Mat 3110 Numerical methods for ODE

Håkon Hoel

Fall 2023

Implicit RK method

An s-stage RK method with the tableau (1)

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|ccc} c_1 & a_{1,1} & \cdots & a_{1,s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s,1} & \cdots & a_{s,s-1} \\ \hline & b_1 & \cdots & b_s \end{array} \tag{1}$$

is called **implicit** if $a_{i,j} \neq 0$ for at least one component with $j \geq i$.

Recall that an s-stage RK method with Butcher tableau (1) has the stepping rule

$$y_{n+1} = y_n + h\Phi(t_n, y_n, y_{n+1}; h)$$

with

$$\Phi(t_n, y_n, y_{n+1}; h) = \sum_{i=1}^{s} b_i k_i,$$

and system of equations

$$k_i = f\left(t_n + c_i h,\ y_n + h\sum_{j=1}^{s} a_{ij} k_j\right) \qquad i = 1, \ldots, s,$$

Solution approach: introduce $F : \mathbb{R}^s \to \mathbb{R}^s$ where

$$F_i(k_1, \ldots, k_s) = k_i - f\left(t_n + c_i h,\ y_n + h\sum_{j=1}^{s} a_{ij} k_j\right) \quad i = 1, \ldots, s$$

and solve $F(k_1, \ldots, k_s) = 0$ using e.g. Newton's method (for every timestep).

Implicit vs explicit

- Implicit methods tend to be more stable than explicit methods.
- So one can often solve problems robustly with larger $h > 0$ with implicit methods than explicit.
- Implicit methods are more suitable for **stiff problems**, involving dynamics on different timescales, like

$$y' = \begin{pmatrix} -1 & 1/100 \\ 0 & -100 \end{pmatrix} y,$$

  where, $y_2(t) = e^{-100t} y_2(0)$ may vary on a faster timescale than

$$y_1(t) = e^{-t} y_1(0) + \text{“small contribution from”}\, y_2.$$

- A drawback is that implicit methods can be more computationally costly than explicit methods, as one needs to solve implicit equation for y_{n+1} every iteration.

Runge–Kutta methods in $\mathbb{R}^d$

Convergence theory, order of accuracy extends to one-step methods when $d \geq 2$.

And practically, computer implementation of RK can be made dimension-independent. For example:

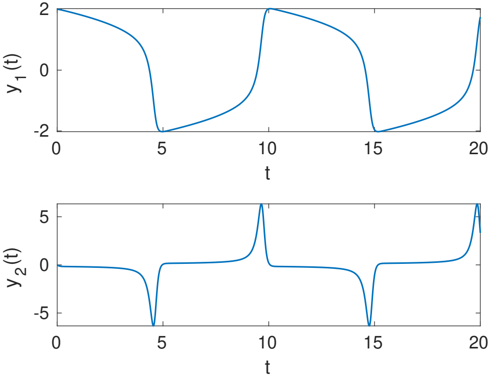
```
function Phi= RK4(t,y,h, f)
  k1 = f(t,y);
  k2 = f(t+h/2, y + (h/2)*k1);
  k3 = f(t+h/2, y + (h/2)*k2);
  k4 = f(t+h, y + h*k3);
  Phi = (k1+2*k2+2*k3+k4)/6;
end
```

Can be used to solve Van der Pol oscillator IVP:

$$\left.\begin{aligned} y_1' &= y_2 \\ y_2' &= 4(1-y_1^2)y_2 - y_1 \end{aligned}\right\} \quad t \in [0,20] \qquad \text{with} \quad y(0) = (2,0) \quad \text{by } \ldots$$

```
f = @(t,x) [x(2); 4*(1-x(1)^2)*x(2)-x(1)];
y0 = [2;0];   a = 0;   b = 20;   N=1000;   h = (b-a)/N;
t = (0:N)*h;   y = zeros(2,N+1);   y(:,1) = y0;

for n=1:N
    y(:,n+1) = y(:,n) + h*RK4(t(n),y(:,n),h, f);
end
```



Example suitable for implicit RK

Consider the ODE

$$y' = Ay \qquad \text{with} \quad y(0) = y_0$$

and $A \in \mathbb{R}^{N\times N}_{sym}$ with all eigenvalues being real-valued and strictly negative:

$$\lambda_N \leq \lambda_{N-1} \leq \ldots \leq \lambda_1 < 0.$$

Implies that the ODE is dissipative: $\|y(t+h)\|_2 < \|y(t)\|_2$ (for $y(t) \neq 0$).

1) Explicit Euler,

$$y_{j+1} = y_j + hAy_j = (I + hA)y_j,$$

preserves dissipativity iff

$$\|I + hA\|_2 < 1 \iff |1 + h\lambda_N| < 1 \iff h < \frac{2}{|\lambda_N|}.$$

2) For implicit Euler, the linear system of equations

$$y_{j+1} = y_j + hAy_{j+1} \implies y_{j+1} = (I - hA)^{-1} y_j$$

preserves dissipativity iff

$$\|(I - hA)^{-1}\|_2 < 1 \iff |1 - \lambda_j h|^{-1} < 1 \quad \forall j \quad \text{(true for all } h > 0\text{!)}$$

Region of absolute stability

For modified Euler (RK2), we showed that

$$R(z) = 1 + z + z^2/2$$

for any RK3 method (any 3-stage explicit RK method with order of accuracy 3), we obtain similarly that

$$R_3(z) = 1 + z + z^2/2 + z^3/6 \quad \text{etc.}$$

For lazy people, like myself, boundaries $|R(z)| = 1$ can be estimated numerically:

```
RK2 = @(z) 1+z +z.^2/2;
RK3 = @(z) 1+z +z.^2/2 + z.^3/6;
[X,Y] = meshgrid(-6:0.01:6,-6:0.01:6);
contour(X,Y,abs(RK2(z)),[0 1],'r', 'linewidth',1.4
contour(X,Y,abs(RK3(z)),[0 1],'r', 'linewidth',1.4
```

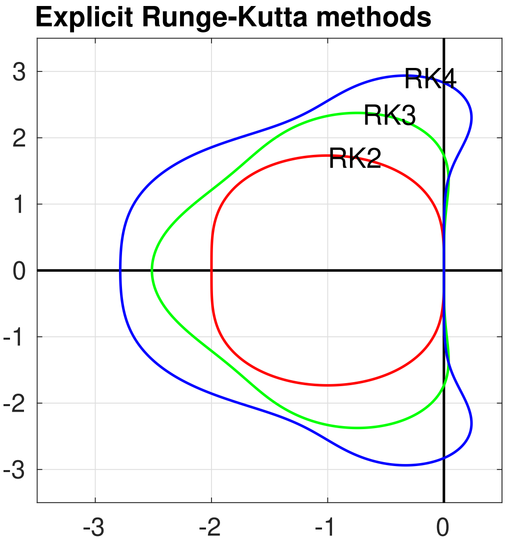


Figure: Region of absolute stability for RK 2,3 and 4 is the **interior** of the respective curves.

Implicit vs explicit RK, test problem

$$y' = -10y, \qquad y(0) = 1, \qquad \text{unique sol} \quad y(t) = e^{-10t}.$$

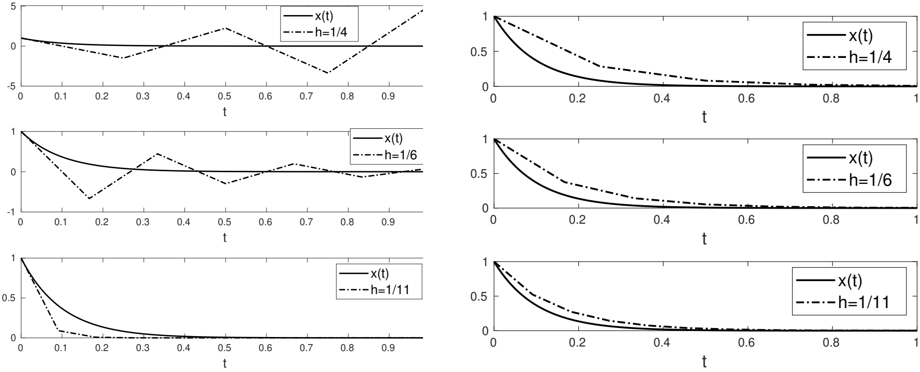


Figure: **Left:** Explicit Euler, stable when $h < 2/|-10| = 1/5$. **Right**: Implicit Euler, unconditionally stable.

Adaptive timestepping

When ODE path $y(t)$ strongly varies in time, efficiency of numerical integration (accuracy vs cost) can improve by using using a variable/adaptive stepsize h.
The Van der Pol oscillator

$$\left.\begin{aligned} x_1' &= x_2 \\ x_2' &= 4(1-x_1^2)x_2 - x_1 \end{aligned}\right\} \quad t \in [0, 20] \qquad \text{with} \quad x(0) = (2, 0)$$

can be written as 2nd order ODE:

$$x_1'' = \underbrace{4(1-x_1^2)x_1'}_{\text{nonlinear "damping"}} - \underbrace{x_1}_{\text{oscillation}}$$

Matlab's variable-stepsize integrator ode45:

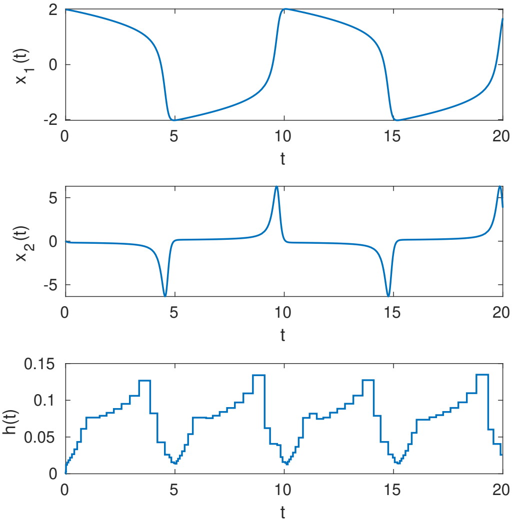


Figure: Bottom: Stepsize of the numerical integrator ($h(t) := h_j$ for $t \in [t_j, t_{j+1})$). We observe that $h(t)$ is small when $|x'(t)|$ is large.

Using ode45

```
f = @(t,x) [x(2); 4*(1-x(1)^2)*x(2)-x(1)];
x0 = [2;0];
t0 = 0, T = 20 % time interval [t0,T]
[t,x] = ode45(f,[t0 T],x0); %numerical integrator
%input: ode45(righthandside, time-interval, initial condition)
%output: [t,x] with  t is the adaptive mesh on which the problem was
% x is a 2 \times length(t) matrix with solution x(:,k) = x_k(t) for

%% ploting results
subplot(3,1,1)
plot(t,x(:,1), 'linewidth',1.2 );xlabel('t');ylabel('x_1(t)')
subplot(3,1,2)
plot(t,x(:,2), 'linewidth',1.2 );xlabel('t');ylabel('x_2(t)')
subplot(3,1,3)
stairs(t(1:end-1), diff(t), 'linewidth',1.3);xlabel('t');ylabel('h(t)
```

Local-error adaptive timestepping

Assume access to two methods: Φ and Φ^*, which respectively produce the solutions

$$y_{j+1} = y_j + h_j \Phi(t_j, y_j; h_j), \qquad \text{and} \quad y^*_{j+1} = y^*_j + h_j \Phi^*(t_j, y^*_j; h_j).$$

1. Assume that Φ^* is more accurate than Φ, and "view" Φ^* as exact solver.
2. Approximate error of Φ made over $(t_n, t_n + h_n)$:

$$S_n(h_n) = h_n \|\Phi(t_n, y^*_n; h_n) - \Phi^*(t_n, y^*_n; h_n)\|$$

3. If $S_n(h_n)$ is too large reduce timestep $h_n = h_n/2$, elseif too small, increase timestep $h_n = 2h_n$, else keep h_n, solve ODE over $(t_n, t_n + h_n)$ using Φ^* and move to $t_{n+1} = t_n + h_n$.
4. Output solution $\{y^*_n\}$.

ODE23 – embedded RK methods

Matlab The Runge-Kutta 2(3) method is given by

$$
\begin{array}{c|ccc}
0 & 0 & & \\
1 & 1 & 0 & \\
1/2 & 1/4 & 1/4 & 0 \\
\hline
 & 1/2 & 1/2 & 0 \\
 & 1/6 & 1/6 & 2/3
\end{array}
$$

where (c, A, b) with $b = (1/2, 1/2, 0)$ produces a (3-stage) RK method Φ of order 2 and same (c, A) with $(1/6, 1/6, 2/3)$ produces method Φ^* of order 3.

Benefit: Both methods share the functions k_1 and k_2!

ODE45 – Dormand–Prince

0							
1/5	1/5						
3/10	3/40	9/40					
4/5	44/45	−56/15	32/9				
8/9	19372/6561	−25360/2187	64448/6561	−212/729			
1	9017/3168	−355/33	46732/5247	49/176	−5103/18656		
1	35/384	0	500/1113	125/192	−2187/6784	11/84	
	35/384	0	500/1113	125/192	−2187/6784	11/84	0
	5179/57600	0	7571/16695	393/640	−92097/339200	187/2100	1/40

Φ and Φ^* used in ODE45. Top line b produces explicit order 5 method Φ^*, lower line b produces explicit order 4 method Φ. The functions $k_1, \ldots, k_6$ are shared for between methods!