

Lecture 15: Error Analysis for time-stepping routines

Accuracy:

- Error analysis for time-stepping schemes is of a similar nature to the error analysis for difference formulas from lecture 11.
- The analysis will also apply to the Time and Space stepping schemes in Lecture 22.
- We are interested in the local discretization error and global discretization error.

- Local error is given by:

$$\epsilon_{k+1} = y(t_{k+1}) - (y(t_k) + \Delta t \cdot \phi)$$

- Global error is given by: $E_k = y(t_k) - y_k$

For the Forward Euler's method we have for a time step Δt for $t \in [a, b]$, and after K steps $\Delta t \cdot K = b - a$ that

$$\epsilon_k = \frac{\Delta t^2}{2} \frac{d^2 \mathbf{y}(c_k)}{dt^2} \sim O(\Delta t^2)$$

$$\begin{aligned} E_k &= \sum_{j=1}^K \frac{\Delta t^2}{2} \frac{d^2 \mathbf{y}(c_j)}{dt^2} \\ &\approx K \frac{\Delta t^2}{2} \frac{d^2 \mathbf{y}(c)}{dt^2} \\ &= \frac{(b-a)\Delta t}{2} \frac{d^2 \mathbf{y}(c)}{dt^2} \sim O(\Delta t) \end{aligned}$$

The accuracy of other methods can be analyzed in a similar fashion by beginning with the Taylor expansion of the solution in order to calculate ϵ_k and E_k .

<i>Method</i>	<i>Local</i>	<i>Global</i>
<i>Euler</i>	$O(\Delta t^2)$	$O(\Delta t)$
<i>2nd O Runge – Kutta</i>	$O(\Delta t^3)$	$O(\Delta t^2)$
<i>4th O Runge – Kutta</i>	$O(\Delta t^5)$	$O(\Delta t^4)$
<i>2nd O Adams – Bashforth</i>	$O(\Delta t^3)$	$O(\Delta t^2)$

Round-off and choosing the step size: We obtain similar results for round-off and truncation error of the forward difference (Lecture 11) as for the Forward Euler method.

To minimize the total error ($E = E_{trunc} + E_{round}$) as a function of step size Δt :

$$\Delta t = \left(\frac{2e_r}{M} \right)^{1/3}$$

Stability: Accuracy is only relevant of course if the scheme is stable.

A general theory of stability can be developed for any one-step time-stepping scheme. Consider the recursion relation for an $M \times M$ system of ODEs

$$\mathbf{y}_{n+1} = \mathbf{A}\mathbf{y}_n$$

after N steps the recursion results in

$$\mathbf{y}_N = \mathbf{A}^N \mathbf{y}_0$$

and further applying the power of matrices from lecture 7 we obtain:

$$\mathbf{A}^N = \mathbf{S}^{-1} \mathbf{\Lambda}^N \mathbf{S}$$

\mathbf{S} is the matrix whose columns are the eigenvectors of \mathbf{A} . $\mathbf{\Lambda}$ is the diagonal matrix whose entries are the eigenvalues of \mathbf{A} .

From the last equation we are only interested in the eigenvalues since instability occurs when $\Re(\lambda_i) > 1$ for $i = 1, 2, \dots, M$.

For two step schemes (Adams methods) the same procedure can be applied starting with the method written as:

$$\mathbf{y}_{n+1} = \mathbf{A}\mathbf{y}_n + \mathbf{B}\mathbf{y}_{n-1}$$

In practice: For a simple ODE:

$$\frac{dy}{dt} = \lambda y$$

$$y(0) = y_0$$

The Forward Euler method (including round-off) results in

$$y_N = (1 + \lambda\Delta t)^N (y_0 + e)$$

Which results in a cumulative error

$$E_N = (1 + \lambda\Delta t)^N e$$

and the Backward Euler method (including round-off) results in

$$y_N = \left(\frac{1}{1 - \lambda\Delta t} \right)^N (y_0 + e)$$

Which results in a cumulative error

$$E_N = \left(\frac{1}{1 - \lambda\Delta t} \right)^N e$$

Thus for $|1 + \lambda\Delta t| > 1$ the Forward Euler method is unstable and for $\frac{1}{|1 - \lambda\Delta t|} > 1$ the Backward Euler method is unstable.

What to do with this information?

When implementing your own numerical Adams type ODE solver, this procedure allows you to choose a Δt that falls in the range of Δt 's that satisfy the stability condition for the method of choice.

Lecture 15.5: More ode45 examples

Suppose the problem to be solved is:

$$\frac{dy}{dt} = t^2 + y(t) \quad y(0) = 1 \quad t \in [0, 5]$$

You would use two .m files, one function file for the right hand side (rhs) of the equations for $\frac{dy}{dt}$ and one main file to call ode45 (which calls the function for $\frac{dy}{dt}$).

```
% This code goes inside file simple_rhs.m
```

```
function rhs = simple_rhs(t,y)
    rhs = t^2 + y;
```

```
% This code goes inside main file simple_IVP.m
```

```
tspan = [0,5];
y0 = 1;
[t,y] = ode45('simple_rhs',tspan, y0);
```

```
% or
```

```
[t,y] = ode45(@simple_rhs,tspan, y0);
```

ode45 chooses its own Δt as large as possible to maintain the local accuracy tolerance.

To set your own tolerances and your own points where you want to evaluate the solutions (rather than just the end points)

```
% This code goes inside file simple_rhs.m
```

```
function rhs = simple_rhs(t,y)
    rhs = t^2 + y;
```

```
% This code goes inside main file simple_IVP.m
```

```
tspan = [0:.1:5];
y0 = 1;
TOL = 1e-5;
options = odeset('RelTol',TOL,'AbsTol',TOL);
[t,y] = ode45('simple_rhs',tspan, y0, options);
```

```
% or
```

```
[t,y] = ode45(@simple_rhs,tspan, y0, options);
```

If you have a function which takes in more variables other than t and y ;

```
% This code goes inside file simple_rhs.m
```

```
function rhs = simple_rhs(t,y,p)
    rhs = p(1)*t^2 + p(2)*y;
```

```
% This code goes inside main file simple_IVP.m
```

```
p = [5, 2];
```

```
tspan = [0:.1:5];
```

```
y0 = 1;
```

```
[t,y] = ode45('simple_rhs',tspan,y0,options,p);
```

```
% or
```

```
[t,y] = ode45(@simple_rhs,tspan, y0, [],p);
```