Computational Physics II

Lecture notes

Wolfgang Dobler

Revision: 1.50
Date: 2007/07/31 21:16:31

Contents

Chapter 1

More on Runge–Kutta methods

1.1 Runge–Kutta revisited

Explicit s-stage (*stage* = *substep*) Runge–Kutta scheme:

$$\begin{aligned}
\tau_1 &= t_0 \,, & \boldsymbol{\eta}_1 &= \mathbf{y}_0 \,, & \mathbf{k}_1 &= h\mathbf{f}(\tau_1, \boldsymbol{\eta}_1) \,, && (1.1)\\
\tau_2 &= t_0 + c_2 h \,, & \boldsymbol{\eta}_2 &= \mathbf{y}_0 + a_{21}\mathbf{k}_1 \,, & \mathbf{k}_2 &= h\mathbf{f}(\tau_2, \boldsymbol{\eta}_2) \,, && (1.2)\\
\tau_3 &= t_0 + c_3 h \,, & \boldsymbol{\eta}_3 &= \mathbf{y}_0 + a_{31}\mathbf{k}_1 + a_{32}\mathbf{k}_2 \,, & \mathbf{k}_3 &= h\mathbf{f}(\tau_3, \boldsymbol{\eta}_3) \,, && (1.3)\\
&\vdots & &\vdots & &\vdots && (1.4)\\
\tau_s &= t_0 + c_s h \,, & \boldsymbol{\eta}_s &= \mathbf{y}_0 + a_{s1}\mathbf{k}_1 + a_{s2}\mathbf{k}_2 + \cdots + a_{s,s-1}\mathbf{k}_{s-1} \,, & \mathbf{k}_s &= h\mathbf{f}(\tau_s, \boldsymbol{\eta}_s) \,, && (1.5)\\
t_1 &= t_0 + h \,, & \mathbf{y}_1 &= \mathbf{y}_0 + b_1\mathbf{k}_1 + b_2\mathbf{k}_2 + \cdots + b_{s-1}\mathbf{k}_{s-1} + b_s\mathbf{k}_s \,. & & && (1.6)
\end{aligned}$$

The coefficients c_i, a_{ik} and b_i are conveniently represented in a *Butcher tableau*

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^{\mathsf{T}} \end{array} \quad = \quad \begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & & \ddots & & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array} \tag{1.7}$$

where all omitted elements a_{ij} vanish.

Examples

Euler scheme (= 1-step first order scheme).

Explicit Euler scheme

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \tag{1.8}$$

Implicit Euler scheme:

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{1.9}$$

2-stage 2nd order Example: the "tangent scheme"

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array} , \tag{1.10}$$

3-stage 3rd order Example: "classical" 3-stage Runge–Kutta scheme

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \tag{1.11}$$

4-stage 4rd order Example: classical 4-th order Runge–Kutta scheme:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \tag{1.12}$$

Note: 5th order requires 6 stages!

1.2 Step-size control

The time step δt cannot be larger than any of the physical time scales involved. However, these vary widely from problem to problem, and can even vary in time for one and the same problem. Hence we want a procedure to automatically set and adjust the time step δt.

δt will depend on the desired accuracy, so we need to monitor (an estimate of) the error.

Local error:

$$\delta_n \equiv y_n - y_n^{(\text{exact})} ,$$

assuming y_{n-1} was correct

Global error:

$$\Delta_n \equiv y_n - y_n^{(\mathrm{exact})} ,$$

assuming y_0 was correct

For small δt:

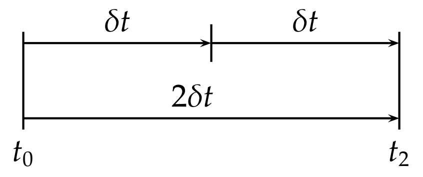
$$|\Delta_n| \approx \left| \sum_{i=1}^{n} \delta_i \right| \leq \sum_{i=1}^{n} |\delta_i|$$

Error control: Estimate δ_i and try to ensure that $\delta_i < \varepsilon$ for given error tolerance ε. If not, replace step with new time step using smaller δt.

1.2.1 The Milne device

How do we get the estimate $\tilde{\delta}_i$?

One method is to compare 2 steps at δt with one step at $2\delta t$:



If we know our method is of order p, then

$$y_1^{(\delta t)} = y_0 + C\,\delta t^{p+1} + O\left(\delta t^{p+2}\right) , \tag{1.13}$$

$$y_2^{(\delta t)} = y_0 + 2C\,\delta t^{p+1} + O\left(\delta t^{p+2}\right) , \tag{1.14}$$

$$y_2^{(2\delta t)} = y_0 + C\,2^p \delta t^{p+1} + O\left(\delta t^{p+2}\right) . \tag{1.15}$$

This allows us to determine C and thus the error $2C\delta t^{p+1}$ of $y_2^{(\delta t)}$:

$$y_2^{(2\delta t)} - y_2^{(\delta t)} = (2^{p+1} - 2)C\,\delta t^p + O\left(\delta t^{p+1}\right) , \tag{1.16}$$

thus the error estimate for two steps is

$$\tilde{\delta}_1 + \tilde{\delta}_2 = \frac{1}{2^p - 1}(y_2^{(2\delta t)} - y_2^{(\delta t)}) . \tag{1.17}$$

How much extra effort does it take to get this error estimate?

without error estimate: $2s$ evaluations of right-hand-side for s-step scheme;

with error estimate: $2s + (s-1)$ evaluations of right-hand-side ($\mathrm{rhs}(t_0, y_0)$ is already known).

Thus, the ratio is

$$\frac{3-1/s}{2} \approx \frac{3}{2} \,. \tag{1.18}$$

More generally: Combine 2 methods of equal order with known error ratio to determine $\tilde{\delta}$. This method is called the *Milne device*.

Note: We could even combine the two methods (single and double step) to a higher-order method using Richardson extrapolation[1] But nowadays people prefer. . .

1.2.2 . . . Embedded Runge-Kutta schemes

Idea: use two schemes, one of order p, the other of order p+1. Then

$$\begin{aligned} y_1^{(p)} &= y_1^{(exact)} + C\delta t^{p+1} + O\left(\delta t^{p+2}\right) \,, && (1.19)\\ y_1^{(p+1)} &= y_1^{(exact)} + O\left(\delta t^{p+2}\right) \,. && (1.20)\\ & && (1.21) \end{aligned}$$

Thus,

$$\begin{aligned} \tilde{\delta}_1^{(p)} = y_1^{(p)} - y_1^{(p+1)} &= C\,\delta t^{p} + O\left(\delta t^{p+1}\right) && (1.22)\\ &= \delta_1^{(p)}\left[1 + O\left(\delta t\right)\right] \,, && (1.23) \end{aligned}$$

is an estimator for the local error of $y_1^{(p)}$.

In practise: use error estimate $\tilde{\delta}_1^{(p)}$, but continue next step with $y_1^{(p+1)}$, for which we have no error estimate, but which is very likely to be more accurate (*local extrapolation*).

Apparent problem: doing two Runge–Kutta schemes in tandem is expensive.

Solution: Construct schemes that share the same coefficients c_i, a_{ik} (Fehlberg).

As an example, consider the Cash–Karp scheme

$$\begin{array}{c|cccccc} 0 & & & & & & \\ \frac{1}{5} & \frac{1}{5} & & & & & \\ \frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & \\ \frac{3}{5} & \frac{3}{10} & -\frac{9}{10} & \frac{6}{5} & & & \\ 1 & -\frac{11}{54} & \frac{5}{2} & -\frac{70}{27} & \frac{35}{27} & & \\ \frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & \\ \hline & \frac{37}{378} & 0 & \frac{250}{621} & \frac{125}{594} & 0 & \frac{512}{1771} \\ \hline & \frac{2825}{27648} & 0 & \frac{18575}{48384} & \frac{13525}{55296} & \frac{277}{14336} & \frac{1}{4} \end{array} \tag{1.24}$$

[1]We would then not have an error estimate for that method, but using $\tilde{\delta}$ for the lower-order method gives normally a quite conservative error estimate. See 'local extrapolation' below.

The two lowest lines represent the coefficients b_i for two different schemes: one of 5th order (second last line) and one of 4th order (last line).

How to calculate the time step for the next step or the refinement step: From $|\delta| \sim \delta t^p$ and the requirement that ideally $|\delta| = \varepsilon$, we find that the old and new value of δt are related by

$$\frac{|\delta|}{\varepsilon} = \left(\frac{\delta t_{\rm old}}{\delta t_{\rm new}}\right)^{p+1} , \tag{1.25}$$

and hence

$$\delta t_{\rm new} = \delta t_{\rm old} \left(\frac{\varepsilon}{|\delta|}\right)^{1/p+1} \tag{1.26}$$

F90 code:

Cash–Karp-4-5

```
do while (t0 < tmax)
    call do_step(t0,y0,dt, y4th,y5th)
    absdelta = abs(y4-y5)
    dt = 0.9*dt*(epsi/absdelta)**(1/(p+1))  ! 0.9 = safety factor
    if (absdelta < epsi) then
        ! prepare next step
        t0 = t+dt
        y0 = y5th                  ! local extrapolation
    else
        ! need to redo step with previous t0, y0 and new dt
    endif
enddo
```

Example: Consider the Curtiss–Hirschfelder equation

$$\dot{y} = -50(y - \cos t) \qquad \text{with } y(0) = 1 , \tag{1.27}$$

which has the solution

$$y = \frac{2500}{2501}\cos t + \frac{50}{2501}\sin t + \frac{1}{2501}e^{-50t} . \tag{1.28}$$

Figure 1.1 shows the solution f and time step δt as a function of time.

1.3 Stability

For the linear ODE

$$\dot{y} = \gamma y \tag{1.29}$$

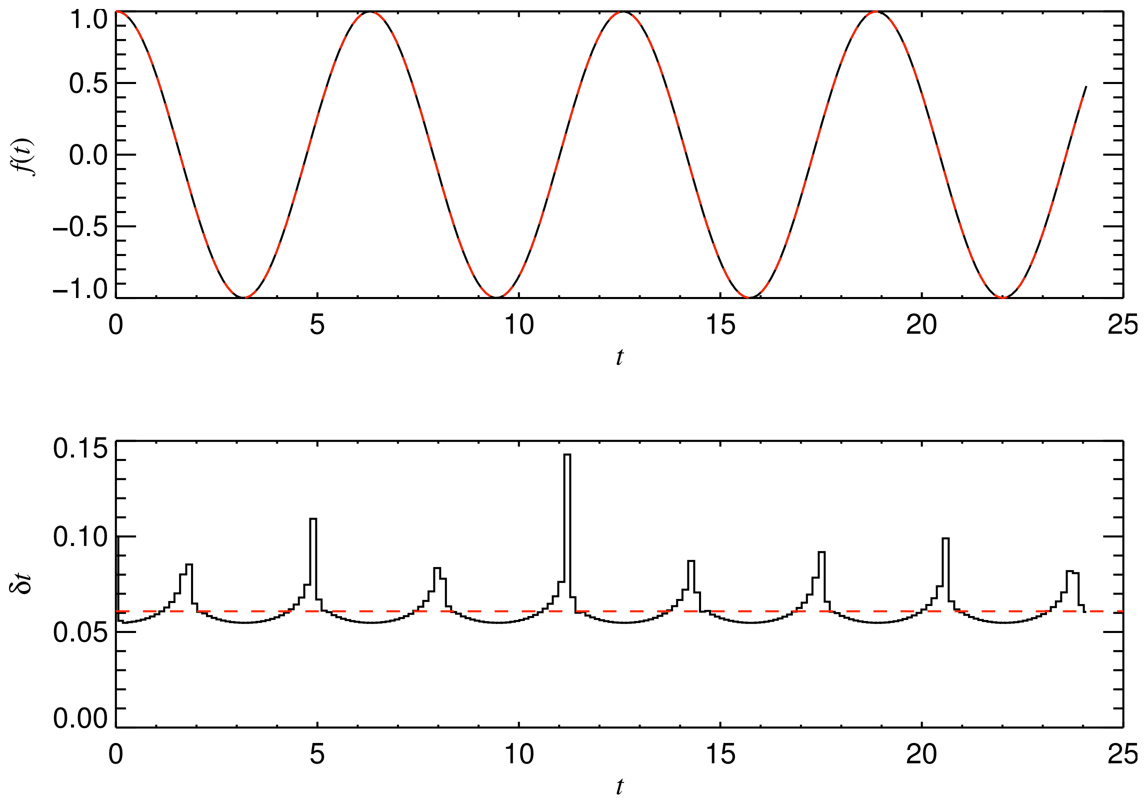


Figure 1.1: Top: Numerical solution $f(t)$ (black continuous line) and analytical solution (red dashed line) for the Curtiss–Hirschfelder equation (1.27). Bottom: Dynamical time step $\delta t(t)$. The numerical method used is the Cash–Karp embedded Runge–Kutta scheme. The dotted line here shows the average time step.

with the complex constant γ, we know that the solution

$$y(t) = y_0 e^{\gamma t} \tag{1.30}$$

decays for $t \to \infty$ if and only if $\Re\gamma < 0$.

A time-stepping scheme is *stable* (for this equation) if its approximations y_n tend to zero for $n \to \infty$, otherwise it is *unstable*.

If we apply the explicit Euler scheme to Eq. (1.29), we find

$$y_1 = y_0 + \delta t \gamma y_0 = (1 + \gamma\, \delta t) y_0 = A y_0 \,, \tag{1.31}$$

where A is the complex amplification factor. If $|A| < 1$, the numerical solution y_n decays for $n \to \infty$, while for $|A| > 1$ it grows unboundedly. Thus, we get the stability requirement

$$|1 + \gamma\, \delta t| < 1 \,. \tag{1.32}$$

For any explicit two-step second-order (i.e. $s = p = 2$) scheme,

$$A = 1 + \Gamma + \frac{\Gamma^2}{2} \,, \tag{1.33}$$

where we have introduced

$$\Gamma \equiv \gamma\, \delta t \,. \tag{1.34}$$

For any explicit $s = p = 3$ scheme, we get

$$A = 1 + \Gamma + \frac{\Gamma^2}{2} + \frac{\Gamma^3}{6} \,, \tag{1.35}$$

and for $s = p = 4$ scheme, we get

$$A = 1 + \Gamma + \frac{\Gamma^2}{2} + \frac{\Gamma^3}{6} + \frac{\Gamma^4}{24} \,. \tag{1.36}$$

Figure 1.2 shows the stable and unstable values of Γ for these Runge–Kutta schemes with $s = p$ from 1 to 4. One can clearly see that explicit Runge–Kutta schemes have only a limited area of stability.

1.4 Systems of ordinary differential equations

Runge–Kutta methods lend themselves directly to the solution of systems of ordinary differential equations, provided the initial conditions all refer to the same time t_0. The only thing that changes is that now the variable y and the right-hand side of the equation become vectors:

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t) \,. \tag{1.37}$$

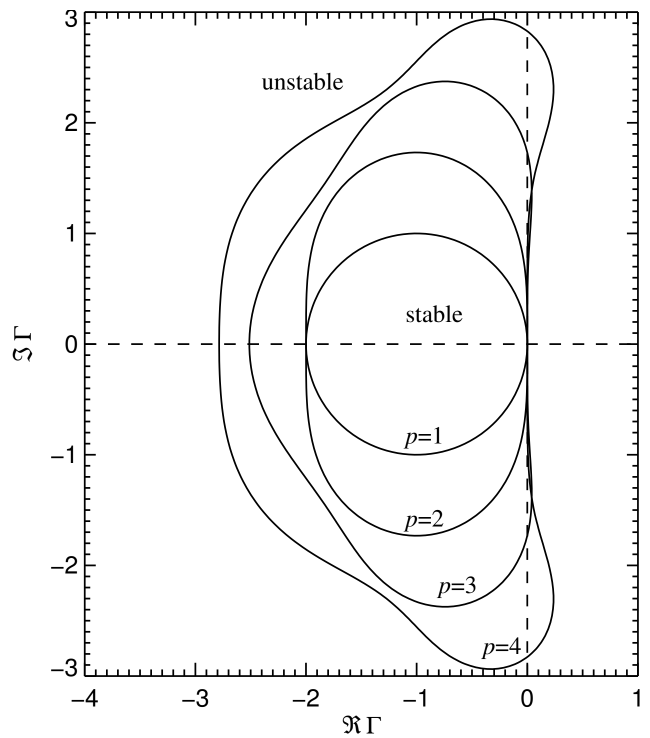


Figure 1.2: Complex stability domains for explicit Runge–Kutta methods of order 1 to 4 applied to Eq. (1.29).

Note: If a scalar Runge–Kutta solver is coded in a programming language with array syntax (Fortran90, Octave, IDL, PerlDL, etc.), it will work out of the box for systems of equations as well.

Note: A k^{th} order differential equation

$$y^{(k)} = f\left(y, \dot{y}, \ldots, y^{(k-2)}, y^{(k-1)}, t\right) \tag{1.38}$$

can always be transformed into a system of k first-order ODEs, using the substitution

$$z_1 \equiv y\,, \quad z_2 \equiv \dot{y}\,, \quad \ldots, \quad z_{k-1} \equiv y^{(k-2)}\,, \quad z_k \equiv y^{(k-1)}\,. \tag{1.39}$$

The resulting system takes the form

$$\begin{aligned}
\dot{z}_k &= f(z_0, z_1, \ldots, z_{k-2}, z_{k-1}, t)\,, && (1.40)\\
\dot{z}_{k-1} &= z_{k-2}\,, && (1.41)\\
&\vdots \\
\dot{z}_2 &= z_3\,, && (1.42)\\
\dot{z}_1 &= z_2\,. && (1.43)\\
& && (1.44)
\end{aligned}$$

Stability of a linear system A linear autonomous system of k ODEs has the form

$$\dot{\mathbf{y}} = \mathcal{M}\mathbf{y} , \tag{1.45}$$

where $\mathcal{M}$ is a square $k \times k$ matrix. For arbitrary initial conditions, the solution vector $\mathbf{y}_n$ will tend to zero if, and only if all eigenvalues μ_i of $\mathcal{M}$ satisfy $\Re \mu_i < 0$.

For time-stepping schemes, this translates to [for Eq. (1.45)]

$$\mathbf{y}_{n+1} = \mathcal{A}\mathbf{y}_n , \tag{1.46}$$

with the amplification matrix $\mathcal{A}$. The scheme is stable if and only if

$$\varrho(\mathcal{A}) < 1 , \tag{1.47}$$

where $\varrho(\mathcal{A}) \equiv \max_i |a_i|$ is the spectral radius (maximum modulus of eigenvalues a_i) of $\mathcal{A}$. In other words, stability is equivalent to

$$|a_i| < 1 \quad \forall\, a_i .$$

Example: For the (explicit) Euler scheme, we have

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \delta t\, \mathcal{M} \cdot \mathbf{y}_n = (\mathbb{1} + \delta t\, \mathcal{M})\mathbf{y}_n , \tag{1.48}$$

thus $\mathcal{A} = \mathbb{1} + \delta t\, \mathcal{M}$.

Let $\mathbf{e}_k$ and μ_k represent the eigenvectors and eigenvalues of $\mathcal{M}$:

$$\mathcal{M} \cdot \mathbf{e}_k = \mu_k \mathbf{e}_k . \tag{1.49}$$

Then

$$\mathcal{A} \cdot \mathbf{e}_k = \mathbf{e}_k + \mu_k \mathbf{e}_k = (1 + \delta t\, \mu_k)\mathbf{e}_k , \tag{1.50}$$

and thus the $\mathbf{e}_k$ are also eigenvectors of $\mathcal{A}$, and the corresponding eigenvalues are

$$a_k = 1 + \delta t\, \mu_k . \tag{1.51}$$

The stability requirement for the Euler scheme is thus

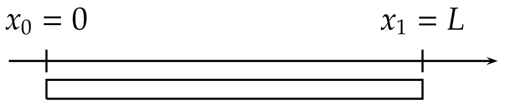
$$|1 + \delta t\, \mu_k| < 1 , \tag{1.52}$$

in analogy to Eq. (1.32).

1.5 Boundary-value problems

If a system of differential equations has boundary conditions specified for different times t_0, t_1 (and possibly more), then we have a *boundary value problem*, rather than an *initial value problem*. As in Sec. 1.4, such a system of ODEs may originate from one or several higher-order equations.

Example: Consider the steady state of a thermally conducting rod of length L. At the left end ($x = 0$), the temperature is constant, while the right end ($x = L$) is insulating. Along the rod, a heating function $q(x)$ (units: W/m^3) is applied.



The thermal heat flux (strictly speaking: energy flux density, units W/m^2) is given by

$$F = -\lambda \frac{dT}{dx} , \tag{1.53}$$

where λ is the *thermal conductivity*. Thus, the boundary condition $F = 0$ at $x = L$ leads to $dT/dx = 0$.

Along the rod, the volume heating rate q must equal div $\mathbf{F}$, and thus

$$\frac{d}{dx}\left(\lambda \frac{dT}{dx}\right) + q(x) = 0 . \tag{1.54}$$

Thus, the complete problem is

$$\begin{aligned} (\lambda T')' &= -q(x) , && (1.55)\\ T(0) &= T_0 , && (1.56)\\ T'(L) &= 0 . && (1.57) \end{aligned}$$

To use our Runge–Kutta methods, we need to transform this second-order equation into two first-order ones; a natural choice for the auxiliary variable is F (but we could also just use dT/dx):

$$\begin{aligned} F'(x) &= q(x) , && (1.58)\\ T'(x) &= -\frac{1}{\lambda} F(x) , && (1.59)\\ T(0) &= T_0 , && (1.60)\\ F(L) &= 0 . && (1.61) \end{aligned}$$

1.5.1 Shooting method

One of the most popular methods for solving boundary value problems is the *shooting method*, where one integrates the corresponding initial value problem from one end of the interval to the other, using guesses for the initial values not specified by boundary conditions, and then tuning the guessed values such that the boundary condition at the other end of the interval is satisfied.

In our example, the shooting method goes like this:

1. Start at $x = 0$ with $T(0) = 0$ and $F(0) = \varphi$ (which initially is arbitrary)
2. Using a standard method for initial value problems (e.g. any Runge–Kutta method), integrate Eqs. (1.58), (1.59) from $x = 0$ to L, which yields a value $F(L;\varphi)$ for F, which will depend on the φ chosen.

We thus have a mapping $\varphi \mapsto f(\varphi) \equiv F(L;\varphi)$. Now we simply apply any convenient root-finding method to the equation $f(\varphi) = 0$ to find the appropriate value of φ.

General problem: N first-order equations:[2]

$$\frac{dy_i}{dt} = f_i(y_1, y_2, \ldots, y_N, t) \qquad i = 1, 2, \ldots, N\,. \tag{1.62}$$

n_1 left and $n_2 = N - n_1$ right boundary conditions:

$$\begin{aligned} y_i(t_1) &= A_i \qquad i = 1, \ldots, n_1\,, & (1.63)\\ y_{i+j}(t_2) &= B_i \qquad i = 1, \ldots, n_2\,, & (1.64) \end{aligned}$$

where $n_1 + n_2 = N$ and $0 \le j \le n_1$ [3] Integrate the initial value problem from t_1 with n_1 correct boundary conditions and n_2 guessed values $y_{n_1+1}(t_1) = \varphi_1, \ldots, y_N(t_1) = \varphi_{n_2}$. The boundary conditions (1.64) then yield n_2 equations for the n_2 unknowns $\varphi_1, \ldots, \varphi_{n_2}$. This system can be solved with any (multidimensional) root-finding routine.

1.5.2 Problems that can be reduced to standard boundary value problem

Eigenvalue problem

An eigenvalue problem

$$\begin{aligned} \frac{dy_i}{dt} &= f_i(y_1, y_2, \ldots, y_N, \lambda) \qquad i = 1, 2, \ldots, N\,, & (1.65)\\ y_i(t_1) &= A_i \qquad i = 1, \ldots, n_1\,, & (1.66)\\ y_{i+j}(t_2) &= B_i \qquad i = 1, \ldots, n_2 & (1.67) \end{aligned}$$

only has solutions for certain values of the eigenvalue λ. Here $n_1 + n_2 = N{+}1$ and $0 \le j \le n_1{-}1$, and the problem would be overdetermined, were it not for the additional free parameter λ.

We add the additional variable $y_{N+1} = \lambda$ satisfying the equation

$$\frac{dy_{N+1}}{dt} = 0 \qquad (\lambda \text{ is constant after all})\,, \tag{1.68}$$

and end up with a standard boundary value problem for the $N + 1$ variables $y_1, \ldots, y_{N+1}$.

[2] This assumes a certain ordering of the variables which can always be constructed.

[3] j determines the 'overlap' of boundary conditions: if $j = n_1$, every variable has exactly one boundary condition to satisfy. Conversely, if $j < n_1$, then $n_1 - j$ variables have two boundary conditions (one at t_1 and one at t_2), while another $n_1 - j$ variables have no boundary condition.

Example: Consider the acoustic eigenmodes of a string of tension F with linear mass density ('mass load') $\eta \equiv dm/dx$. The Helmholtz equation for the displacement $y(x)$ as a function of x is

$$y'' = -k^2 y \,, \tag{1.69}$$

where the wave number k is related to the oscillation frequency ω via

$$k^2 = \frac{\eta}{F}\omega^2 \,; \tag{1.70}$$

the quantity $\lambda \equiv \omega^2$ is our eigenvalue. As the string is fixed on either side, we have the two boundary conditions

$$y(0) = 0 \,, \qquad y(L) = 0 \,. \tag{1.71}$$

So we have a second-order equation and two boundary conditions — how do we fix the additional degree of freedom?

The answer is: we can choose one more boundary condition, because the amplitude of an eigenfunction is arbitrary, while we want our problem to have a unique solution. For example, we can require $y'(0) = z(0) = 1$ and then get

$$\begin{aligned} y' &= z \,, && (1.72)\\ z' &= -\frac{\eta}{F}\,\lambda\, y \,, && (1.73)\\ \lambda' &= 0 \,, && (1.74) \end{aligned}$$

with the boundary conditions

$$\begin{aligned} y(0) &= 0 \,, && (1.75)\\ y(L) &= 0 \,, && (1.76)\\ z(0) &= 1 \,. && (1.77) \end{aligned}$$

Free boundary problems

Consider the case of N equations

$$\begin{aligned} \frac{dy_i}{dt} &= f_i(y_1, y_2, \ldots, y_N) \qquad && i = 1, 2, \ldots, N \,, && (1.78)\\ y_i(t_1) &= A_i && i = 1, \ldots, n_1 \,, && (1.79)\\ y_{i+j}(t_2) &= B_i && i = 1, \ldots, n_2 \,, && (1.80) \end{aligned}$$

where the position of the left boundary t_1 is given, but the position t_2 of the right boundary is unknown. Similar to the eigenvalue problem above, the position t_2 adds one degree of freedom, so we require $N+1$ boundary conditions for a well-defined solution (thus at least one variable will have two boundary conditions), and $n_1 + n_2 = N{+}1$ and $0 \le j \le n_1{-}1$.

In this case, we can introduce the additional variable $y_{N+1} \equiv t_2 - t_1$, together with the equation

$$\frac{dy_{N+1}}{dt} = 0 \qquad (t_1 \text{ and } t_2 \text{ cannot depend on } t!) \,, \tag{1.81}$$

Substituting the independent variable,

$$t - t_1 \equiv \tau \, y_{N+1} \qquad \text{(and thus } dt = y_{N+1}\, d\tau) \ , \tag{1.82}$$

we get

$$\frac{dy_i}{d\tau} = f_i(y_1, y_2, \dots, y_N)\, y_{N+1} \qquad i = 1, 2, \dots, N \ , \tag{1.83}$$

$$\frac{dy_{N+1}}{d\tau} = 0 \ , \tag{1.84}$$

$$y_i(t_1) = A_i \qquad i = 1, \dots, n_1 \ , \tag{1.85}$$

$$y_{i+j}(t_2) = B_i \qquad i = 1, \dots, n_2{+}1 \ , \tag{1.86}$$

which again has the standard form of a boundary value problem of $N + 1$ equations.

Chapter 2

Random numbers and Monte Carlo methods

2.1 Basic probability theory

A *random variable* X can take values x (or y, or $2\psi - 1$, or 0.7).

$X = 0.7$ is an *event*, as is $X \leq 1.4$. An event ω has a *probability* $P(\omega)$ with

$$0 \leq P(\omega) \leq 1 \; . \tag{2.1}$$

The union of two events has the probability

$$P(\omega_1 \cup \omega_2) = P(\omega_1) + P(\omega_2) - P(\omega_1, \omega_2) \tag{2.2}$$

where we have used the notation $P(\omega_1, \omega_2) \equiv P(\omega_1 \cap \omega_2)$ (the property that both ω_1 *and* ω_2 occur).

Discrete probability distributions: Random variable X takes on discrete set of values; typically equidistant values The *probability distribution function* is defined as

$$F_X(n) \;\equiv\; P(X \leq n) = \sum_{k \leq n} p_k \; , \tag{2.3}$$

where

$$p_n \;\equiv\; P(X{=}n) \tag{2.4}$$

is the probability of the event 'X equals n'.

Continuous probability distributions: Random variable X takes on value from an interval, often $(-\infty, \infty)$. The *probability distribution function* is

$$F_X(x) \equiv P(X \leq n) = \int_{-\infty}^{x} f_X(x')\, dx' \ , \tag{2.5}$$

where

$$f_X(x) \equiv \frac{dF_X(x)}{dx} = \frac{P(x < X < x+dx)}{dx} \tag{2.6}$$

is called *probability density function, PDF* or *probability density* or *density function*.

Note 1: For a discrete distribution, we can write

$$f_X(x) \equiv \frac{dF_X(x)}{dx} = \sum_n p_n \delta(x-x_n) \ , \tag{2.7}$$

where $\delta(\cdot)$ is Dirac's delta function.

Note 2: For any distribution, $F_X(-\infty) = 0$ and $F_X(\infty) = 1$.

Examples: Bernoulli distribution: X can take on values 0 or 1.

$$p_0 = p \ , \qquad p_1 = 1-p \ . \tag{2.8}$$

which implies

$$F_X(0) = p \ , \qquad F_X(1) = 1 \ . \tag{2.9}$$

Uniform distribution $\mathcal{U}$: If $U \sim \mathcal{U}(0,1)$,[1]

[rectangle function]

then

$$f_U(x) = \begin{cases} 0 & x < 0 \\ 1 & 0 \leq x \leq 1 \\ 0 & x > 1 \end{cases} \ , \tag{2.10}$$

which implies

$$F_U(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x \leq 1 \\ 1 & x > 1 \end{cases} \ . \tag{2.11}$$

[1] This is a short notation we will use for a few important distributions. It reads 'U is distributed according to the uniform distribution function with offset 0 and width 1'.

2.1.1 Expectation value, variance, covariance

The *expectation value* EX of X is defined as

$$EX \equiv \int_{-\infty}^{\infty} x \, dF_X(x) . \tag{2.12}$$

For a continuous distribution, this becomes

$$EX = \int_{-\infty}^{\infty} x \, f_X(x) \, dx , \tag{2.13}$$

and for a discrete one

$$EX = \sum_{k=-\infty}^{\infty} x_k \, p_k . \tag{2.14}$$

If X is a random variable, then an arbitrary function $g(X)$ is, too. So we can define

$$Eg(X) \equiv \int_{-\infty}^{\infty} g(x) \, dF_X(x) = \begin{cases} \int_{-\infty}^{\infty} g(x) \, f_X(x) \, dx \\ \sum_{k=-\infty}^{\infty} g(x_k) \, p_k \end{cases} \tag{2.15}$$

We can write

$$f_X(x) = \int \delta(\xi - x) \, f_X(\xi) \, d\xi = E\delta(X - x) . \tag{2.16}$$

Note: The expectation value (2.12) is linear in X, thus in particular

$$E(X + Y) = EX + EY . \tag{2.17}$$

The *variance* $V(X)$ is defined as

$$V(X) \equiv E(X - EX)^2 , \tag{2.18}$$

which can be rewritten as

$$V(X) = E[X^2 - 2X \, EX + (EX)^2] = EX^2 - 2EX \, EX + (EX)^2 = EX^2 - (EX)^2 . \tag{2.19}$$

The variance of $X + Y$ is

$$\begin{aligned} V(X + Y) &= E(X - EX + Y - EY)^2 \\ &= V(X) + V(Y) + 2E(X - EX)(Y - EY) \\ &= V(X) + V(Y) + 2 \, \mathrm{Kov}(X, Y) , \end{aligned} \tag{2.20}$$

where

$$\mathrm{Kov}(X,Y) \equiv E(X-EX)(Y-EY) = EXY - EX\,EY \tag{2.21}$$

is the *covariance* of X and Y.

The *correlation coefficient*

$$\varrho(X,Y) \equiv \frac{\mathrm{Kov}(X,Y)}{\sqrt{V(X)V(Y)}} \tag{2.22}$$

satisfies $-1 \le \varrho(X,Y) \le 1$.

If $\mathrm{Kov}(X,Y) = \varrho(X,Y) = 0$ (e.g. if X and Y are independent, see below), then

$$V(X+Y) = V(X) + V(Y)\ . \tag{2.23}$$

2.1.2 Joint and conditional probabilities

Consider 2 random variables X, Y.

$$F_{X,Y}(x,y) \equiv P(X{\le}x, Y{\le}y) \tag{2.24}$$

is the *joint* distribution function, and

$$f_{X,Y}(x,y) \equiv \frac{P(x{<}X{<}x{+}dx,\ y{<}Y{<}y{+}dy)}{dx\,dy} = \partial_x\partial_y F_{X,Y}(x,y) \tag{2.25}$$

is the joint probability density function.

If Y is irrelevant, X has the density function

$$f_X(x) = P(x{<}X{<}x{+}dx, Y \text{ irrelevant}) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)\,dy \tag{2.26}$$

(sometimes called the *marginal distribution*), and similar for f_Y.

Expectation values are now

$$Eg(X,Y) = \iint g(x,y) f_{X,Y}(x,y)\,dx\,dy\ . \tag{2.27}$$

If

$$f_{X,Y}(x,y) = f_X(x) f_Y(y)\ , \tag{2.28}$$

X and Y are called *independent* random variables. In that case,

$$\begin{aligned}\mathrm{Kov}(X,Y) &= EXY - EX\,EY = \iint f_{X,Y}(x,y)\,dx\,dy - EX\,EY \\ &= \int f_X(x)\,dx \int f_Y(y)\,dx - EX\,EY = 0\ .\end{aligned} \tag{2.29}$$

But $\mathrm{Kov}(X, Y) = 0$ (X and Y are *uncorrelated*) is *not* sufficient for stochastic independence. Counter example:

$$X \sim \mathcal{U}(0,1) , \qquad Y = 4X(1-X) . \tag{2.30}$$

The *conditional probability* of ω_1 under the condition ω_2 is

$$P(\omega_1|\omega_2) = \frac{P(\omega_1, \omega_2)}{P(\omega_2)} . \tag{2.31}$$

For example,

$$P(X{\leq}x|Y{<}2) = \frac{P(X{\leq}x|Y{<}2)}{P(Y{<}2)} . \tag{2.32}$$

If ψ_k ($k = 0, 1, \ldots$) is a complete set of mutually exclusive events, i.e.

$$\psi_k \cap \psi_l = \emptyset \text{ for } k \neq l , \qquad \text{and } \sum_k P(\psi_k) = 1 , \tag{2.33}$$

then

$$\sum_k P(\omega|\psi_k)P(\psi_k) = \sum_k P(\omega, \psi_k) = P(\omega) , \tag{2.34}$$

i.e.

$$P(\omega) = \sum_k P(\omega|\psi_k)P(\psi_k) . \tag{2.35}$$

Equation (2.35) is called the *total probability theorem*.

2.1.3 Distribution of sums of random variables

Let X and Y be two random variables with probability density functions f_X and f_Y, respectively. What is the distribution of $Z \equiv X + Y$?

We can write the probability density of Z as

$$f_Z(z) = E\delta(X + Y - z) = \iint dx\, dy\, f_{X,Y}(x, y)\, \delta(x{+}y{-}z) = \int_{-\infty}^{\infty} f_{X,Y}(x, z{-}x)\, dx . \tag{2.36}$$

If X and Y are independent, this becomes

$$f_Z(z) = \int_{-\infty}^{\infty} f_X(x) f_Y(z{-}x)\, dx , \tag{2.37}$$

i.e. the density function of the sum $X + Y$ is the *convolution* of the individual density functions.

2.1.4 Individual distributions

Geometric distribution:

$$p_n = p(1-p)^n\ , \qquad F_X(n) = 1-(1{-}p)^{n+1} \tag{2.38}$$

Mean value and variance:

$$EX = \frac{1{-}p}{p}\ , \qquad V(X) = \frac{1{-}p}{p^2}\ . \tag{2.39}$$

Binomial distribution $\mathcal{B}(N,p)$:

$$P(n|N) = \binom{N}{n} p^n (1-p)^{N-n} \tag{2.40}$$

Mean value and variance:

$$EX = Np\ , \qquad V(X) = Np(1{-}p)\ . \tag{2.41}$$

Poisson distribution:

$$P(n) = \frac{\lambda^n e^{-\lambda}}{n!}\ . \tag{2.42}$$

Mean value and variance:

$$EX = \lambda\ , \qquad V(X) = \lambda\ . \tag{2.43}$$

Interpretation: radioactive decay, decay rate γ [decays/s], time interval Δt, parameter $\lambda = \alpha\Delta t$. Probability of getting n decays during the time interval is $P(n)$.

Uniform distribution $\mathcal{U}(x_0,w)$:

$$F_X(x) = \begin{cases} 0 & x < x_0 \\ \dfrac{x-x_0}{w} & x_0 \le x \le x_0{+}w \\ 1 & x > x_0{+}w \end{cases}\ , \qquad f_X(x) = \begin{cases} 0 & x < x_0 \\ \dfrac{1}{w} & x_0 \le x \le x_0{+}w \\ 0 & x > x_0{+}w \end{cases} \tag{2.44}$$

Normal distribution $\mathcal{N}(\mu,\sigma)$:

$$F_X(x) = \frac{1 + \operatorname{erf}\dfrac{x{-}\mu}{\sqrt{2}\sigma}}{2}\ , \qquad f_X(x) = \frac{1}{2\pi\sigma^2} e^{-(x-\mu)^2/(2\sigma^2)}\ , \tag{2.45}$$

where $\operatorname{erf} z = (2/\sqrt{\pi}) \int_0^z e^{-\zeta^2}\, d\zeta$ is the *error function*.
Mean value and variance:

$$EX = \mu\ , \qquad V(X) = \sigma^2\ . \tag{2.46}$$

Exponential distribution:

$$F_X(x) = 1 - e^{-\alpha x} \;, \qquad f_X(x) = \alpha e^{-\alpha x} \;. \tag{2.47}$$

Mean value and variance:

$$EX = \frac{1}{\alpha} \;, \qquad V(X) = \frac{1}{\alpha^2} \;. \tag{2.48}$$

Interpretation: radioactive decay, α is the decay rate; x is waiting time for first decay event.

Gamma distribution:

$$F_X(x) = \frac{\gamma(\beta, \alpha x)}{(\beta{-}1)!} \text{ (incomplete gamma function)} \;, \qquad f_X(x) = \frac{x^{\beta-1}\alpha^{\beta} e^{-\alpha x}}{(\beta{-}1)!} \;. \tag{2.49}$$

Mean value and variance:

$$EX = \frac{1}{\alpha} \;, \qquad V(X) = \frac{1}{\alpha^2} \;. \tag{2.50}$$

Interpretation: radioactive decay, decay rate α; x is waiting time for βth decay event.

Cauchy distribution:

$$F_X(x) = \frac{1}{2} + \frac{1}{\pi} \arctan \frac{x}{b} \;, \qquad f_X(x) = \frac{b}{\pi} \frac{1}{b^2 + x^2} \;. \tag{2.51}$$

Mean value undefined, variance ∞.

2.2 Generating random numbers with a given distribution

2.2.1 Congruential generators

Consider the following recursion:

$$x_{n+1} = 3\,x_n \mod 7 \;. \tag{2.52}$$

Start e.g. with $x_1 = 5$, then x_n follows the sequence $5, 1, 3, 2, 6, 4, 5, \ldots$, i.e. goes through all values (except 0, which is absorbing) before cycling. Divide x_n by 7 to get sequence $0.71, 0.14, 0.43, 0.29, 0.86, 0.57, 0.71, \ldots$.

On the other hand,

$$x_{n+1} = 3\,x_n \mod 7 \;. \tag{2.53}$$

has two shorter cycles: $5, 3, 6, 5, \ldots$ and $2, 4, 1, 2, \ldots$

Linear congruential method:

$$x_{n+1} = a\, x_n + c \quad \mathrm{mod}\ m\ . \tag{2.54}$$

To get realizations of $Y \sim \mathcal{U}(0,1)$, use $y_n = x_n/m$.

Important to find good values of A, c and m — and test the (pseudo) random numbers. E.g. Parker & Miller: $a = 7^5 = 16808$, $c = 0$, $m = 2^{31} - 1 = 2147483647$ (will cycle after at most 2×10^9 iterations).

2.2.2 Other distributions

Transformation method

Methods like the congruential generators yield uniformly distributed random numbers on $(0,1)$. We often need to map these to random numbers with other distributions – so what happens to the probability density function if we transform a random variable $X \mapsto g(X)$?

Let $Y = g(X)$, and assume $g(x)$ is monotonically non-decreasing, then

$$P(X < x) = P[g(X) < g(x)]\ , \tag{2.55}$$

or

$$F_X(x) = F_Y(g(x)) = F_Y(y)\ , \tag{2.56}$$

where $y \equiv g(x)$.

Note that taking the derivative w.r.t. x, we get

$$f_X(x) = f_Y[g(x)]\, g'(x) = f_Y(y) \frac{dy}{dx}\ , \tag{2.57}$$

which can be written as

$$f_X(x)\, dx = f_Y(y)\, dy\ . \tag{2.58}$$

Going back to Eq. (2.56), we now replace X by $Y \sim \mathcal{U}(0,1)$ and Y by X:

$$F_U(u) = F_X[g(u)]\ , \tag{2.59}$$

and since $F_U(u) = u$, we find

$$F_X(x) = u\ , \tag{2.60}$$

or

$$x = g(u) = F_X^{-1}(u)\ , \tag{2.61}$$

provided we can invert the function $F_X(\cdot)$.

We thus get the following method to construct random numbers X with the density function $f_X(x)$:

1. Integrate $f_X(x)$ to get
$$F_X(x) = \int\limits_{-\infty}^{x} f_X(x')\,dx'$$
2. Obtain uniformly distributed random numbers u_i
3. Solve the equation
$$F_X(x_i) = u_i\ . \tag{2.62}$$
for x_i (i.e. invert F_X). The values x_i will sample the desired distribution.

Example: To construct random numbers X that are exponentially distributed,
$$f_X(x) = \alpha\, e^{-\alpha x} \qquad x \geq 0\ , \tag{2.63}$$
we find
$$F_X(x) = 1 - e^{-\alpha x} \qquad x \geq 0\ , \tag{2.64}$$
and thus
$$1 - e^{-\alpha x_i} = u_i\ , \tag{2.65}$$
which leads to
$$x_i = -\frac{1}{\alpha}\ln(1 - u_i)\ . \tag{2.66}$$
If u_i is uniformly distributed over $[0, 1]$, then $1 - u_i$ is as well, so instead of Eq. (2.67), we can use the slightly simpler
$$x_i = -\frac{1}{\alpha}\ln \tilde{u}_i\ , \tag{2.67}$$
where $\tilde{u}_i$ are uniformly distributed random numbers.

Can we use this to construct normally distributed random numbers? Only if we are willing to somehow numerically calculate the inverse error function. But there is a better method, that allows us to construct two normally distributed random variables at once.

Consider $X \sim \mathcal{N}(0, 1)$ and $Y \sim \mathcal{N}(0, 1)$, which are supposed to be independent. The joint probability density is
$$f_{X,Y}(x, y) = f_X(x)\, f_Y(y) = \frac{1}{2\pi}e^{-(x^2+y^2)/2} = \frac{1}{2\pi}e^{-r^2/2}\ , \tag{2.68}$$
where $r = \sqrt{x^2 + y^2}$. For $R \equiv \sqrt{X^2 + Y^2}$, we find the probability distribution function
$$F_R(r) = \int\limits_{r'\leq r} f_{X,Y}(x, y)\, dx'\, dy' = \frac{1}{2\pi}\int\limits_{r'\leq r} e^{-r'^2/2}\, dx'\, dy'\ . \tag{2.69}$$
Switching to polar coordinates, we have $dx\, dy = d\varphi\, r\, dr$ and thus
$$F_R(r) = P(R \leq r) = \frac{1}{2\pi}\int\limits_0^{2\pi} d\varphi' \int\limits_0^{r} dr'\, r'\, e^{-r'^2/2} = 1 - e^{-r^2/2}\ . \tag{2.70}$$

Setting $F_R(r_i) = 1 - F_U(u_i) = 1 - u_i$ as above[2], we get

$$r_i = \sqrt{-2 \ln u_i}\ , \tag{2.71}$$

so we now know how to construct random numbers R. To get X and Y, we have to multiply R with a random phase factor,

$$X = R \cos \Phi\ , \qquad Y = R \sin \Phi\ , \tag{2.72}$$

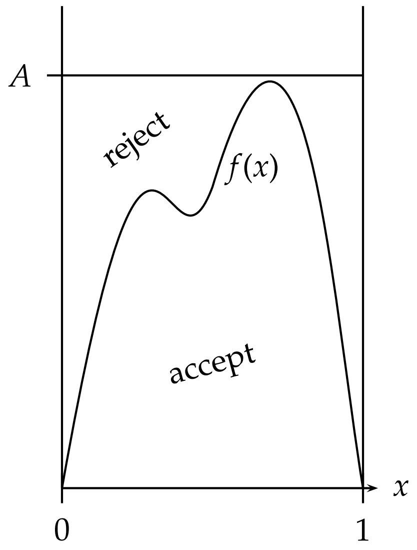
where $\Phi \sim \mathcal{U}(0, 2\pi)$. One can verify that the resulting random numbers X, Y are statistically independent and are normally distributed.

Thus, to construct pairs of normally distributed random numbers, we have the following procedure:

1. Obtain a pair (u_i, v_i) of uniformly distributed random numbers;
2. calculate
$$r_i = \sqrt{-2 \ln u_i}\ , \qquad \varphi_i = 2\pi v_i\ ; \tag{2.73}$$
3. the normally distributed variables are
$$x_i = r_i \cos \varphi_i\ , \qquad y_i = r_i \sin \varphi_i\ . \tag{2.74}$$

Acceptance-rejection method

Consider a distribution with a density $f(x)$ that lives only on the interval $[0, 1]$:



Pick a point (x_i, y_i) in that rectangle at random:

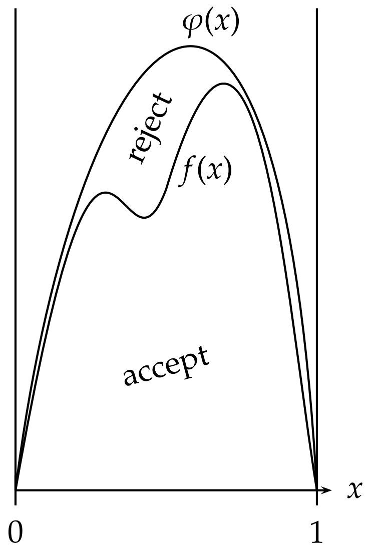
$$X \sim \mathcal{U}(0, 1)\ , \qquad Y \sim \mathcal{U}(0, A)\ . \tag{2.75}$$

[2]We could use $F_R(r_i) = F_U(u_i) = u_i$, but again $1 - U$ has the same characteristics as U

If we now discard all pairs (x_i, y_i) that are above the curve $y = f(x)$, then the probability of a certain x value will be equal to $f(x)$. Formally, this becomes

$$P[x<X<x+dx \,|\, Y<f(X)] = f(x)\,dx \,. \tag{2.76}$$

If we use the method in this form, we may have to discard a lot of points, but the method can be somewhat generalized: consider a distribution with density $f(x)$ and introduce a *comparison function* $\varphi(x)$ with $f(x) \leq \varphi(x)$ everywhere (in the motivation section above, $\varphi(x)$ was just a constant function).



If we can choose random points (x_i, y_i) that are below $\varphi(x)$, but have constant density in x and y, we can discard all points with $f(x_i) < y_i < \varphi(x_i)$ and find that the x coordinates of the remaining points again have probability density function $f(x)$. So how can we distribute the points evenly below the curve $\varphi(x)$?

If we use $X \sim \mathcal{U}(0, 1)$ and for each x_i choose a y_i distributed according to $Y \sim \mathcal{U}[0, \varphi(x_i)]$, then we will have too many points where $\varphi(x)$ is small, and too few where it is large. We must compensate for this by having a density in X that is proportional to $\varphi(x)$. This is our original problem, but now for the comparison function $\varphi(x)$, which we can control. If we choose $\varphi(x)$ such that it can be analytically integrated and the integral can be inverted, then we can use the transformation method to get random numbers x_i distributed according to the probability density function

$$f_X(x) = \frac{\varphi(x)}{\int\limits_{-\infty}^{\infty} \varphi(x')\,dx'} \,. \tag{2.77}$$

This leads us to the following recipe.

1. Choose a comparison function $\varphi(x)$ that satisfies

$$\varphi(x) \geq f(x) \tag{2.78}$$

everywhere, but such that it can be analytically integrated:

$$\widetilde{\Phi}(x) = \frac{\int\limits_{-\infty}^{x} \varphi(x')\,dx'}{\int\limits_{-\infty}^{\infty} \varphi(x')\,dx'} \tag{2.79}$$

and $\widetilde{\Phi}$ can be analytically inverted;

2. obtain a pair (u_i, v_i) of uniformly distributed random numbers;
3. calculate
$$x_i = \widetilde{\Phi}^{-1}(u_i)\ , \qquad \text{and} \qquad y_i = v_i\,\varphi(x_i)\ . \tag{2.80}$$
4. accept x_i as next random number if
$$y_i < f(x_i)\ , \tag{2.81}$$
otherwise reject it and start again.

Note1: The efficiency of our acceptance-rejection method (i.e. the fraction of accepted points) is

$$\eta = \frac{\int\limits_{-\infty}^{\infty} f(x)\,dx}{\int\limits_{-\infty}^{\infty} \varphi(x)\,dx} = \frac{1}{\int\limits_{-\infty}^{\infty} \varphi(x)\,dx}\ . \tag{2.82}$$

It depends on how close the comparison function is to the function $f(x)$. The constant function which we used in the motivation above is often quite inefficient.

Note2: For many density functions, the Cauchy profile

$$\varphi(x) = \frac{A\,b}{\pi}\frac{1}{b^2 + (x - x_0)^2} \tag{2.83}$$

is a good choice as comparison function. In this case, one has

$$\widetilde{\Phi}(x) = \frac{1}{2} + \frac{1}{\pi}\arctan\frac{x - x_0}{b}\ , \tag{2.84}$$

and

$$\widetilde{\Phi}^{-1}(u) = x_0 + b\tan[\pi(u - 1/2)]\ . \tag{2.85}$$

2.2.3 Superposition method

If the density $f_X(x)$ is the linear superposition of (a finite number of or infinitely many) density functions for which we can generate random numbers (e.g. because we know the distribution functions and can invert them),

$$f_X(x) = \sum_k c_k f_{X_k}(x) , \qquad \text{where} \sum_k c_k = 1 , \tag{2.86}$$

then we can generate random numbers X as follows.

1. Obtain two uniformly distributed random numbers u, v.
2. Use u to draw a random number k with probability $P(K{=}k) = c_k$ (i.e. treat c_k as probabilities of a discrete random variable, see § 2.2.4 below)
3. Use v to generate a random number Y according to the distribution for X_k.

To see that this indeed yields random numbers with the desired distribution, we note that

$$\sum_{k=0}^{\infty} c_k \, P(X{\le}x \,|\, K{=}k) \;=\; \sum_{k=0}^{\infty} P(K{=}k)\, P(X{\le}x \,|\, K{=}k) \;=\; P(X{\le}x) \tag{2.87}$$

because of the total probability theorem Eq. (2.35).

Example: To generate a random number x for the distribution with

$$f_X(x) = \frac{1}{3} + \frac{2}{3} 2\,x , \qquad 0 \le x \le 1 , \tag{2.88}$$

we

1. draw u and v,
2. use u to choose which distribution to choose: if $u \le 1/3$, we choose a random number with uniform distribution, $f_X(x) = 1$, else one with linear distribution $f_X(x) = 2x$,
3. and use v to draw the final number. To draw a uniformly distributed number, we just use $x = v$. For the linear distribution, we have $F_X(x) = x^2$, thus $F_X^{-1}(y) = \sqrt{y}$, and thus $x = \sqrt{v}$.

Written even more as a recipe:

1. Draw $u, v \sim \mathcal{U}(0, 1)$.
2. $$\begin{cases} x = v & \text{if } u \le \dfrac{1}{3} \\ x = \sqrt{v} & \text{if } u > \dfrac{1}{3} \end{cases} \tag{2.89}$$

This method is very powerful, as it works for an arbitrary linear superposition.

2.2.4 Discrete distributions

If we have a discrete distribution with probabilities p_k and the distribution function

$$F_X(k) = \sum_{m=0}^{k} p_k \, , \tag{2.90}$$

the following yields the desired random numbers: Obtain a uniformly distributed number u, then

$$\text{if } u \leq F_X(0) \qquad \text{choose } x = 0 \, , \tag{2.91}$$
$$\text{if } F_X(0) < u \leq F_X(1) \qquad \text{choose } x = 1 \, , \tag{2.92}$$
$$\text{if } F_X(1) < u \leq F_X(2) \qquad \text{choose } x = 2 \, , \tag{2.93}$$
$$\text{if } F_X(2) < u \leq F_X(3) \qquad \text{choose } x = 3 \, , \tag{2.94}$$
$$\vdots$$

This works because the probability that u is in a certain interval is equal to the length of this interval, and the length of the kth interval is $F_X(k) - F_X(k-1) = p_k$.

Example: For the binomial distribution $\mathcal{B}(N, p)$ with $N = 5$, p=1/3, we have

$$p_0 = \frac{32}{243} \, , p_1 = \frac{80}{243} \, , p_2 = \frac{80}{243} \, , p_3 = \frac{40}{243} \, , p_4 = \frac{10}{243} \, , p_5 = \frac{1}{243} \, , \tag{2.95}$$

and, calculating the partial sums,

$$F_X(0) = \frac{32}{243} \, , F_X(1) = \frac{112}{243} \, , F_X(2) = \frac{192}{243} \, , F_X(3) = \frac{231}{243} \, , F_X(4) = \frac{242}{243} \, , F_X(5) = \frac{243}{243} \, . \tag{2.96}$$

Thus,

$$\text{if } u \leq \frac{32}{243} \qquad \text{choose } x = 0 \, , \tag{2.97}$$
$$\text{if } \frac{32}{243} < u \leq \frac{112}{243} \qquad \text{choose } x = 1 \, , \tag{2.98}$$
$$\text{if } \frac{112}{243} < u \leq \frac{192}{243} \qquad \text{choose } x = 2 \, , \tag{2.99}$$
$$\text{if } \frac{192}{243} < u \leq \frac{231}{243} \qquad \text{choose } x = 3 \, , \tag{2.100}$$
$$\text{if } \frac{231}{243} < u \leq \frac{242}{243} \qquad \text{choose } x = 4 \, , \tag{2.101}$$
$$\text{if } \frac{242}{243} < u \leq \frac{243}{243} \qquad \text{choose } x = 5 \, . \tag{2.102}$$

2.3 The central limit theorem

The sum $\sum X_i$ of N independent random variables has the probability density function $\bigotimes_i f_{X_i}(x)$, i.e. the convolution of the individual PDFs. As convolutions are best dealt with in Fourier space, let us introduce the Fourier transform of the probability density function,

$$\chi_X(t) \equiv \int_{-\infty}^{\infty} e^{itx} f_X(x)\, dx \,, \tag{2.103}$$

which is called the *characteristic function* of X. and can be written as

$$\chi(t) = Ee^{itX} \,. \tag{2.104}$$

The latter form works also for discrete distributions.

If χ is sufficiently smooth (i.e. if f_X decays sufficiently fast), we can expand $\ln \chi$ in a Taylor series:

$$\ln \chi(t) = \sum_{n=0}^{\infty} \kappa_n \frac{(it)^n}{n!} \,. \tag{2.105}$$

Here κ_n are called the *cumulants* of the distribution, and are related to the central moments

$$\begin{aligned}
\mu &\equiv EX \,, && (2.106)\\
\mu_2 &\equiv E(X-\mu)^2 \,, && (2.107)\\
\mu_3 &\equiv E(X-\mu)^3 \,, && (2.108)\\
\mu_4 &\equiv E(X-\mu)^4 \,, && (2.109)\\
\ldots
\end{aligned}$$

via

$$\begin{aligned}
\kappa_0 &= 0 \,, && (2.110)\\
\kappa_1 &= \mu \,, && (2.111)\\
\kappa_2 &= \mu_2 \,, && (2.112)\\
\kappa_3 &= \mu_3 \,, && (2.113)\\
\kappa_4 &= \mu_4 - 3\mu_2^2 \,, && (2.114)\\
\ldots
\end{aligned}$$

For a normal distribution $\mathcal{N}(\mu,\sigma)$, we get

$$\begin{aligned}
\chi(t) &= \frac{1}{\sqrt{2\pi\sigma^2}}\int\limits_{-\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\, e^{itx}\, dx && (2.115)\\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\int\limits_{-\infty}^{\infty} e^{-\frac{x^2-2\mu x+\mu^2-2\sigma^2 ixt}{2\sigma^2}}\, dx && (2.116)\\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\int\limits_{-\infty}^{\infty} e^{-\frac{[x-(\mu+\sigma^2 it)]^2 x-2\mu\sigma^2 it+\sigma^2 t}{2\sigma^2}}\, dx && (2.117)\\
&= e^{it\mu-\frac{\sigma^2 t}{2}}\,, && (2.118)
\end{aligned}$$

or

$$\ln\chi(t) = \mu it - \frac{\sigma^2 t}{2}\,. \qquad (2.119)$$

Hence,

$$\begin{aligned}
\kappa_0 &= 0\,, && (2.120)\\
\kappa_1 &= \mu\,, && (2.121)\\
\kappa_2 &= \sigma^2\,, && (2.122)\\
\kappa_3 &= 0\,, && (2.123)\\
\kappa_4 &= 0\,, && (2.124)\\
\dots
\end{aligned}$$

If we add two random variables, $Z = X + Y$, the resulting PDF is

$$f_Z(z) = f_X \otimes f_Y = \int\limits_{-\infty}^{\infty} f_X(x)\, f_Y(z-x)\, dx\,. \qquad (2.125)$$

The resulting characteristic function is simply

$$\chi_Z(t) = \chi_X(t)\,\chi_Y(t)\,. \qquad (2.126)$$

If we divide a random variable X by a number n,

$$Y = \frac{X}{n}\,, \qquad (2.127)$$

the PDF becomes [3]

$$f_Y(y) = n f_X(ny)\,, \qquad (2.129)$$

[3] This follows from

$$f_X(x)\, dx = f_Y(y)\, dy\,. \qquad (2.128)$$

and the resulting characteristic function is

$$\chi_Y(t) = \int e^{ity}\, n\, f_X(ny)\, dy = \int e^{i(t/n)x} f_X(x)\, dx = \chi_X(t/n)\ . \tag{2.130}$$

So what is the characteristic function of the arithmetic mean of n random variables

$$Y = \frac{1}{n}\sum_{k=1}^{n} X_k\ ? \tag{2.131}$$

According to Eqs. (2.126) and (2.130), we need to take the product and scale the argument:

$$\chi_Y(t) = \prod_{k=1}^{n} \chi_k(t/n)\ . \tag{2.132}$$

We can write this as

$$\ln \chi_Y(t) = \sum_{k=1}^{n} \ln \chi_k(t/n)\ , \tag{2.133}$$

or, for the cumulants,

$$\begin{aligned}
\kappa_0^{(Y)} &= 0\ , && (2.134)\\
\kappa_1^{(Y)} &= \frac{\sum \kappa_1^{(i)}}{n}\ , && (2.135)\\
\kappa_2^{(Y)} &= \frac{\sum \kappa_2^{(i)}}{n^2}\ , && (2.136)\\
\kappa_3^{(Y)} &= \frac{\sum \kappa_3^{(i)}}{n^3}\ , && (2.137)\\
&\dots
\end{aligned}$$

In terms of the moments, this means that

$$\begin{aligned}
\mu^{(Y)} &= \frac{\sum \mu_1^{(i)}}{n} = \langle \mu \rangle\ , && (2.138)\\
\mu_2^{(Y)} &= \frac{\sum \mu_2^{(i)}}{n^2} = \frac{\langle \mu_2 \rangle}{n}\ , && (2.139)\\
\mu_3^{(Y)} &= \frac{\sum \mu_3^{(i)}}{n^3} = \frac{\langle \mu_3 \rangle}{n^2}\ , && (2.140)\\
&\dots && (2.141)
\end{aligned}$$

If n is large, we can neglect the moments higher than 2 and find that we asymptotically get a normal distribution with

$$\mu = \langle \mu \rangle\ , \qquad \sigma = \sqrt{\frac{\langle \mu_2 \rangle}{n}}\ . \tag{2.142}$$

At the same time, the *skewness* $\gamma_1 \equiv \mu_3/\sigma^3$ and higher cumulants that the normal distribution does not have tend to zero. This statement is called *central limit theorem*.

Example 1: Averaging exponentially distributed random variables. For the exponential distribution $X \sim \mathcal{E}(\alpha)$, we have

$$\chi_X(t) = \int\limits_{x=0}^{\infty} \alpha e^{-\alpha t} e^{itx}\, dx = \frac{1}{1 - \frac{it}{\alpha}} \; . \tag{2.143}$$

Thus,

$$\ln \chi_X(t) = -\ln\left(1 - \frac{it}{\alpha}\right) = \frac{it}{\alpha} + \frac{(it)^2}{2\alpha^2} + \ldots \tag{2.144}$$

and

$$\kappa_1 = \frac{1}{\alpha}\,, \qquad \kappa_2 = \frac{1}{\alpha^2}\,, \qquad \kappa_3 = \frac{2}{\alpha^3}\,, \qquad \ldots \tag{2.145}$$

Averaging n independent exponentially distributed random variables, we get

$$\ln \chi_Y(t) = -n \ln\left(1 - \frac{it}{\alpha n}\right) = \frac{it}{\alpha} + \frac{(it)^2}{2\alpha^2 n} + \frac{(it)^3}{3\alpha^2 n^2} + \ldots \tag{2.146}$$

and thus

$$\mu = \frac{1}{\alpha}\,, \qquad \sigma = \frac{1}{\alpha\sqrt{n}}\,, \qquad \mu_3 = \frac{2}{\alpha^3 n^2}\,, \qquad \ldots \tag{2.147}$$

Thus for large n, the distribution of Y approaches a normal distribution:

$$X \sim \mathcal{N}\left(\frac{1}{\alpha}, \frac{1}{\alpha\sqrt{n}}\right) \text{ for } n \to \infty \tag{2.148}$$

The skewness goes like

$$\gamma_1 = \frac{2}{\sqrt{n}} \to 0 \text{ for } n \to \infty \;\; , \tag{2.149}$$

implying that the distribution gets more symmetric with increasing n.

Example 2: Averaging Cauchy-distributed random variables. For the Cauchy distribution, we have

$$\chi_X(t) = \int\limits_{x=0}^{\infty} \frac{b}{\pi} \frac{1}{b^2 + x^2} e^{itx}\, dx = e^{-b|t|} \; . \tag{2.150}$$

Thus,

$$\ln \chi_X(t) = -b|t| \; . \tag{2.151}$$

For the average Y of n Cauchy-distributed variables, we find

$$\ln \chi_Y(t) = -nb\left|\frac{t}{n}\right| = -b|t| \tag{2.152}$$

— this is *unchanged*. Conclusion: averaging Cauchy-distributed random variables is useless, as it does not change (an in particular not narrow) the distribution.

Reason: None of the cumulants/moments κ_i or μ_i exists, because $\ln \chi$ has no continuous derivatives (which in turn reflects the slow decay of $f_X(x)$ for $|x| \to \infty$). Thus, the central limit theorem does not hold for this distribution.

2.4 Monte Carlo integration

Monte Carlo integration is the approximate calculation of (mostly multi-dimensional) integrals by averaging over a large sample of random numbers.

Example: Let $x_i, i = 1, \dots N$ be exponentially distributed random numbers. Then the expectation value $E \sin X$ is given by

$$E \sin X = \int\limits_{x=0}^{\infty} \sin x \, e^{-x} \, dx \, , \tag{2.153}$$

which can be approximated by the average

$$E \sin X \approx \langle \sin x_i \rangle \equiv \frac{\sum_{i=1}^{N} \sin x_i}{N} \, . \tag{2.154}$$

The standard deviation of the error will be

$$\delta = \frac{\sigma(x_i)}{\sqrt{N-1}} = \sqrt{\frac{\left\langle \sin x_i^2 \right\rangle - \langle \sin x_i \rangle^2}{N-1}} \, , \tag{2.155}$$

thus we can write

$$\int\limits_{x=0}^{\infty} \sin x \, e^{-x} \, dx \approx \langle \sin x_i \rangle \pm \sqrt{\frac{\left\langle \sin x_i^2 \right\rangle - \langle \sin x_i \rangle^2}{N-1}} \, . \tag{2.156}$$

Comparing with the exact value $I = 1/2$, we find values and errors as in Table 2.1.

Table 2.1: Example of Monte Carlo integration of $I \equiv \int_{x=0}^{\infty} \sin x \, e^{-x} \, dx$. The errors are for the values found in this experiment and will be different in a different realization.

N	Monte Carlo approximant I_N	error
10	0.46591	-0.017
100	0.52414	0.012
1 000	0.50023	0.00012
10 000	0.50525	0.0026
100 000	0.49846	-0.00077
1 000 000	0.49937	-0.00031
10 000 000	0.49976	-0.00012

As we can see, the error decreases quite slowly, and we know that it goes like $\delta \sim 1/N^{1/2}$. Using the trapezoid rule, we would have an error that scales like $1/N^2$, and other rules like high-order Newton–Cotes, Romberg or Gauss–Laguerre would give yet much faster convergence.

However, if we have a 10-dimensional integral, then the trapezoid rule (applied to each of the 10 directions) will have an error that scales like $1/N_x^2 = N^{1/5}$, where N is the total number of points. The error of the Monte Carlo method, on the other hand, will still go like $1/N^{1/2}$ and thus decline considerably faster. The cost will be comparable for a 4-dimensional integral. Even for higher-order methods there will be a dimensionality d starting from which Monte Carlo integration is cheaper.

Another, often more important reason for using Monte Carlo integration for multi-dimensional integrals is the fact that it is easily copes with integrals over complicated domains, while higher-order methods are essentially restricted to Cartesian products. For example to calculate the volume of the intersection of a cone with a sphere,

$$V \equiv \int\limits_{\substack{x^2+y^2<z^2 \\ (x-1)^2+y^2+z^2<4}} dV \tag{2.157}$$

we can uniformly sample random points in a box that encloses the volume in question ($[-1,2]\times[-2,2]\times[-2,2]$ will do), and then count the points inside the volume. The fraction of positive counts will be approximately equal to the ratio of V to the volume of the whole box. This leads us to the following code:

Monte Carlo 1

```
real, dimension(3) :: a
real, dimension(2) :: xr=(/-1,2/), yr=(/-2,2/), zr=(/-2,2/)
real               :: Lx=xr(2)-xr(1), Ly=yr(2)-yr(1), Lz=zr(2)-zr(1)
integer            :: count
!
count = 0
do i=1,N
    call random_number(a)     ! [x,y,z]
    x = xr(1)+a(1)*Lx; y = yr(1)+a(2)*Ly; z = zr(1)+a(3)*Lz
    if ((x**2+yy**2 < z**2) .and. ((x-1)**2+y**2+z**2 < 4)) then
        count = count+1
    endif
enddo
print*, 'Volume ~ ', count*Lx*Ly*Lz/N
```

Calculating the integral

$$V \equiv \int\limits_{x^2+y^2<z^2,\ (x-1)^2+y^2+z^2<4} f(\mathbf{x})\, dV \tag{2.158}$$

is almost as simple:

Monte Carlo 2

```
real, dimension(3) :: a
real, dimension(2) :: xr=(/-1,2/), yr=(/-2,2/), zr=(/-2,2/)
real               :: Lx=xr(2)-xr(1), Ly=yr(2)-yr(1), Lz=zr(2)-zr(1)
real                :: sum
!
sum = 0.
do i=1,N
    call random_number(a)     ! [x,y,z]
    x = xr(1)+a(1)*Lx; y = yr(1)+a(2)*Ly; z = zr(1)+a(3)*Lz
    if ((x**2+yy**2 < z**2) .and. ((x-1)**2+y**2+z**2 < 4)) then
        sum = sum + f(x,y,z)
    endif
enddo
print*, 'Integral ~ ', sum*Lx*Ly*Lz/N
```

Basic theorem of Monte Carlo integration:

$$\int g(\mathbf{x})f(\mathbf{x})\,dx^d \approx \langle g(\mathbf{x}_i)\rangle \pm \sqrt{\frac{\langle g(\mathbf{x}_i)^2\rangle - \langle g(\mathbf{x}_i)\rangle^2}{N-1}}\,, \tag{2.159}$$

where x_i are d-dimensional random vectors, distributed according to the PDE $f_{\mathbf{X}}(\mathbf{x}) = f(\mathbf{x})$.

Note: One can attempt to minimize the variance of $g(x_i)$ by absorbing some of g's variability into f, thus mapping

$$f \mapsto \widetilde{f}\,, \qquad g \mapsto \widetilde{g} \tag{2.160}$$

with $\widetilde{g}\widetilde{f} = gf$. This method is called *importance sampling*, because it basically works by sampling more points where the integrand gf is larger.

In the extreme case where $\widetilde{g} = 1$, we would get $\langle g^2\rangle - \langle g\rangle^2 = 0$, and thus have no error at all. However, the resulting integral

$$\int_{\mathcal{V}} \widetilde{f}(\mathbf{x})\,dx^d = 1 \tag{2.161}$$

does not tell us anything new, and in fact we have now replaced the problem of calculating the integral $\int g(\mathbf{x})f(\mathbf{x})\,dx^d$ by that of generating random numbers x_i that are distributed according to $\widetilde{f}(\mathbf{x}) = g(\mathbf{x})f(\mathbf{x})$ over $\mathcal{V}$, which is a more difficult problem.

Nevertheless, a transformation of type (2.160) can sometimes dramatically reduce the statistical error.

2.4.1 Error estimates

While it is sometimes possible to directly estimate the statistical error of a Monte Carlo result, this is by no means the rule. It is however quite simple to estimate the error based on a number N of independent realizations of the Monte Carlo process. Since Monte Carlo

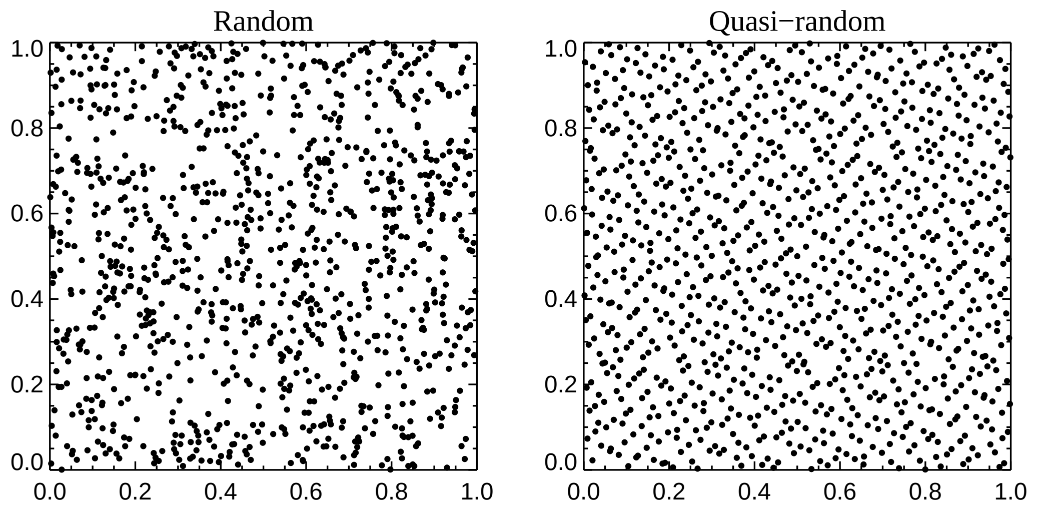


Figure 2.1: Random numbers vs. "quasi-random" numbers. The coordinates (x_i, y_i) of the points on the left are independent, uniformly distributed random numbers. The points on the right are "quasi-random" points (which are in fact completely deterministic), generated by the 'sobseq' routine from [NR90]. In both cases, 1024 points are shown.

simulations require a large number of realizations anyway, this simply means that we group our realizations in N sets.

Within each set, we determine the average $\gamma_i \equiv \langle g \rangle_i, i{=}1, \ldots, N$ of the quantity we are calculating, and calculate the average and standard deviation

$$\overline{\gamma} = \frac{\sum\limits_{i=1}^{N} \gamma_i}{N}, \qquad \sigma^2 = \frac{\sum\limits_{i=1}^{N} (\gamma_i - \overline{\gamma})^2}{N{-}1}. \tag{2.162}$$

The Monte Carlo estimate of $\langle g \rangle$ is then

$$\langle g \rangle = \overline{\gamma} \pm \frac{\sigma}{\sqrt{N}} \tag{2.163}$$

In practise, a value of 10..20 for N gives reasonably good results.

2.4.2 Quasi-random numbers

The fact that the error of a Monte-Carlo integral goes like $1/\sqrt{N}$ follows directly from Eq. (2.23). Using independent random numbers, we cannot obtain a better scaling.

A similar method called "Quasi-Monte Carlo method" utilizes a deterministic sequence of points. These points form "quasi-random" sequences and 'avoid each other' and thus sample the hypercube more efficiently, as can be seen in Fig. **??**. The scaling of the error for Quasi-Monte Carlo integration is $\delta \sim \ln^d N/N$ (and thus close to $1/N$) for the integration of smooth functions over the full d-dimensional hypercube.

If we integrate over a subvolume of the hypercube, the error is be dominated by the points near the (hyper)surface of that subvolume, which 'by chance' happen to be inside (and thus counted) or outside (and thus discarded). The thickness of that layer around the hypersurface is $\sim N^{1/d}$ (this is the typical separation of points in one direction, here in the direction normal to the hypersurface), and this is equal to the fraction of points that are close to the boundary (only the direction normal to the hypersurface is selective, the other directions are irrelevant for whether a point is a boundary point).

The number of boundary points is thus $\sim N^{1/d} N = N^{(d-1)}/d$. According to the statistical 'square root law', this leads to statistical fluctuations of order $\sim N^{(d-1)/2d}$, which contribute to the integral (after dividing by N) as $1/N^{(d+1)/2d}$. This exponent is shown in Table 2.2 for different dimensionalities d. For $d = 3$, it is still markedly different from the scaling exponent 1/2 for the Monte Carlo method, thus Quasi-Monte Carlo will be worth while in 3 dimensions.

Table 2.2: Scaling exponent over dimensionality d for Quasi-Monte Carlo integration over a volume that is not a Cartesian product.

d	$-\frac{d+1}{2d}$	num. value
1	0	0.000
2	−1/4	−0.250
3	−1/3	−0.333
4	−3/8	−0.375
5	−2/5	−0.400
6	−5/12	−0.417
8	−7/16	−0.438
10	−9/20	−0.450
15	−7/15	−0.467
20	−19/40	−0.475

2.5 The Metropolis et al algorithm

Metropolis et al have come up with an algorithm for Monte Carlo methods that is very popular ever since. It provides a method to generate random numbers by a random process ('random walk') that are asymptotically distributed according to any given distribution. *Asymptotic* here means that the process needs some time to equilibrate before the numbers have the desired distribution. Another constraint to keep in mind is the fact that the random numbers are typically strongly correlated. For many applications this is not important, because one needs to average over many correlation 'times' anyway.

2.5.1 Random processes

A discrete *random process* is a sequence of random variables $X_1, X_2, \ldots$.

For a *Markovian random process* ('Markov chain') the distribution of X_{k+1} is completely determined by X_k (Markov chains are 'memory-less'). The dependence on the previous state is characterized by the *transition probability*

$$P(x \to x') = P(X_{n+1}{=}x' \,|X_n{=}x) \tag{2.164}$$

(for a continuous distribution[4]), or

$$P(i \to k) = P(X_{n+1}{=}k \,|X_n{=}i) \tag{2.165}$$

(for a discrete one).

Under certain, not very restrictive conditions, a Markov process leads to a stationary distribution, which is not modified by $P(i \to k)$. A straight-forward way of enforcing that the stationary distribution is $f(i)$ is to require *detailed balance*:

$$f(i)\, P(i \to k) = f(k)\, P(k \to i)\,, \tag{2.166}$$

which can also be written as

$$\frac{P(i \to k)}{P(k \to i)} = \frac{f(k)}{f(i)}\,. \tag{2.167}$$

Since both transition probabilities have to be ≤ 1, we can choose

$$P(i \to k) = \begin{cases} 1\,, & f(k) \geq f(i)\,, \\ \dfrac{f(k)}{f(i)}\,, & f(k) < f(i)\,. \end{cases} \tag{2.168}$$

"Example": In thermodynamics, we are often interested in partition sums of the form

$$Ef = \frac{\sum\limits_i f(E_i)\, e^{-\beta E_i}}{\sum\limits_i e^{-\beta E_i}} \tag{2.169}$$

(or corresponding integrals), where $\beta \equiv 1/(k_{\mathrm{B}} T)$.

To obtain a series of energies E_i with the Boltzmann distribution $p_i \sim e^{-\beta E_i}$, we can use Eq. (2.168) and get

$$P(E_i \to E_k) = \begin{cases} 1\,, & E_k \leq E_i\,, \\ e^{-\beta(E_k - E_i)}\,, & E_k > E_i\,. \end{cases} \tag{2.170}$$

To evaluate the average (2.169), we can thus proceed as follows:

1. Start with an arbitrary state i.

[4] This is a very imprecise notation, since for a continuous distribution $P(X = x_0)$ is always zero (X has a finite probability to be in an interval, but if the interval length tends to zero, the probability does, too). But it should be clear enough how to make sense of it when needed.

2. Consider switching to another state k which would lead to an energy difference $\Delta E \equiv E_k - E_i$:

   $$\begin{cases} \text{if } \Delta E \le 0 & \text{switch unconditionally;} \\ \text{if } \Delta E > 0 & \text{draw random number } u \text{, then} \begin{cases} \text{if } u < e^{-\beta\Delta E} & \text{switch;} \\ \text{if } u \ge e^{-\beta\Delta E} & \text{do not switch;} \end{cases} \end{cases} \tag{2.171}$$

   This decision procedure can be compactly expressed as:

   $$\text{Switch if } u < e^{-\beta\Delta E} \,, \tag{2.172}$$

   as u will never exceed 1.
3. Repeat step 2 *many* times.
4. Evaluate Ef as $\langle f_i \rangle$ over the states i,
   (a) generously discarding the first states it took for the process to reach equilibrium;
   (b) making sure that you do count 'new' states even if they resulted from not switching — if you do not switch from a state, its probability is accordingly higher.

2.5.2 Ising model of a ferromagnet

The Ising model considers an array of spins $s_i = \pm 1$ that only interact with their nearest neighbours. The total energy is given by

$$E = -\varepsilon \sum_{\text{neighbours } (i,k)} s_i s_k \,, \tag{2.173}$$

where ε is a positive energy and summation is over all pairs of nearest neighbours (such that each combination of neighbours is counted only once). The total energy can be minimized by aligning all spins parallel with each other, which results in an energy $-\varepsilon$ per spin.

The one-dimensional Ising model shows smooth temperature dependence (the magnetization decreases, and thermal energy increases, continuously with increasing temperature).

For the one-dimensional Ising model, we find

$$E = -\varepsilon \sum_i s_i s_{i+1} \,. \tag{2.174}$$

In two or more dimensions, however, the Ising model exhibits a *phase transition*: There is a critical temperature where magnetization goes to zero and the heat capacity is infinitely large (because of latent heat).

One way of doing a Monte Carlo simulation to get the thermodynamical properties of the Ising model is to choose a spin position at random and then decide whether to flip it or not. The energy difference when flipping the ith spin from s_i to $-s_i$ is

$$\Delta E = 2\,\varepsilon\,s_i \sum_{k(i)} s_k\ , \tag{2.175}$$

where the sum is over all spins $k(i)$ that are nearest neighbours to i. In one dimension, this becomes

$$\Delta E = 2\,\varepsilon\,s_i(s_{i-1} + s_{i+1})\ . \tag{2.176}$$

The heat capacity can be calculated from

$$c_v \propto \frac{1}{N}\left(\frac{\varepsilon}{k_{\mathrm{B}}T}\right)^2 \left(\left\langle E^2\right\rangle - \langle E\rangle^2\right) \tag{2.177}$$

and the magnetic susceptibility from

$$\chi \propto \frac{1}{N}\frac{\varepsilon}{k_{\mathrm{B}}T}\left(\left\langle M^2\right\rangle - \langle M\rangle^2\right)\ , \tag{2.178}$$

where $M \propto \sum S_i$ is the total magnetization.

Note 1: The right hand sides of Eqs. (2.177) are essentially the variances of E and M. As c_v and χ diverge at a phase transition, total energy and magnetization will be subject to huge fluctuations close to the critical temperature.

Note 2: In a finite system ($N < \infty$), the phase transition will be smoothed out. It only occurs in the limit $N \to \infty$.

Note 3: The normalization factor $1/Z \equiv 1/\sum e^{-\beta E_i}$ never needs to be evaluated.

2.5.3 Quantum Monte Carlo integration

Monte Carlo methods have various applications in quantum mechanics. One of them is the *variational quantum Monte Carlo method* for finding the ground state of many-particle problems.

For an n-particle problem, the energy of a state $\psi(\mathbf{x}) = \langle \mathbf{x}|\psi\rangle$ is

$$\begin{aligned} E &= \frac{\int \psi^*(\mathbf{x})\hat{H}\psi(\mathbf{x})\,dx^{3n}}{\int \psi^*(\mathbf{x})\psi(\mathbf{x})\,dx^{3n}} && (2.179)\\ &= \int \frac{\psi^*(\mathbf{x})\psi(\mathbf{x})}{\int \psi^*(\mathbf{x})\psi(\mathbf{x})\,dx^{3n}}\;\frac{1}{\psi(\mathbf{x})}\hat{H}\psi(\mathbf{x})\;dx^{3n} && (2.180)\\ &= \int f(\mathbf{x})\;\frac{1}{\psi(\mathbf{x})}\hat{H}\psi(\mathbf{x})\;dx^{3n}\ , && (2.181) \end{aligned}$$

where $\mathbf{x}$ is a $3n$-dimensional position vector (3 dimensions for each particle) and $\hat{H}$ is the Hamiltonian.

$$f(\mathbf{x}) \equiv \frac{\psi^*(\mathbf{x})\psi(\mathbf{x})}{\int \psi^*(\mathbf{x})\psi(\mathbf{x})\, dx^{3n}} \tag{2.182}$$

is a (joint) probability density.

Eq. (2.181) has the form (2.159) of a multidimensional Monte Carlo integral and can thus be evaluated using Monte Carlo integration.

To generate the random positions $\mathbf{X}$, we can use the Metropolis et al algorithm, starting with random coordinates $\mathbf{X}_0$ and then accepting or rejecting a change $\Delta\mathbf{X}$ in all six coordinates (alternatively, one could do one coordinate at a time).

After discarding a certain number of initial values, we calculate the average of

$$\tilde{E}_i = \frac{1}{\psi(\mathbf{x}_i)} \hat{H}\psi(\mathbf{x}_i) , \tag{2.183}$$

which will converge to the value of the integral (2.179).

Again, the normalization factor $1/\int |\psi(\mathbf{x})|^2\, dx^{3n}$ is never needed, so we can work with unnormalized trial wave functions $\psi(r)$ here.

Note 1: As a rule of thumb, an acceptance ratio of ≈ 0.5 is optimal (some say 0.2 has advantages); we can tune the amplitude of $\Delta\mathbf{X}$ to achieve this. [We cannot do that for the Ising model where phase space is discrete.]

Note 2: As always with the Metropolis et al algorithm, the random positions $\mathbf{X}$ are strongly correlated. This does not affect the precision of the integral, provided we integrate over many correlation times. However, simple estimates of the statistical error will be far too optimistic, since effectively the number of independent values will not be equal to N (the number of Metropolis steps), but N/N_{corr}, where N_{corr} is the correlation 'time'.

The method described in Sec. 2.4.1 will however work, provided the size of the sets is considerably larger than the correlation 'time'; this will always be the case if you have good statistics and the number of sets is reasonably small.

Example: Consider the one-dimensional potential well

$$U(x) = \begin{cases} \infty , & x < 0 , \\ \alpha x , & x \geq 0 . \end{cases} \tag{2.184}$$

The stationary Schrödinger equation for the wave function $\psi(x)$ is

$$-\frac{\hbar^2}{2m}\psi'' + \alpha x \psi = E\psi , \qquad x \geq 0 , \tag{2.185}$$

with the boundary conditions $\psi(0) = \psi(\infty) = 0$. Here $\hbar = h/2\pi$ is Planck's constant, m the particle mass, and E is the energy of the eigenstate $\psi(x)$.

Using appropriate units, this can be written as

$$\hat{H}\psi \equiv -\psi'' + x\psi = E\psi , \qquad x \geq 0 . \tag{2.186}$$

For an arbitrary 'trial' wave function $\Phi(x)$, the energy E_0 of the ground state satisfies the inequality

$$E_0 \leq \frac{\int \Phi^*(x)\hat{H}\Phi(x)\, dx}{\int \Phi^*(x)\Phi(x)\, dx} , \tag{2.187}$$

and equality holds only if $\Phi(x) \propto \psi_0(x)$ is the correct eigenfunction.

Using the trial function

$$\Phi(x) = \begin{cases} Axe^{-kx} , & x \geq 0 \\ 0 , & x < 0 , \end{cases} \tag{2.188}$$

we can use the Metropolis et al algorithm to approximate the integral for any given value of k. Varying k and choosing the minimum value obtained, we get an approximation for E_0.

To apply the algorithm, we write the right-hand-side of Eq. (2.187) as

$$E = \int \frac{\Phi^*(x)\Phi(x)}{\int \Phi^*(x)\Phi(x)\, dx} \frac{1}{\Phi(x)}\hat{H}\Phi(x)\, dx \equiv \int f(x)g(x)\, dx , \tag{2.189}$$

and find that

$$f(x) \propto \Phi(x)^2 = \begin{cases} x^2e^{-2kx} , & x \geq 0 \\ 0 , & x < 0 \end{cases} \tag{2.190}$$

and

$$g(x) \equiv \frac{1}{x}e^{kx}\left(-\partial_x^2 + x\right)xe^{-kx} = \frac{1}{x}e^{kx}\left(-k^2x + 2k + x^2\right)e^{-kx} = -k^2 + \frac{2k}{x} + x . \tag{2.191}$$

We sample random positions according to $f(x)$ by doing a random walk in the following way. We draw a displacement vector Δx with components uniformly distributed[5] over the interval $L/2, L/2$, evaluate $f(x)$ at the new position $x_k = x_i + \Delta x$, and accept x_k with probability

$$p_{\text{acc}} = \min\left(1, \frac{f(x_k)}{f(x_i)}\right) , \tag{2.192}$$

i.e. draw a uniformly distributed random number u and

[5]We could choose other distributions, e.g. $\Delta X \sim \mathcal{N}(0, L)$, but uniformly distributed numbers are generated fastest.

Note: It turns out that distributions with non-vanishing expectation value give wrong results here. Even if $E\Delta X = 0$, the results seem to be off when the probability density function of the distribution is not symmetric. This is quite surprising and I do not understand this behaviour.

$$\text{accept } x_k \text{ if } u < \frac{f(x_k)}{f(x_i)}.$$

If the acceptance ratio r_{acc} is considerably below 0.5, we decrease the interval length L, if it is larger, we increase L until we get $r \approx 0.5$. Averaging $g(x)$ over the positions x_i (counting the same position several times if a shift was rejected), we get

$$E \approx \langle g(x_i) \rangle \ . \tag{2.193}$$

Chapter 3

Optimization

We cover only one optimization method that follows as a corollary from Monte Carlo simulations.

3.1 Simulated annealing

There are many different methods for optimizing different functions. If the function depends on a large number of variables, *simulated annealing* can be used to find an approximation to the optimum. [*Genetic algorithms* occupy a similar ecological niche, but they are more complex and a bit tedious to apply for optimizing continuous functions].

The idea of simulated annealing is borrowed from what happens when steel (essentially an alloy of iron and carbon) cools down. When the cooling process is fast, only very small crystallites are formed and the material is malleable. To harden steel, it is *annealed*, i.e. heated again for some time and slowly cooled. This allows larger crystals to grow, which make the material elastic. If one anneals too much (too long and slowly), the crystals grow too large, and the material becomes brittle.

Energetically, the formation of large crystals is profitable, i.e. it brings the system closer to the global energy minimum. The small crystallites on the other hand create many local energy minima, and annealing is a way to use thermal fluctuations to "kick" the system out of these local minima and approach the global minimum.

The same ideas can be applied to get close to the global minimum of a function of many variables. As an example, consider the *travelling salesman problem*: A salesman needs to travel from city C_1 to N other cities $C_2, \dots C_N$ and back to C_1. He knows the distance between any pair of cities and wants to minimize the total distance

$$D \equiv \sum_i |\mathbf{x}(C_i) - \mathbf{x}(C_{i+1})| \tag{3.1}$$

he has to travel. This optimization problem is Np-complete, which means it cannot exactly be solved in reasonable time if N is large.

A simulated-annealing approach (to find an approximation to the global optimum, not the optimum itself) would look like this:

1. Start with an arbitrary cyclic sequence of cities (e.g. sort alphabetically).
2. Choose a "temperature" T and use the Metropolis algorithm to exchange, or not exchange, a randomly picked pair of cities. The acceptance probability is
$$p_{\text{acc}} = e^{-\beta(D_{\text{new}}-D_{\text{old}})} , \tag{3.2}$$
where $\beta = 1/T$. Repeat this m times.
3. According to a *schedule*, slowly "cool" the system by lowering temperature T. A commonly used schedule is linear in "time",
$$T(t) = T_0\left(1 - \frac{t}{\Delta t}\right) , \tag{3.3}$$
where t is a measure of the number of tries.
4. Once temperature is very small or zero, we have an approximation to the global minimum.

Note 1: Slower cooling will (in general) get you closer to the global minimum.

Note 2: Before starting a simulated-annealing scheme, you should estimate which temperature values are relevant. For example, if distances between cities for the travelling-salesman problem vary by a few thousand kilometres, your initial 'temperature' should be at least that large.

On the other hand, once your 'temperature' is 10 times smaller than a typical ΔD, the probability of accepting a change towards larger 'energy' is $\exp(-\Delta D/T) \approx 1/20\,000$, so you will see very few changes below this temperature. A good strategy is to monitor the number of changes for a faster (superficial) cooling schedule and use the results to fix the endpoints of a slower (more in-depth) schedule.

Chapter 4

Partial differential equations

4.1 Classification

Partial differential equations are differential equations for functions $f(x_1, x_2, \dots, x_n)$ of several variables. For linear equations of second order in two independent variables,

$$a\frac{\partial^2 f}{\partial x^2} + 2b\frac{\partial^2 f}{\partial x \partial y} + c\frac{\partial^2 f}{\partial x^2} \quad + d\frac{\partial f}{\partial x} + e\frac{\partial f}{\partial y} \quad + gf + h \ = \ 0 \tag{4.1}$$

a useful classification is based on the discriminant $b^2 - ac$ of the quadratic form $F(x, y) \equiv ax^2 + 2bxy + cy^2$:

$b^2 - ac < 0$**:** *Elliptic equation.*
Example: Poisson equation

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y}^2 = -h(x, y) \ . \tag{4.2}$$

Typically steady (time-independent) problems like steady-state heat conduction, elasticity (of thin strings and membranes), stationary wave or Schrödinger equations.

$b^2 - ac < 0$**:** *Parabolic equation.*
Example: (Time-dependent) heat conduction equation

$$\frac{\partial^2 T}{\partial t} - \chi\frac{\partial^2 f}{\partial x}^2 = q(x, t) \ . \tag{4.3}$$

Typically time-dependent problems with damping like diffusion and heat conduction or time dependent Schrödinger equation.

$b^2 - ac < 0$**:** *Hyperbolic equation.*
Example: (Time-dependent) heat conduction equation

$$\frac{\partial^2 T}{\partial t} - \chi\frac{\partial^2 f}{\partial x}^2 = q(x, t) \ . \tag{4.4}$$

Typically time-dependent problems with wave propagation like advection problems, equations for sound waves, electromagnetic waves (possibly with damping: telegrapher's equation).

4.2 Finite differences

Consider a smooth function $f(x)$ sampled on an equidistant grid x_k, which gives the values $f_k = f(x_k)$.[1] We define the following finite difference operators:

$$\begin{array}{lr}
\text{shift operator} & (T_\pm f)_k = f_{k\pm1} \,, \qquad (4.5)\\
\text{forward difference operator} & (\Delta f)_k = f_{k+1} - f_k \,, \qquad (4.6)\\
\text{central difference operator} & (\delta f)_k = f_{k+1/2} - f_{k-1/2} \,, \qquad (4.7)\\
\text{backward difference operator} & (\nabla f)_k = f_k - f_{k-1} \,, \qquad (4.8)\\
\text{averaging operator} & (Mf)_k \equiv \overline{f_k} = \dfrac{f_{k+1/2} + f_{k-1/2}}{2} \,. \qquad (4.9)
\end{array}$$

There are many interrelations between these operators, e.g.

$$\begin{aligned}
T_- &= T_+^{-1} \,, && && (4.10)\\
\Delta &= T_+ - 1 && \curvearrowright\ T_+ = 1 + \Delta \,, && (4.11)\\
\nabla &= 1 - T_- && \curvearrowright\ T_- = 1 - \nabla \,, && (4.12)\\
\delta &= T_+^{1/2} - T_+^{-1/2} && \,, && (4.13)\\
M &= T_+^{1/2} + T_+^{-1/2} && \,. && (4.14)
\end{aligned}$$

Powers of these operators are obtained by iteratively applying them several times. Schematically:

$$\begin{array}{c||c|c|c}
x_2 & f_2 & & \\
 & & \Delta f_1 & \\
x_1 & f_1 & & \Delta^2 f_0 \\
 & & \Delta f_0 & \\
x_0 & f_0 & &
\end{array}
\qquad
\begin{array}{c||c|c|c}
x_1 & f_1 & & \\
 & & \delta f_{1/2} & \\
x_0 & f_0 & & \delta^2 f_0 \\
 & & \delta f_{-1/2} & \\
x_{-1} & f_{-1} & &
\end{array}
\qquad
\begin{array}{c||c|c|c}
x_n & f_n & & \\
 & & \nabla f_n & \\
x_{n-1} & f_{n-1} & & \nabla^2 f_n \\
 & & \nabla f_{n-1} & \\
x_{n-2} & f_{n-2} & &
\end{array}$$

We can define arbitrary analytical functions of the operators via power series, e.g.

$$e^\Delta \equiv 1 + \Delta + \frac{\Delta^2}{2!} + \frac{\Delta^3}{3!} + \ldots \tag{4.15}$$

Once we accept the idea that functions can be applied to operators, we can e.g. invert Eq. (4.13) using the relation

$$\frac{t^{1/2} - t^{-1/2}}{2} = \frac{e^{(1/2)\ln t} - e^{-(1/2)\ln t}}{2} = \sinh\frac{\ln t}{2} \,,$$

[1]Even if the values x_k are not equidistant, the y can often be written as a smooth function $x(\xi_k)$ of an equidistant variable ξ. In that case, $f_k = f(x(\xi_k))$ is obtained from a smooth function $\xi \mapsto f(x(\xi))$ on an equidistant grid ξ_k, so the following still applies, albeit with some modification.

and find

$$\delta = 2\sinh\frac{\ln T_+}{2}\ , \tag{4.16}$$

which can immediately be inverted to

$$\ln T_+ = 2\,\mathrm{arsinh}\,\frac{\delta}{2}\ , \tag{4.17}$$

a result we are going to use later. Similarly,

$$M = \cosh\frac{\ln T_+}{2} = \sqrt{1+\sinh^2\frac{\ln T_+}{2}} = \sqrt{1+\frac{\delta^2}{4}}\ . \tag{4.18}$$

Newton's interpolation formula: Interpolation can be understood as a fractional shift operation of the knwon data:

$$\begin{aligned} f(x) &= T_+^t f_0 = (1+\Delta)^t f_0 = \sum_{k=0}^{n}\binom{t}{k}\Delta^k f_0 + R_n && (4.19)\\ &= f_0 + \binom{t}{1}\Delta f_0 + \binom{t}{2}\Delta^2 f_0 + \ldots + \binom{t}{n}\Delta^n f_0 + R_n\ , \qquad t := \frac{x-x_0}{h} && (4.20)\\ & && (4.21)\end{aligned}$$

Remainder term

$$R_n = \binom{t}{n+1} h^{n+1} f^{(n+1)}(x_0+n\vartheta h)\ , \qquad 0<\vartheta<1\ . \tag{4.22}$$

Stirling's interpolation formula:

$$\begin{aligned} f(t) &= f_0 + t\,\overline{\delta f_{\pm\frac{1}{2}}} + \frac{t^2}{2!}\,\delta^2 f_0 + \frac{t(t^2-1)}{3!}\,\overline{\delta^3 f_{\pm\frac{1}{2}}} + \frac{t^2(t^2-1)}{4!}\,\delta^4 f_0 + \\ &\quad + \frac{t(t^2-1)(t^2-4)}{5!}\,\overline{\delta^5 f_{\pm\frac{1}{2}}} + \ldots && (4.23)\\ &= f_0 + \binom{t}{1}\left(\overline{\delta f_{\pm\frac{1}{2}}} + \frac{t}{2}\delta^2 f_0\right) + \binom{t+1}{3}\left(\overline{\delta^3 f_{\pm\frac{1}{2}}} + \frac{t}{4}\delta^4 f_0\right) + && (4.24)\\ &\quad + \binom{t+1}{5}\left(\overline{\delta^5 f_{\pm\frac{1}{2}}} + \frac{t}{6}\delta^6 f_0\right) + + \ldots && (4.25)\end{aligned}$$

with

$$\overline{\delta^k f_{\pm\frac{1}{2}}} := \frac{\delta^k f_{\frac{1}{2}} + \delta^k f_{-\frac{1}{2}}}{2} \tag{4.26}$$

Example: Interpolate the following function values:

$x =$	-1	0	1	2
$f(x) =$	-7	1	-1	-7

The finite-difference tableau becomes

k	x_k	f_k	Δf	$\Delta^2 f$	$\Delta^3 f$
2	2	-7			
			-6		
1	1	-1		-4	
			-2		6
0	0	1		-10	
			8		
-1	-1	-7			

We have $t = x + 1$, and Newton's formula becomes

$$\begin{aligned} f &= f_0 + \binom{t}{1}\Delta f_0 + \binom{t}{2}\Delta^2 f_0 + \binom{t}{3}\Delta^3 f_0 && (4.27)\\ &= -7 + 8t - 10\frac{t(t-1)}{2} + 6\frac{t(t-1)(t-2)}{6} && (4.28)\\ &= -7 + 8(x+1) - 5(x+1)x + (x+1)x(x-1) && (4.29)\\ &= x^3 - 5x^2 + 2x + 1\ . && (4.30)\end{aligned}$$

Taylor's theorem: We introduce the derivative operator D with

$$Df_0 \equiv f_0'\ . \tag{4.31}$$

If $f(x)$ is sufficiently smooth, it has a Taylor series

$$f(x_0 + h) = \sum_{j=0}^{\infty} \frac{h^j f^{(j)}}{j!}\ , \tag{4.32}$$

which in terms of our operators becomes

$$T_+ f_0 = \sum_{j=0}^{\infty} \frac{(hD)^j}{j!} f_0 = e^{hD} f_0\ . \tag{4.33}$$

Thus,

$$T_+ = e^{hD}\ , \tag{4.34}$$

which can (formally) be inverted as

$$hD \ = \ \ln T_+ \ = \ \ln(1 + \Delta) \ = \ -\ln(1 - \nabla) \ = \ 2\,\mathrm{arsinh}\,\frac{\delta}{2}\ . \tag{4.35}$$

We can use this and similar formulæ to express the derivative operator as a power series in Δ, ∇ or δ:

$$hD = \ln(1+\Delta) = \frac{\Delta}{1} - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} - + \dots \tag{4.36}$$

$$= -\ln(1-\nabla) = \frac{\nabla}{1} + \frac{\nabla^2}{2} + \frac{\nabla^3}{3} + \dots\ . \tag{4.37}$$

For the central differences, a few manipulations [I1996] result in[2]

$$hD = M\left(1+\frac{\delta^2}{4}\right)^{-1/2} 2\,\mathrm{arsinh}\,\frac{\delta}{2} \tag{4.39}$$

$$= M\delta\left[\sum_{j=0}^{\infty}(-1)^j\binom{2j}{j}\left(\frac{\delta^2}{16}\right)^j\right]\left[\sum_{j=0}^{\infty}\frac{(-1)^j}{2j+1}\binom{2j}{j}\left(\frac{\delta^2}{16}\right)^j\right] \tag{4.40}$$

$$= M\left(\delta - \frac{\delta^3}{6} + \frac{\delta^5}{30} - \frac{\delta^7}{140} + \frac{\delta^9}{630} - + \dots\right), \tag{4.41}$$

More explicitly, these operator identities read

$$h\,f_0' = \Delta f_0 - \frac{\Delta^2 f_0}{2} + \frac{\Delta^3 f_0}{3} - \frac{\Delta^4 f_0}{4} + - \dots \tag{4.42}$$

$$= \nabla f_0 + \frac{\nabla^2 f_0}{2} + \frac{\nabla^3 f_0}{3} + \frac{\nabla^4 f_0}{4} + \dots \tag{4.43}$$

$$= \overline{\delta f_0} - \frac{\overline{\delta^3 f_0}}{6} + \frac{\overline{\delta^5 f_0}}{30} - \frac{\overline{\delta^7 f_0}}{140} + - \dots\ . \tag{4.44}$$

For the second-order derivatives, we get

$$h^2D^2 = \ln^2(1+\Delta) = \ln^2(1-\nabla) = 4\,\mathrm{arsinh}^2\,\frac{\delta}{2}, \tag{4.45}$$

and thus (eventually)

$$h^2\,f_0'' = \Delta^2 f_0 - \Delta^3 f_0 + \frac{11}{12}\Delta^4 f_0 - \frac{5}{6}\Delta^5 f_0 + - \dots \tag{4.46}$$

$$= \nabla^2 f_0 + \nabla^3 f_0 + \frac{11}{12}\nabla^4 f_0 + \frac{5}{6}\nabla^5 f_0 + \dots \tag{4.47}$$

$$= \delta^2 f_0 - \frac{\delta^4 f_0}{12} + \frac{\delta^6 f_0}{90} - \frac{\delta^8 f_0}{560} + - \dots\ . \tag{4.48}$$

[2] The reason this expression is so complex is that we need an expansion in $M\delta f_0, M\delta^3 f_0, M\delta^5 f_0, \dots$ (because these operators live on the main grid, just as f_i), while the straight-forward Taylor expansion yields

$$hD = 2\,\mathrm{arsinh}\,\frac{\delta}{2} = \delta\sum_{j=0}^{\infty}\frac{(-1)^j}{2j+1}\binom{2j}{j}\left(\frac{\delta^2}{16}\right)^j = \delta - \frac{\delta^3}{24} + \frac{3}{640}\delta^5 - + \dots, \tag{4.38}$$

and thus leads to an expansion in $\delta f_0, \delta^3 f_0, \dots$, all terms of which are defined on the shifted (half-index) grid.

Using the function values instead of the difference operators, these formulæ (truncated at different levels) become

$$f_0' = \frac{-f_{-1} + f_{-1}}{2h} + O\left(h^2\right) \tag{4.49}$$

$$f_0' = \frac{f_{-2} - 8f_{-1} \quad + 8f_1 - f_2}{12h} + O\left(h^4\right) \tag{4.50}$$

$$f_0' = \frac{-f_{-3} + 9f_{-2} - 45f_{-1} \quad + 45f_1 - 9f_2 + f_3}{60h} + O\left(h^6\right) \tag{4.51}$$

and

$$f_0'' = \frac{f_{-1} - 2f_0 + f_{-1}}{h^2} + O\left(h^2\right) \tag{4.52}$$

$$f_0'' = \frac{-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2}{12h^2} + O\left(h^4\right) \tag{4.53}$$

$$f_0'' = \frac{2f_{-3} - 27f_{-2} + 270f_{-1} - 490f_0 + 270f_1 - 27f_2 + f_3}{180h^2} + O\left(h^6\right) \tag{4.54}$$

4.3 Elliptic problems

Consider the Poisson equation

$$\Delta f \equiv \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = g(x, y) . \tag{4.55}$$

Introducing an equidistant grid

$$x_k = x_0 + kh , \qquad y_k = y_0 + kh \tag{4.56}$$

(with identical grid spacing for x and y), we have a two-dimensional array $f_{kl} \equiv f(x_k, y_l)$. To second order in δx and δy, we can approximate the second derivatives by

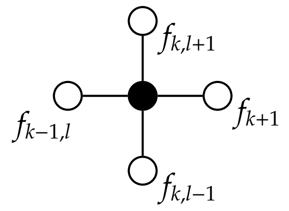
$$\left(\frac{\partial^2 f}{\partial x^2}\right)_{kl} = \frac{f_{k+1,l} - 2f_{k,l} + f_{k-1,l}}{h^2} + O\left(h^2\right) , \tag{4.57}$$

$$\left(\frac{\partial^2 f}{\partial y^2}\right)_{kl} = \frac{f_{k,l+1} - 2f_{k,l} + f_{k,l-1}}{h^2} + O\left(h^2\right) , \tag{4.58}$$

$$\tag{4.59}$$

and thus

$$(\Delta f)_{kl} = \frac{f_{k+1,l} + f_{k-1,l} + f_{k,l+1} + f_{k,l-1} - 4f_{k,l}}{h^2} + O\left(h^2\right) . \tag{4.60}$$



Using this, we can write a *finite-difference version* of the Poisson equation in the form

$$\frac{f_{k+1,l} + f_{k-1,l} + f_{k,l+1} + f_{k,l-1} - 4f_{k,l}}{h^2} = g_{kl} \ . \tag{4.61}$$

Solving this equation, we will get an approximation to the correct $f(x_k, y_l)$ that improves if we increase the number of grid points (i.e. decrease h).

Note: The linear system (4.61) gives rise to a *sparse* matrix, where most elements are zero (each line of the matrix has only 5 non-zero elements). Solving this with a general-purpose method like full *LU* decomposition would be an extreme waste of computing time (as we would mostly multiply some numbers by zero), however there are special methods to solve such systems, typically by iteration (see below).

4.3.1 Fourier method

In Fourier space, the Poisson equation (4.55) becomes an algebraic equation:

$$-\mathbf{k}^2 \tilde{f}(\mathbf{k}) = \tilde{g}(\mathbf{k}) \ . \tag{4.62}$$

Here

$$\tilde{f}(\mathbf{k}) = \mathcal{F}\{f(\mathbf{x}); \mathbf{k}\} = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(\mathbf{x})\, e^{-i\mathbf{k}\cdot\mathbf{x}}\, dx^2 \tag{4.63}$$

is the forward Fourier transform, with

$$f(\mathbf{x}) = \mathcal{F}^{-1}\{\tilde{f}(\mathbf{k}); \mathbf{x}\} = \int_{-\infty}^{\infty} \tilde{f}(\mathbf{k})\, e^{i\mathbf{k}\cdot\mathbf{x}}\, dk^2 \tag{4.64}$$

being the corresponding backward transform.

Equation (4.62) yields simply

$$\tilde{f}(\mathbf{k}) = -\frac{\tilde{g}(\mathbf{k})}{\mathbf{k}^2} \ , \tag{4.65}$$

or

$$f(\mathbf{x}) = -\mathcal{F}^{-1}\left\{ \frac{\mathcal{F}\{g(\mathbf{x}), \mathbf{k}\}}{\mathbf{k}^2}, \mathbf{x} \right\} \ . \tag{4.66}$$

For $k \equiv |\mathbf{k}| \to 0$ there is an apparent problem because we divide by k^2, but $k = 0$ simply represents a constant term in $f(\mathbf{x})$, and as Eq. (4.55) is not affected by a constant shift in $f(\mathbf{x})$, we can choose $\tilde{f}(\mathbf{0}) = 0$ (and re-add a constant in the end, if necessary).

If we solve the Poisson equation not in an infinite volume, but in a finite box with periodic boundary conditions, the backward Fourier integral (4.64) turns into a Fourier series, while integration in the forward transform (4.63) is only over the volume of the box.

In numerical mathematics, however, even the positions $\mathbf{x}$ in configuration space are discrete, as we *discretize* the box volume by introducing an equidistant grid. In that case, we are left with the *discrete Fourier transform*

$$\tilde{f}(\mathbf{k}_{jl}) \;=\; \mathcal{F}\{f(\mathbf{x}_{mn});\mathbf{k}_{jl}\} \;=\; \sum_{m,n} f(\mathbf{x}_{mn})\, e^{-i\mathbf{k}_{jl}\cdot\mathbf{x}_{mn}} \;=\; \sum_{m,n} f(\mathbf{x}_{mn})\, e^{-2\pi i\left(\frac{jm}{N_x}+\frac{ln}{N_y}\right)} \tag{4.67}$$

$$f(\mathbf{x}_{mn}) \;=\; \mathcal{F}^{-1}\{\tilde{f}(\mathbf{k}_{jl});\mathbf{x}_{mn}\} \;=\; \frac{1}{N}\sum_{j,l} f(\mathbf{k}_{jl})\, e^{\mathbf{k}_{jl}\cdot\mathbf{x}_{mn}} \;=\; \frac{1}{N}\sum_{j,l} f(\mathbf{k}_{jl})\, e^{2\pi i\left(\frac{jm}{N_x}+\frac{ln}{N_y}\right)}\,, \tag{4.68}$$

where $\delta k_x = 2\pi/L_x$, $\delta k_y = 2\pi/L_y$, and summation is over all points in the box.

To approximately solve the Poisson equation on the grid points, we use Eq. (4.66) and avoid division by zero for $\mathbf{k}_{jl} = 0$ as discussed above.

We thus have the following recipe:

1. Choose an equidistant grid $\mathbf{x}_{mn}$ to cover the box; make sure it is compatible with the periodic boundary conditions (no point counted twice).
2. Calculate $g(\mathbf{x}_{mn})$ on the grid.
3. Calculate the Fourier transform $\tilde{g}(\mathbf{k}_{jl})$
4. Use
$$\tilde{f}(\mathbf{k}_{jl}) = \begin{cases} -\dfrac{\tilde{g}(\mathbf{k}_{jl})}{k^2}\,, & k \neq 0 \\ 0\,, & k = 0 \end{cases} \tag{4.69}$$
to calculate $\tilde{f}(\mathbf{k}_{jl})$.
5. Transform back to get $f(\mathbf{x}_{jl})$.

How do we calculate the discrete Fourier transform? Many software packages (and IDL) provide the *Fast Fourier Transform* (FFT). This is a very efficient method, particularly if the number of points N_x and N_y are powers of 2. In two dimensions, the operation count of the Fast Fourier Transform is

$$n_{\text{ops}} \;\propto\; N_x \ln N_x \times N_y \ln N_y \;\propto\; N \ln^2 N\,, \tag{4.70}$$

where $N = N_x N_y$ is the total number of points; this is almost linear in N, as the logarithm grows very slowly.

Note 1: For Dirichlet boundary conditions $f|_{\partial V} = 0$, one uses the Fourier sine transform instead of the complex Fourier transform. Similarly, von Neumann boundary conditions require the Fourier cosine transform.

Note 2: The Fourier method can only be applied to equations with constant coefficients. For that case, it is one of the most efficient methods to solve elliptic PDEs.

4.3.2 Multigrid method

A very efficient way of solving the discretized Poisson equation is the *multigrid method* which, together with the main grid, introduces a number of coarser grids, typically with two, four, etc. times the grid spacing in each direction.

Consider the linear system (4.61). Under certain conditions (which are typically met), it can be solved iteratively using

$$f_{k,l} = \frac{f_{k+1,l} + f_{k-1,l} + f_{k,l+1} + f_{k,l-1} - h^2 g_{kl}}{4} \ . \tag{4.71}$$

There are two variants of this iteration scheme. For *Jacobi iteration*, at a given stage the right hand sides are evaluated for all grid points and only then the values f_{kl} are updated. Convergence is two times faster with *Gauss–Seidel* iteration, where Eq. (4.71) is evaluated for each point in sequence, using the latest updated values on the right hand side. Gauss–Seidel iteration also requires only half the memory of Jacobi iteration and has the important advantage of damping the highest wave number. This is the scheme we will consider here.

The iterative solution of Eq. (4.71) is quite slow. While the small scales ($\sim \delta x$) are quickly converging, it is the largest scales that take very long to reach their 'equilibrium' values due to the (local) iteration. For these large scales, however, one would not need the fine grid, and on a coarser grid they would converge much faster. Hence the idea of multigrid methods: Solve the problem iteratively on grids of different resolution by

1. *coarse-graining* (downsampling) using the *restriction matrix R*:

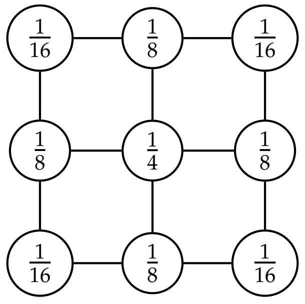
$$\mathbf{r}_{\text{coarse}} = \mathcal{R}\mathbf{r}_{\text{fine}} \ , \tag{4.72}$$

   and

2. *fine-graining* (refining = interpolation), using the *prolongation matrix P*:

$$\delta\mathbf{f}_{\text{fine}} = \mathcal{P}\delta\mathbf{f}_{\text{coarse}} \ . \tag{4.73}$$

A popular choice for the restriction matrix is represented by



This matrix acts as a *lowpass filter*: signals at the Nyquist frequency $f_{\text{Ny,fine}}$ of the fine grid are completely filtered out, while signals at $f_{\text{Ny,coarse}}$ are damped as little as possible.

This is crucial for the scheme to be efficient, as any contribution of $f_{\mathrm{Ny,fine}}$ would only be misinterpreted as a larger frequency on the coarser grid (*aliasing*).

The refinement is done using linear interpolation.

To see how the method works, consider the differential equation

$$\mathcal{L}f = g \, , \tag{4.74}$$

which, discretized at the grid spacing h, becomes

$$\mathcal{L}_h f_h = g_h \, . \tag{4.75}$$

If we have an approximate solution $\tilde{f}_h$, we can introduce the error

$$\delta f_h = f_h - \tilde{f}_h \, , \tag{4.76}$$

and find

$$-\mathcal{L}_h \delta f_h = \mathcal{L}_h \tilde{f}_h - \mathcal{L}_h f_h = \mathcal{L}_h \tilde{f}_h - g_h \, , \tag{4.77}$$

i.e.

$$-\mathcal{L}_h \delta f_h = r_h \, , \tag{4.78}$$

where $r_h \equiv \mathcal{L}_h \tilde{f}_h - g_h$ is the *residual*, which is a measure of how well our approximate solution $\tilde{f}$ solves the original problem.

Coarse-graining r_h to the coarser grid with spacing $H = 2h$,

$$r_H = \mathcal{R} r_h \, , \tag{4.79}$$

we arrive at

$$-\mathcal{L}_H \delta f_H = r_H \, , \tag{4.80}$$

which we solve (this is faster than on the finer grid) to obtain δf_H.

Then we fine-grain (interpolate) δf_H onto the finer grid,

$$\delta \tilde{f}_h = \mathcal{P} \delta f_H \, , \tag{4.81}$$

and calculate the new value of $\tilde{f}_h$ as

$$\tilde{f}_h^{\mathrm{new}} = \tilde{f}_h + \delta \tilde{f}_h \, . \tag{4.82}$$

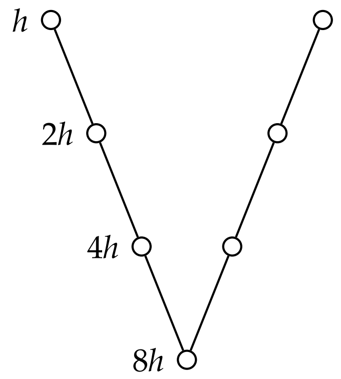
Formulated as a recipe, this two-grid method reads:

0. Start with a guess $\tilde{f}_h$ on the fine grid (e.g. choose 0).
1. Calculate the residual $r_h = \mathcal{L}_h \tilde{f}_h - g_h$, and coarse-grain it, $r_H = \mathcal{R} r_h$.
2. Solve $\mathcal{L}_H \, \delta f_H = -r_H$ on the coarser grid, fine-grain the correction, $\delta \tilde{f}_h = \mathcal{P} \delta f_H$, and add it to the initial guess. $\tilde{f}_h^{\mathrm{new}} = \tilde{f}_h + \delta \tilde{f}_h$.
3. Do one Gauss–Seidel iteration.
4. Continue with 1 until $\delta \tilde{f}_h$ is small enough.

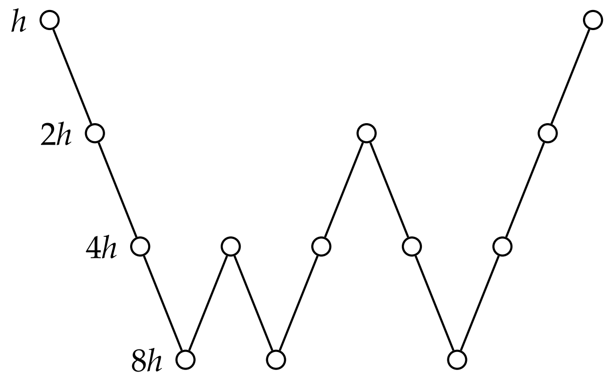
To turn this from a two- into a multi-grid method, we simply use another two-grid scheme for obtaining δf_H at stage 2, etc. The best way to code a multigrid method is obviously recursive.

Note 1: The numerical cost for the multigrid method is roughly $\propto N = N_x N_y$ and thus comparable to Fourier methods. However, multigrid methods can be used for equations with variable coefficients and also for nonlinear equations.

Note 2: The multigrid scheme thus described is referred to as 'V' cycle (the name should be evident from the scheme below). There are other popular cycles like the 'W' cycle.



The 'V' cycle.



A 'W' cycle (many other W cycles exist).

4.4 Parabolic problems

The heat conduction equation

$$\frac{\partial T}{\partial t} = \chi \frac{\partial^2 T}{\partial x^2} \tag{4.83}$$

is the prototype of a parabolic differential equation. To solve it numerically, we need to discretize in space and time:

$$T_k^l \equiv T(x_k, t_l) \; . \tag{4.84}$$

Starting from an initial condition T_k^l, which we assume to be known everywhere, we need to construct the solution at the next time, T_k^{l+1}.

4.4.1 Explicit scheme

A simple scheme is obtained from the discretization

$$\frac{T_k^{l+1} - T_k^l}{\delta t} = \chi \, \frac{T_{k-1}^l - 2T_k^l + T_{k+1}^l}{\delta x^2} \; . \tag{4.85}$$

We can explicitly solve for the unknown T_k^{l+1},

$$T_k^{l+1} \;=\; T_k^l + \frac{\chi\,\delta t}{\delta x^2}\left(T_{k-1}^l - 2T_k^l + T_{k+1}^l\right) \;=\; CT_{k-1}^l + (1{-}2C)T_k^l + CT_{k+1}^l \; , \tag{4.86}$$

where

$$C_{\text{dif}} \equiv \frac{\chi \delta t}{\delta x^2} \tag{4.87}$$

is the (diffusive) *Courant number*; we will mostly omit the index 'dif' and refer to C.

According to the discretizations used, this *explicit scheme* is of first order in time (Euler stepping), and of second order in space (centred second derivative).

4.4.2 Fully implicit scheme

Another scheme is obtained from the discretization

$$\frac{T_k^{l+1} - T_k^l}{\delta t} = \chi \, \frac{T_{k-1}^{l+1} - 2T_k^{l+1} + T_{k+1}^{l+1}}{\delta x^2} \; . \tag{4.88}$$

This time, we have a coupled system of equations for the unknown T_k^{l+1},

$$-\, CT_{k-1}^{l+1} + (1 + 2C)T_k^{l+1} - CT_{k+1}^{l+1} = T_k^l \tag{4.89}$$

Again, this *fully implicit scheme* is of first order in time (Euler stepping), and of second order in space (centred second derivative).

Note: The system (4.89) is a tridiagonal system and can thus be solved much more efficiently than a general linear system.

Nevertheless, it takes more numerical effort than the explicit scheme and (like any implicit scheme) has the major disadvantage that it leads to nonlinear systems of equations if the heat conduction equation is nonlinear (i.e. if χ depends on temperature T).

4.4.3 General implicit and Crank-Nicholson schemes

We can mix the explicit and the fully implicit schemes with a weighting factor $0 \le q \le 1$ to obtain the *general implicit scheme*:

$$\frac{T_k^{l+1} - T_k^l}{\delta t} = \frac{\chi}{\delta x^2} \left[q \left(T_{k-1}^{l+1} - 2T_k^{l+1} + T_{k+1}^{l+1} \right) + (1{-}q) \left(T_{k-1}^l - 2T_k^l + T_{k+1}^l \right) \right] \; . \tag{4.90}$$

The special cases $q = 0$ and $q = 1$ correspond to the explicit and fully implicit schemes, respectively.

The linear system takes the form

$$-\, qCT_{k-1}^{l+1} + (1{+}2qC)T_k^{l+1} - qCT_{k+1}^{l+1} \;=\; (1{-}q)CT_{k-1}^l + [1 - 2(1{-}q)C]T_k^l + (1{-}q)CT_{k+1}^l \; , \tag{4.91}$$

which is again a tridiagonal system.

The general implicit scheme is typically of first order in time and of second order in space, but in the special case $q = 1/2$, the temporal order is 2. This scheme,

$$-\frac{C}{2}T_{k-1}^{l+1} + (1+C)T_k^{l+1} - \frac{C}{2}T_{k+1}^{l+1} = \frac{C}{2}T_{k-1}^{l} + (1-C)T_k^{l} + \frac{C}{2}T_{k+1}^{l} , \tag{4.92}$$

is called *Crank–Nicholson scheme*.

4.4.4 Stability

If we apply the explicit scheme with a relatively large time step, we find that oscillatory perturbations grow rapidly and make the numerical solution quickly useless. On the other hand, for small δt the solution is very well behaved and settles to the physical steady state.

To assess the *stability* of a scheme and predict the useful range of δt, we use *von Neumann stability analysis*: Assume that at time t_l, temperature T varies harmonically in x,

$$T_j^l = e^{ikx_j} . \tag{4.93}$$

For each of these Fourier modes (characterized by the wave number k), investigate the amplitude factor A given by

$$T_j^{l+1} = AT_j^l . \tag{4.94}$$

If $|A| < 1$, the Fourier mode is stable, otherwise it is unstable. If at least one Fourier mode is unstable, the scheme is unstable, too, as any weak perturbation containing that Fourier mode will grow exponentially and make the solution unusable.

What is the range of wave numbers that are meaningful on a grid of spacing δx? The lowest wave number is 0, while the highest wave number corresponds to a zigzag profile with a period of $2\,\delta x$, thus $k \le 2\pi/(2\delta x) = \pi/\delta x \equiv k_{\mathrm{Ny}}$.[3] Here $k_{\mathrm{Ny}} \equiv \pi/\delta x$ is called the *Nyquist wave number*. For a finite number of grid points, the wave numbers will be discrete, with a spacing $\delta k = 2\pi/L_x$, where $L_x = N_x\,\delta x$ is the interval length. By increasing N_x, we can thus decrease δk as far as we want, hence we will treat k as a continuous variable.

Von Neumann stability analysis:

Assume

$$T_j^l = e^{ikx_j} . \tag{4.96}$$

Consider the wave length k to be a continuous number in the interval

$$0 \le k \le k_{\mathrm{Ny}} , \tag{4.97}$$

Analyze amplification factor A: if $A \le 1 \forall k$, scheme is stable, otherwise unstable.

[3] The fact that k_{Ny} is the highest wave number that can be distinguished on a grid of spacing δx can also be seen from

$$e^{i(k_{\mathrm{Ny}}+k')x_l} = (-1)^l e^{ik'x_l} = e^{i(-k_{\mathrm{Ny}}+k')x_l} , \tag{4.95}$$

which implies that the wave number $k_{\mathrm{Ny}}+k'$ is equivalent to $-k_{\mathrm{Ny}}+k'$.

Often we will work with the dimensionless wave number

$$\kappa \equiv k\,\delta x\,, \tag{4.98}$$

for which $\kappa_{\mathrm{Ny}} = \pi$.

Let us now see when the explicit scheme is stable. Plugging Eq. (4.93) into (4.86), we find

$$Ae^{ikx} = e^{ikx}\left[Ce^{-ik\delta x} + (1-2C) + Ce^{ik\delta x}\right]\,, \tag{4.99}$$

or

$$A \;=\; 1 - 2C + 2C\cos k\delta x \;=\; 1 - 2C(1-\cos\delta x)\,. \tag{4.100}$$

As $1-\cos\xi \le 0$, the amplitude factor can never exceed 1. However, instability occurs also if $A < -1$. The cos function takes its minimum of -1 for $k\delta x = \pi$, which will thus be the least stable Fourier mode. The stability threshold is thus where

$$-1 = 1 - 2C[1-(-1)]\,, \tag{4.101}$$

or $C = \dfrac{1}{2}$.

We thus find that the explicit scheme (4.86) is stable only if the *Courant criterion*

$$C \le \frac{1}{2} \tag{4.102}$$

is met. In practise, C should be chosen well below that threshold. If the Courant criterion is violated, the smallest scales are the most unstable ones. Also, because $A < -1$, the unstable modes will change sign from one time step to the next.

Interpretation of Courant criterion: "Information must not travel more than half a grid cell in one time step." (will become clearer for hyperbolic problems).

Next, let us look at the stability of the fully implicit scheme. Here we find

$$A\left[-Ce^{-ik\delta x} + 1{+}2C - Ce^{ik\delta x}\right] = 1\,, \tag{4.103}$$

or

$$A = \frac{1}{1 + 2C(1-\cos k\delta x)}\,. \tag{4.104}$$

One can easily see that

$$\frac{1}{1+4C} \le A \le 1\,, \tag{4.105}$$

and thus *the fully implicit scheme is always stable.*

Interpretation: No restriction on information propagation as solving the tridiagonal system propagates information across the whole grid.

Note 1: As A is strictly positive, there is no change of sign for any of the Fourier modes.

Note 2: Stability does not necessarily imply correctness. If we choose a very large time step (large Courant number), we will get an evolution that is considerably different from the exact one. However, the dissipative nature of parabolic problems leads to one finite state (thermal equilibrium) and this often makes the qualitative behaviour similar to the exact solution even when δt is large.

Finally, let us look at the stability of the general implicit scheme. Here,

$$A = \frac{1 - (1-q)2C(1-\cos k\delta x)}{1 + q2C(1-\cos k\delta x)} , \tag{4.106}$$

and in particular

$$A = \frac{1 - C(1-\cos k\delta x)}{1 + C(1-\cos k\delta x)} , \tag{4.107}$$

for the Crank–Nicholson scheme, and one can show that the general implicit scheme is

$$\begin{cases} \text{unconditionally stable} & \text{for } q \geq 1/2 , \\ \text{only stable if } C \leq \dfrac{1}{2(1-2q)} & \text{for } q < 1/2 . \end{cases} \tag{4.108}$$

The Crank–Nicholson scheme is thus unconditionally stable.[4]

4.4.5 Schemes to avoid: the Dufort–Frankel scheme; (in)consistency

There is a great deal of arbitrariness in choosing the discretizations for solving partial differential equations, and seemingly small changes to the scheme can make a huge difference.

As an example, consider the plausible scheme

$$\frac{T_k^{l+1} - T_k^{l-1}}{2\delta t} = \chi \, \frac{T_{k-1}^l - 2T_k^l + T_{k+1}^l}{\delta x^2} , \tag{4.109}$$

which is nice because it is of second order in time. Unfortunately, this scheme turns out to be *unconditionally unstable*.

The reason for the instability is related to the fact that even and odd time steps are somewhat decoupled (in the sense that if the left-hand side is used to step from one odd time step to the following one, the right-hand side represents only the even time step in between).

[4] This result holds for linear stability for constant thermal diffusivity χ. For more complicated settings, the fact that the Crank–Nicholson scheme is just on the border between conditional and unconditional stability may lead to unsatisfactory stability properties. In that case, a more implicit scheme ($q > 1/2$) may be needed. But you should not sacrifice second-order accuracy in time unless you are sure you need to.

To mitigate this, we can replace T_k^l on the right-hand side by the average $(T_k^{l-1} + T_k^{l+1})/2$:

$$\frac{T_k^{l+1} - T_k^{l-1}}{2\delta t} = \chi \frac{T_{k-1}^l - (T_k^{l-1} + T_k^{l+1}) + T_{k+1}^l}{\delta x^2} , \tag{4.110}$$

which can be written as

$$T_k^{l+1} = \frac{CT_{k-1}^l + \left(\frac{1}{2} - C\right) T_k^{l-1} + CT_{k+1}^l}{\frac{1}{2} + C} . \tag{4.111}$$

Von Neumann Stability analysis gives

$$A = \frac{2C \cos k\delta x \pm \sqrt{1 - 4C^2 \sin^2 k\delta x}}{1 + 2C} , \tag{4.112}$$

which turns out to always satisfy $|A| \leq 1$.

However, Eq. (4.111) is an *explicit* equation for T_k^{l+1}. The only additional price we have to pay compared to our explicit scheme is the fact that we need to retain two levels of values (T_k^l and T_k^{l-1}), but then we are rewarded with second-order accuracy in time, and with absolute stability.

So it looks like we have found an explicit scheme that is unconditionally stable! Unfortunately, there is a downside to the Dufort–Frankel scheme that makes me strongly recommend against its use: it solves the wrong equations. To see this, let us expand T around x_k and t_l up to second order:

$$\begin{aligned} T_{k\pm 1}^l &= T \pm \partial_x T \delta x + \frac{\partial_x^2 T}{2} \delta x^2 + O\left(\delta x^2\right) , && (4.113) \\ T_k^{l\pm 1} &= T \pm \partial_t T \delta t + \frac{\partial_t^2 T}{2} \delta t^2 + O\left(\delta t^3\right) , && (4.114) \end{aligned}$$

where we have dropped the indices $()_k^l$ for brevity. Equation (4.111) then becomes

$$\frac{2\partial_t T \, \delta t}{2\delta t} = \chi \frac{2T + \partial_x^2 T \delta x^2 - (2T + \partial_t^2 T \delta t^2)}{\delta x^2} + O\left(\delta t^2\right) + O\left(\delta x^2\right) + O\left(\delta t^4/\delta x^2\right) . \tag{4.115}$$

Ignoring the terms that get small when $\delta t \sim \delta x \to 0$, we are left with

$$\partial_t T = \chi \partial_x^2 T - \frac{\chi \, \delta t^2}{\delta x^2} \partial_t^2 T \tag{4.116}$$

This equation is of completely different type than the heat-conduction equation (it is the *telegrapher's equation*, which is hyperbolic), and certainly not what we wanted to solve in the first place.

We thus find that the Dufort–Frankel scheme – although constructed as a discretization of the heat conduction equation – in fact represents the discretization of a different equation. The discretization leading to the scheme was not *consistent*.

If we insist on using the Dufort–Frankel scheme for the heat conduction equation, we will need to make the additional term small. In order to achieve this, δt must decrease faster than δx, and we essentially have the requirement $\delta t \ll \chi \delta x^2$ or $C \ll 1$. The unconditional stability of the scheme does *not* allow us to choose a larger time step than for other explicit schemes.

To gain some peace of mind, let us verify the consistency of our general implicit scheme. Here we get

$$\frac{\partial_t T \delta t + \frac{\partial_t^2 T}{2} \delta t^2}{\delta t} = \chi \frac{\partial_x^2 T \delta x^2 + O\left(\delta x^2 \delta t\right) + O\left(\delta x^4\right)}{\delta x^2} , \tag{4.117}$$

and hence

$$\partial_t T = \chi \partial_x^2 T + O\left(\delta t\right) + O\left(\delta x^2\right) . \tag{4.118}$$

In the limit of both time step and grid spacing tending to zero, we obtain exactly the heat conduction equation, thus the general implicit scheme is consistent, together with all of its special cases like the explicit scheme, the fully implicit scheme, and the Crank–Nicholson scheme.

4.4.6 Boundary conditions

A Dirichlet boundary condition $T_0 = a(t)$ translates into the first line of

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ -qC & 1{+}2qC & -qC & 0 & \dots & 0 \\ 0 & -qC & 1{+}2qC & -qC & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \end{pmatrix} \begin{pmatrix} T_0^{l+1} \\ T_1^{l+1} \\ T_2^{l+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} a(t_{l+1}) \\ (1{-}q)CT_0^l + [1{-}2(1{-}q)C]T_1 + (1{-}q)CT_2^l \\ (1{-}q)CT_1^l + [1{-}2(1{-}q)C]T_2 + (1{-}q)CT_3^l \\ \vdots \end{pmatrix} \tag{4.119}$$

A von Neumann boundary condition $(\partial T/\partial x)_0 = b(t)$ can be discretized as

$$\frac{T_1 - T_0}{\delta x} = b(t) \tag{4.120}$$

with first-order accuracy.[5] This maps to the first line of

$$\begin{pmatrix} 1 & -1 & 0 & 0 & \dots & 0 \\ -qC & 1{+}2qC & -qC & 0 & \dots & 0 \\ 0 & -qC & 1{+}2qC & -qC & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \end{pmatrix} \begin{pmatrix} T_0^{l+1} \\ T_1^{l+1} \\ T_2^{l+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} -\delta x\, b(t_{l+1}) \\ (1{-}q)CT_0^l + [1{-}2(1{-}q)C]T_1 + (1{-}q)CT_2^l \\ (1{-}q)CT_1^l + [1{-}2(1{-}q)C]T_2 + (1{-}q)CT_3^l \\ \vdots \end{pmatrix} \tag{4.121}$$

[5] It can be shown that an $N - 1$th order boundary scheme is consistent with an Nth order scheme for interior points. Thus, the Crank–Nicholson scheme together with a first-order boundary scheme will still be of second order in space and time.

4.4.7 Non-homogeneous equation

It is pretty straight-forward to adapt our schemes for the non-homogeneous equation

$$\frac{\partial T}{\partial t} - \chi \frac{\partial^2 T}{\partial x^2} = h(x,t) \ . \tag{4.122}$$

The general implicit scheme takes the form

$$\begin{aligned} -qCT_{k-1}^{l+1} + (1{+}2qC)T_k^{l+1} - qCT_{k+1}^{l+1} &= (1{-}q)CT_{k-1}^{l} + [1-2(1{-}q)C]T_k^l + (1{-}q)CT_{k+1}^{l} \\ &\quad + \delta t\, h(x_k, t_{l+1/2}) \ , \end{aligned} \tag{4.123}$$

which is still a tridiagonal system as before.

There is some freedom in the choice of the time argument for h, and we could just as well have chosen $h(x_k, t_l)$ or $h(x_k, t_{l+1})$. The present choice [to evaluate $h(x,t)$ at the time $t_{l+1/2} \equiv t_l + \delta t/2$] has the advantage that the Crank–Nicholson scheme

$$-\frac{C}{2}T_{k-1}^{l+1} + (1{+}C)T_k^{l+1} - \frac{C}{2}T_{k+1}^{l+1} = \frac{C}{2}T_{k-1}^{l} + (1{-}C)T_k^{l} + \frac{C}{2}T_{k+1}^{l} + \delta t\, h(x_k, t_{l+1/2}) \ , \tag{4.124}$$

will still be of second order in time.

4.4.8 Higher-order explicit schemes

The accuracy of the schemes used so far was second order in space and first or second order in time. Many people are content with this – but they should not, as it is relatively simple to use higher-order schemes and increase accuracy. One reason not to do so may be the better stability properties of implicit methods, which are more difficult to implement in higher order. Another complication with higher-order methods are often the boundary conditions, although most of this can be dealt with elegantly using ghost zones (see 4.5.2). But for explicit time stepping with periodic boundary conditions there is definitively no excuse for not using high-order schemes.

As high-order methods are going to be our favourite method for solving hyperbolic equations (which we will in fact turn into parabolic ones), we leave the discussion for Section 4.5.2 below.

4.5 Hyperbolic problems

The standard example of a hyperbolic equation is the wave equation

$$\frac{\partial^2 f}{\partial t^2} = c^2 \frac{\partial^2 f}{\partial x^2} \ . \tag{4.125}$$

It describes waves moving with phase velocity c in both directions and has the general solution

$$f(x,t) = \varphi_{\mathrm{r}}(x{-}ct) + \varphi_{\mathrm{l}}(x{+}ct) \ . \tag{4.126}$$

The wave operator can be factored as

$$\left(\frac{\partial}{\partial t} - c\frac{\partial}{\partial x}\right)\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right) f = 0 , \tag{4.127}$$

and if we are only interested in signals propagating to the right, we can drop the first differential operator and get

$$\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right) f = 0 , \tag{4.128}$$

or

$$\frac{\partial f}{\partial t} = -u\frac{\partial f}{\partial x} , \tag{4.129}$$

where we have replaced the symbol c with u. This equation has the general solution

$$f(x,t) = \varphi_{\mathrm{r}}(x{-}ut) . \tag{4.130}$$

The advection equation (4.129) is *not* a hyperbolic equation, as it is only of first order. Nevertheless it will be our prototype for this section, as it has all the properties of hyperbolic systems that are relevant for us. Also, note that Eq. (4.127) implies that the wave equation can be written as a system of two coupled advection equations

$$\frac{\partial f}{\partial t} = u\frac{\partial g}{\partial x} , \tag{4.131}$$

$$\frac{\partial g}{\partial t} = u\frac{\partial f}{\partial x} . \tag{4.132}$$

4.5.1 Low-order schemes

A straight-forward discretization of the advection equation

$$\frac{\partial f}{\partial t} = -u\frac{\partial f}{\partial x} , \tag{4.133}$$

is

$$\frac{f_k^{l+1} - f_k^l}{\delta t} = -u\,\frac{f_{k+1}^l - f_{k-1}^l}{2\delta x} , \tag{4.134}$$

which is first-order accurate in time and second-order accurate in space. The scheme turns out to be consistent.

Von Neumann stability analysis gives

$$\frac{A-1}{\delta t} = -u\frac{i\,\sin k\delta x}{\delta x} , \qquad \text{or} \qquad A = 1 - i\,C_{\mathrm{adv}} \sin k\delta x , \tag{4.135}$$

where

$$C_{\mathrm{adv}} \equiv \frac{u\delta t}{\delta x} \tag{4.136}$$

is the *advective Courant number*; we will mostly omit the index 'adv', unless this can lead to confusion with the diffusive Courant number.

The modulus of the amplification factor A is thus

$$|A|^2 = 1 + C^2 \sin^2 k\delta x \geq 1 \ , \tag{4.137}$$

and thus, unfortunately, this scheme is unconditionally unstable. As [NR77] put it:

"The resulting finite-difference approximation [...] is called the FTCS representation (Forward Time Centred Space) [...] It's a fine example of an algorithm that is easy to derive, takes little storage, and executes quickly. Too bad it doesn't work!"

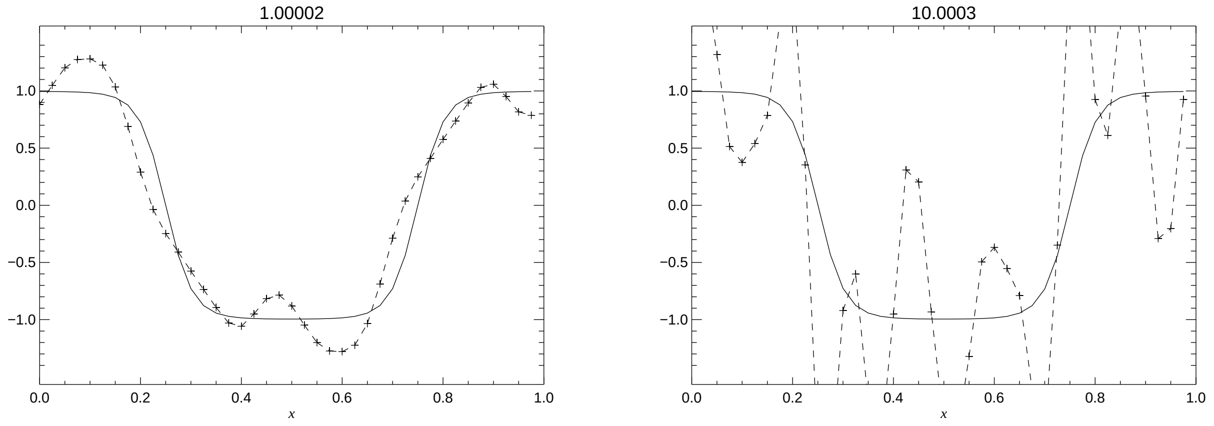


Figure 4.1: The $O\left(\delta t, \delta x^2\right)$ (first-order time step, second-order spatial derivatives) scheme applied to the advection problem (4.129) with $u = 1$ and periodic boundary conditions. The time step is extremely small ($\delta t = 0.0005$). The solid line shows the exact solution (identical to the initial profile), while the crosses and dashed line show the numerical solution. Left: t=1 (i.e. the pattern has travelled once through the interval $[0, 1]$. Right: t=10 (the pattern has travelled ten times through the interval).

The instability is illustrated in Figure 4.1.

The Lax scheme

Replacing f_k^l by $(f_{k-1}^l + f_{k+1}^l)/2$ on the left-hand side of Eq. (4.134), we obtain the *Lax scheme*

$$\frac{f_k^{l+1} - \dfrac{f_{k-1}^l + f_{k+1}^l}{2}}{\delta t} = -u \frac{f_{k+1}^l - f_{k-1}^l}{2\delta x} \ , \tag{4.138}$$

or

$$f_k^{l+1} = \frac{f_{k-1}^l + f_{k+1}^l}{2} - C \, \frac{f_{k+1}^l - f_{k-1}^l}{2} \ . \tag{4.139}$$

The amplification factor is

$$A = \cos k\delta x - i\, C \sin k\delta x \ , \tag{4.140}$$

and

$$|A|^2 \ = \ \cos^2 k\delta x + C^2 \sin^2 k\delta x \ = \ 1 + (C^2 - 1) \sin^2 k\delta x \ . \tag{4.141}$$

The Lax scheme is thus conditionally stable, and the stability criterion is the *Courant–Friedrichs–Lewy stability criterion* (or simply *Courant criterion*)

$$C \leq 1 \,, \tag{4.142}$$

which holds in a similar form for most or all explicit schemes.

Interpretation of the Courant criterion: the scheme is only stable if information (which travels at speed u) crosses no more than one cell per time step. This seems pretty intuitive, as during one step, our scheme, which extends just one point to the left and one to the right, does not have access to information from further away. On the other hand, we should remember that von Neumann stability analysis is base on the *global* Fourier modes and their evolution is governed by the *phase velocity*. Thus, the propagation of information (which propagates at the *group velocity*) does probably not give a consistent interpretation.

Why is the Lax scheme stable (provided δt satisfies the Courant condition), while (4.134) is not? The answer becomes clear if we write the Lax scheme in the form

$$\frac{f_k^{l+1} - f_k^l}{\delta t} = -u \frac{f_{k+1}^l - f_{k-1}^l}{2\delta x} + \frac{f_{k-1}^l - 2f_k^l + f_{k+1}^l}{2\delta t} \,. \tag{4.143}$$

This suggests that the Lax scheme is the scheme (4.134) applied to the equation

$$\begin{aligned} \frac{\partial f}{\partial t} &= -u \frac{\partial f}{\partial x} + \frac{\delta x^2}{2\delta t} \frac{\partial^2 f}{\partial x^2} & (4.144) \\ &= -u \frac{\partial f}{\partial x} + \frac{\delta x}{2C} \frac{\partial^2 f}{\partial x^2} \,. & (4.145) \end{aligned}$$

Additional second-derivative terms like the one appearing here are called *numerical diffusivity* (or *numerical viscosity*) terms. They damp the largest wave numbers (that we will not get right anyway), but have relatively little impact on the small wave numbers (large scales) that we are interested in.

Is the Lax scheme consistent? Equation (4.144) tells us that it is generally not. However, one will typically us a time step that is slightly (by a factor of 1/2 or so) below the Courant threshold, thus keeping C constant when reducing the grid size δx. In this case, as Eq. (4.145) shows, the additional term tends to zero as δx does. Thus, in practise, the Lax scheme is (often) consistent, but when choosing very small time steps, it is not.

The upwind scheme

The advection equation only advects information in one direction (the positive x direction if $u > 0$). The Lax scheme, however, takes into account information from both neighbouring points, which is part of the reason why its numerical diffusivity is quite large. We can avoid this by discretizing

$$\frac{f_k^{l+1} - f_k^l}{\delta t} = \begin{cases} -|u| \, \dfrac{f_{k+1}^l - f_k^l}{\delta x} \,, & \text{for } u < 0 \\[2ex] -|u| \, \dfrac{f_k^l - f_{k-1}^l}{\delta x} \,, & \text{for } u > 0 \end{cases} \tag{4.146}$$

This is called the *upwind scheme* (because only information from the upwind or upstream direction is used); it is first-order accurate in time and space and is stable, provided that

$$C \le 1 \; . \tag{4.147}$$

The upwind scheme can be written as

$$\frac{f_k^{l+1} - f_k^l}{\delta t} = -u \, \frac{f_{k+1}^l - f_{k-1}^l}{2\delta x} + |u| \, \frac{f_{k-1}^l - 2f_k^l + f_{k+1}^l}{2\delta x} \; . \tag{4.148}$$

It discretizes

$$\frac{\partial f}{\partial t} = -u \, \frac{\partial f}{\partial x} + \frac{|u| \, \delta x}{2} \, \frac{\partial^2 f}{\partial x^2} + O\left(\delta t\right) + O\left(\delta x^2\right) \; . \tag{4.149}$$

We thus again have numerical diffusivity in the scheme, but this time the scheme is consistent for any (admissible) values of δt and δx. Another advantage over the Lax scheme is that if $u = u(x)$, viscosity is only applied "where it is needed", i.e. there is little diffusion in regions where $|u|$ is small.

And yet, the upwind scheme is only first-order accurate in space (and time), and its numerical diffusivity is still relatively large (where $|u|$ is large). Higher-order methods are much better.

More low-order schemes

There are a number of schemes designed to be less diffusive than the Lax or the upwind scheme and some of them are of second order in both time and space (e.g. the Lax–Wendroff scheme or the staggered leapfrog scheme). We will not discuss them here (as we will use something better), and just refer to [NR77] here.

TVD schemes

Another class of schemes we mention only briefly are the *total variation diminishing* (TVD) schemes. These are schemes that use a nonlinear filter to minimize the total variation

$$V[f] \equiv \sum |f_i - f_{i-1}| \tag{4.150}$$

due to the small scales[6]. This suppresses the formation of Nyquist zigzags that are in many cases the source of instability.

[6] For the one-dimensional advection problem (4.129) [but with possibly space-and time-dependent advection velocity $u(x,t)$], the analytical total variation

$$V[f] \equiv \int \left| \frac{df}{dx} \right| dx \tag{4.151}$$

[for which Eq. (4.150) is a discretization] is known to be constant. This is because the effect of advection is to displace and stretch or compress the initial profile, which does not change the difference between successive minima and maxima.

However, in more than one dimension, or for equations more complicated than the plain advection equation, there is no analog for this property.

These schemes give impressive results for simple test problems (and can be applied for real-life applications), but the nonlinear character of the filtering makes them somewhat dubious and very difficult to analyze.

Conservative schemes

Many schemes use the fact that advection-type equations can be written in the form

$$\frac{\partial f}{\partial t} = -\nabla(\mathbf{F}) \; , \tag{4.152}$$

where $\mathbf{F}$ is a *flux function*. In our example, $F = uf$ if u is constant. This representation is called *conservation form*, as it is related to conservation laws (e.g. for momentum and total energy), and the class of *conservative schemes* employs this form to obtain exact conservation of these quantities (up to round-off error). Note that the conservation form of equations can be quite unnatural compared to the *primitive equations* (continuity, Navier-Stokes, induction, . . .) that we are used to deal with.

While it may sound impressive that these schemes manage to exactly conserve energy and momentum, it must be kept in mind that these conservation properties just constrain the trajectory of the system to a 6^N-4-dimensional hypersurface in phase space if N is the number of grid points. To get the correct trajectory, one needs another 6^N-3 conservation properties, none of which is known. Thus, while conservative schemes are good at conserving a few known properties, they can be just as bad at getting the phase-space trajectory right as are other schemes of the same order.

And while non-conservative schemes allow checking the known conservation laws as a simple accuracy test, conservative schemes provide no such straight-forward quality measure.

4.5.2 Higher-order schemes

The basic idea of higher-order explicit schemes is quite simple: Treat time and spatial discretization completely separately. We thus use a high-order spatial discretization like Eq. (4.54) for the right-hand side. Once we know how to calculate the RHS, we can apply Runge–Kutta for time stepping.

The scheme we are going to use is third-order in time and sixth-order in space.

Spectral characteristics of finite-difference stencils

Important insight into the properties of a finite-difference scheme is obtained by applying it to a harmonic function

$$f(x) = e^{ikx} \; , \qquad \text{where again } 0 \le |k| \le k_{\mathrm{Ny}} \; . \tag{4.153}$$

Applying the exact first and second derivative operators to the harmonic function (4.153) would yield

$$\partial_x e^{ikx} = ik e^{ikx} \; , \qquad \partial_x^2 e^{ikx} = -k^2 e^{ikx} \; , \tag{4.154}$$

thus the *spectral transfer functions*

$$H^{(1)}(k) \equiv e^{-ikx} D^{(1)} e^{ikx} , \tag{4.155}$$

$$H^{(2)}(k) \equiv e^{-ikx} D^{(2)} e^{ikx} \tag{4.156}$$

indicate the quality of the finite-difference approximations $D^{(1)}$, $D^{(2)}$: for exact derivatives $D^{(1)} = D, D^{(2)} = D^2$ one would get [7]

$$H^{(1)}(k) = ik , \qquad H^{(2)}(k) = -k^2 . \tag{4.157}$$

Finite-difference approximations $D^{(1)}$ and $D^{(2)}$ to the first and second order derivatives will give rise to deviations,

$$H^{(1)}(k) = ik\Theta , \qquad H^{(2)}(k) = -k^2\Theta^{(2)} , \tag{4.158}$$

where Θ and $\Theta^{(2)}$ are very close to 1 for small wave numbers (large scales), but differ strongly from 1 near the Nyquist wave number.

For the second-order first derivative operator (4.49), we get

$$H^{(1)}(k) = ik \frac{\sin\kappa}{\kappa} = ik\left(1 - \frac{\kappa^2}{6} + \cdots\right) , \tag{4.159}$$

where $\kappa \equiv k\,\delta x$. For the fourth-order operator (4.50), we get

$$H^{(1)}(k) = ik \frac{8\sin\kappa - \sin 2\kappa}{6\kappa} = ik\left(1 - \frac{\kappa^4}{30} + \cdots\right) , \tag{4.160}$$

and for sixth order (4.51)

$$H^{(1)}(k) = ik \frac{45\sin\kappa - 9\sin 2\kappa + \sin 3\kappa}{30\kappa} = ik\left(1 - \frac{\kappa^6}{140} + \cdots\right) . \tag{4.161}$$

Note that all of these expressions become zero at the Nyquist frequency. This is unavoidable for centred finite-difference operators on a non-staggered grid.

Similarly, for the second-order second derivative operator (4.52), we get

$$H^{(2)}(k) = -k^2\, 2\frac{1-\cos\kappa}{\kappa^2} = -k^2\left(1 - \frac{\kappa^2}{12} + \cdots\right) , \tag{4.162}$$

for fourth order (4.53)

$$H^{(2)}(k) = -k^2 \frac{15 - 16\cos\kappa + \cos 2\kappa}{6\,\kappa^2} = -k^2\left(1 - \frac{\kappa^4}{9} + \cdots\right) , \tag{4.163}$$

and for sixth order (4.54)

$$H^{(2)}(k) = -k^2 \frac{245 - 270\cos\kappa + 27\cos 2\kappa - 2\cos 3\kappa}{90\,\kappa^2} = -k^2\left(1 - \frac{\kappa^6}{560} + \cdots\right) . \tag{4.164}$$

Figure 4.2 shows the spectral transfer functions for a number of schemes from order 2 up to 20. One can easily see how all schemes yield good approximations to the exact derivative for small k, but for intermediate wave numbers (say, half the Nyquist wavenumber $\kappa_{\mathrm{Ny}} = k_{\mathrm{Ny}}\delta x = \pi$) only higher orders reproduce the exact derivatives with sufficient accuracy.

[7] Such exact numerical derivative operators are indeed implemented by *spectral schemes* which apply a Fourier transform, multiply in Fourier space by ik or $-k^2$, and then transform back.

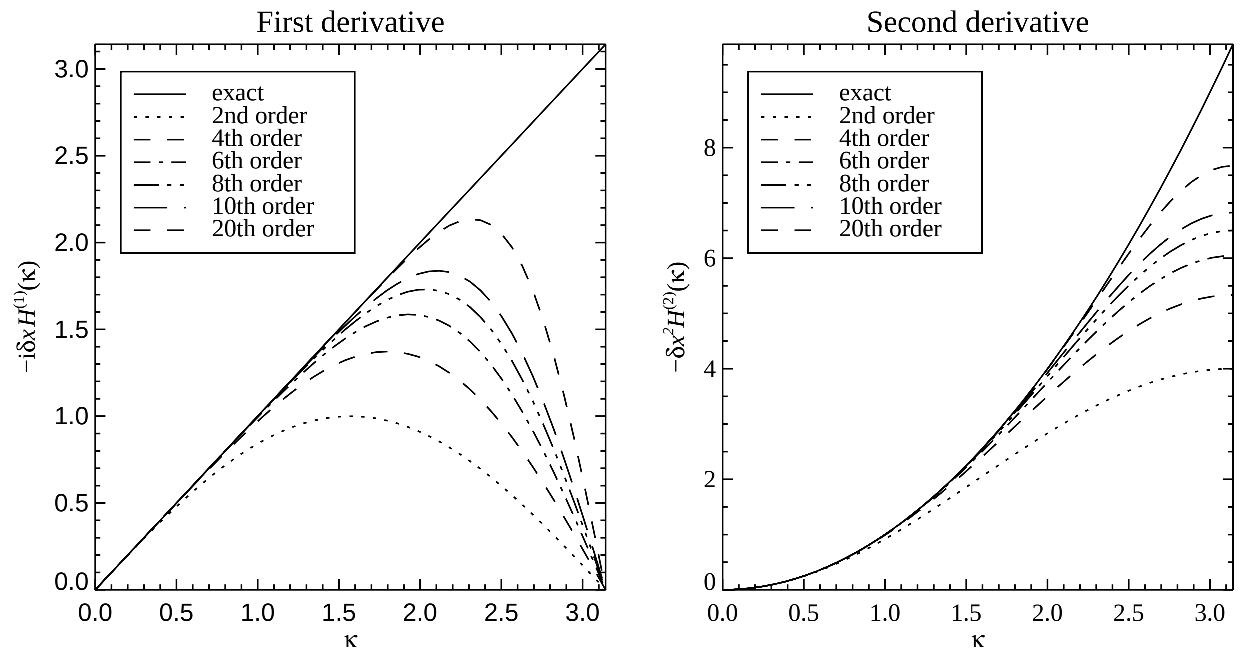


Figure 4.2: Spectral transfer functions $H(k\delta x) \equiv e^{-ik} D e^{ik}$ as a function of $\kappa = k\delta x$ for centred finite-difference schemes of different orders. Left: transfer function for the first derivative operator, $D^{(1)}$, multiplied by $-i\,\delta x$. Right: spectral transfer function for the second derivative operator, $D^{(2)}$, multiplied by $-\delta x^2$. The solid lines show the transfer function of the exact derivative operator (which is reproduced by spectral schemes).

Note: The fact that the numerical first derivative of a Nyquist signal is zero has several consequences. First, we note that for a Nyquist signal there is no propagation due to the advection term $uD^{(1)}$, thus signals with wave numbers κ close to π will quickly get out of phase with the larger-scale signals (see *phase error* in Sec. 4.5.2 below).

Second, an iterated first derivative $D^{(1)}D^{(1)}f$ will be zero for a Nyquist signal, and thus not have any damping effect, while the second derivative $D^{(2)}f$ gives rise to dissipative damping. As a consequence, any iterated first derivative term on the right-hand side of our partial differential equations *must* be re-written in terms of a second derivative. E.g. the heat-conduction term $\partial_x(\lambda \partial_x T)$ must be used in the form $\partial_x\lambda\, \partial_x T + \lambda\partial_x^2 T$. If this rule is not applied, calculations typically work reasonably well for some time, but Nyquist signals slowly grow in amplitude (due to boundary effects and nonlinearities) and eventually make the numerical solution unusable.

Stability

Under the advection equation, a harmonic profile

$$f(x,0) = e^{ikx} \tag{4.165}$$

evolves as

$$f(x,t) = e^{ik(x-ut)} = e^{-ikut} f(x,0)\ , \tag{4.166}$$

thus the exact amplification factor is

$$A_{\text{exact}} = e^{-iku}\ . \tag{4.167}$$

The amplification factor of the discretized scheme will differ from this value:

$$A = e^{-iuk+\gamma+i\omega} , \qquad \gamma, \omega \in \mathbb{R} . \tag{4.168}$$

The quantity γ is a growth rate and gives rise to the *amplitude error*, while ω represents the *phase error*.

The question of stability boils down to the sign of γ. If $\gamma > 0$ for some modes, then the energy in these modes will grow and eventually dominate the solution and render it useless. Reducing the *modulus* of γ in this case will not remove the instability — it only increases the time for which it can be ignored.

If $\gamma < 0$, on the other hand, energy in the corresponding modes will decrease. This implies that there is some *numerical dissipation* at work, but normally this only affects the smaller scales. By decreasing the time step, both amplitude and phase error will be decreased, so if $\gamma \le 0$ for all modes, one can control the errors by adjusting the time step δt.

Similar conclusions can be drawn for the diffusion term, which has only an amplitude error γ_{diff}. Here, however, instability will only occur if γ_{diff} overcomes the natural decay of the modes.

Table 4.1: Leading-order terms (in δt) of the growth rate γ and phase drift ω for time-stepping schemes of different order m. The quantities Θ and $\Theta^{(2)}$ measure the quality of the spatial schemes. Note that in the absence of diffusion $\gamma < 0$ (indicating stability) only for $m = 3, 4; 7, 8; 11, 12 \ldots$.

m	γ	ω	γ_{diff}
1	$\frac{(uk\Theta)^2}{2}\delta t$	$uk(1-\Theta) + \frac{(uk\Theta)^3}{3}\delta t^2$	$\nu k^2(1-\Theta^{(2)}) - \frac{\nu^2 k^4 {\Theta^{(2)}}^2}{2}\delta t$
2	$\frac{(uk\Theta)^4}{8}\delta t^3$	$uk(1-\Theta) - \frac{(uk\Theta)^3}{6}\delta t^2$	$\nu k^2(1-\Theta^{(2)}) + \frac{\nu^3 k^6 {\Theta^{(2)}}^3}{6}\delta t^2$
3	$-\frac{(uk\Theta)^4}{24}\delta t^3$	$uk(1-\Theta) - \frac{(uk\Theta)^5}{30}\delta t^4$	$\nu k^2(1-\Theta^{(2)}) - \frac{\nu^4 k^8 {\Theta^{(2)}}^4}{24}\delta t^3$
4	$-\frac{(uk\Theta)^6}{144}\delta t^5$	$uk(1-\Theta) + \frac{(uk\Theta)^5}{120}\delta t^4$	$\nu k^2(1-\Theta^{(2)}) + \frac{\nu^5 k^{10} {\Theta^{(2)}}^5}{120}\delta t^4$

Table 4.1 shows the leading order in δt of the amplitude and phase errors for time-stepping schemes of orders 1 to 4. Without artificial diffusivity, only the third- and fourth-order schemes are stable ($\gamma < 0$), provided that the time step is sufficiently small (see § 4.5.2 below). Although this result is formulated for the advection problem (4.129), exactly the same stability conditions hold in the case of linear sound waves

$$\partial_t \ln \varrho = -\partial_x v \tag{4.169}$$
$$\partial_t v = -c_{\text{s}}^2 \partial_x \ln \varrho \tag{4.170}$$

if the advection speed u is replaced by the speed of sound c_{s}. In the case of sound waves in a medium that moves at speed u, the relevant velocity is $\max(|u \pm c_{\text{s}}|)$.

To conclude, we can say that

For advection and similar problems the amplitude error, and thus the stability of the scheme, is determined by the time-stepping scheme, while the phase error is normally dominated by the spatial discretization.

Artificial viscosity

When solving partial differential equations that are more realistic than our simple advection problem, even third- and fourth order time-stepping schemes require some amount of diffusivity/viscosity due to boundary effects, nonlinearities or just to minimize the consequences of the phase error. Like in the case discussed above, this viscosity will always tend to zero for $\delta x \to 0$. The recommended minimum value of viscosity for the $O\left(\delta t^3, \delta x^6\right)$ scheme is[8]

$$\nu = c_\nu \, U_{\max} \, \delta x \tag{4.171}$$

where $U_{\max}$ is the largest velocity in the problem (including propagation speeds of waves), and $c_\nu = 0.01 \,..\, 0.02$.

The length of the time step

Even the explicit schemes labelled as 'stable' are only stable if the time step δt satisfies a Courant condition of the form

$$\delta t \le c_{\mathrm{adv}} \frac{\delta x}{u} \qquad \text{or} \qquad \delta t \le c_{\mathrm{dif}} \frac{\delta x^2}{\nu} \, . \tag{4.172}$$

For $O\left(\delta t^3, \delta x^6\right)$ schemes, the stability boundary is $c_{\mathrm{adv}} = 1.092$ and $c_{\mathrm{dif}} = 0.4157$. For $O\left(\delta t^4, \delta x^6\right)$ schemes, we have $c_{\mathrm{adv}} = 1.783$ and $c_{\mathrm{dif}} = 0.4608$. In practise one should use a time step considerably smaller than the stability limit ($C = 0.5$ or smaller), since at the very limit nonlinearities and boundaries can destabilize the configuration.

Other propagation velocities (like the sound speed) will give rise to similar time step restrictions. The recommended time step for the $O\left(\delta t^3, \delta x^6\right)$ scheme is

$$\delta t = \min\left(0.4\, \delta x / U_{\max},\ 0.08\, \delta x^2 / \nu_{\max}\right) \, , \tag{4.173}$$

where $U_{\max}$ is the largest velocity[9] in the problem and $\nu_{\max}$ the largest diffusivity.

[8] This dependence on velocity and grid spacing is reminiscent of the upwind scheme (Sec. 4.5.1), but for higher-order schemes we need much less diffusivity.

[9] Again, $U_{\max}$ includes propagations speeds of waves and also drift terms like $\partial_x \lambda$ in $\partial_x(\lambda \partial_x T) = \partial_x \lambda \, \partial_x T + \lambda \partial_x^2 T$.

Our standard scheme

This box summarizes the properties of the scheme we normally use to solve partial differential equations.

- 6th-order spatial derivative operators:

$$D^{(1)}f = \frac{-f_{k-3} + 9f_{k-2} - 45f_{k-1} \quad + 45f_{k+1} - 9f_{k+2} + f_{k+3}}{60\,\delta x} \tag{4.174}$$

$$D^{(2)}f = \frac{2f_{k-3} - 27f_{k-2} + 270f_{k-1} - 490f_k + 270f_{k+1} - 27f_{k+2} + f_{k+3}}{180\,\delta x^2} . \tag{4.175}$$

- Artificial viscosity:

$$\nu = c_\nu U_{\max}\delta x$$

  with $c_\nu = 0.01 \ldots 0.02$ (or physical viscosity at least that large);

- 3rd-order Runge–Kutta time stepping (this is a memory-efficient version of third-order Runge–Kutta, although this is far from being obvious here):

$$\begin{array}{c|ccc} 0 & & & \\ \frac{8}{15} & \frac{8}{15} & & \\ \frac{2}{3} & \frac{1}{4} & \frac{5}{12} & \\ \hline & \frac{1}{4} & 0 & \frac{3}{4} \end{array} \tag{4.176}$$

  applied to solve

$$\frac{df_k}{dt} = -uD^{(1)}f_k + \nu D^{(2)}f_k . \tag{4.177}$$

- Time step:

$$\delta t = \min\left(0.4\,\frac{\delta x}{U_{\max}},\ 0.08\,\frac{\delta x^2}{\nu_{\max}}\right) .$$

Boundary conditions

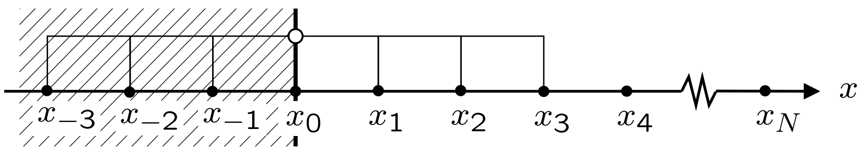


Figure 4.3: Sketch of ghost zones for a seven-point finite-difference stencil on a grid ranging from x_0 to x_N.

So far, our discussion was implicitly assuming that boundary conditions are periodic (or that the interval in x is unbounded). In real life, one often has to use other boundary conditions. We will discuss this just briefly, restricting ourselves to boundary conditions implemented by setting *ghost zone* values. A ghost zone is a layer of fictitious points beyond the boundary which is introduced so that wide finite-difference stencils can be applied even close to the boundary. For our sixth-order (seven-point) stencil, we need three points on each side of the given point, thus we will need three ghost layers on each side if we want to be able to calculate derivatives in the very boundary points. This situation is depicted in Fig. 4.3.

When ghost zones are used, the boundary conditions provide a rule how to set the values in the ghost points. We just present four popular choices of boundary conditions that can be thus implemented:

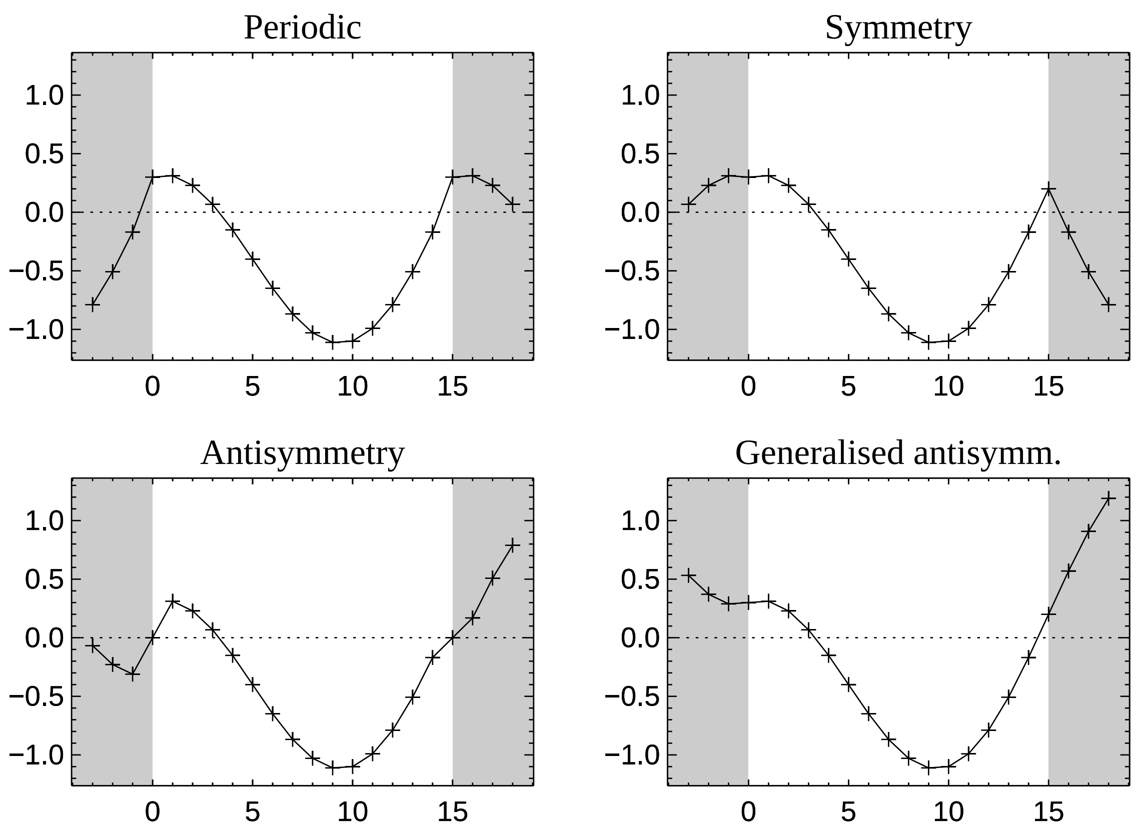


Figure 4.4: The four boundary conditions discussed in the text, applied to an arbitrarily chosen function. The shaded regions to the left and right are the 'ghost zones'.

1. Periodic boundary conditions

$$y_{-1} = y_{N-1} \,, \qquad y_{-2} = y_{N-2} \,, \qquad y_{-3} = y_{N-3} \,. \tag{4.178}$$

2. Symmetry ($y'_0 = 0$)

$$y_{-1} = y_1 \,, \qquad y_{-2} = y_2 \,, \qquad y_{-3} = y_3 \,. \tag{4.179}$$

3. Antisymmetry ($y_0 = 0$)

$$y_{-1} = -y_1 \,, \qquad y_{-2} = -y_2 \,, \qquad y_{-3} = -y_3 \,. \tag{4.180}$$

4. Generalized antisymmetry ($y_0'' = 0$)

$$y_{-1} = 2y_0 - y_1 , \qquad y_{-2} = 2y_0 - y_2 , \qquad y_{-3} = 2y_0 - y_3 . \tag{4.181}$$

Figure 4.4 illustrates these four boundary conditions.

Note that symmetry also implies that $y_0^{(3)} = y_0^{(5)} = \ldots = 0$, and similarly antisymmetry and generalized antisymmetry imply $y_0^{(2)} = y_0^{(4)} = \ldots = 0$. These additional conditions are often compatible with the physical problem (some heat-conduction problems are solved using symmetry conditions like this), but this will not generally be the case. If they are not, the overall order of the numerical scheme is reduced to second order in space, which sounds like a dramatic change for the worse. In practise, however, we find that these boundary conditions give quite good results (the coefficient in front of the $O(\delta x^2)$ error term must be small), and the resulting schemes have very good stability properties.

Application I: Sound waves

Sound waves are propagating pressure perturbations arising from the interaction of velocity, density and pressure.

Continuity equation:

$$\frac{\partial \varrho}{\partial t} + \nabla \cdot (\varrho \mathbf{v}) = 0 , \tag{4.182}$$

or

$$\frac{\partial \varrho}{\partial t} + \mathbf{v} \cdot \nabla \varrho = -\varrho \nabla \cdot \mathbf{v} , \tag{4.183}$$

which we can write as

$$\frac{D \ln \varrho}{Dt} = -\nabla \cdot \mathbf{v} , \tag{4.184}$$

where

$$\frac{D}{Dt} \equiv \frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla \tag{4.185}$$

is the *advective derivative* (sometimes also called *convective* or *Lagrangian derivative*).

Equation of motion: The *Navier–Stokes equation*

$$\frac{D\mathbf{v}}{Dt} = -\frac{\nabla P}{\varrho} + \nu \left(\Delta \mathbf{v} + \frac{1}{3} \nabla \nabla \cdot \mathbf{v} \right) + \mathbf{f}_{\mathrm{ext}} \tag{4.186}$$

describes momentum conservation (although this is not obvious from this form of the equation). The above form holds for a compressible fluid if the kinematics viscosity ν is constant.

Pressure term: If there is a unique relation between P and ϱ — for example an adiabatic, polytropic or isothermal equation of state — we can define the sound speed [10]

$$c_\mathrm{s}^2 \equiv \frac{dP}{d\varrho} \; . \qquad (4.189)$$

This allows us to rewrite the pressure term as follows

$$-\frac{1}{\varrho}\nabla P = -\frac{c_\mathrm{s}^2}{\varrho}\nabla\varrho = -c_\mathrm{s}^2\nabla\ln\varrho \; . \qquad (4.190)$$

Thus, in terms of logarithmic density, our equations become

$$\begin{aligned} \frac{\partial \ln\varrho}{\partial t} + \mathbf{v}\cdot\nabla\ln\varrho &= -\nabla\cdot\mathbf{v} \qquad (4.191)\\ \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}\cdot\nabla)\mathbf{v} &= -c_\mathrm{s}^2\nabla\ln\varrho + \nu\left(\Delta\,\mathbf{v} + \frac{1}{3}\nabla\nabla\cdot\mathbf{v}\right) + \mathbf{f}_\mathrm{ext} \; . \end{aligned}$$

One-dimensional case

For a one-dimensional flow $\mathbf{v} = [v(x,t),0,0]$, $\varrho = \varrho(x,t)$, Eqs. (4.191) and (4.192) simplify to

$$\begin{aligned} \partial_t \ln\varrho &= -v\,\partial_x\ln\varrho - \partial_x v \; , \qquad (4.192)\\ \partial_t v &= -v\partial_x v - \frac{\partial_x P}{\varrho} + \frac{4}{3}\nu\partial_x^2 v \qquad (4.193) \end{aligned}$$

Sound waves

If we linearize equations (4.191), (4.192) using the ansatz

$$\begin{aligned} \ln\varrho &= \ln\varrho_0 + \lambda \; , \qquad (4.194)\\ \mathbf{v} &= \mathbf{0} + \mathbf{u} \qquad (4.195) \end{aligned}$$

[10] For a perfect gas in the adiabatic case (entropy s = const), the equation of state is

$$P = K\varrho^\gamma \; , \qquad (4.187)$$

where $\gamma \equiv c_p/c_v$ is the adiabatic index, i. e. the ratio of specific heat at constant pressure, c_p, to the specific heat at constant volume, c_v, and K is a constant related to the entropy s. For this case we obtain the familiar relation

$$c_\mathrm{s}^2 = \left(\frac{dP}{d\varrho}\right)_s = \gamma\frac{P}{\varrho} = \gamma\frac{\mathcal{R}}{\mu_\mathrm{mol}}T \qquad (4.188)$$

where $\mathcal{R}/\mu_\mathrm{mol}$ is the specific gas constant, T the temperature, and $(\partial/\partial)_s$ denotes the partial derivative for constant entropy s.

A *polytropic equation of state* looks like the adiabatic one (4.187), but with the adiabatic index replaced by an exponent Γ that is treated as a free parameter.

and assuming that $\lambda \ll 1$, $|\mathbf{u}| \ll c_\mathrm{s}$, we obtain the system

$$\frac{\partial \lambda}{\partial t} = -\frac{\partial u}{\partial x}\,, \tag{4.196}$$

$$\frac{\partial u}{\partial t} = -c_\mathrm{s}^2 \frac{\partial \lambda}{\partial x}\,. \tag{4.197}$$

This system has the general solution

$$\lambda = f(x - c_\mathrm{s} t) + g(x + c_\mathrm{s} t) \tag{4.198}$$

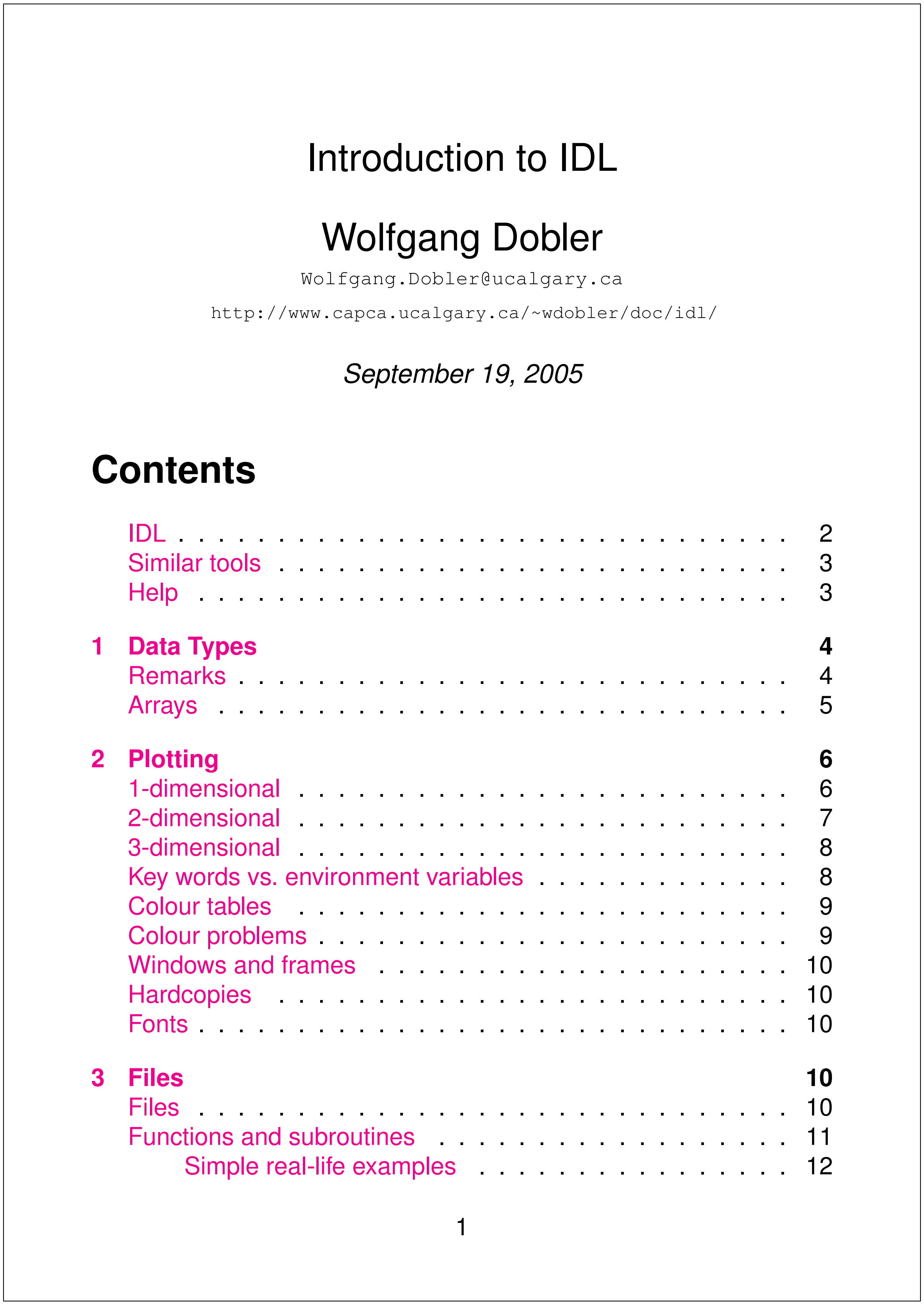
$$u = c_\mathrm{s}\, f(x - c_\mathrm{s} t) - c_\mathrm{s}\, g(x + c_\mathrm{s} t) \tag{4.199}$$

where $f(\cdot)$, $g(\cdot)$ are arbitrary functions.

Nonlinear sound waves are described by our equations (4.191) and (4.192) and the nonlinearities give rise to new phenomena like steepening of wave profiles ans shocks.

Appendix A

Interactive Data Language (IDL)



Introduction to IDL

Wolfgang Dobler

Wolfgang.Dobler@ucalgary.ca

http://www.capca.ucalgary.ca/~wdobler/doc/idl/

September 19, 2005

Contents

Foreword

IDL

– $\approx$ least elegant programming language I have used
– (but I never used *COBOL* or *Visual Basic* . . .)
– most powerful graphics tool I have ever used:
  ∘ full-featured programming language
  $\leadsto$ some aspects are awkward, but you can program around them
– widely used in the scientific community
– interactive
– optional arguments for functions and procedures
– command line editing terrible, but can be fixed (*rlwrap*)

Similar tools

Matlab
Octave (a Matlab clone)
- much more modern language
- much lower graphics quality

SciLab (another Matlab clone)
PerlIDL (*Perl* derivate $\Longrightarrow$ very powerful language)
Python-numeric / **Python-scientific** (*Python* library)
GnuDL
- full IDL syntax
- many functions / subroutines still missing

⋮

Xmgr/Grace
- click-and-cramp
- apparently has a scripting language

Gnuplot
- scripting language, but not more

⋮

Help

Command line: *idlhelp* $\Longrightarrow$ online help

In IDL: Type *'?'* $\Longrightarrow$ online help

In IDL: '`idl> help, var`' $\Longrightarrow$ info on `var`

Literature:

- David Fanning, *IDL Programming Techniques*, 2nd Edition, 2000; ISBN 0-9662383-2-X

Web:

- `http://www.dfanning.com/` (*very* useful)
- RSI Technical Tips at `http://www.rsinc.com/services/prodspec.cfm?product=IDL`

3

– Newsgroup `news:comp.lang.idl-pvwave`

Online handbooks: PDF manuals come with the IDL installation, starting point: *$IDL/docs/onlguide.pdf*

---

1. Data Types

Atomic:

– byte, integer, long (integer)
– float, double, complex
– string

Complex:

– arrays
– structures
– object classes

Beware of

`for i=0,100000 do` *(something)*

— the 2-byte integer i will never attain the value 100000. Instead write

`for i=0L,100000 do` *(something)*

Now i is initialised as *long int* and your loop will (eventually) finish.

Remarks

– not declarative

– 2/3 problem (shared with C, Fortran, . . .):
$2./3. \neq 2/3 = 0$

– Examples:
  - `x = 5` *(integer)*
  - `x = 5D0` *(double)*
  - `z = complex(x,7)` *(guess what)*

– info on variables:

  - `help, x`
  - `help, !p, /STRUCT`
- system variables: '!p', '!x', '!y', '!z', '!d', '!pi' (and others)

Arrays

- `zeros = fltarr(10,20)`
- `ones = make_array(10,20,VAL=1)`
- zero indexing: `zeros[0,0] = 1`
- `count = indgen(10) & print, count`

- coordinate vectors

```
nx=50 & ny=60 & nz=70

x = findgen(nx)    ;; x = 0, 1, . . . , 48, 49

x0=-1. & x1=1.
x = x0 + findgen(nx)/(nx-1.)*(x1-x0)
    ;; x = -1, -0.86, . . . , 1

    ;; alternatively:
x = linspace(-1, 1, nx)    ;; (my routine)
y = linspace(-1.5, 1.5, ny)
z = linspace(0.2, 5, nz)
```

- *rebin-reform*
  reform: re-shape array without changing data
  rebin: duplicate array elements

```
xx = rebin(reform(x, nx, 1), nx, ny)
    ;; coordinate grid array
```

```
yy = rebin(reform(y, 1, ny), nx, ny)
rr = sqrt(xx^2+yy^2)
```

– array syntax

much faster than explicit looping

```
ff = sin(6*xx)*exp(-2*rr)
surface, ff, x, y
```

– array slices:

- `ff[0,0]` , `ff[10,7]`
- `ff[2:5,0]`
- `ff[*,5]`
- `ff[*,3:7]`

– *where* function and array subscripts:

```
bad = where(rr gt 0.5)
```

;; don't use '>' instead of 'gt' $\Longrightarrow$ *dubious results*

```
ff[bad] = 0
surface, ff, x, y
```

---

2. Plotting

1-dimensional

```
f = sin(3*x)*exp(-x)
plot, x, f
plot, x, f, XRANGE=[0,1], COLOR=150
plot, x, f, PSYM=-4
plot, x, f, PSYM=10
```

```
g = sqrt(2*!pi*z)*z^z*exp(-z)
plot, z, g
plot, z, g, /XLOG, /YLOG
plot, z, g, /YLOG
oplot, z, gamma(z+1), LINESTYLE=2, COLOR=150
xyouts, 0.5, 1.5, "Stirling's formula"
```

2-dimensional

```
surface, ff, x, y
for i=0,360,10 do begin $
   surface, ff, x, y, AZ=25+i & wait, 0.1

shade_surf, ff, x, y

xsurface, ff

contour, ff, x, y
contour, ff, x, y, /FILL
contour, ff, x, y, /FILL, NLEVELS=60
```

;; (More or less) the same, but shorter:

```
contourfill, ff, x, y, /GRID   ;; (my routine)
```

```
gg = cos(xx)*exp(-rr)
velovect, ff, gg, x, y
vel, ff, gg
vel, ff, gg, LEN=0.2, NVECS=1000
```

Combining different types of plotting:

```
contourfill, ff, x, y
contour, ff, x, y, NLEVELS=20, /OVERPLOT
velovect, ff, gg, x, y, /OVERPLOT
```

3-dimensional

```
xxx = rebin(reform(x, nx, 1, 1), nx, ny, nz)
yyy = rebin(reform(y, 1, ny, 1), nx, ny, nz)
zzz = rebin(reform(z, 1, 1, nz), nx, ny, nz)
rrr = sqrt(xxx^2+yyy^2)
phi = atan(yyy,xxx)
m = 1
kz = 2*!pi/(z[nz-1]-z[0])
fff = rrr^2*exp(-4*rrr^2)*cos(m*phi-kz*zzz)

shade_volume, fff, 0.9*max(fff), vert, poly
scale3, $    ;; ($-sign = continuation character)
  XRANGE=[0,nx], YRANGE=[0,ny], ZRANGE=[0,nz]
image = POLYSHADE(vert, poly, /T3D)
loadct, 3
TV, image
```

Key words vs. environment variables

```
f = cos(z)
plot, z, f, XRANGE=[0,6]
```

vs.

```
!x.range = [0,6]
plot, z, f
plot, z, sin(z)
```

!x is a *structure* and *!x.range* accesses one slot of it:

```
help, /STRUCTURE, !x
```

keyword	*env. variable*
`title`	`!p.title`
`color`	`!p.color`
`charsize`	`!p.charsize`
`linestyle`	`!p.linestyle`
`psym`	`!p.psym`
`thick`	`!p.thick`
	`!p.multi`
`{x,y,z}charsize`	`!{x,y,z}.charsize`
`{x,y,z}margin`	`!{x,y,z}.margin`
`{x,y,z}range`	`!{x,y,z}.range`
`{x,y,z}style`	`!{x,y,z}.style`
`{x,y,z}title`	`!{x,y,z}.title`

Colour tables

```
contourfill, ff, x, y
loadct, 5   ;; loads colour table No. 5
contourfill, ff, x, y
loadct, 16   ;; loads colour table No. 16
contourfill, ff, x, y
xloadct   ;; interactively pick colour table
```

Colour problems

If you only get different shades of red, try

```
device, DECOMPOSE=0
```

in your IDL startup file (☞ below).

Windows and frames

Open a new window:

```
  window, 1
```

Plot several graphs in one window

```
  !p.multi = [0,3,2]
  for i=0,5 do plot, x, x^i, XRANGE=[0,1]
  !p.multi = 0   ;; reset to single plot
```

Hardcopies

```
  set_plot, 'PS'
  plot, z, f
  device, /CLOSE
  set_plot, 'X'
  ;; or (my commands):
  psa, FILE='tmp.ps', THICK=2
  plot, z, f
  pse
```

Fonts

```
  plot, x, f, XTITLE='!8B!6!Dnorm!N - !7w'
```

You can also use PostScript fonts (requires some setup; default with my *psa*, *pse*) or TrueType fonts

3. Files and Functions

Files

Write

```
x1 = linspace(0,10,50)
y1 = cos(x1)
```

to file *incl1.pro* and

```
@incl1
plot, x1, y1
end
```

to file *short.pro*

Now you can run it with

```
idl>  .r short
```

You can however *not* run *incl1.pro* this way:

```
idl>  .r incl1
% End of file encountered before end of program.
```

since the `end` is missing $\Longrightarrow$ inconsistency.

Functions and subroutines

Write

```
function htan, x
  if (x lt 0) then begin
    res = tanh(x)
  endif else begin
    res = tan(x)
  endelse
  return, res
end
```

to file *htan.pro* and

```
pro jabber, x, y, z, BRILLIG=bril
  if (keyword_set(bril)) then print, 'Brillig'
  print, '(x,y,z) =', x, y, z
end
```

to file *jabber.pro*

Now you can use the new function *htan*

```
  idl>  print, htan(0.7)
```

and procedure *jabber*

```
idl>  jabber, 5, 3, htan(-2)
```

Simple real-life examples

(yet simplified)

idl/lib/default.pro

```
pro default, var, val
  if (n_elements(var) eq 0) then var=val
end
```

idl/lib/minmax.pro

```
function minmax, f
  on_error, 2    ;; return to caller on error
  return, [min(f),max(f)]
end
```

idl/lib/contourfill.pro

```
pro contourfill, z, x, y, $
               NLEVELS=nlevels, _EXTRA=_extra
  if (n_elements(nlevels) eq 0) then nlevels=60
  contour, array, x, y, $
    NLEVELS=nlevels, /FILL, _EXTRA=_extra
end
```

Startup file; journalling

You want to be able to use your own procedures from everywhere.

1. Put your (general purpose) scripts into directory `~/idl/lib/` (or `~/idl/pro/,...`)
2. Tell *IDL* to read *~/.idlrc* at startup (in *~/.cshrc*):

```
export IDL_STARTUP=$HOME/.idlrc
setenv IDL_STARTUP $HOME/.idlrc
```

3. Add your directory to the *IDL* search path. In *~/.idlrc*, write

```
device,decompose=0  ;; (probably needed for indexed colour)

!EDIT_INPUT = 1000  ;; (increase length of history)

!path = !path + ':~/idl/lib:' $
    + expand_path('+~wdobler/idl/lib') $
    + expand_path('+~wdobler/f90/pencil-code/lib')
    ;; (append your and my directories to search path)
```

Journalling creates a script of your IDL session

⟹ turn experiments into scripts by adding '`end`'

```
idl> journal, 'jou.pro'  ;; activate journalling

;; (interactively try some IDL statements)

flush, !journal  ;; ensure journal file is up-to-date

;; (copy journal file, edit if necessary and add 'end')
;; (...)

idl> journal  ;; deactivate journalling
```

Subroutines vs. working in global scope

Subroutines:

- `sub1, a, b, c`
- allow for good programming style
- local variables (⟹ no name clashes)
- need *common blocks* for global communication

Global scope:

13

- `.r glob1`
- interactive access to *all* data
  (⟶ idea of an interactive language)
- caution needed: don't overwrite variables

Recommendation: use subroutines for general-purpose tasks only; work in global scope with your data

4. Interacting with Fortran

```
Fortran code                          IDL program
  real, dimension(5,7,7) :: a           a = fltarr(5,7,7)
  double precision :: d                 d = 0D0
  integer, i,k,l                        i=0L & j=0L & k=0L
    ...                                   ;; Fortran integers are
                                          ;;   IDL long ints

                                        close, 1   ;; (just to be sure)
                                        openu, 1, /F77
                                          ;; open unformatted,
                                          ;; assume F77 records
  write(1) a, i                         read, 1, a, i
  write(1) d, k, l                      read, 1, d, k, l
                                        close, 1
```

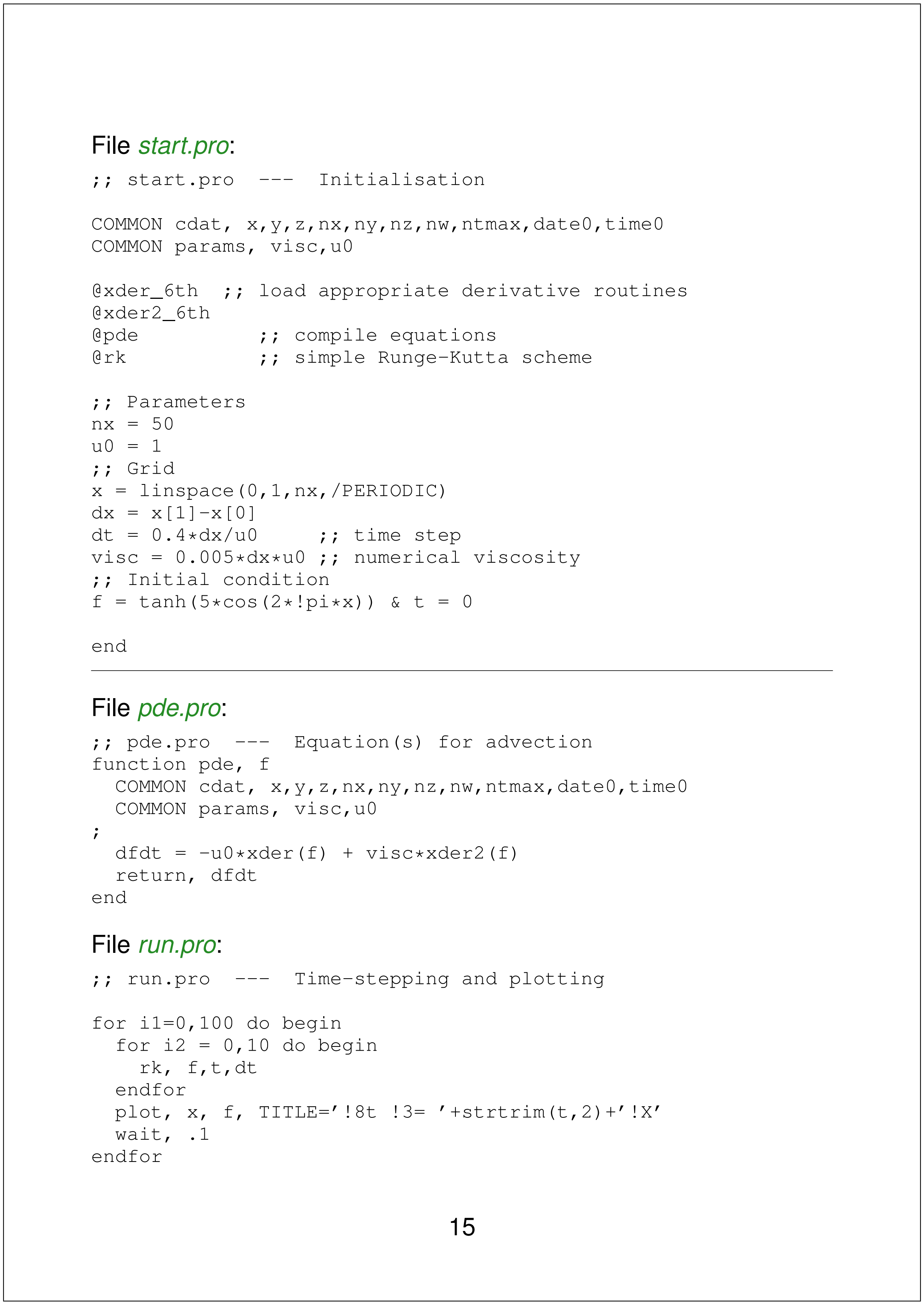
Doing it all in *IDL*

Philosophy: Want to be able to do the same things with your data as in Fortran

Thus (to work with finite-difference code): need derivative operators (*xder*, *xder2*, etc.)

Add time-stepping $\Longrightarrow$ don't need Fortran at all.

Example: advection of passive scalar



File *start.pro*:

```
;; start.pro  ---  Initialisation

COMMON cdat, x,y,z,nx,ny,nz,nw,ntmax,date0,time0
COMMON params, visc,u0

@xder_6th  ;; load appropriate derivative routines
@xder2_6th
@pde         ;; compile equations
@rk          ;; simple Runge-Kutta scheme

;; Parameters
nx = 50
u0 = 1
;; Grid
x = linspace(0,1,nx,/PERIODIC)
dx = x[1]-x[0]
dt = 0.4*dx/u0     ;; time step
visc = 0.005*dx*u0 ;; numerical viscosity
;; Initial condition
f = tanh(5*cos(2*!pi*x)) & t = 0

end
```

File *pde.pro*:

```
;; pde.pro  ---  Equation(s) for advection
function pde, f
  COMMON cdat, x,y,z,nx,ny,nz,nw,ntmax,date0,time0
  COMMON params, visc,u0
;
  dfdt = -u0*xder(f) + visc*xder2(f)
  return, dfdt
end
```

File *run.pro*:

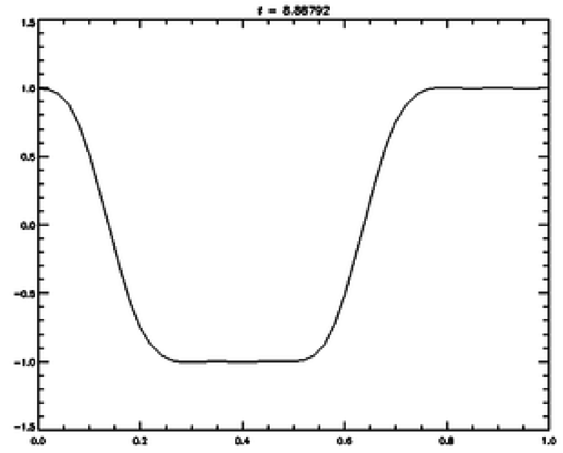
```
;; run.pro  ---  Time-stepping and plotting

for i1=0,100 do begin
  for i2 = 0,10 do begin
    rk, f,t,dt
  endfor
  plot, x, f, TITLE='!8t !3= '+strtrim(t,2)+'!X'
  wait, .1
endfor
```

15

```
end
```

Now run this code:

```
  idl>  .r start
  idl>  .r run
  idl>  .r run
```



Appendix B

Concurrent Versions System (CVS)

A Quick Introduction to CVS

Wolfgang Dobler[*]

Revision: 1.25 , Date: 2005/11/28 22:32:29

Contents

[*]Please send comments etc. to `Wolfgang.Dobler@ucalgary.ca`

1 What CVS does

CVS (Concurrent Versions System)

– is a system that lets groups of people work simultaneously on groups of files (for instance a numerical code, a LaTeXpaper, a set of HTML pages, etc.)

– allows you to retrieve older versions of (the important files in) a directory tree, identified by date, specific tags, ...

– forces you (to some extent) to write log messages for any changes you make to your code. These messages are recorded and with '*cvs log ...*' you can obtain a full annotated changelog history for a given file or module

– can also be used to keep your computing environment in sync (e.g. '*~/bin*', '*~/idl/lib*', '*~/tex/include*') across different computers

2 Nomenclature

Some specific terms you should know:

repository: The directory structure where CVS stores the files it manages, together with some administrative files

module: Essentially, a directory tree subject to version control. More formally, a module is a directory tree listed in '*CVSROOT/modules*', which can be accessed under the module name instead of the full path

tags, rtags: Labels attached to the files (possibly directories) and modules, allowing to identify them more easily

revision: A numerical or alpha-numerical tag identifying the version of a file

check in (commit) / check out: Write your modified version of a file/module to the repository (commit); retrieve the latest or a particularly specified version from the repository (check out)

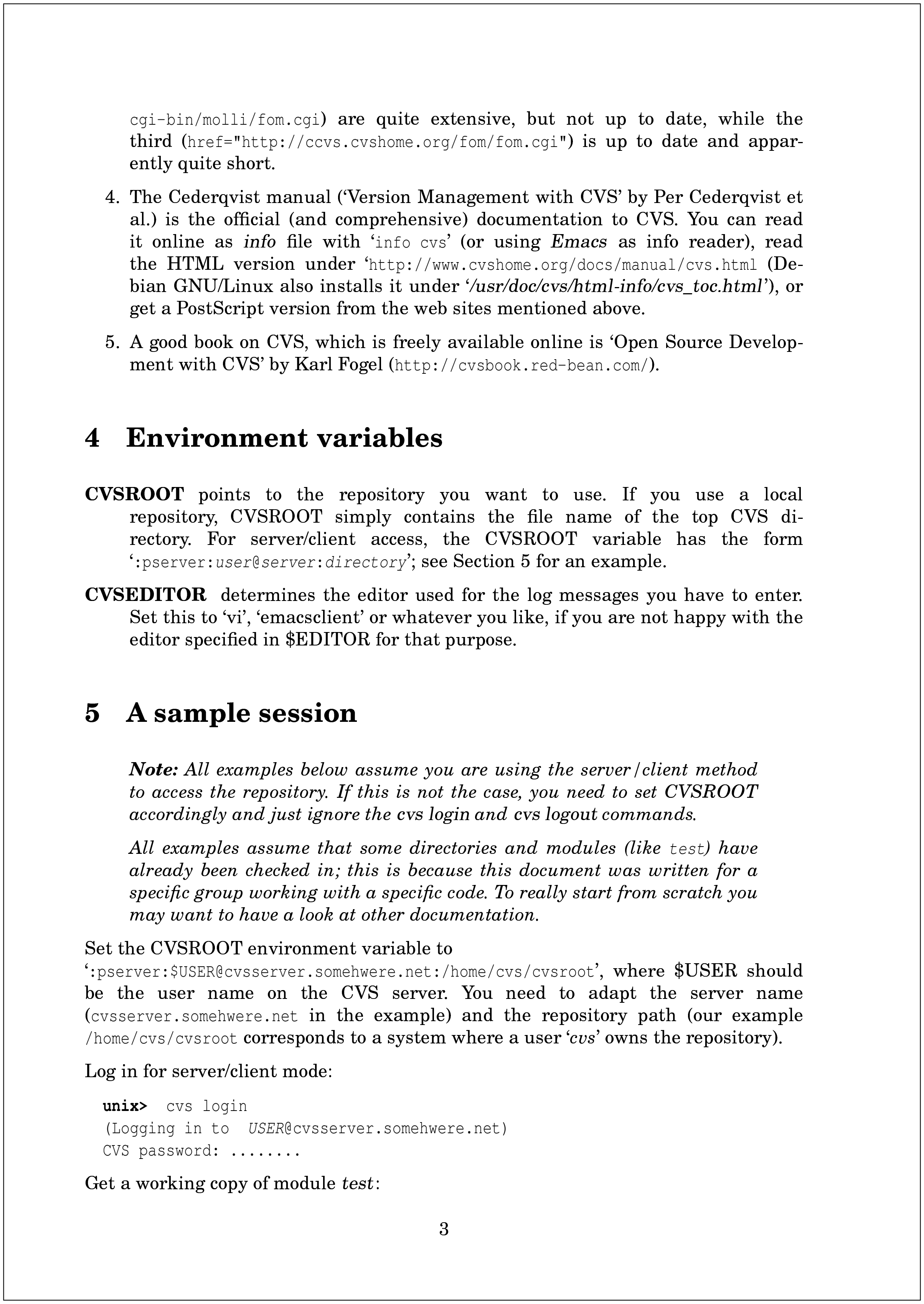
3 Getting help

There are several levels of information available for the CVS commands.

1. '`cvs -H` *command*' or '`cvs --help` *command*' gives an overview of *command*. A (brief) overview of the '`cvs`' command itself is obtained by '`cvs -H`'.

2. The CVS manpage ('`man cvs`') shows a brief overview over all CVS commands, followed by a detailed list of general options and a more detailed description of the individual commands.

3. `http://www.cvshome.org/` is the standard reference site for CVS. Apparently, three lists of frequently asked questions are available, two of which (`http://www.loria.fr/~molli/cvs/cvs-FAQ-1.4/cvsfaq0.html` and `http://www.loria.fr/`



cgi-bin/molli/fom.cgi) are quite extensive, but not up to date, while the third (href="http://ccvs.cvshome.org/fom/fom.cgi") is up to date and apparently quite short.

4. The Cederqvist manual ('Version Management with CVS' by Per Cederqvist et al.) is the official (and comprehensive) documentation to CVS. You can read it online as *info* file with '`info cvs`' (or using *Emacs* as info reader), read the HTML version under '`http://www.cvshome.org/docs/manual/cvs.html` (Debian GNU/Linux also installs it under '*/usr/doc/cvs/html-info/cvs_toc.html*'), or get a PostScript version from the web sites mentioned above.
5. A good book on CVS, which is freely available online is 'Open Source Development with CVS' by Karl Fogel (`http://cvsbook.red-bean.com/`).

4 Environment variables

CVSROOT points to the repository you want to use. If you use a local repository, CVSROOT simply contains the file name of the top CVS directory. For server/client access, the CVSROOT variable has the form '`:pserver:`*`user`*`@`*`server`*`:`*`directory`*'; see Section 5 for an example.

CVSEDITOR determines the editor used for the log messages you have to enter. Set this to 'vi', 'emacsclient' or whatever you like, if you are not happy with the editor specified in $EDITOR for that purpose.

5 A sample session

> ***Note:*** *All examples below assume you are using the server/client method to access the repository. If this is not the case, you need to set CVSROOT accordingly and just ignore the* **cvs login** *and* **cvs logout** *commands.*
>
> *All examples assume that some directories and modules (like* `test`*) have already been checked in; this is because this document was written for a specific group working with a specific code. To really start from scratch you may want to have a look at other documentation.*

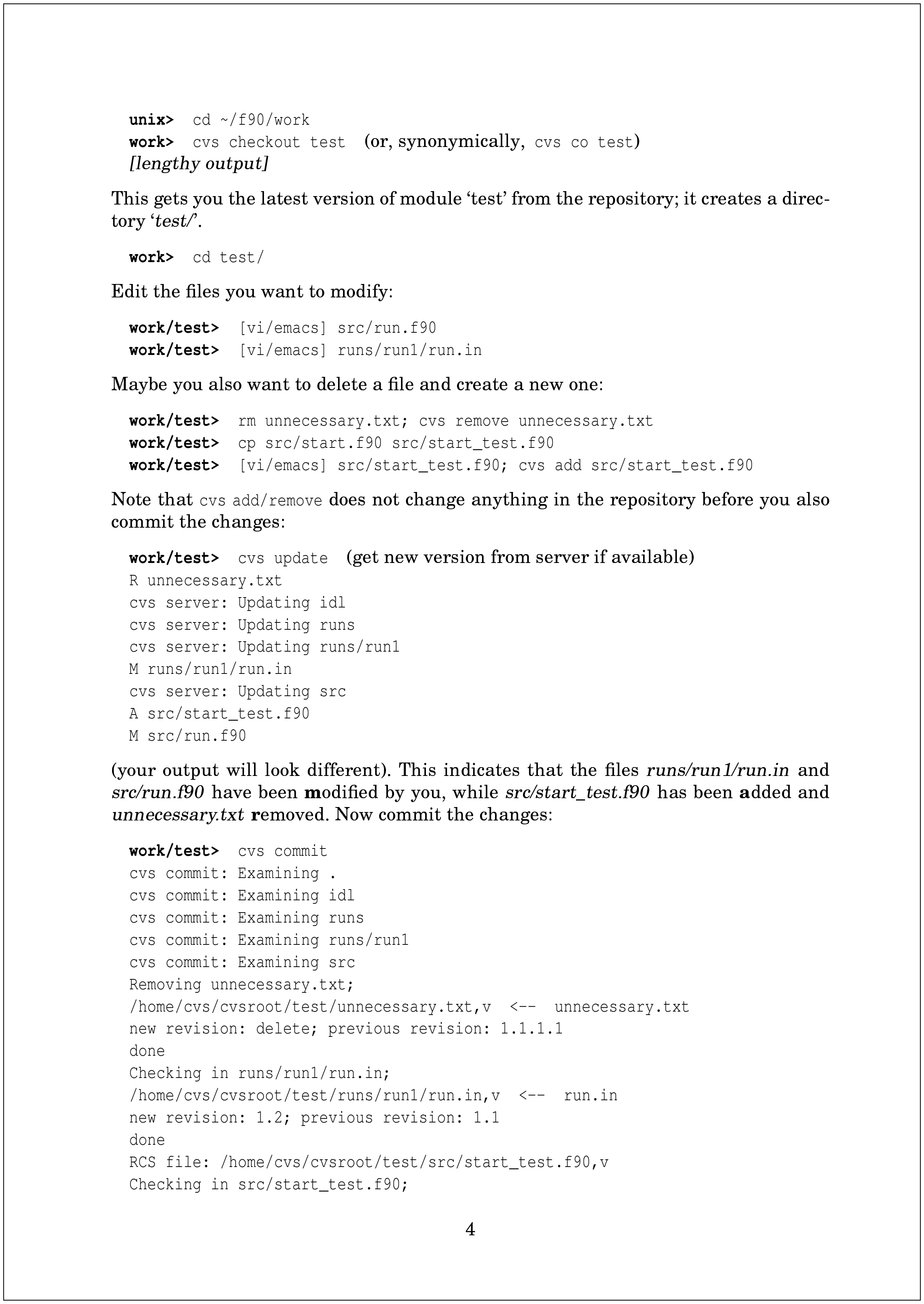
Set the CVSROOT environment variable to
'`:pserver:$USER@cvsserver.somehwere.net:/home/cvs/cvsroot`', where $USER should be the user name on the CVS server. You need to adapt the server name (`cvsserver.somehwere.net` in the example) and the repository path (our example `/home/cvs/cvsroot` corresponds to a system where a user '*cvs*' owns the repository).

Log in for server/client mode:

```
unix>  cvs login
(Logging in to  USER@cvsserver.somehwere.net)
CVS password: ........
```

Get a working copy of module *test*:

3



```
unix>  cd ~/f90/work
work>  cvs checkout test  (or, synonymically, cvs co test)
[lengthy output]
```

This gets you the latest version of module ‘test’ from the repository; it creates a directory ‘*test/*’.

```
work>  cd test/
```

Edit the files you want to modify:

```
work/test>  [vi/emacs] src/run.f90
work/test>  [vi/emacs] runs/run1/run.in
```

Maybe you also want to delete a file and create a new one:

```
work/test>  rm unnecessary.txt; cvs remove unnecessary.txt
work/test>  cp src/start.f90 src/start_test.f90
work/test>  [vi/emacs] src/start_test.f90; cvs add src/start_test.f90
```

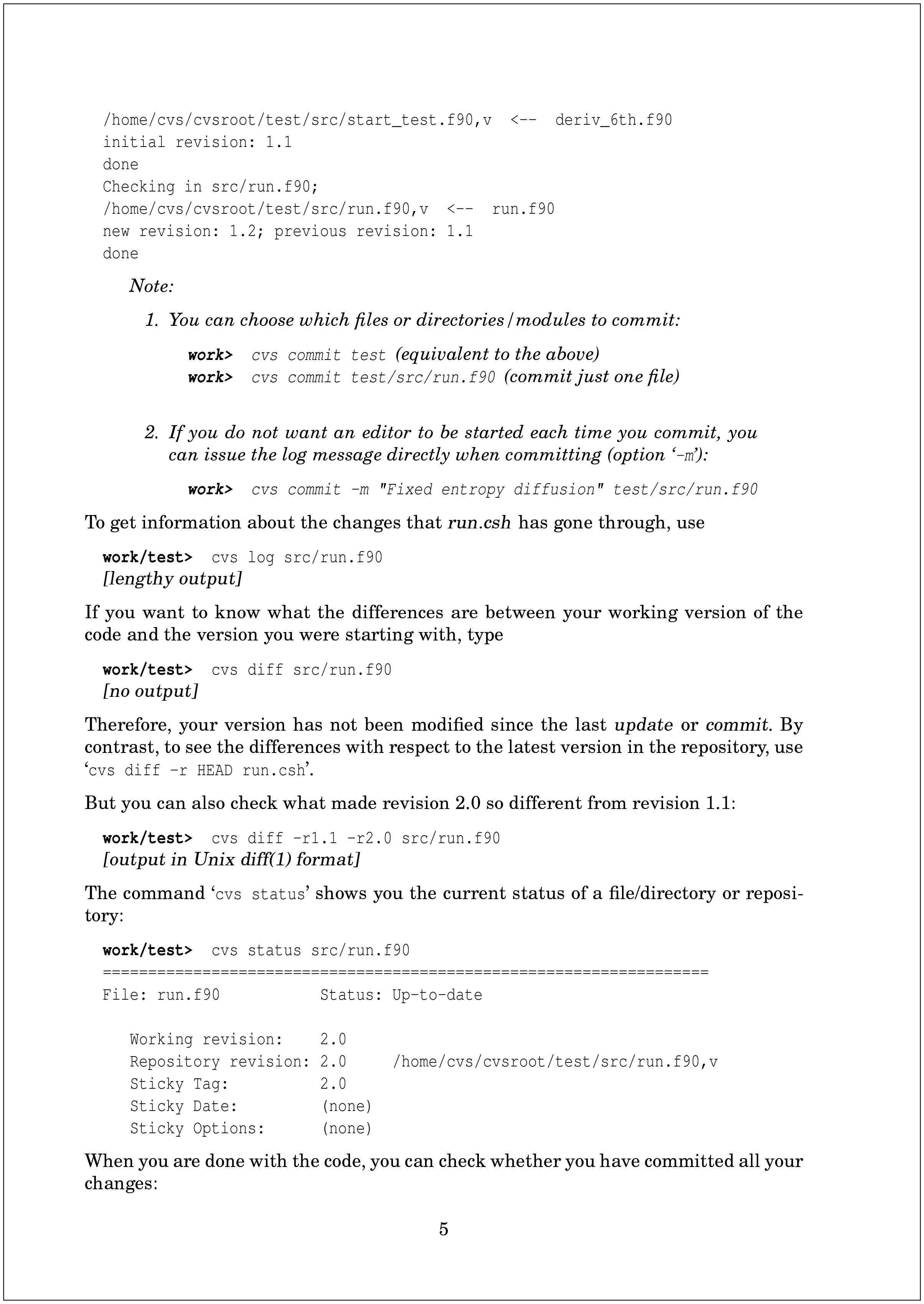
Note that `cvs add/remove` does not change anything in the repository before you also commit the changes:

```
work/test>  cvs update  (get new version from server if available)
R unnecessary.txt
cvs server: Updating idl
cvs server: Updating runs
cvs server: Updating runs/run1
M runs/run1/run.in
cvs server: Updating src
A src/start_test.f90
M src/run.f90
```

(your output will look different). This indicates that the files *runs/run1/run.in* and *src/run.f90* have been **m**odified by you, while *src/start_test.f90* has been **a**dded and *unnecessary.txt* **r**emoved. Now commit the changes:

```
work/test>  cvs commit
cvs commit: Examining .
cvs commit: Examining idl
cvs commit: Examining runs
cvs commit: Examining runs/run1
cvs commit: Examining src
Removing unnecessary.txt;
/home/cvs/cvsroot/test/unnecessary.txt,v  <--  unnecessary.txt
new revision: delete; previous revision: 1.1.1.1
done
Checking in runs/run1/run.in;
/home/cvs/cvsroot/test/runs/run1/run.in,v  <--  run.in
new revision: 1.2; previous revision: 1.1
done
RCS file: /home/cvs/cvsroot/test/src/start_test.f90,v
Checking in src/start_test.f90;
```

4



```
/home/cvs/cvsroot/test/src/start_test.f90,v  <--  deriv_6th.f90
initial revision: 1.1
done
Checking in src/run.f90;
/home/cvs/cvsroot/test/src/run.f90,v  <--  run.f90
new revision: 1.2; previous revision: 1.1
done
```

Note:

1. *You can choose which files or directories/modules to commit:*

   work> `cvs commit test` *(equivalent to the above)*
   work> `cvs commit test/src/run.f90` *(commit just one file)*

2. *If you do not want an editor to be started each time you commit, you can issue the log message directly when committing (option '`-m`'):*

   work> `cvs commit -m "Fixed entropy diffusion" test/src/run.f90`

To get information about the changes that *run.csh* has gone through, use

```
work/test>  cvs log src/run.f90
```
[lengthy output]

If you want to know what the differences are between your working version of the code and the version you were starting with, type

```
work/test>  cvs diff src/run.f90
```
[no output]

Therefore, your version has not been modified since the last *update* or *commit*. By contrast, to see the differences with respect to the latest version in the repository, use '`cvs diff -r HEAD run.csh`'.

But you can also check what made revision 2.0 so different from revision 1.1:

```
work/test>  cvs diff -r1.1 -r2.0 src/run.f90
```
[output in Unix diff(1) format]

The command '`cvs status`' shows you the current status of a file/directory or repository:

```
work/test>  cvs status src/run.f90
===================================================================
File: run.f90           Status: Up-to-date

   Working revision:    2.0
   Repository revision: 2.0     /home/cvs/cvsroot/test/src/run.f90,v
   Sticky Tag:          2.0
   Sticky Date:         (none)
   Sticky Options:      (none)
```

When you are done with the code, you can check whether you have committed all your changes:

5

```
work/test>  cd ..   (You must be immediately above
the directory you were working on)
work>  cvs release test
M start.csh
You have [1] altered files in this repository.
Are you sure you want to release directory 'test': n
** 'release' aborted by user choice.
```

In this example you had not, and entered 'n' to cancel the release. Now commit the modified file *start.csh* and release again:

```
work>  cvs commit -m "Set nwidth to 17" test/start.csh
work>  cvs release -d test  (Be careful when using the '-d' option!)
You have [0] altered files in this repository.
Are you sure you want to release (and delete) directory 'test': y
```

With the '-d' option, *release* removes the working copy of the module, provided you tell it to do so by answering 'y' to the prompt. *If you hurry at this point, you can lose data*.

At the end of your session, you can

```
work/test>  cvs logout
```

which will remove the entry for the given CVS server from the file '*~/.cvspass*'. This means that the next time you want to access the server again from the same machine, you will be asked for the CVS password again.

6 Tags and revision numbers

CVS identifies revisions with unique version numbers, like 1.1.1.4 or 2.15. Often it is much more convenient to refer to a given revision with a symbolic tag. You can attach a tag to your working copy with

```
work/test>  cvs rtag jets-hydro-5 test
```

Do not use rtag in the form 'cvs rtag hydro-5 .' — rtag needs the module name as last argument and will otherwise tag all the files you have under CVS control, affecting any other modules as well.

There is also a command 'cvs tag'. The bottom line is that you use *tag* to tag individual files, but *rtag* for the whole module.

A more detailed discussion of the differences between tag and rtag is given in one of the FAQs (http://www.loria.fr/~molli/fom-serve/cache/211.html): The end result of both commands is that a [tag], or symbolic name, is attached to a single revision in each of a collection of files. The differences lie in:

- *The collection of files they work on.*

  "rtag" works on the collection of files referred to by a "module" name as defined in the "modules" file, or a relative path within the Repository.

  "tag" works on files and directories specified on the command line within the user's working directory. (Default is '.')

> *Both commands recursively follow directory hierarchies within the named files and directories.*
>
> – *The revisions they choose to tag.*
>
> *"rtag" places a tag on the latest committed revision of each file on the branch specified by the '-r' option. By default it tags the Main Branch.*
>
> *"tag" places a tag on the BASE (i.e. last checked out, updated or committed) revision of each file found in the working directory. (The BASE revision of a file is the one stored in the ./CVS/Entries file.)*
>
> *[. . .]*

If you want to bring all your files up to revision 3.0 (including those that haven't changed), you might invoke

```
work/test>  cvs commit -r 3.0
```

This only works if none of the files in the module had a revision number higher than 3.0. It is probably a good idea to check with your collaborators before you decide to increase the major revision number (the first digit of the revision number).

7 Conflicts

If you want to *commit* a file (say, *run.csh*), but someone else has in the meantime committed a later version of it than the one you were working with, '*commit*' will speak very roughly to you:

```
unix>  cvs commit -m 'Removed a few module references'
cvs commit: Examining src
cvs commit: Up-to-date check failed for `src/run.f90'
cvs [commit aborted]: correct above errors first!
```

What you should do now is *update* the file '*src/run.f90*' (or the whole directory '*src*'):

```
unix>  cvs update src
cvs server: Updating src
RCS file: /home/cvs/cvsroot/test/src/run.f90,v
retrieving revision 2.0
retrieving revision 2.1
Merging differences between 2.0 and 2.1 into run.f90
M src/run.f90
```

If you are lucky — i. e. if the changes appeared in different files, or even if they are located in non-overlapping regions of the same file — the two versions are automatically merged and everything is OK. If you have doubts, take a look a the merged file. (If you are unlucky, you must manually resolve the conflict, see below.) Now *src/run.f90* contains both modifications together[1], and you can commit the merged file:

[1] If this is not what you wanted, you can reconstruct your version of *test/src/makefile* with '`cvs update -j ...`'

```
unix>  cvs commit -m "Made important changes and merged" src
Checking in run.f90;
/var/local/cvsroot/test/src/run.f90,v  <--  run.f90
new revision: 1.3; previous revision: 1.2
done
```

If you are *really* lucky, the merged code still compiles. . .

However, if you and your colleague have modified the same part of the code, the conflict can not be resolved automatically by merging, and you obtain a warning

```
unix>  cvs update
cvs server: Updating src
RCS file: /home/cvs/cvsroot/test/src/run.f90,v
retrieving revision 2.1
retrieving revision 2.2
Merging differences between 2.1 and 2.2 into run.f90
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in src/run.f90
C src/run.f90
```

Now you have to resolve the conflict manually by editing the file, fixing it and running '*cvs commit*.' The file *src/run.f90* looks like this:

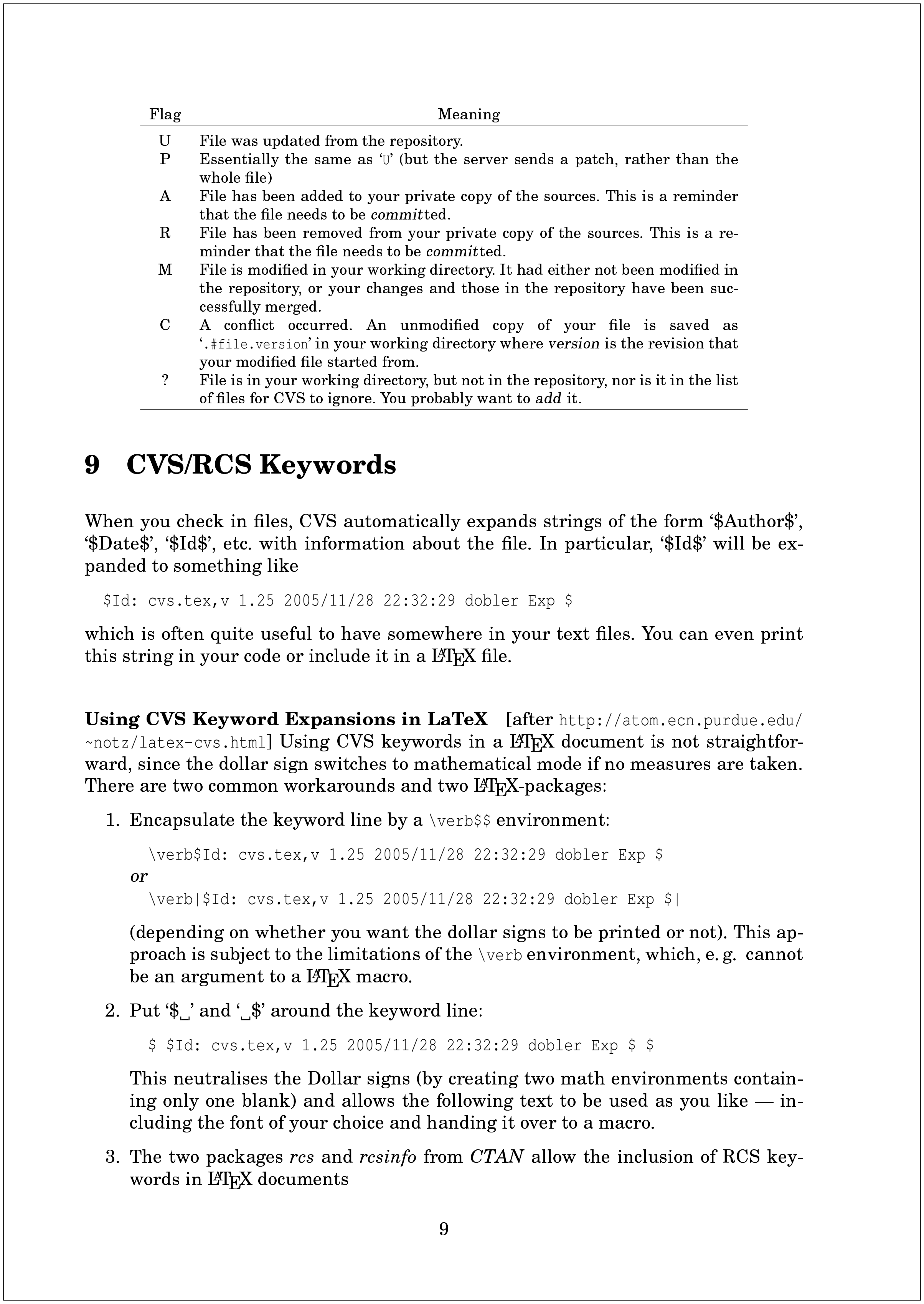
```
...
!
    use Mpicomm
<<<<<<< run.f90
!    use Cdata
!    use Deriv
=======
    use Cdata ! Really use it
    use Deriv
>>>>>>> 2.2
    use Sub
    use Timestep
...
```

The line between '<<<<<<<' and '=======' represents your changes of *run.csh*, while the part between '=======' and '>>>>>>>' has been committed in version 2.2 of the file by your swift colleague.

You can find your version of the file in the hidden file '`src/.#run.csh.2.1`'.

8 Flags issued by '`update`'

In the previous example, '*update*' flagged *test/src/makefile* with the capital letter '`M`' and *test/run.csh* with '`C`'. Here is a list of all the flags used by '*update*'.



Flag	Meaning
U	File was updated from the repository.
P	Essentially the same as '`U`' (but the server sends a patch, rather than the whole file)
A	File has been added to your private copy of the sources. This is a reminder that the file needs to be *commit*ted.
R	File has been removed from your private copy of the sources. This is a reminder that the file needs to be *commit*ted.
M	File is modified in your working directory. It had either not been modified in the repository, or your changes and those in the repository have been successfully merged.
C	A conflict occurred. An unmodified copy of your file is saved as '`.#file.version`' in your working directory where *version* is the revision that your modified file started from.
?	File is in your working directory, but not in the repository, nor is it in the list of files for CVS to ignore. You probably want to *add* it.

9 CVS/RCS Keywords

When you check in files, CVS automatically expands strings of the form '$Author$', '$Date$', 'Id', etc. with information about the file. In particular, 'Id' will be expanded to something like

```
$Id: cvs.tex,v 1.25 2005/11/28 22:32:29 dobler Exp $
```

which is often quite useful to have somewhere in your text files. You can even print this string in your code or include it in a LaTeX file.

Using CVS Keyword Expansions in LaTeX [after `http://atom.ecn.purdue.edu/~notz/latex-cvs.html`] Using CVS keywords in a LaTeX document is not straightforward, since the dollar sign switches to mathematical mode if no measures are taken. There are two common workarounds and two LaTeX-packages:

1. Encapsulate the keyword line by a `\verb$$` environment:

   ```
   \verb$Id: cvs.tex,v 1.25 2005/11/28 22:32:29 dobler Exp $
   ```
   or
   ```
   \verb|$Id: cvs.tex,v 1.25 2005/11/28 22:32:29 dobler Exp $|
   ```

   (depending on whether you want the dollar signs to be printed or not). This approach is subject to the limitations of the `\verb` environment, which, e. g. cannot be an argument to a LaTeX macro.

2. Put '$␣' and '␣$' around the keyword line:

   ```
   $ $Id: cvs.tex,v 1.25 2005/11/28 22:32:29 dobler Exp $ $
   ```

   This neutralises the Dollar signs (by creating two math environments containing only one blank) and allows the following text to be used as you like — including the font of your choice and handing it over to a macro.

3. The two packages *rcs* and *rcsinfo* from *CTAN* allow the inclusion of RCS keywords in LaTeX documents

9

10 Creating a repository

If you want to create a new repository, you use *cvs init*:

```
unix>  cvs -d ~/cvsroot init
```

— this creates a repository in your home directory. To access this repository, you should set CVSROOT accordingly:

```
unix>  setenv CVSROOT ~/tmp/cvsroot
```

11 Nota Bene

- If you are uncertain about what a given command might do, run it with '`cvs -n <command>`' first.

  ```
  unix>  cvs -n update   (Does not change any file)
  unix>  cvs -n commit   (Does not change any file)
  ```

  The '`-n`' flag tells CVS to do a 'dry run' of the command and not change any files.
- When creating new directories, remember to explicitly *add* them. The commands '*update*' and '*release*' will show the new directory flagged with a question mark, but nobody will keep you from finally deleting your working copy and thus getting rid of all the files created in your new directory.
- If you rename files, you must *remove* the old file and *add* the new one:

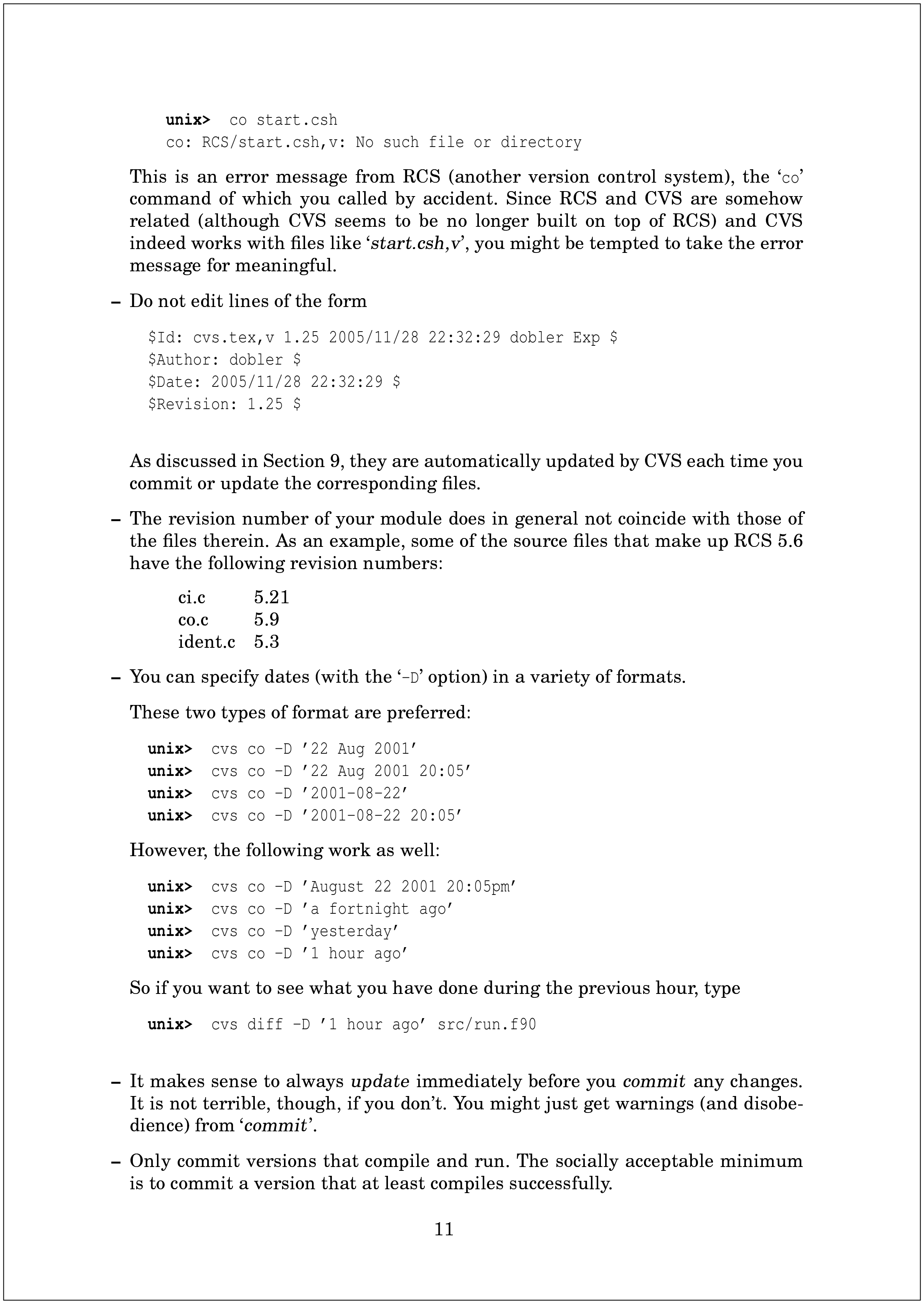
  ```
  unix>  mv old new
  unix>  cvs add new
  unix>  cvs remove old
  ```

  Remember that for these changes to take place in the repository, you must still *commit* them.

  > *The new file will know nothing about the modification history prior to this operation. If you want to rename a file, retaining the full history, then you need direct access to the repository.*[2] *You* copy *the file from 'old,v' to 'new,v'. Then you do* cvs remove *on the old version. This ensures that* cvs update *removes the old version.*

- Before you *release* a modified working copy, you must *commit* it — otherwise, you get warnings about modified files (marked with '`M`' in front of the file name) and should then definitely not continue the *release*, unless you want to lose the changes you have made.
- Keep in mind that CVS simply ignores symbolic links. However, there should be no need to link the *src/* directory and files any more, since all the supposed advantages of this technique are features of CVS.
- Do not forget the leading `cvs` for the CVS commands. Otherwise you might end up with cryptic error messages like in the following example

[2] This trick is due to Karl Fogel's book mentioned in Section 3.



```
unix>  co start.csh
co: RCS/start.csh,v: No such file or directory
```

This is an error message from RCS (another version control system), the ‘`co`’ command of which you called by accident. Since RCS and CVS are somehow related (although CVS seems to be no longer built on top of RCS) and CVS indeed works with files like ‘*start.csh,v*’, you might be tempted to take the error message for meaningful.

– Do not edit lines of the form

```
$Id: cvs.tex,v 1.25 2005/11/28 22:32:29 dobler Exp $
$Author: dobler $
$Date: 2005/11/28 22:32:29 $
$Revision: 1.25 $
```

As discussed in Section 9, they are automatically updated by CVS each time you commit or update the corresponding files.

– The revision number of your module does in general not coincide with those of the files therein. As an example, some of the source files that make up RCS 5.6 have the following revision numbers:

ci.c	5.21
co.c	5.9
ident.c	5.3

– You can specify dates (with the ‘`-D`’ option) in a variety of formats.

These two types of format are preferred:

```
unix>  cvs co -D '22 Aug 2001'
unix>  cvs co -D '22 Aug 2001 20:05'
unix>  cvs co -D '2001-08-22'
unix>  cvs co -D '2001-08-22 20:05'
```

However, the following work as well:

```
unix>  cvs co -D 'August 22 2001 20:05pm'
unix>  cvs co -D 'a fortnight ago'
unix>  cvs co -D 'yesterday'
unix>  cvs co -D '1 hour ago'
```

So if you want to see what you have done during the previous hour, type

```
unix>  cvs diff -D '1 hour ago' src/run.f90
```

– It makes sense to always *update* immediately before you *commit* any changes. It is not terrible, though, if you don't. You might just get warnings (and disobedience) from ‘*commit*’.

– Only commit versions that compile and run. The socially acceptable minimum is to commit a version that at least compiles successfully.

11

If someone else has made changes simultaneously and your *update*d code doesn't compile any more, either fix this problem before *commit*ting, or create a separate branch for your version of the code.

– Be minimalistic about the files you keep in the working directory. Remember that there is no need to retain files (say, IDL programs) that were once useful and might, perhaps, possibly, under special circumstances be needed again in the distant future.

  In particular, do not keep old versions of files in your working directory. For example, if you modify version 1.5 of '*run.f90*' in an experimental way, just change it; if you then need the original version, retrieve it with '`cvs co -r 1.5 test run.f90`'

– To put a new project under CVS control, go to its top directory and `import` it:

```
unix>  cd ~/f90/projects/solitons
solitons>  cvs import -m "Import of soliton code v. 0.05" \
                    f90/solitons ncl-mhd Solitons_0-05
```

  Here '*f90/solitons*' is the name of the module in the repository, '*ncl-mhd*' is a "vendor tag" (unimportant in our case) and '*Solitons_0-05*'[3] is a *tag* attached to this imported revision, allowing to refer to it later.

  The files are imported with revision number 1.1.1.1, which is specific for the vendor branch. At the same time, the files have revision number 1.1 and only this is used for the version on the trunk. So, the first time you modify an imported file, its revision number gets increased to 1.2, the next time to 1.3, and so on.

  Importing a directory does not make it a checked out version of it (i. e. the directory where you called *cvs import* will not contain a '*CVS*' subdirectory. One simple way of turning your '*solitons*' directory into a CVS-controlled one after the import is to do

```
projects/solitons>  cd ..
projects>  mv solitons solitons-deleteme-eventually
projects>  cvs co -d solitons f90/solitons
```

  You can keep the original directory around for some time in case you forgot to check in some files, but from now on you will work with the checked-out version.

12 My top ten CVS commands

Here are my top ten commands, i. e. (ordered by frequency of use) the main CVS commands in my repertoire:

1. `cvs -qn update`
2. `cvs update -d`

[3] '*Solitons-0.05*' would look much nicer (in my opinion), but tags must not contain any of the characters '`$,.:;`'

3. `cvs commit`
4. `cvs diff` *<file>* *(difference to original version)*
   or
   `cvs diff -r HEAD` *<file>* *(difference to latest repository version).*

   A variant is
   `cvs diff -u -rHEAD <file> | a2ps -Eudiff --prologue=diff -Pdisplay`
   to pretty-print difference between local and latest version
5. `cvs checkout` *<module>*
6. `cvs add` *<file>*
7. `cvs checkout -d` *<directory>* *<module>*
   (check out module into specific directory)
8. `cvs log` *<file>* `| less`
9. `cvs annotate` *<file>*
10. `cvs import` *<repository>* *<vendor-tag>* *<release-tag>*

Note: The `-q` flag ('be somehow quiet') is so useful for larger projects that you may want to put the line

```
cvs -q
```

into '*~/.cvsrc*' to have it always set.

13 Other user interfaces

1. **VC (minor) mode:** If Emacs is your operating system of choice, you can use *VC* mode as a front end to *CVS*. Normally, Emacs (at least versions ≥ 21) automatically detects which files are under *CVS* and adds a string like "CVS:1.15" to your mode line.

   Useful key strokes are

   `C-x C-q` and **`C-x v v`:** *vc-next-action*, do *cvs commit* or *cvs update*, whichever makes more sense

   `C-x v i`: *vc-register*, i. e. *cvs add*

   `C-x v =`: *vc-diff*, does *cvs diff* on buffer file

   `C-x v l`: *vc-print-log*, shows output from *cvs log* in separate buffer

   Less essential, but useful key strokes are

   `C-x v u`: *vc-revert-buffer*, reverts to the version buffer file was based on (i. e. undoes all changes)

C-x v ~: *vc-version-other-window*, loads a specific version into another buffer (allows for *ediff*)

C-x v a: *vc-update-change-log*, extracts log information and writes or adds it to a ChangeLog file

C-x v h: *vc-insert-headers*, inserts 'Id' as a comment

C-x v g: *vc-annotate*, do *cvs annotate* with colors indicating different versions. . .

2. **pcl-cvs:** There is another *CVS* front end for Emacs, called *pcl-cvs*. I do not use it and think that *VC* mode is the way to go, but if you are interested in *pcl-cvs*, here is a short description.

   To get started, just type '`M-x cvs-update RET`' and enter the name of a directory where you have a checked-out CVS module:

```
PCL-CVS release 1.05 from CVS release $Name:  $.
Copyright (C) 1992, 1993 Per Cederqvist
Pcl-cvs comes with absolutely no warranty; for details consult the manual.
This is free software, and you are welcome to redistribute it under certain
conditions; again, consult the TeXinfo manual for details.

In directory /home/dobler/f90/mhdf/work/test:
  Updated     run.csh

In directory /home/dobler/f90/mhdf/work/test/src:
  Modified ci run.f90
  Updated     mhd1.f90
  Unknown     mhd2.f90
---------- End -----
```

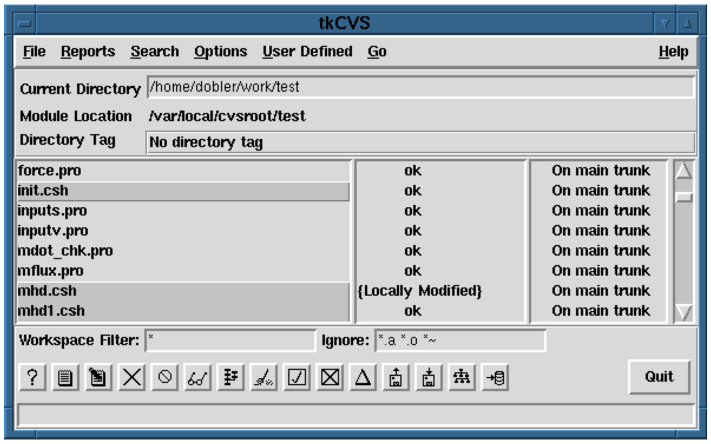
   Now you can do lots of fancy things with a few key strokes. Look at the *info* documentation for *pcl-cvs* for details.

   One of the good points about *pcl-cvs* is that you have the marvelous tools '*ediff*' and '*emerge*' at hand (and even automatically invoked) if you need them.

   On my *Debian* system, I had problems getting pcl-cvs to run. See `http://www.kis.uni-freiburg.de/~dobler/docs` for how I solved them.

   Warning to Vi users: There is a reported case of a Vi user who converted to Emacs just to be able to use *pcl-cvs*; so better watch out.

3. **tkcvs:** If you prefer graphical user interfaces (the ones where you have to mechanically repeat the same sequence of 20 mouse clicks all of the time), try *tkcvs*.



Current Directory /home/dobler/work/test
Module Location /var/local/cvsroot/test
Directory Tag No directory tag

force.pro	ok	On main trunk
init.csh	ok	On main trunk
inputs.pro	ok	On main trunk
inputv.pro	ok	On main trunk
mdot_chk.pro	ok	On main trunk
mflux.pro	ok	On main trunk
mhd.csh	{Locally Modified}	On main trunk
mhd1.csh	ok	On main trunk

Workspace Filter: * Ignore: *.a *.o *~

You will find it via the *cvshome* web page; it should be possible to install it on any Unix machine running a recent version of *tcl/tk*.

4. There are also graphical clients for Macintosh (*MacCVSClient*, see `http://www.cvshome.org/cyclic/cvs/mac.html`) and Windows (*WinCVS*, see `http://www.cvshome.org/cyclic/cvs/windows.html`)

Appendix

A Overview over CVS commands

The following overview over the basic CVS commands has been adapted from the 'CVS tutorial' (`http://www.loria.fr/~molli/cvs/cvs-tut/cvs_tutorial_toc.html`) by Gray Whatson.

> Most of the below commands should be executing while in the directory you checked out. If you did a '`cvs checkout malloc`' then you should be in the malloc sub-directory to execute most of these commands. '`cvs release`' is different and must be executed from the directory above.
>
> **`cvs add`** and **`cvs remove`**
> It can be that the changes you want to make involve a completely new file, or removing an existing one. The commands to use here are:
>
> `cvs add` *'filename'*
> `cvs remove` *'filename'*
>
> You still have to do a '*commit*' after these commands to make the additions and removes actually take affect. You may make any number of new files in your copy of the repository, but they will not be committed to the central copy unless you do a '`cvs add`'.
>
> CVS remove does not actually remove the files from the repository. It only removes them from the "current list" and puts the files in the CVS Attic. When another person checks out the module in the future they will not get the files that were removed. But if you ask for older versions that had the file before it was removed, the file will be checked out of the Attic.
>
> **`cvs admin`**
> This is the CVS interface to assorted administrative facilities. Some of them have questionable usefulness for CVS but exist for historical purposes. Some of the questionable options are likely to disappear in the future. This command *does* work recursively, so extreme care should be used.
>
> **`cvs annotate`**
> Gives you an annotated listing of the current version of a file, containing for each line information about in which version, by whom and when it was written. It does *not* contain information about deleted or modified lines (to get this, use '`cvs diff`' on the two versions you are interested in).

```
unix>  cvs annotate start.csh
Annotations for start.csh
***************
1.1 (brandenb 22-Apr-99): ! src/start.x
1.2 (nbmvr    10-Jul-99):
```

```
1.2 (nbmvr    10-Jul-99): -8.,8.,               :zmin,zmax
1.2 (nbmvr    12-Jul-99): .05,2.,1.,40060000., :rin,rqu,xboxmax,rLL
1.1 (brandenb 22-Apr-99): -.25,1.5,.1499,8,    :r1,r2,height,nwidth
1.1 (brandenb 22-Apr-99): 1.,0,0,0,            :mu0,B0,Bphi0,eps_quadru
1.1 (brandenb 22-Apr-99): 1.666667,.1,0.,      :gamma,beta,HH0
1.1 (brandenb 22-Apr-99): 0,0,                 :nsmooth,nsmoothrun
1.1 (brandenb 22-Apr-99): 1.,0.1,              :frac1,d1mask
1.3 (dobler   12-Jul-99): 1,2.,0               :isymm,scale,iffree
1.1 (brandenb 22-Apr-99): 0.0000001,           :ampl
1.1 (brandenb 22-Apr-99): EOF
1.1 (brandenb 22-Apr-99):
1.1 (brandenb 22-Apr-99): rm -f tmp/n.dat
1.1 (brandenb 22-Apr-99): rm -f t*.dat
1.1 (brandenb 22-Apr-99):
1.1 (brandenb 22-Apr-99): #
1.3 (dobler   12-Jul-99): #  iffree  --  initialise B force-free
```

cvs checkout (or cvs co)
To make a local copy of a module's files from the repository execute '`cvs checkout module`' where module is an entry in your modules file (see below). This will create a sub-directory module and check-out the files from the repository into the sub-directory for you to work on.

cvs commit
When you think your files are ready to be merged back into the repository for the rest of your developers to see, execute '`cvs commit`'. You will be put in an editor to make a message that describes the changes that you have made (for future reference). Your changes will then be added to the central copy.

When you do a '*commit*', if you haven't updated to the most recent version of the files, CVS tells you this; then you have to first *update*, resolve any possible clashes, and then redo the *commit*.

cvs diff
To see the differences between your version of the files, and the version in the repository you started from, do:

`cvs diff` *'filename(s)'*

If you want to compare to the latest version in the repository, use

`cvs diff -r HEAD` *'filename(s)'*

cvs history
To find out information about your CVS repositories use the '`cvs history`' command. By default '*history*' will show you all the entries that correspond to you. Use the '`-a`' option to show information about everyone.

cvs history -a -o
shows you (a)ll the checked (o)ut modules

cvs history -a -T

reports (a)ll the r(T)ags for the modules

cvs history -a -e
reports (a)ll the information about (e)verything

cvs import
Use 'import' to incorporate an entire source distribution from an outside source (e.g., a source vendor) into your source repository directory. You can use this command both for initial creation of a repository, and for wholesale updates to the module from the outside source. *Note Tracking sources::, for a discussion on this subject

cvs init
Create a CVS repository if it doesn't exist.

cvs log
To see the commit messages for files, and who made them, use:

`cvs log` *'filename(s)'*

cvs login, logout
Connect to, and disconnect from, the CVS server when using the *server/client* mode of accessing the repository (which we do).

cvs rdiff
Create 'patch' format diffs between releases

Builds a Larry Wall format patch(1) file between two releases, that can be fed directly into the 'patch' program to bring an old release up-to-date with the new release. (This is one of the few CVS commands that operates directly from the repository, and doesn't require a prior checkout.) The diff output is sent to the standard output device.

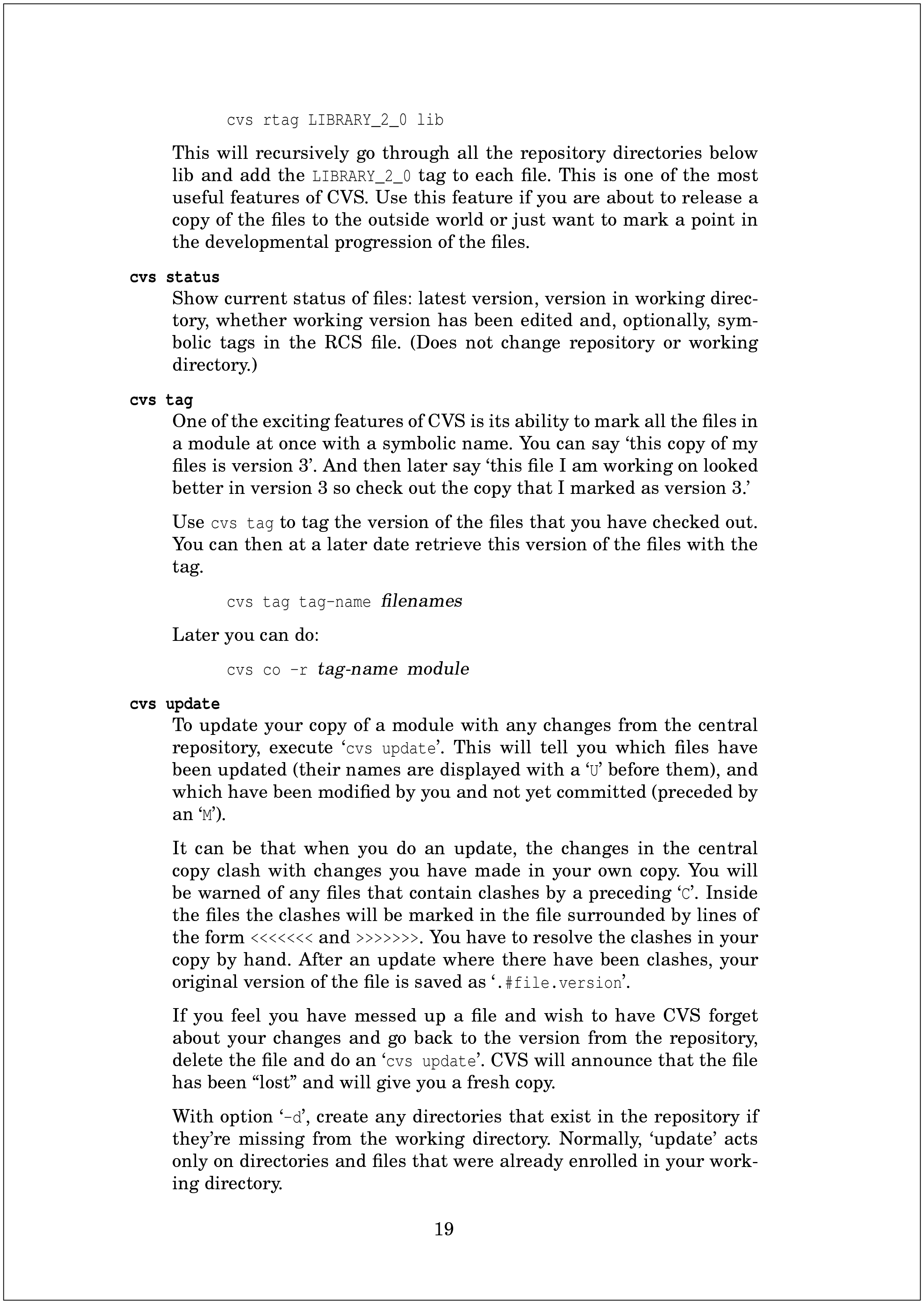
cvs release
When you are done with your local copy of the files for the time being and want to remove your local copy use '`cvs release module`'. This must be done in the directory above the module sub-directory you which to release. It safely cancels the effects of '`cvs checkout`'. Usually you should do a commit first.

If you wish to have CVS also remove the module sub-directory and your local copy of the files then you do '`cvs release -d module`'.

NOTE: Take your time here. CVS will inform you of files that may have changed or it does not know about (watch for the '`?`' lines) and then will ask you to confirm this action. Make sure you want to do this.

cvs remove
See *cvs add.*

cvs rtag
Like '*tag*', '*rtag*' marks the current versions of files but it does not work on your local copies but on the files in the repository. To tag all my libraries with a version name I can do:



```
cvs rtag LIBRARY_2_0 lib
```

This will recursively go through all the repository directories below lib and add the `LIBRARY_2_0` tag to each file. This is one of the most useful features of CVS. Use this feature if you are about to release a copy of the files to the outside world or just want to mark a point in the developmental progression of the files.

cvs status

Show current status of files: latest version, version in working directory, whether working version has been edited and, optionally, symbolic tags in the RCS file. (Does not change repository or working directory.)

cvs tag

One of the exciting features of CVS is its ability to mark all the files in a module at once with a symbolic name. You can say ‘this copy of my files is version 3’. And then later say ‘this file I am working on looked better in version 3 so check out the copy that I marked as version 3.’

Use `cvs tag` to tag the version of the files that you have checked out. You can then at a later date retrieve this version of the files with the tag.

`cvs tag tag-name` *filenames*

Later you can do:

`cvs co -r` *tag-name module*

cvs update

To update your copy of a module with any changes from the central repository, execute ‘`cvs update`’. This will tell you which files have been updated (their names are displayed with a ‘`U`’ before them), and which have been modified by you and not yet committed (preceded by an ‘`M`’).

It can be that when you do an update, the changes in the central copy clash with changes you have made in your own copy. You will be warned of any files that contain clashes by a preceding ‘`C`’. Inside the files the clashes will be marked in the file surrounded by lines of the form <<<<<<< and >>>>>>>. You have to resolve the clashes in your copy by hand. After an update where there have been clashes, your original version of the file is saved as ‘`.#file.version`’.

If you feel you have messed up a file and wish to have CVS forget about your changes and go back to the version from the repository, delete the file and do an ‘`cvs update`’. CVS will announce that the file has been “lost” and will give you a fresh copy.

With option ‘`-d`’, create any directories that exist in the repository if they’re missing from the working directory. Normally, ‘update’ acts only on directories and files that were already enrolled in your working directory.

19

cvs edit,editors,watch,watchers,unedit
These are commands that are irrelevant for us.

B Branches

B.1 Accessing branches

CVS allows different branches of one module to be worked on simultaneously. You can branch from an earlier version, work on that branch and finally merge your changes into the latest revision on the main branch.

To check out the branch labelled '*S-const-branch*' of module '*test*', type

```
work/test>  cvs update -r S-const-branch
```

(or '`cvs co -r S-const-branch`' if you do not have a working copy).

If you now commit changes, they will be saved on the branch '*S-const-branch*':

```
work/test>  cvs commit
[...]
work/test>  cvs status
===================================================================
File: start.csh          Status: Up-to-date

   Working revision:    1.32.2.3
   Repository revision: 1.32.2.3     /var/local/cvsroot/test/start.csh,v
   Sticky Tag:          S-const-branch (branch: 1.32.2)
[...]
```

The version number 1.32.2 is the number of the branch that was split off revision 1.32. Note that the branch tags stick to the branch (i. e. checking out the version with the tag '*S-const-branch*' will always give you the latest version on that branch), while revision tags are tied to one revision (like e. g. 1.32), although you can update them if you like.

'`cvs log`' lists you the tags, including branch tags:

```
work/test>  cvs log start.csh
RCS file: /var/local/cvsroot/test/start.csh,v
Working file: start.csh
head: 1.35
branch:
locks: strict
access list:
symbolic names:
        S-const-branch: 1.32.0.2
        pre-S-const-branch: 1.32
keyword substitution: kv
total revisions: 38;    selected revisions: 38
```

```
description:
----------------------------
revision 1.35
[ ... ]
----------------------------
revision 1.32
branches:  1.32.2;
[ ... ]
```

B.2 Creating branches

[from the FAQ]:

Suggested technique:

1. Attach a non-branch tag to all the revisions you want to branch from (i. e. the branch point revisions).
2. When you decide you really need a branch, attach a branch tag to the same revisions marked by the non-branch tag.
3. "Checkout" or "update" your working directory onto the branch.

Schematically, this means

(Write information about tags-to-come to Tags.list and commit)

```
unix>  cvs rtag <branch_point_tag> <module>
unix>  cvs rtag -b -r <branch_point_tag> <branch_tag> <module>
unix>  cvs checkout -r <branch_tag> <module>
```

The first step refers to the case where you are keeping a list of tags in a file '*Tags.list*'; you should update this file before you branch, so the information about the branch points is up to date on both trunk and branch.

C Tips, tricks and troubleshooting

C.1 User level tips and tricks

How can I *checkout* a directory without getting all its subdirectories? Use the '`-l`' flag of *checkout* (or *update*) to avoid recursion through the directory tree:

```
unix>  cvs co -l -d runs pencil-runs
```

To only get a sparse tree, say `runs/forced/halo1/`, you will have to apply this technique sequentially:

```
unix>  cvs co -l -d runs pencil-runs
unix>  cd runs;   cvs up -dl forced
unix>  cd forced; cvs up -dl halo1
```

The combination of '`-l`' and '`-d`' creates subdirectories without recursing.

My cvsroot has changed (new server name, . . .) – how do I update my checked out copies?

For each copy, *cd* to the top directory, then do

```
  unix>  oldroot=':pserver:USER\@OLD.HOST.DOM:/OLD/PATH'
  unix>  newroot=':pserver:USER\@NEW.HOST.DOM:/NEW/PATH'
  unix>  find . -path '*CVS/Root' \
          | xargs fgrep -l "${oldroot/\\\\@/@}" \
          | xargs perl -i.bak -pe "s{$oldroot}{$newroot}"
```

where (you guessed it) you replace all uppercase names and paths with real stuff. If you run this once, it creates a backup ‘*Root.bak*’ of each ‘*Root*’ file it adapts. When running a second time, however, the first backup will get overwritten.

C.2 Administration

C.2.1 Problems with the CVS pserver

Here is a checklist that proved useful.

1. Have you set up your repository correctly?

   `cvs -d /home/User/CVS init`   *(or wherever the repository should go)*

   You *do* need the ‘-d . . . ’, since otherwise CVS takes the value from CVSROOT — which points to a directory that is not yet set up for CVS.

If you try to `cvs login`, but get no connection:

```
cvs [login aborted]: connect to ...:2401 failed: Connection refused
```

2. (From the Cederqvist manual)
   Try

   ```
   telnet servername 2401
   ```

   After connecting, send any text (for example "foo" followed by return). If CVS is working correctly, it will respond with

   ```
   cvs [pserver aborted]: bad auth protocol start: foo
   ```

3. Does your system know about the *service* cvs? If ‘*/etc/inetd.conf*’ operates with service names instead of port numbers (i. e. if the first entry of each inetd line is a name, rather than a number) your cvs entry there,

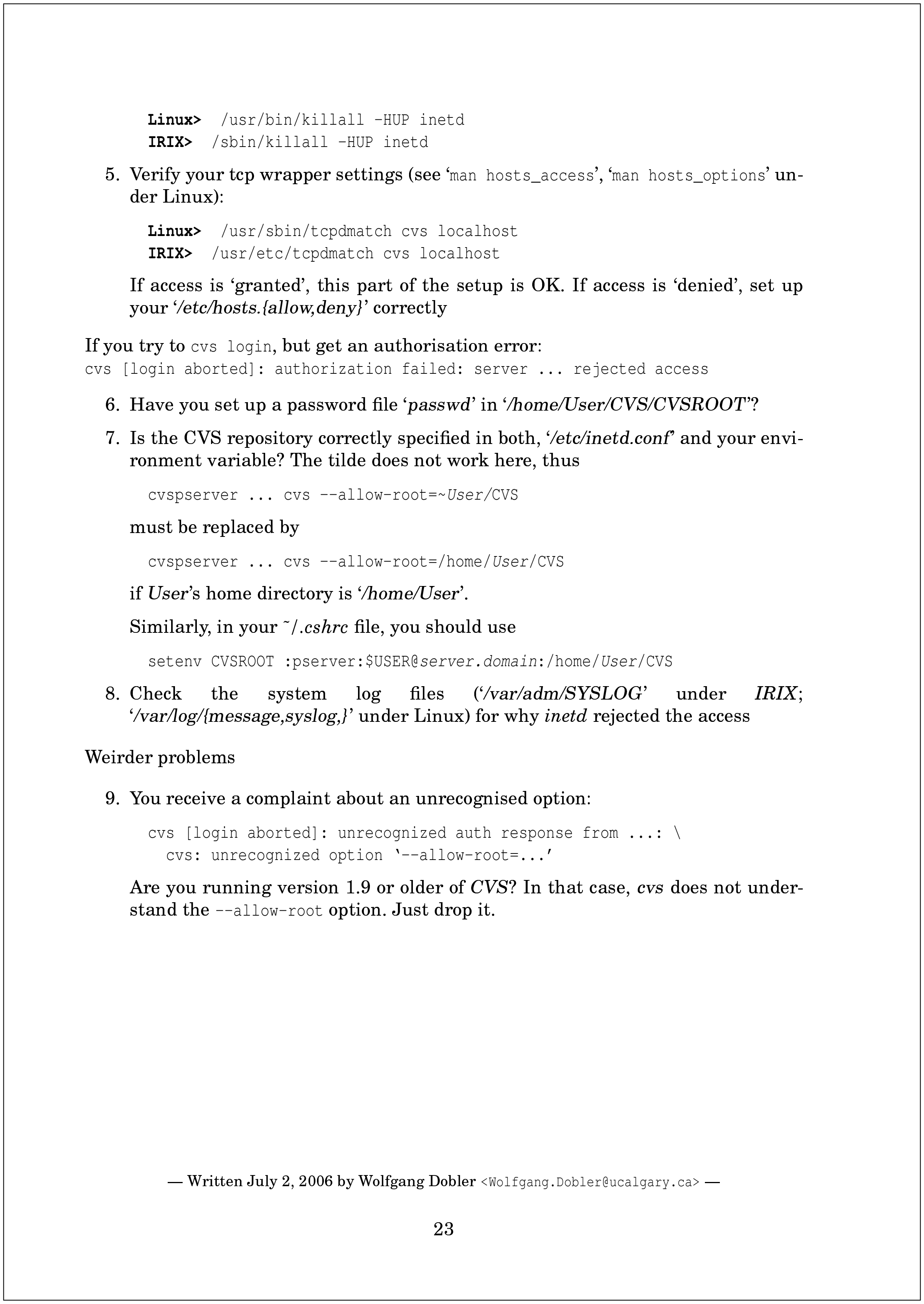
   ```
   cvspserver stream tcp nowait root \
   /usr/local/bin/cvs cvs --allow-root=/home/User/CVS pserver
   ```

   — then ‘*/etc/services*’ must define the service *cvspserver*:

   ```
   cvspserver      2401/tcp
   ```

   to tell inetd to start cvs when there is a request on port 2401.

4. Have you restarted *inetd* after changing ‘*/etc/{inetd.conf,services}*’?



```
Linux>  /usr/bin/killall -HUP inetd
IRIX>  /sbin/killall -HUP inetd
```

5. Verify your tcp wrapper settings (see '`man hosts_access`', '`man hosts_options`' under Linux):

   ```
   Linux>  /usr/sbin/tcpdmatch cvs localhost
   IRIX>  /usr/etc/tcpdmatch cvs localhost
   ```

   If access is 'granted', this part of the setup is OK. If access is 'denied', set up your '*/etc/hosts.{allow,deny}*' correctly

If you try to `cvs login`, but get an authorisation error:

```
cvs [login aborted]: authorization failed: server ... rejected access
```

6. Have you set up a password file '*passwd*' in '*/home/User/CVS/CVSROOT*'?
7. Is the CVS repository correctly specified in both, '*/etc/inetd.conf*' and your environment variable? The tilde does not work here, thus

   ```
   cvspserver ... cvs --allow-root=~User/CVS
   ```

   must be replaced by

   ```
   cvspserver ... cvs --allow-root=/home/User/CVS
   ```

   if *User*'s home directory is '*/home/User*'.

   Similarly, in your ~/*.cshrc* file, you should use

   ```
   setenv CVSROOT :pserver:$USER@server.domain:/home/User/CVS
   ```

8. Check the system log files ('*/var/adm/SYSLOG*' under *IRIX*; '*/var/log/{message,syslog,}*' under Linux) for why *inetd* rejected the access

Weirder problems

9. You receive a complaint about an unrecognised option:

   ```
   cvs [login aborted]: unrecognized auth response from ...: \
     cvs: unrecognized option '--allow-root=...'
   ```

   Are you running version 1.9 or older of *CVS*? In that case, *cvs* does not understand the `--allow-root` option. Just drop it.

— Written July 2, 2006 by Wolfgang Dobler <Wolfgang.Dobler@ucalgary.ca> —

23

Appendix C

The Pencil Code

http://www.nordita.dk/software/pencil-code

Bibliography

[BF90] R. Davies, A. Rea, and D. Tsaptsinos (1995) *Introduction to FORTRAN 90 – Student Notes*, `http://www.pcc.qub.ac.uk/tec/courses/f90/stu-notes/f90-stu.html`

[CK] W. Cheney and D. Kincaid (2003) *Numerical Mathematics and Computing*, 5th edition, Brooks/Cole, Monterey.

[GPMan] T. Williams and C. Kelly (2004) *gnuplot — an interactive plotting program* (official gnuplot manual), `http://www.gnuplot.info/docs/gnuplot.pdf`.

[GPTut] H. P. Gavin (2004) *Gnuplot 4.0 – A Brief Manual and Tutorial*, `http://www.duke.edu/~hpgavin/gnuplot.html`

[F95] M. Metcalf and J. K. Reid (1999) *Fortran 90/95 explained*, 2nd edition, Oxford University Press, Oxford.

[I1996] Arieh Iserles (1996) *A First Course in the Numerical Analysis of Differential Equations*, Cambridge University Press, Cambridge.

[MEx] M. L. Abell and J. P. Braselton (1997) *Mathematica by Example*, 2nd edition, Academic Press, Toronto.

[NR77] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (1996) *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, 2nd edition, Cambridge University Press, Cambridge. [Online available at `http://www.library.cornell.edu/nr`]

[NR90] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (1996) *Numerical Recipes in Fortran 90: The Art of Scientific Computing*, 2nd edition, Cambridge University Press, Cambridge. [Online available at `http://www.library.cornell.edu/nr`]

[Penc] W. Dobler, A. Brandenburg and others (2005) *The Pencil Code: A High-Order MPI code for MHD Turbulence. User's and Reference Manual*, `http://www.nordita.dk/software/pencil-code/doc/manual.pdf`

[PSS1983] L. A. Pozdniakov, I. M. Sobol and R. A. Suniaev (1983) *Comptonization and the shaping of X-ray source spectra - Monte Carlo calculations*, Soviet Scientific Reviews, Section E: Astrophysics and Space Physics Reviews (ISSN 0143-0432) **2**, 1983, p. 189–331. Translation.