Chapter 8

More on Runge–Kutta methods

8.1 Runge–Kutta revisited

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

$$\begin{aligned}
\tau_1 &= t_0\,, & \boldsymbol{\eta}_1 &= \mathbf{y}_0\,, & \mathbf{k}_1 &= h\mathbf{f}(\tau_1,\boldsymbol{\eta}_1)\,, && (8.1)\\
\tau_2 &= t_0 + c_2 h\,, & \boldsymbol{\eta}_2 &= \mathbf{y}_0 + a_{21}\mathbf{k}_1\,, & \mathbf{k}_2 &= h\mathbf{f}(\tau_2,\boldsymbol{\eta}_2)\,, && (8.2)\\
\tau_3 &= t_0 + c_3 h\,, & \boldsymbol{\eta}_3 &= \mathbf{y}_0 + a_{31}\mathbf{k}_1 + a_{32}\mathbf{k}_2\,, & \mathbf{k}_3 &= h\mathbf{f}(\tau_3,\boldsymbol{\eta}_3)\,, && (8.3)\\
&\vdots & &\vdots & &\vdots && (8.4)\\
\tau_s &= t_0 + c_s h\,, & \boldsymbol{\eta}_s &= \mathbf{y}_0 + a_{s1}\mathbf{k}_1 + a_{s2}\mathbf{k}_2 + \ldots + a_{s,s-1}\mathbf{k}_{s-1}\,, & \mathbf{k}_s &= h\mathbf{f}(\tau_s,\boldsymbol{\eta}_s)\,, && (8.5)\\
t_1 &= t_0 + h\,, & \mathbf{y}_1 &= \mathbf{y}_0 + b_1\mathbf{k}_1 + b_2\mathbf{k}_2 + \ldots + b_{s-1}\mathbf{k}_{s-1} + b_s\mathbf{k}_s\,. & & && (8.6)
\end{aligned}$$

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

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^{\mathsf{T}} \end{array} \quad = \quad \begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & & \ddots & & \\ c_s & a_{s1} & a_{s2} & \ldots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \ldots & b_{s-1} & b_s \end{array} \qquad (8.7)$$

where all omitted elements a_{ij} vanish.

Examples

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

Explicit Euler scheme

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \tag{8.8}$$

Implicit Euler scheme:

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{8.9}$$

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

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}\,, \tag{8.10}$$

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

$$\begin{array}{c|ccc} 0 & & & \\ \frac{1}{2} & \frac{1}{2} & & \\ 1 & -1 & 2 & \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array} \tag{8.11}$$

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

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \tag{8.12}$$

Note: 5th order requires 6 stages!

8.2 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:

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

assuming y_0 was correct

For small δt:

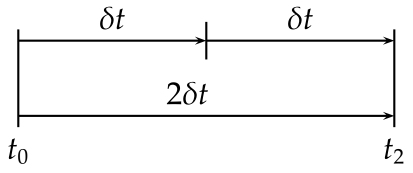
$$|\Delta_n| \;\approx\; \left|\sum_{i=1}^{n} \delta_i\right| \;\leq\; \sum_{i=1}^{n} |\delta_i|$$

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.

8.2.1 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

$$y_1^{(\delta t)} = y_0 + C\,\delta t^{p+1} + O\left(\delta t^{p+2}\right) \; , \tag{8.13}$$

$$y_2^{(\delta t)} = y_0 + 2C\,\delta t^{p+1} + O\left(\delta t^{p+2}\right) \; , \tag{8.14}$$

$$y_2^{(2\delta t)} = y_0 + C\,2^p \delta t^{p+1} + O\left(\delta t^{p+2}\right) \; . \tag{8.15}$$

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

$$y_2^{(2\delta t)} - y_2^{(\delta t)} = (2^{p+1} - 2)C\,\delta t^p + O\left(\delta t^{p+1}\right) \; , \tag{8.16}$$

thus the error estimate for two steps is

$$\tilde{\delta}_1 + \tilde{\delta}_2 = \frac{1}{2^p - 1}(y_2^{(2\delta t)} - y_2^{(\delta t)}) \; . \tag{8.17}$$

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

$$\frac{3-1/s}{2} \approx \frac{3}{2} \,. \tag{8.18}$$

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. . .

8.2.2 . . . Embedded Runge-Kutta schemes

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

$$\begin{aligned} y_1^{(p)} &= y_1^{(exact)} + C\delta t^{p+1} + O\left(\delta t^{p+2}\right) \,, & (8.19)\\ y_1^{(p+1)} &= y_1^{(exact)} + O\left(\delta t^{p+2}\right) \,. & (8.20)\\ & & (8.21) \end{aligned}$$

Thus,

$$\begin{aligned} \tilde{\delta}_1^{(p)} = y_1^{(p)} - y_1^{(p+1)} &= C\,\delta t^{p} + O\left(\delta t^{p+1}\right) & (8.22)\\ &= \delta_1^{(p)}\left[1 + O\left(\delta t\right)\right] \,, & (8.23) \end{aligned}$$

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).

[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.

As an example, consider the Cash–Karp scheme

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & \\
\frac{3}{5} & \frac{3}{10} & -\frac{9}{10} & \frac{6}{5} & & & \\
1 & -\frac{11}{54} & \frac{5}{2} & -\frac{70}{27} & \frac{35}{27} & & \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & \\
\hline
 & \frac{37}{378} & 0 & \frac{250}{621} & \frac{125}{594} & 0 & \frac{512}{1771} \\
\hline
 & \frac{2825}{27648} & 0 & \frac{18575}{48384} & \frac{13525}{55296} & \frac{277}{14336} & \frac{1}{4}
\end{array} \tag{8.24}$$

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

$$\frac{|\delta|}{\varepsilon} = \left(\frac{\delta t_{\mathrm{old}}}{\delta t_{\mathrm{new}}}\right)^{p+1} , \tag{8.25}$$

and hence

$$\delta t_{\mathrm{new}} = \delta t_{\mathrm{old}} \left(\frac{\varepsilon}{|\delta|}\right)^{1/p+1} \tag{8.26}$$

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
```

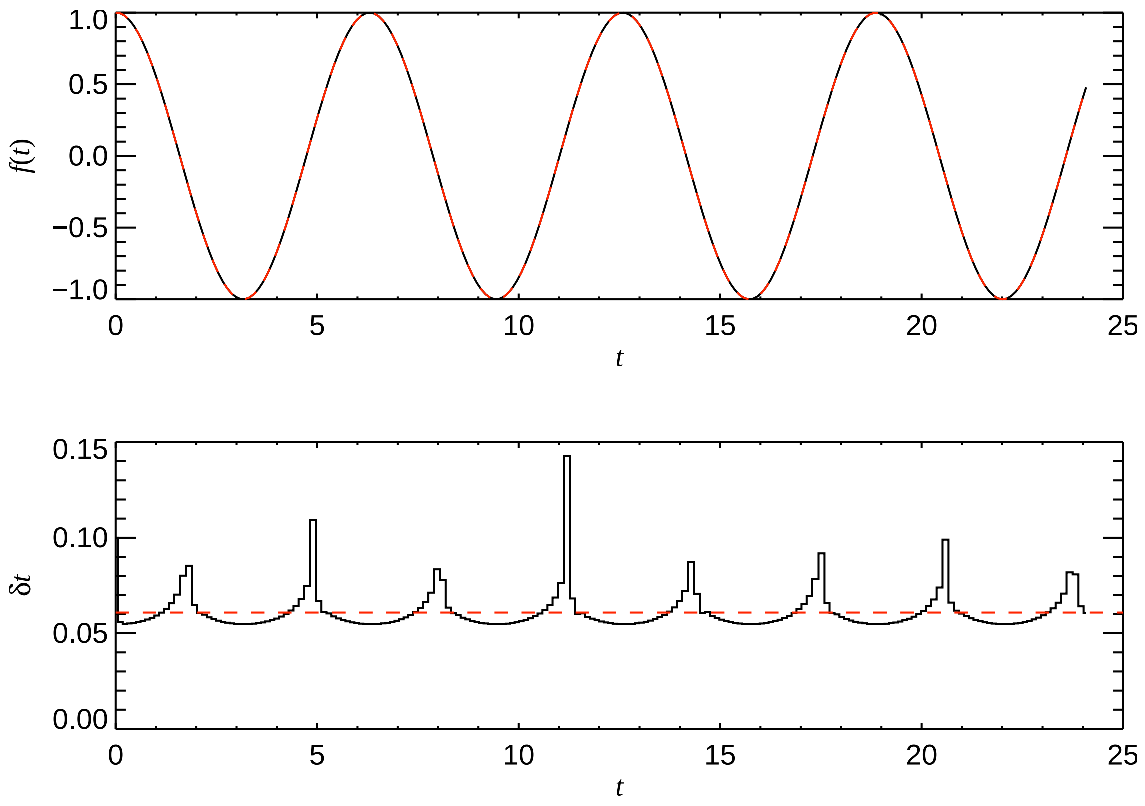


Figure 8.1: Top: Numerical solution $f(t)$ (black continuous line) and analytical solution (red dashed line) for the Curtiss–Hirschfelder equation (8.27). Bottom: Dynamical time step $\delta t(t)$. The numerical method used is the Cash–Karp embedded Runge–Kutta scheme. The dotted line here shows the average time step.

Example: Consider the Curtiss–Hirschfelder equation

$$\dot{y} = -50(y - \cos t) \qquad \text{with } y(0) = 1 \, , \tag{8.27}$$

which has the solution

$$y = \frac{2500}{2501}\cos t + \frac{50}{2501}\sin t + \frac{1}{2501}e^{-50t} \, . \tag{8.28}$$

Figure 8.1 shows the solution f and time step δt as a function of time.

8.3 Stability

For the linear ODE

$$\dot{y} = \gamma y \tag{8.29}$$

with the complex constant γ, we know that the solution

$$y(t) = y_0 e^{\gamma t} \tag{8.30}$$

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. (8.29), we find

$$y_1 = y_0 + \delta t \gamma y_0 = (1 + \gamma\,\delta t) y_0 = A y_0 \;, \tag{8.31}$$

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

$$|1 + \gamma\,\delta t| < 1 \;. \tag{8.32}$$

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

$$A = 1 + \Gamma + \frac{\Gamma^2}{2} \;, \tag{8.33}$$

where we have introduced

$$\Gamma \equiv \gamma\,\delta t \;. \tag{8.34}$$

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

$$A = 1 + \Gamma + \frac{\Gamma^2}{2} + \frac{\Gamma^3}{6} \;, \tag{8.35}$$

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

$$A = 1 + \Gamma + \frac{\Gamma^2}{2} + \frac{\Gamma^3}{6} + \frac{\Gamma^4}{24} \;. \tag{8.36}$$

Figure 8.2 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.

8.4 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:

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t) \;. \tag{8.37}$$

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.

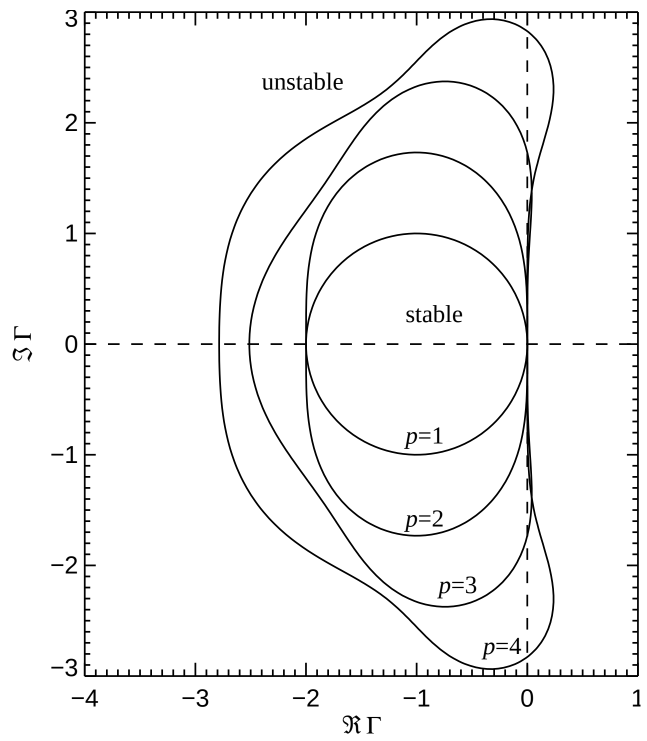


Figure 8.2: Complex stability domains for explicit Runge–Kutta methods of order 1 to 4 applied to Eq. (8.29).

Note: A k^{th} order differential equation

$$y^{(k)} = f\left(y, \dot{y}, \ldots, y^{(k-2)}, y^{(k-1)}, t\right) \tag{8.38}$$

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

$$z_1 \equiv y\ , \quad z_2 \equiv \dot{y}\ , \quad \ldots, \quad z_{k-1} \equiv y^{(k-2)}\ , \quad z_k \equiv y^{(k-1)}\ . \tag{8.39}$$

The resulting system takes the form

$$\begin{aligned} \dot{z}_k &= f(z_0, z_1, \ldots, z_{k-2}, z_{k-1}, t)\ , & (8.40)\\ \dot{z}_{k-1} &= z_{k-2}\ , & (8.41)\\ &\vdots \\ \dot{z}_2 &= z_3\ , & (8.42)\\ \dot{z}_1 &= z_2\ . & (8.43)\\ & & (8.44) \end{aligned}$$

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

$$\dot{\mathbf{y}} = \mathcal{M}\mathbf{y}\ , \tag{8.45}$$

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. (8.45)]

$$\mathbf{y}_{n+1} = \mathcal{A}\mathbf{y}_n \,, \tag{8.46}$$

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

$$\varrho(\mathcal{A}) < 1 \,, \tag{8.47}$$

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

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \delta t\, \mathcal{M} \cdot \mathbf{y}_n = (\mathbb{1} + \delta t\, \mathcal{M})\mathbf{y}_n \,, \tag{8.48}$$

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

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

$$\mathcal{M} \cdot \mathbf{e}_k = \mu_k \mathbf{e}_k \,. \tag{8.49}$$

Then

$$\mathcal{A} \cdot \mathbf{e}_k = \mathbf{e}_k + \mu_k \mathbf{e}_k = (1 + \delta t\, \mu_k)\mathbf{e}_k \,, \tag{8.50}$$

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

$$a_k = 1 + \delta t\, \mu_k \,. \tag{8.51}$$

The stability requirement for the Euler scheme is thus

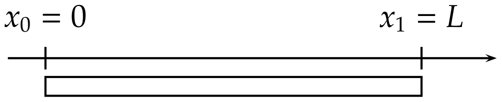
$$|1 + \delta t\, \mu_k| < 1 \,, \tag{8.52}$$

in analogy to Eq. (8.32).

8.5 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. 8.4, 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

$$F = -\lambda \frac{dT}{dx} , \tag{8.53}$$

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

$$\frac{d}{dx}\left(\lambda \frac{dT}{dx}\right) + q(x) = 0 . \tag{8.54}$$

Thus, the complete problem is

$$\begin{aligned} (\lambda T')' &= -q(x) , && (8.55)\\ T(0) &= T_0 , && (8.56)\\ T'(L) &= 0 . && (8.57) \end{aligned}$$

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):

$$\begin{aligned} F'(x) &= q(x) , && (8.58)\\ T'(x) &= -\frac{1}{\lambda}F(x) , && (8.59)\\ T(0) &= T_0 , && (8.60)\\ F(L) &= 0 . && (8.61) \end{aligned}$$

8.5.1 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. (8.58), (8.59) 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]

$$\frac{dy_i}{dt} = f_i(y_1, y_2, \ldots, y_N, t) \qquad i = 1, 2, \ldots, N \; . \tag{8.62}$$

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

$$\begin{aligned} y_i(t_1) &= A_i \qquad i = 1, \ldots, n_1 \; , & (8.63) \\ y_{i+j}(t_2) &= B_i \qquad i = 1, \ldots, n_2 \; , & (8.64) \end{aligned}$$

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 (8.64) 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.

8.5.2 Problems that can be reduced to standard boundary value problem

Eigenvalue problem

An eigenvalue problem

$$\begin{aligned} \frac{dy_i}{dt} &= f_i(y_1, y_2, \ldots, y_N, \lambda) \qquad & i = 1, 2, \ldots, N \; , & \quad (8.65) \\ y_i(t_1) &= A_i & i = 1, \ldots, n_1 \; , & \quad (8.66) \\ y_{i+j}(t_2) &= B_i & i = 1, \ldots, n_2 & \quad (8.67) \end{aligned}$$

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

$$\frac{dy_{N+1}}{dt} = 0 \qquad (\lambda \text{ is constant after all}) \; , \tag{8.68}$$

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

$$y'' = -k^2 y \,, \tag{8.69}$$

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

$$k^2 = \frac{\eta}{F}\omega^2 \,; \tag{8.70}$$

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

$$y(0) = 0 \,, \qquad y(L) = 0 \,. \tag{8.71}$$

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

$$\begin{aligned} y' &= z \,, && (8.72)\\ z' &= -\frac{\eta}{F}\lambda\, y \,, && (8.73)\\ \lambda' &= 0 \,, && (8.74) \end{aligned}$$

with the boundary conditions

$$\begin{aligned} y(0) &= 0 \,, && (8.75)\\ y(L) &= 0 \,, && (8.76)\\ z(0) &= 1 \,. && (8.77) \end{aligned}$$

Free boundary problems

Consider the case of N equations

$$\begin{aligned} \frac{dy_i}{dt} &= f_i(y_1, y_2, \ldots, y_N) && i = 1, 2, \ldots, N \,, && (8.78)\\ y_i(t_1) &= A_i && i = 1, \ldots, n_1 \,, && (8.79)\\ y_{i+j}(t_2) &= B_i && i = 1, \ldots, n_2 \,, && (8.80) \end{aligned}$$

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 $0 \leq j \leq n_1-1$.

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

$$\frac{dy_{N+1}}{dt} = 0 \qquad (t_1 \text{ and } t_2 \text{ cannot depend on } t!) \,, \tag{8.81}$$

Substituting the independent variable,

$$t - t_1 \equiv \tau\, y_{N+1} \qquad \text{(and thus } dt = y_{N+1}\, d\tau\text{)}\ , \tag{8.82}$$

we get

$$\frac{dy_i}{d\tau} = f_i(y_1, y_2, \ldots, y_N)\, y_{N+1} \qquad i = 1, 2, \ldots, N\ , \tag{8.83}$$

$$\frac{dy_{N+1}}{d\tau} = 0\ , \tag{8.84}$$

$$y_i(t_1) = A_i \qquad i = 1, \ldots, n_1\ , \tag{8.85}$$

$$y_{i+j}(t_2) = B_i \qquad i = 1, \ldots, n_2{+}1\ , \tag{8.86}$$

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

8.6 Appendix

8.6.1 Lab exercises

Question 16 *Step-size controlled solution of the Curtiss–Hirschfelder equation*

The code for this problem can be found at
`http://www.capca.ucalgary.ca/top/teaching/phys499+535/Section8/Curtiss-Hirschfelder`

(a) Numerically solve the Curtiss–Hirschfelder equation for at least $t = 50$.

(b) Adapt '`run.pro`' to make it overplot minimum and maximum of δt instead of the average.

(c) Find the largest error tolerance *err* that is acceptable (decision based on the two curves).

(d) Add a third panel showing the difference between the numerical solution and $\cos t$. Compare with exact result, and find again the largest acceptable value of *err*.

Question 17 *Van der Pol equation*

Now "vectorize" the files from the Curtiss–Hirschfelder experiments and use them to solve the van der Pol equations

$$\ddot{y} = -y + \mu(1 - y^2)\dot{y} \tag{8.87}$$

for $\mu = 5$.

Note: It is best to start with a full copy of the Curtiss–Hirschfelder directory.

(a) Write Eq. (8.87) as a system of two first-order equations.

(b) Now adapt the files `start.pro`, `pde.pro`, `run.pro`, and '`print.pro`' to make them work with a 2-element array |f|.

(c) Adapt '`README`', so you can still use this setup in a year from now. . .

(d) Plot $y(t)$, $\dot{y}(t)$, $\dot{y}(y)$, and $\delta t(t)$.

(e) How does the parameter μ affect the behaviour of the time step $\delta t(t)$?

Question 18 *Multi-dimensional root finding in IDL*

(a) Numerically, find the position of the maxima of

$$f(x) = \frac{x^3}{e^x - 1} \tag{8.88}$$

and

$$g(x) = \frac{1/x^5}{e^{1/x} - 1} \, . \tag{8.89}$$

Use IDL's |broyden()| function for root finding.

Which of the keywords of |broyden()| do you need to tune to increase the precision of the result?

Now use |broyden()| to find a root of the system

$$\cos x + e^{-y} = 6 \tag{8.90}$$
$$\frac{x^3}{x^2 + y^2} = 2 \tag{8.91}$$

Verify that the numerical solution satisfies the equations (this is a one-liner in IDL).

Question 19 *Heat conduction in a rod*

Consider a thermally conducting thin rod of constant heat conductivity $\lambda = 1$. The left end of the rod ($x = 0$) is kept at constant temperature $T = 0$, the right end ($x = L \equiv \pi$) is thermally insulated, and along the rod a volume heating $q(x) = 2 + \cos x$ is applied.

(a) Find the steady state of the rod and plot temperature and heat flux as functions of x.

Hint: Use the fixed-time-step Runge–Kutta scheme |rk4.pro| to define a function $F(L; \varphi)$ that maps a guess φ for the heat flux $F(0)$ to a value at the right end. Then use |broyden()| to find the correct value of φ.

(b) Replace $q(x)$ by a narrow Gaussian centred somewhere near the middle of the rod. Explain what you get now.