + \mathcal{O}\left(h^3\right) \qquad \Rightarrow \\ y_{n+1} &= y_n + (a+b) hf(x_n, y_n) + bmh^2 \left(\partial_x f(x_n, y_n) + f(x_n, y_n) \partial_y f(x_n, y_n)\right) + \mathcal{O}(h^3) \end{aligned}$$

By comparing this expression with the Taylor expansion of the exact solution $y(x_n+h)$ (see above), we see that the local error becomes $\mathcal{O}(h^3)$ if

$$a + b = 1 \qquad \text{and} \qquad bm = 1/2$$

This is the best that can be achieved with this ansatz — to eliminate the $\mathcal{O}(h^3)$ error term we would need more points.

The two equations for a, b and m have a one-parameter family of solutions. We can, for example, take

$$a = 0,\ b = 1,\ m = \tfrac{1}{2}$$

as in NR, or

$$a = b = \tfrac{1}{2},\ m = 1$$

which is called Heun's method.

6.4.2 Fourth-Order Runge-Kutta

Higher-order Runge-Kutta methods can be obtained by adding more points. This gives improved accuracy at the cost of increased complexity. A very popular compromise is the fourth-order scheme given by Eq. 16.1.3 in NR, which requires four f evaluations. One reason that this scheme is so popular is that in order to get a fifth-order scheme, it turns out that six (and not five) points are needed.

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + h/2, y_n + k_1/2) \\
k_3 &= hf(x_n + h/2, y_n + k_2/2) \\
k_4 &= hf(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + k_1/6 + k_2/3 + k_3/3 + k_4/6 + \mathcal{O}(h^5)
\end{aligned}
$$

6.5 Adaptive Step Size

So far, we have assumed that the step size h is held constant throughout the integration. However, the function $f(x, y)$ may be very different in different parts of space. As a result, there may exist both "easy" regions where a large step can be used and "difficult" ones where a much smaller step size is required.

Therefore, it is often very useful to vary the step size. A natural choice is to vary the step size in such a way that the local error is kept at a constant level. For that, we need to have an estimate of the local error.

A convenient way to estimate the local error is by step doubling. This means that we repeat all calculations using a step size twice as large. By comparing the two calculations, we get an estimate of the local error.

To see how this works, consider fourth-order Runge-Kutta, for which the local error scales as h^5. Let $y(x + h; h/2)$ and $y(x + h; h)$ denote the estimates of the exact solution $y(x + h)$ that are obtained by using two steps of size $h/2$ and one step of size h, respectively. For small h, these estimates should behave as

$$
\begin{aligned}
y(x + h; h/2) - y(x + h) &\sim \left(\frac{h}{2}\right)^5 c(x) + \mathcal{O}(h^6) \\
y(x + h; h) - y(x + h) &\sim h^5 c(x) + \mathcal{O}(h^6)
\end{aligned}
$$

where $c(x)$ is some unknown function. The quantity $\Delta = |y(x+h; h) - y(x+h; h/2)|$ provides a rough estimate of the error in $y(x + h; h)$. To keep the local error under control, we may require that $\Delta < \Delta_{\text{tol}}$, where Δ_{tol} is a predetermined tolerance level.

If $\Delta > \Delta_{\text{tol}}$, we redo the calculation using a smaller step size h'. How small the new step size h' should be can be estimated by using that $\Delta \sim h^5$. This gives

$$
h' \approx S \left(\frac{\Delta_{\text{tol}}}{\Delta}\right)^{1/5} h
$$

where h is the step size that gave us Δ. S is a "safety" factor that is supposed to be less than 1.

If, on the other hand, $\Delta \ll \Delta_{\text{tol}}$, the step size should be increased. How much it can be increased can also be estimated by using that $\Delta \sim h^5$.

The gain from using an adaptive step size can be huge compared to the cost of the additional calculations that are needed to keep track of the local error.

6.6 The Modified Midpoint Method

Consider the same first-order ODE as before,

$$\frac{dy}{dx} = f(x, y)\,.$$

A central-difference approximation of the derivative,

$$\left.\frac{dy}{dx}\right|_{x=x_n} = \frac{y_{n+1} - y_{n-1}}{2h} + O\left(h^2\right)\,,$$

gives us the so-called midpoint method,

$$y_{n+1} = y_{n-1} + 2hf(x_n, y_n)\,,$$

with a local error of $\mathcal{O}(h^3)$. This is a "two-point" method, in which both y_n and y_{n-1} are needed in order to obtain y_{n+1}.

(Compare this with the second order Runge-Kutta method with $a = 0$: $y_{n+1} = y_n + hf(x_n + h/2, y_n + hf(x_n, y_n)/2)$ "=" $y_n + hf(x_{n+1/2}, y_{n+1/2})$)

The *modified* midpoint method has three components:

1. An initial Euler step to get a second starting value:

$$y_1 = y_0 + hf(x_0, y_0)$$

2. $N - 1$ steps with the midpoint method:

$$y_{n+1} = y_{n-1} + 2hf(x_n, y_n) \qquad n = 1, \ldots, N-1$$

3. A "correction" of the last value y_N:

$$y(x_0 + H; h) = \tfrac{1}{2}\left(y_N + y_{N-1} + hf\left(x_N, y_N\right)\right)$$

where $H = Nh$. $y(x_0 + H; h)$ is the final estimate, for step size h, of the exact solution $y(x_0 + H)$.

For fixed x_0 and H and even N, it has been shown (by Gragg) that

$$y\left(x_0+H;h\right)-y\left(x_0+H\right)=c_1h^2+c_2h^4+...$$

In other words, the truncation error contains only even powers of h, a situation we have encountered before (see Romberg's integration method). This property makes the method well suited for Richardson extrapolation.

The so-called Bulirch-Stoer method is based on the modified midpoint method and a Richardson-like extrapolation (but with rational functions rather than polynomials). This method can be a good choice if high accuracy is needed.

6.7 Predictor-Corrector Methods

All methods discussed so far have been *explicit* in that y_{n+1} could be calculated in a direct manner, given one or more input values. There are also *implicit* methods, in which an equation must be solved in order to obtain y_{n+1}. Implicit methods have their advantages, as will be seen below in the discussion of stability. A simple example of an implicit method is obtained by applying the trapezoidal rule for integration to the equation $dy/dx=f(x,y)$. This gives

$$y_{n+1}-y_n=\int_{x_n}^{x_{n+1}}f\left(x,y\right)dx\approx\frac{h}{2}\left[f(x_n,y_n)+f(x_{n+1},y_{n+1})\right] \qquad \text{(i)}$$

This equation for y_{n+1} can be solved by using Newton's method for root finding. Another possibility is to use functional iteration. Consider the recursion formula

$$y^{(k+1)}=g(y^{(k)}) \qquad\qquad g(y)\equiv y_n+\frac{h}{2}\left[f(x_n,y_n)+f(x_{n+1},y)\right] \qquad \text{(ii)}$$

The solution y_{n+1} to equation (i) is a fixed point of this formula; that is, $y_{n+1}=g(y_{n+1})$. If $y^{(k)}$ is close to y_{n+1}, we may linearize the recursion formula:

$$\begin{aligned} y^{(k+1)} &= g\left(y_{n+1}+y^{(k)}-y_{n+1}\right) \\ &\approx g\left(y_{n+1}\right)+\left(y^{(k)}-y_{n+1}\right)\frac{h}{2}\partial_y f\left(x_{n+1},y_{n+1}\right) \\ \Rightarrow \qquad y^{(k+1)}-y_{n+1} &\approx \left(y^{(k)}-y_{n+1}\right)\frac{h}{2}\partial_y f\left(x_{n+1},y_{n+1}\right) \end{aligned}$$

This shows that if our initial guess $y^{(0)}$ is close enough to y_{n+1} for the linearization to be OK, and if h is small enough that $|h\partial_y f(x_{n+1},y_{n+1})/2|<1$, then $|y^{(k)}-y_{n+1}|$ decreases with k. The desired value y_{n+1} can then be found by iteration.

Predictor-corrector methods use some recursion formula like (ii) to improve on an initial guess of y_{n+1}. The recursion formula is applied a fixed number of times, and not until some convergence criterion is fulfilled, which makes predictor-corrector methods explicit rather than implicit.

A simple example of a predictor-corrector method, based on the trapezoidal rule, is:

$$\begin{cases} y^{(0)} = y_n + hf(x_n, y_n) & \text{“predictor” step} \\ y_{n+1} = y^{(1)} = y_n + \dfrac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y^{(0)})] & \text{“corrector” step} \end{cases}$$

Note that this is equivalent to the second order Runge–Kutta with $a = b = 1/2$ and $m = 1$.

A widely used, more advanced predictor-corrector method is the Adams-Bashforth-Moulton method (see NR 16.7).

6.8 Stability

If $n > 1$, it may happen that different components y_i evolve at very different speeds. The problem is then called *stiff*. Stiff problems are generally hard to solve.

Example: Consider

$$\frac{d\mathbf{y}}{dx} = -\mathbf{A}\mathbf{y} \qquad\qquad \mathbf{y}(0) = \mathbf{y}_0$$

where $\mathbf{y}$ is a vector with two components and $\mathbf{A}$ is a 2×2 matrix. Assume that

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i \qquad (i = 1, 2) \qquad\qquad \lambda_2 \gg \lambda_1 > 0$$

and that $\mathbf{y}_0 = c_1\mathbf{v}_1 + c_2\mathbf{v}_2$ for some c_1, c_2. The exact solution is then given by

$$\mathbf{y}(x) = c_1 \mathrm{e}^{-\lambda_1 x}\mathbf{v}_1 + c_2 \mathrm{e}^{-\lambda_2 x}\mathbf{v}_2$$

as can be easily verified. Here, the second term decays much faster than the first one, because $\lambda_2 \gg \lambda_1$. As a result, the second term is not very relevant if we are interested in “timescales” $x \gtrsim 1/\lambda_1$.

Suppose we want to solve this problem numerically by using the Euler method. This gives us the recursion formula

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h(-\mathbf{A}\mathbf{y}_n) = (\mathbf{1} - h\mathbf{A})\mathbf{y}_n$$

(where **1** denotes the unit matrix) which can be easily solved:

$$\begin{aligned}\mathbf{y}_n &= (\mathbf{1} - h\mathbf{A})^n \mathbf{y}_0 = \\ &= c_1(\mathbf{1} - h\mathbf{A})^n \mathbf{v}_1 + c_2(\mathbf{1} - h\mathbf{A})^n \mathbf{v}_2 \\ &= c_1(1 - h\lambda_1)^n \mathbf{v}_1 + c_2(1 - h\lambda_2)^n \mathbf{v}_2\end{aligned}$$

If we take the limit $h \to 0$ for fixed $x = nh$, we should recover the exact solution, and this is indeed the case, because

$$(1 - h\lambda_i)^n = (1 - \lambda_i x/n)^n \to \mathrm{e}^{-\lambda_i x} \qquad n \to \infty \qquad (h \to 0)\,.$$

Stability requires that $|1 - h\lambda_i| \leq 1$ for $i = 1, 2$ (otherwise two solutions corresponding to slightly different initial values will diverge exponentially), and for this to hold it is necessary to take h very small if λ_2 is large,

$$1 - h\lambda_2 \geq -1 \qquad \Rightarrow \qquad h \leq 2/\lambda_2\,.$$

Suppose now we want to study the behavior at large "timescales", $x \gtrsim 1/\lambda_1$. The number of steps needed will then be very large, $x/h \gtrsim \lambda_2/2\lambda_1$. Note that it is the rapidly decaying, "uninteresting" component that dictates what step size we can use.

The Implicit Euler Method

If instead of a forward difference we use a backward difference to approximate the derivative,

$$\left.\frac{d\mathbf{y}}{dx}\right|_{x=x_n} \approx \frac{\mathbf{y}_n - \mathbf{y}_{n-1}}{h}\,,$$

we obtain the so-called implicit Euler method. This may look like a minor modification, but the stability properties change drastically. The recursion formula for the implicit Euler method is

$$\mathbf{y}_n = \mathbf{y}_{n-1} + h(-\mathbf{A}\mathbf{y}_n) \qquad \Rightarrow \qquad \mathbf{y}_n = (\mathbf{1} + h\mathbf{A})^{-1}\mathbf{y}_{n-1}$$

which has the solution

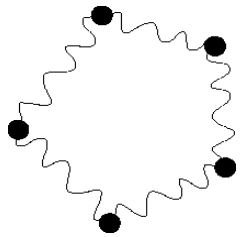
$$\mathbf{y}_n = c_1(1 + h\lambda_1)^{-n}\mathbf{v}_1 + c_2(1 + h\lambda_2)^{-n}\mathbf{v}_2\,.$$

Again, it is easy to verify that the exact solution is recovered as $h \to 0$ for fixed $x = nh$. The relative error in the second term will be large unless h is very small, but this time this error is small in absolute numbers — $|1/(1 + \lambda_i h)| \leq 1$ for all $h > 0$, so the stability problems have disappeared. This is a major advantage compared to the previous method. The disadvantage is that we have to solve an implicit equation for $\mathbf{y}_n$ for each n, which can be prohibitively time-consuming.

Example: Stiff problems are common in physics, especially for systems with many degrees of freedom. As a not obvious example of a stiff problem, consider an ideal ring polymer with potential energy

$$V = \tfrac{1}{2}\sum_{i=1}^{N}(\mathbf{x}_n - \mathbf{x}_{n+1})^2 \qquad\qquad (\mathbf{x}_{N+1} = \mathbf{x}_1)$$

and monomer units of mass 1.



The equations of motion for this system are ($n = 1, \ldots, N$)

$$\begin{aligned}\frac{d^2\mathbf{x}_n}{dt^2} &= -\nabla_{\mathbf{x}_n} V = \\ &= -\tfrac{1}{2}\nabla_{\mathbf{x}_n}\left[(\mathbf{x}_n - \mathbf{x}_{n-1})^2 + (\mathbf{x}_n - \mathbf{x}_{n+1})^2 + \{\mathbf{x}_n \text{ independent terms}\}\right] \\ &= \mathbf{x}_{n-1} + \mathbf{x}_{n+1} - 2\mathbf{x}_n\end{aligned}$$

The general solution can be written as

$$\mathbf{x}_n(t) = \mathbf{A} + \mathbf{B}t + \sum_{k=1}^{N-1}\left[\mathbf{C}_k \cos\left(\tfrac{2\pi kn}{N} - \omega(k)t\right) + \mathbf{D}_k \sin\left(\tfrac{2\pi kn}{N} - \omega(k)t\right)\right]$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}_k$ and $\mathbf{D}_k$ are constant vectors (determined by the initial conditions) and $\omega(k) = 2\sin \pi k/N$. The fastest mode corresponds to $k = N/2$ (if N even) and has a frequency of $\omega_{\max} = 2$. The $k = 1$ and $k = N - 1$ modes are the slowest ones, with a frequency of $\omega_{\min} = 2\sin \pi/N$.

Now, if a method like the explicit Euler method is applied to this system, then the step size must satisfy $1 \gtrsim h\omega_{\max}$ for the integration to be stable. This means that a large number of steps will be needed in order to see a significant change in modes with $\omega = \omega_{\min}$, which are those with longest wavelength; the number of steps required to integrate up to $t = 1/\omega_{\min}$ is

$$\frac{1/\omega_{\min}}{h} \gtrsim \frac{\omega_{\max}}{\omega_{\min}} = \frac{1}{\sin(\pi/N)} \sim \frac{N}{\pi} \to \infty \qquad N \to \infty$$

So, the number of steps needed to explore the long-wavelength or large-scale properties of the system increases with N. The computer time required for each step increases, of course, too.

6.9 Molecular Dynamics (MD)

Classical MD (where quantum-mechanical and relativistic effects are ignored) amounts to integrating Newton's or Hamilton's equations of motion for a system of particles,

$$H = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m_i} + V(\mathbf{x}_1, \ldots, \mathbf{x}_N)$$

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{p}_i/m_i \qquad \frac{d\mathbf{p}_i}{dt} = -\nabla_{\mathbf{x}_i} V \qquad (i = 1, \ldots, N)$$

MD simulations can be used to study both transport and equilibrium properties, and are widely used not least in chemistry. The potential V may contain, for example, Coulomb interactions and van der Waals terms.

Integrating the equations of motion above leaves, of course, the energy $E = H$ invariant ($dH/dt = 0$), as can be easily verified. Nevertheless, molecular dynamics can be used for calculating equilibrium averages in the canonical ensemble, where the energy is fluctuating. Such an average can be written as

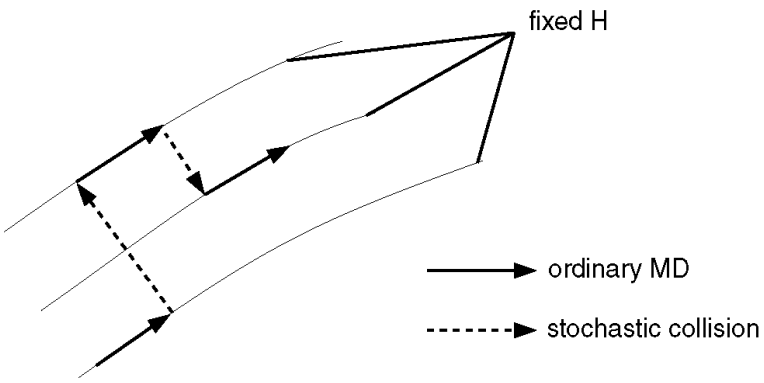
$$\begin{aligned} \langle O \rangle &= \int O \rho_{\text{can}} d^{3N}x d^{3N}p \\ \rho_{\text{can}} &= \frac{\mathrm{e}^{-E/kT}}{\int \mathrm{e}^{-E/kT} d^{3N}x d^{3N}p} \end{aligned}$$

To be able to do this, it is clear that some mechanism must be added to enable the system to make jumps in energy. One solution is to use a so-called Andersen thermostat, which gives us the two-component algorithm:

1) Ordinary MD; that is, integration of the equation of motions.

2) "Stochastic collisions". At randomly chosen or predetermined times t, the momenta of the particles are refreshed, by drawing new values from the equilibrium distribution:

$$\mathbf{p}_i \to \mathbf{p}_i' \qquad \text{where} \qquad P(\mathbf{p}_i') \propto \exp\left(-(\mathbf{p}_i')^2/2m_i kT\right)$$



Having added these collisions, canonical ensemble averages can be obtained as time averages:

$$\int O\rho_{can}d^{3N}xd^{3N}p = \lim_{T\to\infty}\frac{1}{T}\int_0^T O\left(x\left(t\right),p\left(t\right)\right)dt$$

Assuming that the process is ergodic, this can be justified by showing that ρ_{can} is stationary. That $\rho_{\rm can}$ is stationary under the stochastic collisions is obvious, since the new $\mathbf{p}_i'$ is drawn from this distribution. That $\rho_{\rm can}$ is stationary under integration of the equations of motion follows from Liouville's theorem. This theorem says that the flow in phase space ($6N$-dimensional) under the equations of motion is such that the time evolution of a distribution $\rho(x,p,t)$ must satisfy

$$\frac{d\rho}{dt} = \frac{\partial\rho}{\partial t} + \sum_{i=1}^{N}\left(\nabla_{\mathbf{x}_i}\rho\cdot\frac{d\mathbf{x}_i}{dt} + \nabla_{\mathbf{p}_i}\rho\cdot\frac{d\mathbf{p}_i}{dt}\right) = 0\,.$$

So, ρ doesn't change along the trajectories in phase space (the flow is incompressible). If $\rho = \rho(H(x,p),t)$, it can be easily shown that the sum in the expression for $d\rho/dt$ vanishes, which implies that $\partial\rho/\partial t = 0$. In particular, this shows that ρ_{can} is stationary, $\partial\rho_{\rm can}/\partial t = 0$.

6.10 Numerical Integration of the Equations of Motion

Suppose we integrate the equations of motion above from time 0 to some time t. This defines a mapping $T(t)$ of phase space onto itself:

$$T(t): \qquad R^{6N} \ni (x(0),p(0)) \;\mapsto\; (x(t),p(t)) \in R^{6N}$$

Two important properties of this mapping are:

- By Liouville's theorem, it conserves phase space volume — the Jacobian

$$\left|\frac{\partial(x(t),p(t))}{\partial(x(0),p(0))}\right| = 1$$

- It is time reversible in the sense that

$$(x_A, p_A) \;\mapsto\; (x_B, p_B) \quad\Rightarrow\quad (x_B, -p_B) \;\mapsto\; (x_A, -p_A)$$

When using a discretized version of the equation of motions, these properties are generally lost — they hold approximately only. There are, however, discretization methods such that these two properties remain exact.

6.10.1 The Leapfrog Method

This method is not very accurate (the local error is $\mathcal{O}(h^3)$), but has the advantage that it is time reversible and preserves phase space volume. It combines two different elementary steps that represent Euler steps in x and p, respectively:

$$E_x(h): \quad \begin{cases} \mathbf{x}_i(t) \mapsto \mathbf{x}_i(t+h) = \mathbf{x}_i(t) + h\mathbf{p}_i/m_i \\ \mathbf{p}_i(t) \mapsto \mathbf{p}_i(t+h) = \mathbf{p}_i(t) \end{cases}$$

$$E_p(h): \quad \begin{cases} \mathbf{x}_i(t) \mapsto \mathbf{x}_i(t+h) = \mathbf{x}_i(t) \\ \mathbf{p}_i(t) \mapsto \mathbf{p}_i(t+h) = \mathbf{p}_i(t) - h\nabla_{\mathbf{x}_i} V(t) \end{cases}$$

for $i = 1, ..., N$.

Let S denote the mapping

$$\begin{cases} \mathbf{x}_i \to \mathbf{x}_i \\ \mathbf{p}_i \to -\mathbf{p}_i \end{cases} \qquad (i = 1, \ldots, N)$$

An arbitrary mapping M is then time reversible (see above) if $M^{-1} = SMS$. It can be easily verified that $E_x(h)$ and $E_p(h)$ both have this property, but

$$\begin{aligned} \left[E_x(h)\, E_p(h)\right]^{-1} &= E_p(h)^{-1}\, E_x(h)^{-1} \\ &= SE_p(h)\, E_x(h)\, S \neq SE_x(h)\, E_p(h)\, S \end{aligned}$$

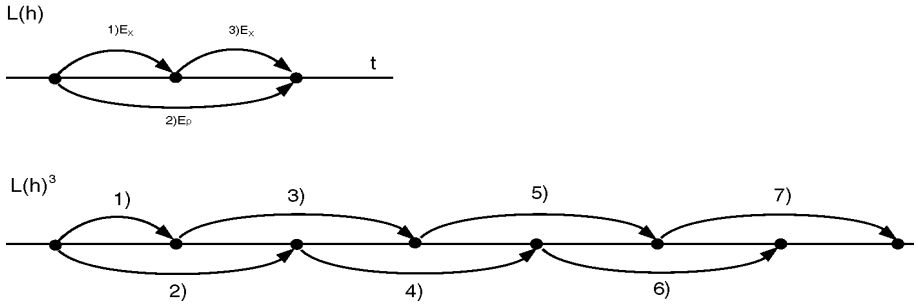
The leapfrog method $L(h)$ is obtained by taking a symmetric product,

$$L(h) = E_x(h/2)E_p(h)E_x(h/2)$$

(x and p can be interchanged). Thanks to the symmetry, $L(h)$ becomes time reversible; that is,

$$L(h)^{-1} = SE_x(h/2)E_p(h)E_x(h/2)S = SL(h)S$$

If we apply $L(h)$ many times, the x and p variables will be updated following the pattern in this figure:



$L(h)$ is not only time reversible, but preserves phase space volume too; that is, the Jacobian

$$\left|\frac{\partial(x(h), p(h))}{\partial(x(0), p(0))}\right| = 1\,.$$

This follows immediately from the time reversibility along with the fact that the Jacobian for S^2 is 1 (because "the determinant of a product is the same as the product of the determinants"). Alternatively, it can be shown that the Jacobians of $E_x(h)$ and $E_p(h)$ both are 1. From this it follows that the Jacobian of $L(h)$, which is the product of three such factors, must be 1 as well.

7 Partial Differential Equations (PDE)

[NR 19.0, 19.1, 19.2, 19.5, 19.6 (19.3, 19.4)]

7.1 Introduction

Our discussion of PDEs will focus on three simple but important equations: the diffusion or heat equation, the Poisson equation and the wave equation. The Poisson equation differs from the other two in that there is no time dependence; this equation is solved for given boundary conditions. The diffusion and wave equations involve time and can be solved by evolving a given initial state in time, subject to given boundary conditions.

The methods to be discussed are *finite-difference methods*, in which the system is put on a space-time grid and the derivatives are approximated with finite differences. This is a simple and useful approach, but there exist many other methods too.

An approach that is widely used, for example, in structural mechanics, is *finite-element methods*. Here, the desired function $u(\mathbf{x}, t)$ is expanded in some basis functions $\varphi_i(\mathbf{x})$,

$$u(\mathbf{x}, t) \approx \sum_{i=1}^{N} a_i(t)\varphi_i(\mathbf{x})$$

The task then is to determine the expansion coefficients $a_i(t)$, by some suitable criterion. Each basis function $\varphi_i(\mathbf{x})$ is nonzero only in a local neighborhood (an element).

There are also *variational methods*. Many physical PDE problems can be reformulated as variational problems; the desired function $u(\mathbf{x}, t)$ is an extremum of some integral I,

$$\delta I[u(\mathbf{x}, t)] = \delta \int d^D x F[u(\mathbf{x}, t)] = 0$$

(for an example, see hand-out for computer exercise 3). If we expand $\varphi_i(\mathbf{x})$ as above, this criterion gives a set of N equations for the expansion coefficients, $\partial I/\partial a_i = 0$, which are called the Rayleigh-Ritz equations. If each basis function $\varphi_i(\mathbf{x})$ is nonzero only in a small element, this is a finite-element method, but this does not have to be the case.

7.2 The Diffusion Equation

Consider a system of particles undergoing random collisions in d dimensions. Let $u(\mathbf{x}, t)$ be the particle density at time t ($\mathbf{x} \in R^d$). The random motion of the particles gives rise to a particle flow, or current, given by

$$\mathbf{j}(\mathbf{x}, t) = -D(\mathbf{x})\nabla u(\mathbf{x}, t)\,,$$

where D is the so-called diffusion coefficient. The dimensions of u and $\mathbf{j}$ are $1/\text{length}^d$ and $1/\text{time}{\cdot}\text{length}^{d-1}$, respectively, so D has dimension $\text{length}^2/\text{time}$. In equilibrium, the net particle flow is zero, $\mathbf{j} = 0$, and the distribution u is uniform.

From particle number conservation follows (via Gauss' theorem) that the current must obey a continuity equation,

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{j} = 0 \qquad \Rightarrow \qquad \frac{\partial u}{\partial t} = \nabla \cdot (D\nabla u)\,.$$

If there is also an external force $\mathbf{F}(\mathbf{x}) = -\nabla U(\mathbf{x})$ acting on the particles, where U is a potential, then the particles will have a drift velocity $\mathbf{v}_{\text{drift}} = \mu\mathbf{F}$, where μ is called the mobility. The drift causes a particle flow, described by the current $\mathbf{j}_{\text{drift}} = u\mathbf{v}_{\text{drift}}$, and the total current now becomes $\mathbf{j} = -D\nabla u + \mathbf{j}_{\text{drift}}$. In equilibrium, the current vanishes, which implies that

$$\nabla u = \frac{\mu u}{D}\mathbf{F}\,.$$

But we know that u is given by $u \propto \mathrm{e}^{-U/kT}$ in equilibrium. Combining these two equilibrium equations, we obtain $D = \mu kT$, which is called the Einstein relation. It follows that the time evolution of u in the presence of an external force $\mathbf{F}$ is given by the equation

$$\frac{\partial u}{\partial t} = \nabla \cdot \left[D\left(\nabla u + \frac{1}{kT} u\mathbf{F} \right) \right]\,.$$

In our discussion, we will make the simplifying assumptions that there is no external force, $\mathbf{F} = 0$, and that the diffusion coefficient D is constant. The equation can then be written as

$$\frac{\partial u}{\partial t} = D\Delta u \qquad \left(\Delta u = \nabla \cdot \nabla u = \sum_{i=1}^{d} \frac{\partial^2 u}{\partial x_i^2} \right)$$

where Δ is called the Laplace operator. We may furthermore assume that $D = 1$ (by a suitable choice of units).

7.2.1 Example of an Exact Solution

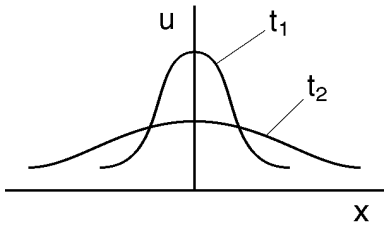
It can be easily verified that

$$u(x,t) = \frac{1}{\sqrt{4\pi t}} \mathrm{e}^{-x^2/4t}$$

solves the one-dimensional diffusion problem [3]

$$\begin{array}{rcll} \dfrac{\partial u}{\partial t} & = & \dfrac{\partial^2 u}{\partial x^2} & \\ u(x,t) & \rightarrow & \delta(x) & \qquad t \searrow 0 \end{array}$$

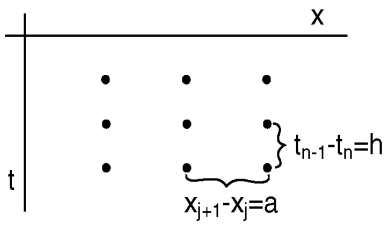
This shows how a distribution initially localized at $x = 0$ evolves in time under diffusive dynamics. The distribution is Gaussian with mean $\langle x \rangle = 0$ for all $t > 0$, and the variance increases linearly with t, $\langle x^2 \rangle = 2t$.



7.2.2 A Simple Finite-Difference Scheme

As a first example of a finite-difference scheme for the one-dimensional diffusion equation, consider the following approximation:

$$\begin{array}{rcl} u\left(x_j, t_n\right) & = & u_j^n \\ \dfrac{u_j^{n+1} - u_j^n}{h} & \approx & \dfrac{\partial u}{\partial t} = \dfrac{\partial^2 u}{\partial x^2} \approx \dfrac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{a^2} \end{array}$$

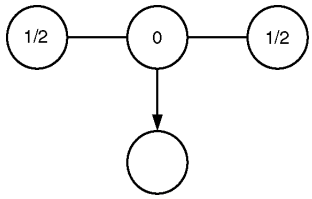


[3] The "delta function" $\delta(x)$ has the characteristic properties that $\int_{-\infty}^{\infty} \delta(x)dx = 1$ and that $\delta(x) = 0$ if $x \neq 0$.

This difference equation becomes particularly simple if the step sizes h and a are chosen so that $h/a^2 = 1/2$, which gives

$$u_j^{n+1} = \tfrac{1}{2}\left(u_{j+1}^n + u_{j-1}^n\right) .$$

Schematically, this equation can be written as



Suppose we start from a state with one $u_j^n = 1$ and all the others zero, to mimic the exact calculation above. By iterating our discrete equation for the time evolution, we obtain the solution

$$\begin{array}{cc} & x \rightarrow \\ t \downarrow & \begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 & \frac{2}{4} & 0 & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{8} & 0 & \frac{3}{8} & 0 & \frac{3}{8} & 0 & \frac{1}{8} & 0 \end{array} \end{array}$$

which has similarities with the exact solution above.

7.2.3 A More Systematic Approach

Let us now solve the same one-dimensional problem in a more systematic manner. To that end, we first discretize x only. Put

$$\begin{aligned} u_j(t) &= u(x_j, t) \qquad\qquad (x_j = ja) \\ \left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_j} &\approx \frac{u_{j+1} - 2u_j + u_{j-1}}{a^2} \end{aligned}$$

The diffusion equation then becomes a system of first-order ODEs,

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u}\,,$$

where $\mathbf{u}(t)$ is a vector with components $u_j(t)$ and the matrix $\mathbf{A}$ is given by

$$\mathbf{A} = \frac{1}{a^2}\begin{pmatrix} \ddots & \ddots & \ddots & & 0 & \\ & 1 & -2 & 1 & & \\ & & 1 & -2 & 1 & \\ & 0 & & \ddots & \ddots & \ddots \\ & & & & & \end{pmatrix}$$

We have encountered this type of problem before. Three possible methods for solving the system of equations are ($\mathbf{u}^n = \mathbf{u}(t_n)$, $t_n = nh$):

1. The explicit Euler method

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{h} = \mathbf{A}\mathbf{u}^n \qquad \Rightarrow \qquad \mathbf{u}^{n+1} = (\mathbf{1} + h\mathbf{A})\mathbf{u}^n$$

2. The implicit Euler method

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{h} = \mathbf{A}\mathbf{u}^{n+1} \qquad \Rightarrow \qquad \mathbf{u}^{n+1} = (\mathbf{1} - h\mathbf{A})^{-1}\mathbf{u}^n$$

3. The trapezoidal rule

$$\mathbf{u}^{n+1} - \mathbf{u}^n = \int_{t_n}^{t_{n+1}} \frac{d\mathbf{u}}{dt} dt \approx \frac{h}{2}(\mathbf{A}\mathbf{u}^n + \mathbf{A}\mathbf{u}^{n+1}) \qquad \Rightarrow$$

$$\mathbf{u}^{n+1} = (\mathbf{1} - \frac{h}{2}\mathbf{A})^{-1}(\mathbf{1} + \frac{h}{2}\mathbf{A})\mathbf{u}^n$$

   This is called the *Crank-Nicolson method*.

The explicit and implicit Euler methods are first order in time, whereas the Crank-Nicolson method is second order. We know, however, that the order of the method is not the only issue in a problem like this; we must also look at the stability properties.

7.2.4 von Neumann Stability Analysis

For the continuous diffusion equation $\partial u/\partial t = \partial^2 u/\partial x^2$, it is possible to find solutions of the form $u(x,t) = T(t)X(x)$. These are given by $u = \mathrm{e}^{-\omega(k)t}\mathrm{e}^{ikx}$, where $\omega(k) = k^2$. For a discretized version of the equation, we can make a similar ansatz,

$$\mathbf{u}^n = \xi(k)^n \tilde{\mathbf{u}}^{(k)} \,,$$

where $\xi(k)$ is a number corresponding to $\mathrm{e}^{-\omega(k)h}$, and $\tilde{\mathbf{u}}^{(k)}$ is a vector with components $\tilde{u}_j^{(k)} = \mathrm{e}^{ikx_j}$, corresponding to the function e^{ikx}. It is not unreasonable to expect $\tilde{\mathbf{u}}^{(k)}$ to be an eigenvector of the matrix $\mathbf{A}$, because e^{ikx} is an eigenfunction of the "operator" $\partial^2/\partial x^2$ (with eigenvalue $-k^2$), and $\mathbf{A}$ is our discretized version of this operator. A direct calculation shows that this is indeed the case:

$$\begin{aligned}(\mathbf{A}\tilde{\mathbf{u}}^{(k)})_j &= \frac{1}{a^2}(\mathrm{e}^{ikx_{j-1}} - 2\mathrm{e}^{ikx_j} + \mathrm{e}^{ikx_{j+1}}) \\ &= \underbrace{-\frac{1}{a^2}\left(2 - \mathrm{e}^{-ika} - \mathrm{e}^{ika}\right)}_{\lambda_k} \tilde{u}_j^{(k)}\end{aligned}$$

So,

$$\mathbf{A}\tilde{\mathbf{u}}^{(k)} = \lambda_k \tilde{\mathbf{u}}^{(k)} \qquad\qquad \lambda_k = -\frac{4}{a^2}\sin^2\frac{ka}{2}$$

Note that $\lambda_k \approx -k^2$ if $ka \ll 1$.

Assuming that the initial state can be written as a superposition of solutions of the form $\mathbf{u}^n = \xi(k)^n \tilde{\mathbf{u}}^{(k)}$, we can now draw conclusions about the stability of the three methods mentioned above.

1. Explicit Euler

   $$\xi(k)^{n+1}\tilde{\mathbf{u}}^{(k)} = \xi(k)^n(\mathbf{1} + h\mathbf{A})\tilde{\mathbf{u}}^{(k)} \qquad \Rightarrow \qquad \xi(k) = 1 + h\lambda_k$$

   Stability requires that $|\xi(k)| \leqslant 1$ for all k, so we must have (note that $\lambda_k \leqslant 0$)

   $$1 + h\lambda_k \geqslant -1 \qquad \Leftrightarrow \qquad \frac{2h}{a^2}\sin^2\frac{ka}{2} \leqslant 1 \qquad\qquad (\text{all } k)$$

   This holds for all k only if $h \leqslant a^2/2$. This is a severe restriction. For the exact solution, we saw that the width of the distribution scaled as $\sqrt{\text{time}}$. This means that the number of steps needed before the distribution reaches a given width L scales as

   $$\text{no. of steps} = \text{time}/h \sim L^2/h \gtrsim (L/a)^2$$

2. Implicit Euler

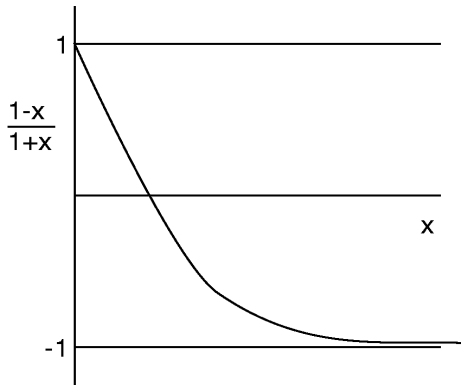
   $$\xi(k)^{n+1}\tilde{\mathbf{u}}^{(k)} = \xi(k)^n(\mathbf{1} - h\mathbf{A})^{-1}\tilde{\mathbf{u}}^{(k)} \qquad \Rightarrow \qquad \xi(k) = \frac{1}{1 - h\lambda_k}$$

   Since $\lambda_k \leqslant 0$, it immediately follows that $0 \leqslant \xi(k) \leqslant 1$ for all k and $h > 0$. Hence, the method is stable for any $h > 0$.

3. Crank-Nicolson

$$\xi(k)^{n+1}\tilde{\mathbf{u}}^{(k)} = \xi(k)^n(\mathbf{1} - \frac{h}{2}\mathbf{A})^{-1}(\mathbf{1} + \frac{h}{2}\mathbf{A})\tilde{\mathbf{u}}^{(k)} \qquad \Rightarrow \qquad \xi(k) = \frac{1 + \frac{h}{2}\lambda_k}{1 - \frac{h}{2}\lambda_k}$$

This implies that $|\xi(k)| \leqslant 1$ for all k and $h > 0$. This follows from the facts that all $\lambda_k \leqslant 0$ and that $|(1-x)/(1+x)| \leqslant 1$ for all $x \geqslant 0$. So, this method is also stable for all $h > 0$.



7.2.5 Implementing the Implicit Euler and Crank-Nicolson Methods

At each implicit Euler step, we need to solve $(\mathbf{1} - h\mathbf{A})\mathbf{u}^{n+1} = \mathbf{u}^n$ for $\mathbf{u}^{n+1}$. For a general $\mathbf{A}$ this would be very time-consuming. However, $\mathbf{A}$ is tridiagonal, which makes it doable.

Assume, for simplicity, that we are studying an interval $0 \leqslant x \leqslant Ja$ and that the boundary conditions are $u(0,t) = C_1$ and $u(Ja,t) = C_2$, where C_1 and C_2 are constants. Let $\mathbf{u}$ be the vector with the $J-1$ components $u_1, \ldots, u_{J-1}$ (u_0 and u_J are left out because they are constant). The problem can then be written

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u} + \mathbf{c}$$

where

$$\mathbf{A} = \begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & 0 & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{pmatrix} \qquad \mathbf{c} = \begin{pmatrix} C_1 \\ 0 \\ \vdots \\ 0 \\ C_2 \end{pmatrix}$$

The $(J-1)\times(J-1)$-matrix $\mathbf{A}$ is called tridiagonal because it has nonzero elements only in three bands along its diagonal. The matrix $\mathbf{1}-h\mathbf{A}$ is tridiagonal too, and this makes it possible to solve the equation $(\mathbf{1}-h\mathbf{A})\mathbf{x}=\mathbf{y}$ for $\mathbf{x}$ in $\mathcal{O}(J)$ operations, by Gauss elimination. This figure illustrates how this is done for an arbitrary tridiagonal $N\times N$-matrix:

$$
\begin{pmatrix}
b_1 & c_1 & & & \\
a_2 & b_2 & c_2 & & \\
 & a_3 & b_3 & c_3 & \\
 & & \ddots & \ddots & \ddots \\
 & & & & \\
\end{pmatrix}
\begin{matrix} -a_2/b_1 \\ \hookleftarrow \end{matrix} \quad \rightarrow
$$

$$
\begin{pmatrix}
b_1 & c_1 & & & \\
0 & b_2' = b_2 - a_2c_1/b_1 & c_2 & & \\
 & a_3 & b_3 & c_3 & \\
 & & \ddots & \ddots & \ddots \\
 & & & & \\
\end{pmatrix}
\begin{matrix} -a_3/b_2' \\ \hookleftarrow \end{matrix} \quad \rightarrow \quad \ldots \rightarrow
$$

$$
\begin{pmatrix}
b_1 & c_1 & & & \\
 & b_2' & c_2 & & \\
 & & b_3' & c_3 & \\
 & & \ddots & \ddots & \\
 & & & b_{N-1}' & c_{N-1} \\
 & & & & b_N'
\end{pmatrix}
\begin{matrix} \\ \\ \\ \\ \hookleftarrow \\ -c_{N-1}/b_N' \end{matrix} \quad \rightarrow \quad \ldots
$$

There are $\approx N$ steps "downwards" and $\approx N$ steps "upwards" and each step requires $\mathcal{O}(1)$ operations, which makes the total cost $\mathcal{O}(N)$.

In the Crank-Nicolson method, there is also a tridiagonal system of equations to be solved at each step. The same method can be used in this case.

7.2.6 Higher Dimensions

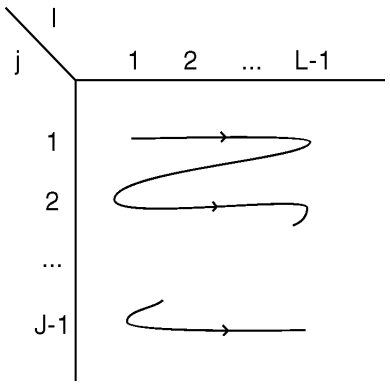
So far, we have restricted ourselves to one-dimensional diffusion ($d=1$). Formally, it is straightforward to extend these methods to higher d. To illustrate that, consider the two-dimensional diffusion problem

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}$$

with fixed boundary conditions, $u = 0$. Assume, for simplicity, that the region studied is rectangular. We can then discretize x and y as follows.

$$\begin{aligned} x_j &= ja \qquad (j = 0, \ldots, J) \\ y_l &= la \qquad (l = 0, \ldots, L) \\ u_{jl}(t) &= u(x_j, y_l, t) \\ \frac{du_{jl}}{dt} &\approx \frac{u_{j+1l} + u_{j-1l} + u_{jl+1} + u_{jl-1} - 4u_{jl}}{a^2} \end{aligned}$$

Put all "internal" (non-constant) u_{jl} in a vector with a single index i, which may be defined as $i = (j-1)(L-1) + l - 1$.



We can then write the discretized diffusion equation as

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u} \qquad (+ \text{ constant vector if there are nonzero boundary values})$$

The Crank-Nicholson method, for example, can be directly applied to this equation, which gives

$$\mathbf{u}^{n+1} = (\mathbf{1} - \tfrac{h}{2}\mathbf{A})^{-1}(\mathbf{1} + \tfrac{h}{2}\mathbf{A})\mathbf{u}^n \,.$$

Formally, this looks precisely as before. However, the matrix $\mathbf{A}$ now has 5 bands instead of 3, which makes the problem computationally harder. The cost of the calculation of $\mathbf{u}^{n+1}$ scales linearly with the number of rows or columns of the matrix in this case too, but this number is larger, $(J-1)\cdot(L-1)$, and the prefactor in this scaling law is higher when the number of bands is 5 instead of 3.

The Alternating Direction Implicit method (ADI)

ADI is a method that, like the Crank-Nicolson method, is second order in time and stable for all $h > 0$. The advantage of ADI compared to Crank-Nicolson is that it

works with tridiagonal matrices. This can be achieved by splitting the matrix $\mathbf{A}$ into two terms,

$$\mathbf{A} = \mathbf{A}_x + \mathbf{A}_y \,,$$

where $\mathbf{A}_x$ and $\mathbf{A}_y$ correspond to $\partial^2/\partial x^2$ and $\partial^2/\partial y^2$, respectively. Both $\mathbf{A}_x$ and $\mathbf{A}_y$ are tridiagonal. To be able to take advantage of that, each ADI step is split into two substeps:

$$\begin{cases} \mathbf{u}^{n+1/2} = \mathbf{u}^n + \frac{h}{2}(\mathbf{A}_x\mathbf{u}^{n+1/2} + \mathbf{A}_y\mathbf{u}^n) & \text{“implicit in } x\text{, explicit in } y''\\ \mathbf{u}^{n+1} = \mathbf{u}^{n+1/2} + \frac{h}{2}(\mathbf{A}_x\mathbf{u}^{n+1/2} + \mathbf{A}_y\mathbf{u}^{n+1}) & \text{“explicit in } x\text{, implicit in } y'' \end{cases}$$

There are two equation systems to be solved, one for each substep, and they are both tridiagonal,

$$\begin{aligned} (\mathbf{1} - \tfrac{h}{2}\mathbf{A}_x)\mathbf{u}^{n+1/2} &= (\mathbf{1} + \tfrac{h}{2}\mathbf{A}_y)\mathbf{u}^n \\ (\mathbf{1} - \tfrac{h}{2}\mathbf{A}_y)\mathbf{u}^{n+1} &= (\mathbf{1} + \tfrac{h}{2}\mathbf{A}_x)\mathbf{u}^{n+1/2} \end{aligned}$$

The method can be summarized in one equation by eliminating $\mathbf{u}^{n+1/2}$, which gives

$$\mathbf{u}^{n+1} = (\mathbf{1} - \tfrac{h}{2}\mathbf{A}_y)^{-1}(\mathbf{1} + \tfrac{h}{2}\mathbf{A}_x)(\mathbf{1} - \tfrac{h}{2}\mathbf{A}_x)^{-1}(\mathbf{1} + \tfrac{h}{2}\mathbf{A}_y)\mathbf{u}^n$$

7.3 The Poisson Equation

Consider the Poisson equation

$$\Delta u(\mathbf{x}) = \nabla \cdot \nabla u(\mathbf{x}) = -\rho(\mathbf{x}) \qquad\qquad \mathbf{x} \in \Omega$$

with the boundary condition $u(\mathbf{x}) = f(\mathbf{x})$, $\mathbf{x} \in \partial\Omega$ ($\partial\Omega$ denotes the boundary of Ω). Such a condition that specifies u itself on the boundary (f is a given function) is called a Dirichlet boundary condition.

This boundary-value problem appears at different places in physics. An example is electrostatics. u is then the electrostatic potential and ρ the charge density (in suitable units).

Suppose we want to solve this problem by finite differencing. The problem is then transformed into a linear algebra problem of the form $\mathbf{Au} = \mathbf{b}$, where $\mathbf{A}$, as before, corresponds to Δ and $\mathbf{b}$ is a vector that is determined by ρ and the function f in the boundary condition.

Solving this system of equations is, in principle, straightforward. The problem is the size of the system; if, for example, we are in $d = 2$ and work with a 1000×1000 grid, then there are 10^6 $\mathbf{u}$ components and 10^{12} elements in the matrix $\mathbf{A}$. This makes it necessary to make use of the fact that $\mathbf{A}$ is sparse (that is, only a tiny fraction of its elements are nonzero).

7.3.1 Relaxation Methods

A widely used strategy for solving the boundary value problem above is to let the system evolve in a fictitious time τ with diffusive dynamics. We then consider the diffusion problem

$$\begin{cases} \dfrac{\partial u}{\partial \tau} = \Delta u + \rho & \mathbf{x} \in \Omega,\ \tau > 0 \\ u = f & \mathbf{x} \in \partial\Omega,\ \tau > 0 \\ \text{some initial condition} & \tau = 0 \end{cases}$$

where ρ and f are the same as in the original problem. In general, u will approach a stationary state as $\tau \to \infty$; that is, $\partial u/\partial \tau \to 0$ as $\tau \to \infty$. This stationary state is the desired solution.

Suppose we evolve u in τ by using a finite-difference method of the form

$$\mathbf{u}^{n+1} = \mathbf{T}\mathbf{u}^n + \text{constant vector}\,.$$

The matrix $\mathbf{T}$ is called the iteration matrix. The convergence rate is governed by the spectral radius ρ_T of this matrix, defined by

$$\rho_T = \max_{\text{eigenvalues } \lambda \text{ of } \mathbf{T}} |\lambda|\ .$$

To see this, we first note that the desired solution must correspond to a fix point of the recursion formula. Denote this fix point by $\mathbf{u}^{\mathrm{f}}$. The deviation from the fix point satisfies

$$\mathbf{u}^{n+1} - \mathbf{u}^{\mathrm{f}} = \mathbf{T}(\mathbf{u}^n - \mathbf{u}^{\mathrm{f}})\,.$$

which implies that

$$|\mathbf{u}^n - \mathbf{u}^{\mathrm{f}}| \sim \rho_T^n \qquad n \to \infty$$

(unless the projection of $\mathbf{u}^0 - \mathbf{u}^{\mathrm{f}}$ onto the eigenvector with maximum $|\lambda|$ happens to be zero). This first of all shows that the method will converge if $\rho_T < 1$. We also see that for the convergence to be fast, ρ_T should be as small as possible.

7.3.2 The Jacobi method

A simple relaxation method is the Jacobi method, which is obtained by using an Euler step in τ. The method is given by

$$\mathbf{u}^{n+1} = \mathbf{T}\mathbf{u}^n + \boldsymbol{\rho}\,,$$

where the iteration matrix $\mathbf{T} = \mathbf{1} + h\mathbf{A}$ and the vector $\boldsymbol{\rho}$ represents the charge density $\rho(\mathbf{x})$ (for simplicity, we assume that $f = 0$). As usual, the matrix $\mathbf{A}$ is our approximation of the Laplace operator Δ. It is the same as before.

Assume that we are in two dimensions and that the region Ω is a square,

$$\Omega = \{(x, y); 0 \leqslant x, y \leqslant Ja\}$$

It is then easy to find the eigenvalues and eigenvectors of $\mathbf{A}$, which are given by

$$\begin{aligned} \mathbf{A}\mathbf{v}^{(k_x,k_y)} &= \lambda_{k_x,k_y}\mathbf{v}^{(k_x,k_y)} \qquad\qquad k_x, k_y = 1, \ldots, J-1 \\ \lambda_{k_x,k_y} &= -\frac{4}{a^2}\left(\sin^2\frac{k_x\pi}{2J} + \sin^2\frac{k_y\pi}{2J}\right) \\ \mathbf{v}^{(k_x,k_y)}_{jl} &= \sin\frac{jk_x\pi}{J}\sin\frac{lk_y\pi}{J} \end{aligned}$$

The eigenvalues of the iteration matrix $\mathbf{T}$ are given by

$$\lambda^T_{k_x,k_y} = 1 + h\lambda_{k_x,k_y}\,.$$

It is clear that $\lambda^T_{k_x,k_y} \leqslant 1$, since $\lambda_{k_x,k_y} \leqslant 0$. The lowest eigenvalue of $\mathbf{T}$ is

$$\lambda^T_{J-1,J-1} = 1 + h\lambda_{J-1,J-1} \approx 1 - 2\frac{4h}{a^2}\,.$$

This shows that in order to have $\rho_T \leqslant 1$, we must take $h \leqslant a^2/4$. In the Jacobi method, one takes $h = a^2/4$. With this choice, the time evolution is given by

$$u^{n+1}_{jl} = \frac{1}{4}(u^n_{j+1l} + u^n_{j-1l} + u^n_{jl+1} + u^n_{jl-1}) + \frac{a^2}{4}\rho_{jl}\,.$$

To find ρ_T and thereby the convergence rate, we note that

$$\max\lambda^T_{k_x,k_y} = 1 + h\lambda_{1,1} = 1 - 2\sin^2\frac{\pi}{2J} \approx 1 - \frac{\pi^2}{2J^2}$$

$$\min\lambda^T_{k_x,k_y} = 1 + h\lambda_{J-1,J-1} = 1 - 2\sin^2\frac{(J-1)\pi}{2J} = \cos\frac{(J-1)\pi}{J} \approx -1 + \frac{\pi^2}{2J^2}$$

So, $\rho_T \approx 1 - \pi^2/2J^2$. The number of iterations, r, required to reduce the error by a factor 10^{-p} satisfies

$$\rho_T^r \sim 10^{-p} \qquad \Rightarrow \qquad r \approx \frac{-p\ln 10}{\ln\rho_T} \approx \frac{2p\ln 10}{\pi^2}J^2\,.$$

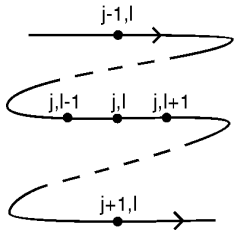
Hence, the number of iterations scales quadratically with J. This makes the method too slow to be of direct practical interest.

7.3.3 The Gauss-Seidel Method

This method is very similar to the Jacobi method. The only difference is that "new" u values are used as soon as they become available. This gives a slightly better convergence rate, but the scaling of the required number of iterations with system size remains quadratic.

The Gauss-Seidel method can be written as

$$u_{jl}^{n+1} = \frac{1}{4}(u_{j+1l}^{n} + \underset{\uparrow}{u_{j-1l}^{n+1}} + u_{jl+1}^{n} + \underset{\uparrow}{u_{jl-1}^{n+1}}) + \frac{a^2}{4}\rho_{jl}$$



The Jacobi and Gauss-Seidel method are slow but important as starting points for more advanced methods.

Successive overrelaxation (SOR; see computer exercise 3) is based on the Gauss-Seidel method. It is almost as simple as the Gauss-Seidel method to implement, but much more efficient — the number of iterations needed scales as J rather than J^2, provided that the overrelaxation parameter ω is carefully chosen.

Multigrid methods (see NR) are more advanced. They are very efficient, but also more complicated than SOR to implement.

7.3.4 The Fourier Transform Method

Consider the boundary value problem

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = -\rho & 0 < x < X,\ 0 < y < Y \\ u(0,y) = u(X,y),\ u(x,0) = u(x,Y) & \text{periodic boundary conditions} \\ u(0,0) = u_0 & \end{cases}$$

A very efficient method for this problem can be obtained by using Fourier decomposition:

$$\begin{aligned} u(x,y) &= \sum_{k_x=-\infty}^{\infty} \sum_{k_y=-\infty}^{\infty} \hat{u}(k_x,k_y) \mathrm{e}^{2\pi i\left(\frac{k_x x}{X}+\frac{k_y y}{Y}\right)} \\ \hat{u}\left(k_x,k_y\right) &= \frac{1}{XY}\int_0^X dx \int_0^Y dy u(x,y) \mathrm{e}^{-2\pi i\left(\frac{k_x x}{X}+\frac{k_y y}{Y}\right)} \end{aligned}$$

Do the same for $\rho(x,y)$. Insertion into the Poisson equation gives

$$-4\pi^2\left[\left(\tfrac{k_x}{X}\right)^2+\left(\tfrac{k_y}{Y}\right)^2\right]\hat{u}\left(k_x,k_y\right) = -\hat{\rho}\left(k_x,k_y\right) ,$$

which is a simple algebraic equation for $\hat{u}$, in place of the original differential equation.

This gives us a three-step "algorithm" for solving the given problem:

1. Transform $\rho \to \hat{\rho}$.
2. Calculate $\hat{u}(k_x,k_y)$ by using the algebraic equation above for all $(k_x,k_y) \neq (0,0)$. For $k_x = k_y = 0$, $\hat{u}$ is determined by the additional condition $u(0,0) = u_0$.
3. Transform $\hat{u} \to u$.

Numerically, these calculations are done using discrete Fourier transforms,

$$f_n = \frac{1}{\sqrt{N}}\sum_{k=1}^{N} \hat{f}_k \mathrm{e}^{\frac{2\pi i k n}{N}} \qquad\qquad \hat{f}_k = \frac{1}{\sqrt{N}}\sum_{n=1}^{N} f_n \mathrm{e}^{-\frac{2\pi i k n}{N}}$$

With the method of Fast Fourier Transform (FFT), the cost of a transform $f_n \mapsto \hat{f}_k$ (or vice versa) becomes $\sim N \ln N$ rather than N^2.

This makes this method very fast. However, it requires that the region considered is rectangular, which is a strong limitation (although the boundary conditions do not have to be periodic).

7.4 Waves

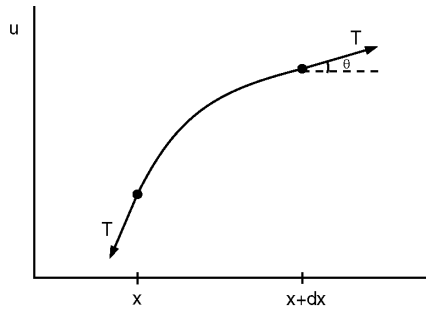
The one-dimensional wave equation

$$\frac{\partial^2 u}{\partial t^2} - v^2 \frac{\partial^2 u}{\partial x^2} = 0$$

describes, for example, the transverse motion of a free string. This can be seen by considering a small string element dx. The string is assumed to be homogeneous with a linear density ρ, so the mass of the small element is ρdx. Furthermore, it is assumed that the tension T is constant, and that the motion is "small" in the sense that the angle θ in the figure below stays small. The equation for the transverse motion of the small element then is

$$\rho dx \frac{\partial^2 u}{\partial t^2} = T \sin\theta|_{x+dx} - T \sin\theta|_x \approx T \tan\theta|_{x+dx} - T \tan\theta|_x \qquad \Rightarrow$$

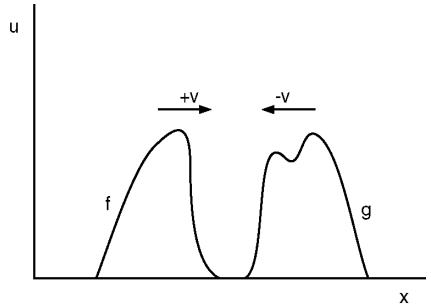
$$\rho dx \frac{\partial^2 u}{\partial t^2} \approx T \frac{\partial u}{\partial x}\Big|_{x+dx} - T \frac{\partial u}{\partial x}\Big|_x \approx T dx \frac{\partial^2 u}{\partial x^2} \qquad \Rightarrow \qquad \frac{\partial^2 u}{\partial t^2} \approx \frac{T}{\rho}\frac{\partial^2 u}{\partial x^2} \qquad (v = \sqrt{T/\rho})$$



The one-dimensional wave equation can be solved by changing variables to $\xi = x - vt$ and $\eta = x + vt$. The equation then becomes

$$\frac{\partial^2 u}{\partial \xi \partial \eta} = 0 \qquad \Rightarrow \qquad u = f(\xi) + g(\eta) = f(x - vt) + g(x + vt)\,,$$

where the functions f and g are determined by the initial values of u itself and the velocity $\partial u/\partial t$ (the equation is second order in time). $f(x-vt)$ and $g(x+vt)$ represent right- and left-moving waves, respectively ($v > 0$).



Instead of the wave equation, we will, for simplicity, consider

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0\,,$$

which has right-moving solutions only; the same change of variables as before gives

$$\frac{\partial u}{\partial \eta} = 0 \qquad \Rightarrow \qquad u = f(\xi) = f(x - vt)\,.$$

It is instructive to solve this problem by separation of variables, which means that we write the solution as a superposition of solutions of the form $u(x, y) = X(x)T(t)$. This ansatz gives

$$\frac{X'(x)}{X(x)} = -\frac{1}{v}\frac{T'(t)}{T(t)} \equiv ik = \text{constant} \qquad \Rightarrow \qquad X(x) \propto \mathrm{e}^{ikx},\ T(t) \propto \mathrm{e}^{-ikvt}\,,$$

where the constant k must be real for X to remain bounded as $x \to \pm\infty$. This shows that $u_k(x,t) = \mathrm{e}^{i(kx-\omega(k)t)}$, where $\omega(k) = kv$, is a solution to the given equation for all real k. The general solution can be written as a continuous superposition of such solutions,

$$u(x,t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} c(k)\mathrm{e}^{i(kx-\omega(k)t)}dk\,,$$

where $c(k)$ is determined by initial data. Suppose a (right-moving) "wave packet" is expressed in this way. The fact that $\omega(k) = kv$ implies that all plane-wave components u_k of the wave packet have the same phase velocity (v). This in turn implies that the wave packet propagates without changing shape.

If the phase velocity $v_\mathrm{f} = \omega(k)/k$ were k dependent, then the shape of a wave packet would change with time. This is called *dispersion*.

This may be further generalized by considering a complex $\omega(k) = \Omega(k) - i\gamma(k)$, where Ω and γ both are real. In this case, it can be shown (by using Parseval's formula) that

$$\begin{aligned}
\int_{-\infty}^{\infty}|u(x,t)|^2dx &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\left|\int_{-\infty}^{\infty}c(k)\mathrm{e}^{i(kx-\omega(k)t)}dk\right|^2 dx \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty}dx\,dk\,dk'c(k)\bar{c}(k')\mathrm{e}^{i(kx-\omega(k)t)}\mathrm{e}^{-i(k'x-\bar{\omega}(k')t)} \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty}dk\,dk'c(k)\bar{c}(k')\mathrm{e}^{i(\bar{\omega}(k')-\omega(k))t)}\int_{-\infty}^{\infty}dx\mathrm{e}^{ix(k-k')} \\
&= \int_{-\infty}^{\infty}|c(k)|^2\mathrm{e}^{-2\gamma(k)t}dk\,.
\end{aligned}$$

From this result we see that the normalization $\int |u|^2 dx$ is preserved as long as ω is real ($\gamma = 0$). If, on the other hand, $\gamma > 0$, the normalization integral decreases with time. This is called *damping*.

7.4.1 Discrete waves

Let us now look at how to solve the equation $\partial u/\partial t = -v\partial u/\partial x$ by finite differencing. As usual, we put $u_j^n = u(x_j, t_n)$, where $x_j = ja$ and $t_n = nh$. For the space derivative, we use a central difference,

$$\frac{\partial u}{\partial x}\Big|_{t=t_n} = \frac{u_{j+1}^n - u_{j-1}^n}{2a} + \mathcal{O}(a^2)$$

The Euler Method

With an Euler step in time, we get

$$\frac{u_j^{n+1} - u_j^n}{h} = -v\frac{u_{j+1}^n - u_{j-1}^n}{2a} \qquad \Rightarrow \qquad u_j^{n+1} = u_j^n - \frac{hv}{2a}(u_{j+1}^n - u_{j-1}^n)$$

To check the stability of the method, we make a von Neumann ansatz, $u_j^n = \xi(k)^n \mathrm{e}^{ikx_j}$. This gives

$$\xi(k) = 1 - \frac{hv}{2a}(\mathrm{e}^{ika} - \mathrm{e}^{-ika}) = 1 - i\frac{hv}{a}\sin ka \qquad \Rightarrow$$

$$|\xi(k)|^2 = 1 + \left(\frac{hv}{a}\right)^2 \sin^2 ka > 1$$

showing that $|u_j^n| = |\xi(k)|^n$ grows exponentially with n for all $h > 0$. So, this method, which may not seem unreasonable, is a disaster — it is always unstable.

The Lax Method

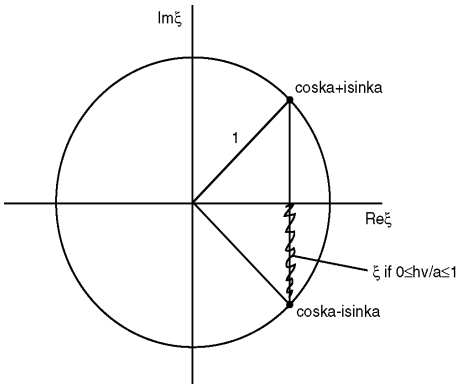
The method above can be improved by a simple modification, which gives the so-called Lax method. The modification is that u_j^n is replaced by the average of u_{j+1}^n and u_{j-1}^n. This gives

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{hv}{2a}(u_{j+1}^n - u_{j-1}^n)\,.$$

This time the ansatz $u_j^n = \xi(k)^n \mathrm{e}^{ikx_j}$ gives

$$\xi(k) = \frac{1}{2}(\mathrm{e}^{ika} + \mathrm{e}^{-ika}) - \frac{hv}{2a}(\mathrm{e}^{ika} - \mathrm{e}^{-ika}) = \cos ka - i\frac{hv}{a}\sin ka\,,$$

and from the figure below it can be seen that $|\xi(k)| \leqslant 1$ for all k if $hv/a \leqslant 1$. So, the method is stable if $h \leqslant a/v$. How does the discrete solution behave?



The k dependence of $\xi(k)$ corresponds to an $\omega(k)$ given by

$$\xi(k) = \mathrm{e}^{-i\omega(k)h} = \cos ka - i\frac{hv}{a}\sin ka\,.$$

If $hv/a = 1$, we see that $h\omega = ka$. So, $\omega = ka/h = kv$, which happens to be the exact result.

If, on the other hand, $hv/a < 1$, then

$$|\xi|^2 = \mathrm{e}^{-2h\gamma} = \cos^2 ka + \left(\frac{hv}{a}\right)^2 \sin^2 ka < 1\,.$$

This means that $\gamma > 0$, so the discrete solution is, in contrast to the exact one, damped. The damping is, however, small if ka is small. The real part Ω of ω satisfies

$$h\Omega = \arctan\left(\frac{(hv/a)\sin ka}{\cos ka}\right) \approx \arctan hvk \approx hvk$$

if ka is small. This shows that $\omega(k)$ for the discrete solution is approximately correct if the wavelength is much larger than a, so that $ka \ll 1$. Short-wavelength modes have large relative errors but are damped, so these errors do not destroy the long-wavelength properties of the solution.

The Leapfrog Method

The Lax method is first order in time and second order in space. A method that is second order in time too can be obtained by using a central-difference approximation for the time derivative, which gives

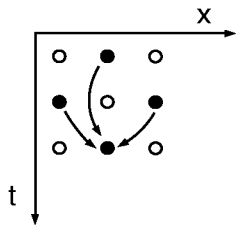
$$u_j^{n+1} = u_j^{n-1} - \frac{vh}{a}\left(u_{j+1}^n - u_{j-1}^n\right)\,.$$

For this method, the ansatz $u_j^n = \xi(k)^n \mathrm{e}^{ikx_j}$ gives

$$\xi(k) = -i\frac{vh}{a}\sin ka \pm \sqrt{1 - \left(\frac{vh}{a}\right)^2 \sin^2 ka}\,.$$

From this it can be easily verified that this method, like the Lax method, is stable if $\dfrac{hv}{a} \leqslant 1$.

This is called the leapfrog method because the space-time grid separates into two sub-lattices that do not influence each other. This may cause numerical problems; the sublattices may "diverge" due to rounding errors. An artificial coupling term is then needed.



7.5 Schrödinger's equation

Consider the one-dimensional Schrödinger equation

$$i\frac{\partial\psi}{\partial t} = -\frac{\partial^2\psi}{\partial x^2} + V(x)\,\psi = H\psi$$

(in units such that $\hbar = 1, m = 1/2$). For $V = 0$, this equation describes diffusion in imaginary time; in fact, with $\tau = it$ and $V = 0$, we obtain $\partial\psi/\partial\tau = \partial^2\psi/\partial x^2$.

After discretizing x, the Schrödinger equation becomes

$$i\frac{d\boldsymbol{\psi}}{dt} = \mathbf{H}_D\,\boldsymbol{\psi}$$

where $\boldsymbol{\psi}$ is a vector with components $\psi_j(t) = \psi(x_j, t)$ and $\mathbf{H}_D$ is a discrete version of H that is assumed to be Hermitian, $\mathbf{H}_D^\dagger = \mathbf{H}_D$.

A fundamental property of the exact solution is that the normalization is preserved,

$$\int_{-\infty}^{\infty} |\psi(x,t)|^2 dx = 1\,.$$

Does this remains true after discretizing x? The discrete analog is

$$\frac{d}{dt}(\boldsymbol{\psi}^\dagger\boldsymbol{\psi}) = \left(\frac{1}{i}\mathbf{H}_D\boldsymbol{\psi}\right)^\dagger \boldsymbol{\psi} + \frac{1}{i}\boldsymbol{\psi}^\dagger\mathbf{H}_D\boldsymbol{\psi} = 0\,.$$

The eigenvalues E_I of $\mathbf{H}_D$ are real, since $\mathbf{H}_D$ is Hermitian. Denote the eigenvectors by $\boldsymbol{\psi}_I$,

$$\mathbf{H}_D\boldsymbol{\psi}_I = E_I\boldsymbol{\psi}_I\,.$$

The eigenvectors may be assumed to be orthonormal.

Let us now consider three possible ways of discretizing t.

1. The explicit Euler method gives

   $$\boldsymbol{\psi}^{n+1} = \boldsymbol{\psi}^n - ih\mathbf{H}_D\boldsymbol{\psi}^n = \mathbf{T}\boldsymbol{\psi}^n\,,$$

   where the iteration matrix $\mathbf{T} = \mathbf{1} - ih\mathbf{H}_D$. The eigenvalues of $\mathbf{T}$ are $\lambda_I = 1 - ihE_I$ and satisfies $|\lambda_I| \geqslant 1$ for all I. Suppose now

   $$\boldsymbol{\psi}^0 = \sum_I c_I\boldsymbol{\psi}_I \qquad \Rightarrow \qquad \boldsymbol{\psi}^n = \sum_I c_I\lambda_I^n\boldsymbol{\psi}_I\,.$$

   From the fact that the eigenvectors are orthonormal, it follows that the normalization

   $$|\boldsymbol{\psi}^n|^2 = \boldsymbol{\psi}^{n\dagger}\boldsymbol{\psi}^n = \sum_I |c_I|^2\lambda_I^{2n}\,.$$

   This implies that $|\boldsymbol{\psi}^n|^2$ is not constant, but increases exponentially with n (for large n).

2. With an implicit Euler step instead, we obtain

   $$\boldsymbol{\psi}^{n+1} = \boldsymbol{\psi}^n - ih\mathbf{H}_D\boldsymbol{\psi}^{n+1} \qquad \Rightarrow \qquad \boldsymbol{\psi}^{n+1} = \mathbf{T}\boldsymbol{\psi}^n$$

   with $\mathbf{T} = (\mathbf{1} + ih\mathbf{H}_D)^{-1}$. The eigenvalues of $\mathbf{T}$ are this time given by $\lambda_I = 1/(1 + ihE_I)$, so $|\lambda_I| \leqslant 1$ for all I. It follows that $|\boldsymbol{\psi}^n|^2$ decreases steadily with n.

3. Finally, we consider the Crank-Nicholson method (or trapezoidal rule),

   $$\boldsymbol{\psi}^{n+1} = \boldsymbol{\psi}^n + \frac{h}{2}(-i\mathbf{H}_D\boldsymbol{\psi}^n - i\mathbf{H}_D\boldsymbol{\psi}^{n+1})\,,$$

   which can be written $\boldsymbol{\psi}^{n+1} = \mathbf{T}\boldsymbol{\psi}^n$ with

   $$\mathbf{T} = \left(\mathbf{1} + i\frac{h}{2}\mathbf{H}_D\right)^{-1}\left(\mathbf{1} - i\frac{h}{2}\mathbf{H}_D\right)\,.$$

The eigenvalues of this $\mathbf{T}$ are

$$\lambda_I = \frac{1 - i\frac{h}{2}E_I}{1 + i\frac{h}{2}E_I} .$$

Since λ_I is a ratio between two numbers that are the complex conjugates of each other, it follows that $|\lambda_I| = 1$ (for all I), and therefore that $|\boldsymbol{\psi}^n|^2$ is constant. So, the Crank-Nicolson method has the advantage that the normalization is preserved.

8 Appendix: χ^2 Minimization

[NR 15.5]

When discussing minimization, we mentioned the Newton method for root finding. This method can be used to solve the equation system $\nabla f(\mathbf{x}) = 0$, and thereby find the minimum of a given cost function f. Usually, this is an impractical method, but it is useful for function fitting.

Suppose we want to fit a set of data points (x_i, y_i) with errors σ_i, $i = 1, \ldots, N$, to a function $y(x; \mathbf{a})$, where $\mathbf{a} = (a_1, \ldots, a_M)$ are parameters ($M < N$). This is usually done by minimizing

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{N} \left(\frac{y_i - y(x_i, \mathbf{a})}{\sigma_i} \right)^2$$

with respect to $\mathbf{a}$.

If the function y depends linearly on the parameters a_i, this amounts to solving a relatively simple linear algebra problem. To see this, assume that y is a linear combination of some basis functions φ_k,

$$y(x; \mathbf{a}) = \sum_{k=1}^{M} a_k \varphi_k(x) .$$

χ^2 can then be expressed as

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{N} \left(\frac{y_i - \sum_{k=1}^{M} a_k \varphi_k(x_i)}{\sigma_i} \right)^2 = \sum_{i=1}^{N} \left(b_i - \sum_{k=1}^{M} A_{ik} a_k \right)^2 = |\mathbf{b} - \mathbf{A}\mathbf{a}|^2 ,$$

where $\mathbf{A}$ is the matrix with elements $A_{ik} = \varphi_k(x_i)/\sigma_i$ and $\mathbf{b}$ is the vector with components $b_i = y_i/\sigma_i$. Putting $\nabla\chi^2(\mathbf{a}) = 0$, we get a linear equation system for the parameters a_i,

$$\mathbf{A}^T\mathbf{A}\mathbf{a} = \mathbf{A}^T\mathbf{b}\,.$$

These equations are called the normal equations and can be solved, for example, by singular value decomposition (SVD; see NR 2.6 and p. 676).

If y is a non-linear function of the parameters a_i, it is necessary to take a different approach. A popular choice is to use the *Levenberg-Marquardt method*, which can be thought of as a combination of steepest descent and the Newton method. In the Levenberg-Marquardt method, we need the gradient of χ^2,

$$\frac{\partial\chi^2}{\partial a_k} = -2\sum_{i=1}^{N}\frac{y_i - y(x_i,\mathbf{a})}{\sigma_i^2}\frac{\partial y(x_i,\mathbf{a})}{\partial a_k}$$

as well as the Hessian,

$$\frac{\partial^2\chi^2}{\partial a_k\partial a_l} = 2\sum_{i=1}^{N}\frac{1}{\sigma_i^2}\left[\frac{\partial y(x_i,\mathbf{a})}{\partial a_k}\frac{\partial y(x_i,\mathbf{a})}{\partial a_l} - (y_i - y(x_i,\mathbf{a}))\frac{\partial^2 y(x_i,\mathbf{a})}{\partial a_k\partial a_l}\right]\,.$$

Here the last term should be small if we are near the minimum and the fit is good (it vanishes if y depends linearly on the parameters). For convenience, this term is neglected, which does not affect the final a_k values and which gives

$$\frac{\partial^2\chi^2}{\partial a_k\partial a_l} \approx A_{kl} = 2\sum_{i=1}^{N}\frac{1}{\sigma_i^2}\frac{\partial y(x_i,\mathbf{a})}{\partial a_k}\frac{\partial y(x_i,\mathbf{a})}{\partial a_l}\,.$$

Put

$$\beta_k = -\frac{1}{2}\frac{\partial\chi^2}{\partial a_k} \qquad \alpha_{kl} = \frac{1}{2}A_{kl} \qquad \delta a_k = a'_k - a_k\,.$$

In this notation, a steepest descent step has the form $\delta a_k = \text{constant}\cdot\beta_k$. The Levenberg-Marquardt method does not use a proper steepest descent step but rather

$$\delta a_k = \frac{1}{\lambda\alpha_{kk}}\beta_k\,,$$

where λ is a parameter (see below) and α_{kk} is introduced because the different components a_k may behave very differently; dividing by α_{kk} makes sense dimensionally. Note that α_{kk} is guaranteed to be positive because we are using the simplified version of the Hessian.

A Newton step can be written as

$$\mathbf{a}' = \mathbf{a} - \mathbf{A}(\mathbf{a})^{-1}\nabla\chi^2(\mathbf{a}) \qquad \Rightarrow \qquad \sum_l \alpha_{kl}\delta a_l = \beta_k \,.$$

The Levenberg-Marquardt method can be seen as an elegant way to interpolate between these two types of step, by changing the parameter λ. This is achieved by considering

$$\sum_{kl} \tilde{\alpha}_{kl}\delta a_l = \beta_k \qquad \text{where} \qquad \tilde{\alpha}_{kl} = \begin{cases} \alpha_{kk}(1+\lambda) & k = l \\ \alpha_{kl} & k \neq l \end{cases}$$

For $\lambda \to 0$, this becomes a Newton step, because the matrices α_{kl} and $\tilde{\alpha}_{kl}$ are the same in this limit. For large λ, on the other hand, we get a modified steepest descent step, because the diagonal elements dominate the matrix $\tilde{\alpha}_{kl}$.

The idea is to start with a large λ which gives a steepest descent-like step, and then gradually decrease λ as we get closer to the minimum, so that the step becomes more and more Newton-like.

Schematically, the algorithm can be written as:

1. Pick some initial $\mathbf{a}$ and a large λ.
2. Solve the equation system for $\delta\mathbf{a}$ and calculate $\chi^2(\mathbf{a}+\delta\mathbf{a})$.
3. If $\chi^2(\mathbf{a}+\delta\mathbf{a}) < \chi^2(\mathbf{a})$, update $\mathbf{a}$ to $\mathbf{a}_{\text{new}} = \mathbf{a}+\delta\mathbf{a}$, decrease λ, and continue at 2. If $\chi^2(\mathbf{a}+\delta\mathbf{a}) \geq \chi^2(\mathbf{a})$, increase λ (decrease the step size) and go back to 2 without changing $\mathbf{a}$.

Iterate till some stopping criterion is fulfilled.