

1 Ordinary Differential Equations

The basic problem in ordinary differential equations (ODEs) is the initial value problem (IVP). The IVP consists of two components, the differential equation and the initial value, such as

$$\begin{aligned}\frac{du}{dt} &= f(u(t)) & (1) \\ u(0) &= u_0 . & (2)\end{aligned}$$

Separating and integrating (1) results in,

$$\int_t^{t+\Delta t} du = \int_t^{t+\Delta t} f(u(s)) ds \quad (3)$$

Completing the integral on the left side of (3) results in

$$u(t + \Delta t) - u(t) = \int_t^{t+\Delta t} f(u(s)) ds \quad (4)$$

or on rearranging (4)

$$u(t + \Delta t) = u(t) + \int_t^{t+\Delta t} f(u(s)) ds . \quad (5)$$

Equation (5) provides a rule for the advancement u from time t to time $t + \Delta t$ in terms of the integral of f over that interval. The remaining step is to choose a particular method of numerical quadrature. We will consider three types of quadrature:

Forward Euler – approximate the value of the function over the interval by the current value.

$$\int_t^{t+\Delta t} f(u(s)) ds \approx f(u(t)) \Delta t \quad (6)$$

substituting (6) into (5) produces the explicit forward Euler time integration method:

$$u(t + \Delta t) = u(t) + f(u(t)) \Delta t \quad (7)$$

Backward Euler – approximate the value of the function over the interval by the future value.

$$\int_t^{t+\Delta t} f(u(s)) ds \approx f(u(t + \Delta t)) \Delta t \quad (8)$$

substituting (8) into (5) produces the implicit backward Euler time integration method:

$$u(t + \Delta t) = u(t) + f(u(t + \Delta t))\Delta t \quad (9)$$

Trapezoid – approximate the value of the function over the interval by the average of the current and future value.

$$\int_t^{t+\Delta t} f(u(s))ds \approx \frac{1}{2} [f(u(t + \Delta t)) + f(u(t))] \Delta t \quad (10)$$

substituting (10) into (5) produces the mixed explicit/implicit Trapezoidal time integration method:

$$u(t + \Delta t) = u(t) + \frac{1}{2} [f(u(t + \Delta t)) + f(u(t))] \Delta t \quad (11)$$

For the next section, the simplification $f(u(t)) = -(\alpha + i\beta)u(t)$ will be made. A common shorthand notation is to write $u(t)$ as U^n , where $t_n = n\Delta t$. Then $u(t + \Delta t)$ is written as U^{n+1} .

1.1 Numerical Solution of ODEs

Forward Euler Method is simplest explicit method. The initial state U^0 is advanced by the rule $U^{n+1} = U^n + \Delta t F(U^n)$. For the special case of a linear functional $F(U) = -(\alpha + i\beta)U$, the integration rule becomes

$$\begin{aligned} U^{n+1} &= U^n + \Delta t F(U^n) \\ &= U^n - \Delta t(\alpha + i\beta)U^n \\ &= U^n (1 - \Delta t(\alpha + i\beta)) \end{aligned} \quad (12)$$

So what does (12) mean? The initial value for U is represented by U^0 . The subsequent iterates represent a march forward in time:

$$\begin{aligned} u(\Delta t) = U^1 &= U^0 (1 - \Delta t(\alpha + i\beta)) \\ u(2\Delta t) = U^2 &= U^1 (1 - \Delta t(\alpha + i\beta)) \\ &= U^0 (1 - \Delta t(\alpha + i\beta))^2 \\ u(3\Delta t) = U^3 &= U^2 (1 - \Delta t(\alpha + i\beta)) \\ &= U^1 (1 - \Delta t(\alpha + i\beta))^2 \\ &= U^0 (1 - \Delta t(\alpha + i\beta))^3 \\ &\vdots \\ u(N\Delta t) = U^N &= U^{N-1} (1 - \Delta t(\alpha + i\beta)) \\ &= U^0 (1 - \Delta t(\alpha + i\beta))^N . \end{aligned}$$

How should this be coded! Using a single variable, U , the above sequence can be implemented with a simple for loop.

```

% define initial condition
U = Uinitial
for k=1:N
    U = U * ( 1 - dt*(a + i b))
end

```

Notice that the variable U is overwritten each time step. This does not pose a problem since it is overwritten only after it is used. Convince yourself that this works!

Backward Euler Method is simplest implicit method. The initial state U^0 is advanced by the rule $U^{n+1} = U^n + \Delta t F(U^{n+1})$. Notice that the new time level is on both sides of the equation. To write this as a numerical scheme, we need the new level ($n+1$) only on one side of the equation. For the special case of a linear functional $F(U) = -(\alpha + i\beta)U$, this separation is easy. The integration rule becomes

$$\begin{aligned}
 U^{n+1} &= U^n + \Delta t F(U^{n+1}) \\
 &= U^n - \Delta t(\alpha + i\beta)U^{n+1} \\
 U^{n+1}(1 + \Delta t(\alpha + i\beta)) &= U^n \\
 U^{n+1} &= \frac{1}{(1 + \Delta t(\alpha + i\beta))} U^n. \quad (13)
 \end{aligned}$$

The scheme (13) is very similar to (12). The primary difference is the particular multiplication factor $\frac{1}{(1 + \Delta t(\alpha + i\beta))}$. Just as before, the iterates represent a march forward in time:

$$\begin{aligned}
 u(\Delta t) = U^1 &= U^0 \left(\frac{1}{1 + \Delta t(\alpha + i\beta)} \right) \\
 u(2\Delta t) = U^2 &= U^1 \left(\frac{1}{1 + \Delta t(\alpha + i\beta)} \right) \\
 &= U^0 \left(\frac{1}{1 + \Delta t(\alpha + i\beta)} \right)^2 \\
 u(3\Delta t) = U^3 &= U^2 \left(\frac{1}{1 + \Delta t(\alpha + i\beta)} \right) \\
 &= U^1 \left(\frac{1}{1 + \Delta t(\alpha + i\beta)} \right)^2 \\
 &= U^0 \left(\frac{1}{1 + \Delta t(\alpha + i\beta)} \right)^3 \\
 &\vdots
 \end{aligned}$$

$$\begin{aligned}
u(N\Delta t) = U^N &= U^{N-1} \frac{1}{1 + \Delta t(\alpha + i\beta)} \\
&= U^0 \left(\frac{1}{1 + \Delta t(\alpha + i\beta)} \right)^N .
\end{aligned}$$

This sequence can be implemented again with a simple for loop.

```

U = Uinitial
for k=1:N
    U = U/( 1 + dt*(a + i b) )
end

```

This works in the same way as the code fragment for the Euler method.

Trapezoidal Method is simplest mixed explicit/implicit method. The initial state U^0 is advanced by the rule $U^{n+1} = U^n + \frac{\Delta t}{2} (F(U^n) + F(U^{n+1}))$. Notice that the right hand side is the average of F at the two time levels. To write this as a numerical scheme, we need separate the two time levels. Again, for the special case of a linear functional $F(U) = -(\alpha + i\beta)U$, this separation is easy. The integration rule becomes

$$\begin{aligned}
U^{n+1} &= U^n + \frac{\Delta t}{2} (F(U^n) + F(U^{n+1})) \\
&= U^n - \frac{\Delta t}{2} (\alpha + i\beta)(U^n + U^{n+1}) \\
U^{n+1} \left(1 + \frac{\Delta t}{2} (\alpha + i\beta) \right) &= U^n \left(1 - \frac{\Delta t}{2} (\alpha + i\beta) \right) \\
U^{n+1} &= \frac{1 - \frac{\Delta t}{2} (\alpha + i\beta)}{1 + \frac{\Delta t}{2} (\alpha + i\beta)} U^n . \tag{14}
\end{aligned}$$

The scheme (14) is a straight forward combination of (12) and (13). Marching (14) forward in time produces:

$$\begin{aligned}
u(\Delta t) = U^1 &= U^0 \left(\frac{1 - \frac{\Delta t}{2} (\alpha + i\beta)}{1 + \frac{\Delta t}{2} (\alpha + i\beta)} \right) \\
u(2\Delta t) = U^2 &= U^1 \left(\frac{1 - \frac{\Delta t}{2} (\alpha + i\beta)}{1 + \frac{\Delta t}{2} (\alpha + i\beta)} \right) \\
&= U^0 \left(\frac{1 - \frac{\Delta t}{2} (\alpha + i\beta)}{1 + \frac{\Delta t}{2} (\alpha + i\beta)} \right)^2
\end{aligned}$$

$$\begin{aligned}
u(3\Delta t) = U^3 &= U^2 \left(\frac{1 - \frac{\Delta t}{2}(\alpha + i\beta)}{1 + \frac{\Delta t}{2}(\alpha + i\beta)} \right) \\
&= U^1 \left(\frac{1 - \frac{\Delta t}{2}(\alpha + i\beta)}{1 + \frac{\Delta t}{2}(\alpha + i\beta)} \right)^2 \\
&= U^0 \left(\frac{1 - \frac{\Delta t}{2}(\alpha + i\beta)}{1 + \frac{\Delta t}{2}(\alpha + i\beta)} \right)^3 \\
&\dots \\
u(N\Delta t) = U^N &= U^{N-1} \left(\frac{1 - \frac{\Delta t}{2}(\alpha + i\beta)}{1 + \frac{\Delta t}{2}(\alpha + i\beta)} \right) \\
&= U^0 \left(\frac{1 - \frac{\Delta t}{2}(\alpha + i\beta)}{1 + \frac{\Delta t}{2}(\alpha + i\beta)} \right)^N .
\end{aligned}$$

Just as above, this sequence can be implemented with a simple for loop.

```

U = Uinitial
for k=1:N
    U = U*( 1 - 0.5*dt*(a + i b) )/( 1 + 0.5*dt*(a + i b) )
end

```

Convince yourself that this works!

Notice that the code for all three schemes are very similar and only take a few lines. Notice also that no vectors have been used here to compute the solution. All variables are scalars. We will see that for purposes of output we will need a storage vector, but this is separate from the solver.

1.2 Form of the function F

A simple linear form for the function F has been assumed in the above examples, but this is commonly not the case. Of greatest interest is the case where F contains a spatial derivative. For example if $F(u) = -c\frac{\partial u}{\partial x}$, the ODE becomes the partial differential equation $\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$. This is the uni-directional wave equation. In this case the variable u depends on both x and t . The discrete form of u then becomes $u(n\Delta t, i\Delta x) = U_i^n$. If the spatial derivative is expressed by the one way difference $\frac{\partial u}{\partial x} \approx \frac{U_i^n - U_{i-1}^n}{\Delta x}$, for $i = 1, \dots, K$, the implicit backward

Euler implementation of the wave equation is

$$\begin{aligned} U_i^{n+1} &= U_i^n + \Delta t F(U^{n+1}) \\ &= U_i^n - c \frac{\Delta t}{\Delta x} (U_i^{n+1} - U_{i-1}^{n+1}) \end{aligned} \quad (15)$$

$$U_i^{n+1} + c \frac{\Delta t}{\Delta x} (U_i^{n+1} - U_{i-1}^{n+1}) = U_i^n \quad (16)$$

$$U_i^{n+1} (1 + c \frac{\Delta t}{\Delta x}) - U_{i-1}^{n+1} = U_i^n \quad (17)$$

This last equation is a matrix problem of the form

$$\mathbb{M} \mathbf{U}^{n+1} = \mathbf{U}^n, \quad (18)$$

where the vectors \mathbf{U}^{n+1} and \mathbf{U}^n are of the form

$$\mathbf{U}^{n+1} = \begin{bmatrix} U_1^{n+1} \\ U_2^{n+1} \\ U_3^{n+1} \\ \vdots \\ U_{K-1}^{n+1} \\ U_K^{n+1} \end{bmatrix} \quad (19)$$

and the $K \times K$ matrix \mathbb{M} , depending on boundary conditions, will look something like:

$$\begin{bmatrix} 1 + \Omega & 0 & 0 & 0 & \cdots & 0 & 0 \\ -\Omega & 1 + \Omega & 0 & 0 & \cdots & 0 & 0 \\ 0 & -\Omega & 1 + \Omega & 0 & \cdots & 0 & 0 \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 + \Omega & 0 \\ 0 & 0 & 0 & 0 & \cdots & -\Omega & 1 + \Omega \end{bmatrix} \quad (20)$$

where $\Omega = c \frac{\Delta t}{\Delta x}$. To advance (18) it is necessary to invert \mathbb{M} such that

$$\mathbf{U}^{n+1} = \mathbb{M}^{-1} \mathbf{U}^n. \quad (21)$$

The inversion of \mathbb{M} for large systems is computationally expensive. Recall how simple the Backward Euler method (13) was for the linear scalar form of F . This is why we willingly use the method to solve an ODE, but think twice before applying it to a PDE.

1.3 Three Level Schemes

The trapezoidal method is unconditionally stable, and has the lowest truncation error of the three schemes presented. Its weakness is that it is implicit. This can be a problem when F is other than the linear scalar function used above.

Information from earlier time levels can be incorporated into the integration formula. This increases storage, but avoids performing more than one evaluation of F per time step. Of these three level schemes, the leapfrog method is the best explicit scheme for oscillation and wave equation problems. The leapfrog scheme is stable, second order, and requires only one function evaluation per time step. Its only drawback is the existence of an undamped computational mode, which slowly amplifies during simulations of nonlinear problems. For linear problems this is not an issue.

The leapfrog scheme is $U^{n+1} = U^{n-1} + 2\Delta t F(U^n)$. This scheme uses three time levels. Implementing the leapfrog scheme requires two initial levels; U^0 and U^1 . The first leapfrog iterate $U^2 = U^0 + 2\Delta t F(U^1)$ actually takes place at the second time level ($t = 2\Delta t$). To prime the leapfrog method, it is necessary to advance the initial scheme one time step using another method such as trapezoidal. The leapfrog + trapezoidal sequence is then

$$\begin{aligned}
 u(\Delta t) = U^1 &= U^0 + \frac{\Delta t}{2} (F(U^0) + F(U^1)) && \text{Trapezoidal} \\
 u(2\Delta t) = U^2 &= U^0 + 2\Delta t F(U^1) && \text{Leapfrog} \\
 u(3\Delta t) = U^3 &= U^1 + 2\Delta t F(U^2) && \text{Leapfrog} \\
 u(4\Delta t) = U^4 &= U^2 + 2\Delta t F(U^3) && \text{Leapfrog} \\
 &\vdots && \\
 u(N\Delta t) = U^N &= U^{N-2} + 2\Delta t F(U^{N-1}) && \text{Leapfrog}
 \end{aligned}$$

The leapfrog sequence can be implemented with a for loop and the three variables Unew, U, and Uold. The code will look roughly like

```

% define initial condition
Uold = Uinitial
% single step of trapezoidal to prime Leapfrog sequence
U = Uold*( 1 - 0.5*dt*(a + i b) )/( 1 + 0.5*dt*(a + i b) )
% loop through Leapfrog
for k=2:N
    Unew = Uold - 2*dt*(a + i b)*U
% shift variables in time for next loop
    Uold = U
    U = Unew
end

```

This code fragment takes the initial condition Uinitial, moves it into Uold, and then takes a single step of Trapezoidal. The output from the Trapezoidal step is used as the middle time level in the leapfrog loop. The old time level is in the Uold variable. The new time level Unew results as output from the leapfrog step. The last step is to shift the variables back in time for the next loop. The current middle level variable becomes the old variable Uold = U, and the new middle time level variable is the current new variable U = Unew.

1.4 Time Step Selection

The selection of the time step Δt depend on satisfying two factors. The first is the stability constraint for the method. For the *forward Euler* method applied to the damping problem, i.e. $\frac{du}{dt} = -\alpha u$, the stability criteria is $\alpha\Delta t_{stability} < 2$. A good general rule is to limit the actual time step $\Delta t \leq \frac{1}{2}\Delta t_{stability}$. If we applied the *backward Euler* method to the damping problem, there is no stability criteria to work from. In that case recall that the truncation error for the *backward Euler* method is first order. This means that the error proportional to some constant times Δt . So to obtain three digits of accuracy, a reasonable time step restriction is $\Delta t < 0.001$.