

## Lecture 14: Numerical Solution of ODE Initial Value Problems cont'd

### Outline

- 0) Review:
  - basic stepping schemes
  - Adaptive stepping
  - Example: Matlab's ode45 routine and decayChains
- 1) Stiff Equations: Decay Chain Example
- 2) Stability and step-size
- 3) Implicit Schemes: Backward Euler
- 4) Stability of Implicit schemes
- 5) Matlab Routines ode15s
- 6) Implicit schemes and non-linear equations (van Der Pol oscillator)

### Review: single step schemes

All can be related to quadrature schemes

$$\frac{dy}{dt} = f(t, y), \quad y(0) = y_0 \quad y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y) dt$$

Euler scheme:

$$k_1 = h f(t_n, y(t_n)) \quad y_{n+1} = y_n + k_1 + O(h^2)$$

Mid-Point scheme

$$\begin{aligned} k_1 &= h f(t_n, y_n) \\ k_2 &= h f(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ y_{n+1} &= y_n + k_2 + O(h^3) \end{aligned}$$

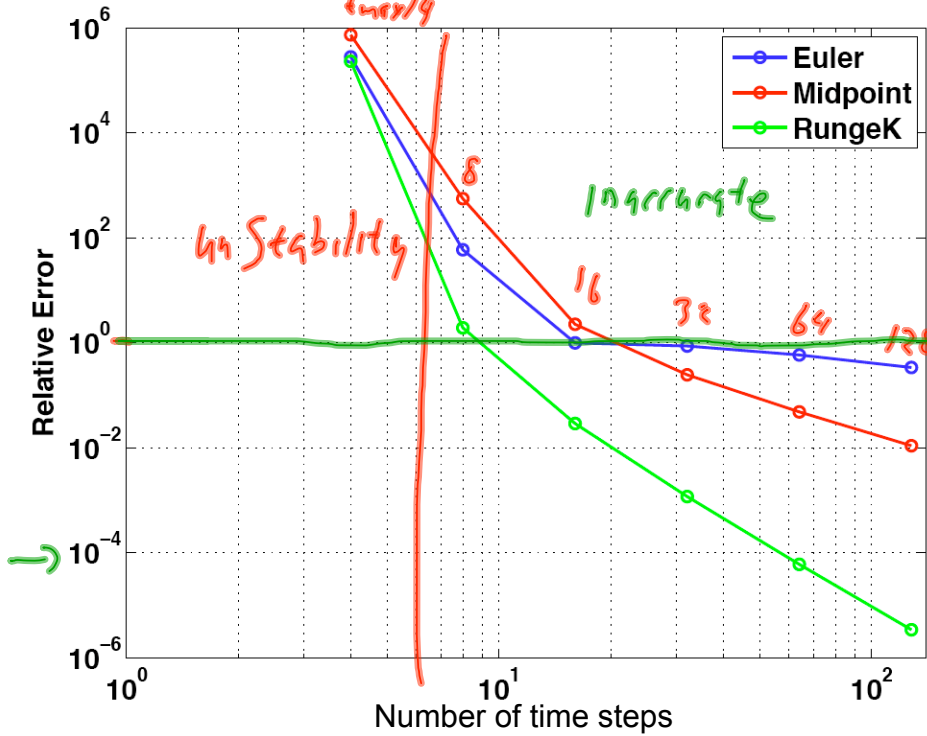


4th order Runge-Kutta

$$\begin{aligned} k_1 &= h f(t_n, y_n) \\ k_2 &= h f(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= h f(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= h f(t_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{k_1}{6} + \frac{(k_2 + k_3)}{3} + \frac{k_4}{6} + O(h^5) \end{aligned}$$

### Comparison of Errors and Accuracy:

Simple decay  $dc/dt = -c$ ,  $c(0)=1$   $t_{max} = 10$



### Adaptive Time stepping:

Step doubling:

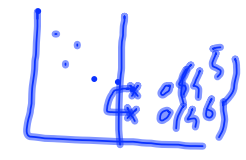
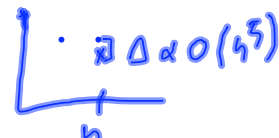
Take one step of size  $h$  and two steps of size  $h/2$ . The difference between the two estimates is the relative truncation error and will be  $O(h^5)$ . Use this information to adjust  $h$ , such that the relative truncation error is less than some tolerance.

Total # of function evaluations per time-step = 11 functions/good step

Embedded Runge-Kutta Schemes:

Numerical Recipes: rkck 5th-6th order Cash-Karp scheme  
 Matlab: ode45 (Dormand, Prince 4-5 pair)

Idea (ode45): take 6 function evaluations. One linear combination is  $O(h^5)$ , the other is  $O(h^6)$ , the relative truncation error is  $O(h^5)$ ?



## Adaptive Time stepping:

### Matlab Example: Radioactive decay chains (linear dynamical system)

Problem:

solve  $du/dt = Au$ , with  $u(0)=u_0$

where  $u$  is a vector of activities (e.g.  $u_0=[1\ 2\ 3]'$ )  
and

$$A = \begin{bmatrix} -\lambda(1) & 0 & 0 \\ \lambda(2) & -\lambda(2) & 0 \\ 0 & \lambda(3) & -\lambda(3) \end{bmatrix}$$

$$\lambda_i = \frac{\log(2)}{\tau_{1/2}}$$

## Matlab Example

### Radioactive decay chains (linear dynamical system)

Step #1: need a function that returns the right hand side  $f(t,u)$

```
function dudt = decayChain(t,u,lambda)
% DECAYCHAIN mfile for radioactive decay of a chain of n nuclides such that
% element_1 -> element_2 -> element_3 ... with decay constants lambda_1, lambda_2 etc
%
% the problem can be written as a linear dynamical system du/dt = Au with
%
% A = [ -lambda_1      0      0 ;
%       lambda_2 -lambda_2      0 ;
%       0      lambda_3 -lambda_3 ]
% etc.
%
% t - current time
% u - vector of activities of decay chain nuclides
% lambda - vector of decay constants
%
% dudt - change in activity with time

% construct sparse decay matrix using spdiags (this is more obscure than it should be)
n = length(u); % number of elements in chain
→ A = spdiags( [ [ lambda(2:end) ; 0 ] -lambda ], [-1 0],n,n);
dudt = A*u;
```

sparse matrices  
full matrices

$$\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix}$$

## Matlab Example

### Radioactive decay chains (linear dynamical system)

Step #2: set options for use by ode-stepping algorithms

examples:

options = odeset; % just use defaults

options = odeset('RelTol',1.e-6,'AbsTol',1.e-8)

options = odeset(options,'OutputFcn','odeplot','Stats','on'); % set default plotting and information options

$$\Delta_i < \max(10^{-3} \text{RelTol} \lambda_i, 10^{-6} \text{AbsTol})$$

Step #3: Set initial conditions, range of integration and parameters

**a0** = [ 1 2 3 ]'; % initial activities

**tRange** = [ 0 1e5 ]; % integrate from 0 to 100,000 years

**lambda** = log(2)./tHalf; % tHalf is vector of half-lives

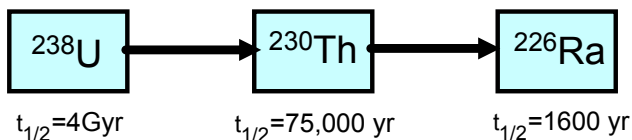
Step #4: Call the ode integration routine, e.g.

[ t, u ] = ode45( @(t, u) decayChain(t,u,lambda), tRange, a0, options);

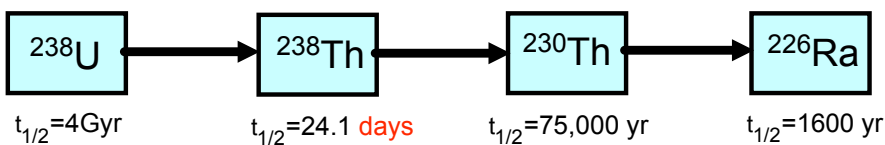
*func(t,u)*

## Stiff Equations:

### Radioactive decay Chains (n-nuclides)



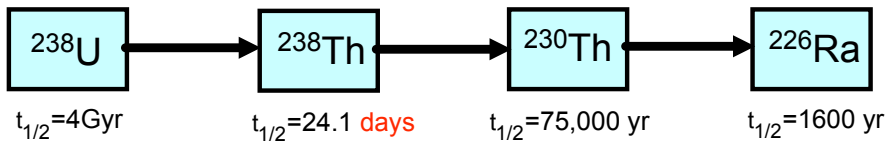
$$\frac{dx}{dt} = Ax$$



$$A = \begin{bmatrix} -\lambda_1 & & & \\ \lambda_2 & -\lambda_2 & & \\ & \lambda_3 & -\lambda_3 & \\ & & \lambda_4 & -\lambda_4 \end{bmatrix}$$

**Stiff Equations:**

Radioactive decay Chains (n-nuclides)



$$y(t) = c_1 e^{-\lambda_1 t} + c_2 e^{-\lambda_2 t} + c_3 e^{-\lambda_3 t} + c_4 e^{-\lambda_4 t}$$

$$c = S^{-1} y_0$$

matlab example: ode45

**Stiff Equations:**Stability of **Explicit** schemes:

1-nuclide decay

Euler step:

$$\frac{dc}{dt} = -\lambda c$$

$$k_1 = hf(t_n, y_n)$$

$$C_{n+1} = C_n - h\lambda C_n$$

$$C_{n+1} = (1 - \lambda h) C_n$$

$$C_k = (1 - \lambda h)^k C_0$$

$$C_1 = (1 - \lambda h) C_0$$

$$C_2 = (1 - \lambda h) C_1 = (1 - \lambda h)^2 C_0$$

stability  $|1 - \lambda h| < 1$

hence  $\lambda h - 1 < 1$

$$h = \tau/\lambda$$

$$h > \tau/\lambda \text{ unstable}$$

**Stiff Equations:**Stability of **Implicit** schemes:

1-nuclide decay

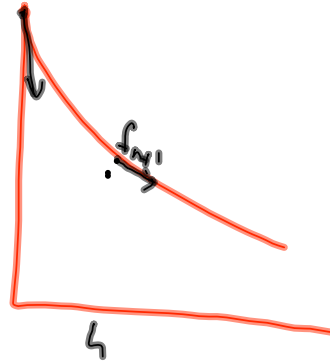
Backward Euler step:

$$C_{n+1} = C_n - h\lambda C_{n+1}$$

$$(1 + \lambda h)C_{n+1} = C_n$$

$$C_{n+1} = \frac{1}{(1 + \lambda h)} C_n \quad h \rightarrow \infty \rightarrow O(h^2)$$

Implicit time step unconditionally stable

**Stiff Equations:**Stability of **Implicit** schemes:

1-nuclide decay

Higher order semi-implicit schemes: Trapezoidal (2nd Order RK)

$$u_{n+1} = u_n + h \left[ \frac{f_n}{2} + \frac{f_{n+1}}{2} \right]$$

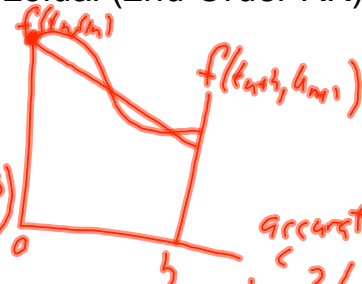
$$C_{n+1} = C_n - \frac{\lambda h}{2} [C_n + C_{n+1}]$$

$$(1 + \frac{\lambda h}{2}) C_{n+1} = (1 - \frac{\lambda h}{2}) C_n$$

ode23s

$$C_{n+1} = \frac{(1 - \lambda h/2)}{(1 + \lambda h/2)} C_n$$

uncond stable



accurate  
 $h = 2/\lambda$

**Stiff Systems of Equations:**Stability of **Explicit** schemes:

n-nuclide decay chains

Euler step:

$$\frac{d\underline{u}}{dt} = A\underline{u}$$

$$\begin{aligned} \underline{u}_{n+1} &= \underline{u}_n + hA\underline{u}_n & \underline{u}_{n+1} &= E\underline{u}_n & \underline{u}_1 &= E\underline{u}_0 \\ &= (I+hA)\underline{u}_n & \underline{u}_k &= E^k\underline{u}_0 & \underline{u}_2 &= E\underline{u}_1 = E^2\underline{u}_0 \end{aligned}$$

$$E = SAS^{-1}$$

$$E^k = S\Lambda^k S^{-1}$$

$$\underline{u}_k = C_i \underline{x}_i$$

$$\underline{u}_0 = C_1 \underline{x}_1 + C_2 \underline{x}_2 + \dots$$

$$E \underline{x}_i = \lambda_i \underline{x}_i$$

$$\underline{u}_k = C_1 \lambda_1^k \underline{x}_1 + C_2 \lambda_2^k \underline{x}_2 + \dots$$

$$\max |\lambda| < 1$$

$$\rho(E) < 1$$

spectral radius

$$\rho(A) = |\lambda|_{\max}$$

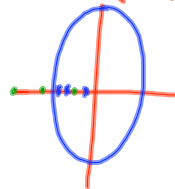
$$\rho(E) = \rho(I+hA)$$

$$\text{if } Ax = \lambda x$$

$$(I+hA)x = x + h\lambda x = (1+h\lambda)x$$

$$\rho(E) = \max_i |1+h\lambda_i| < 1$$

stable requires  
 $h < 2/|\lambda|_{\max}$

**Stiff Systems of Equations:**Stability of **Implicit** schemes:

n-nuclide decay chains

Backward's Euler step:

$$\underline{u}_{n+1} = \underline{u}_n + hA\underline{u}_{n+1}$$

$$(I-hA)\underline{u}_{n+1} = \underline{u}_n$$

$$\underline{u}_{n+1} = (I-hA)^{-1} \underline{u}_n$$

$$E = (I-hA)^{-1}$$

$$\underline{u}_{n+1} = (I-hA)^{-1} \underline{u}_n$$

$$\text{if } \lambda \text{ is real and } \lambda < 0, \text{ then } \lambda e = \frac{1}{1-h\lambda} \quad \rho(E) < 1 \text{ independent of } h$$

if conditions are stable

### Stiff Systems of Equations:

Matlab routines for stiff equations: Best bet ode15s:

Same syntax and options...but works well for stiff problems

General Recommendations:

- 1) Start by trying explicit ode45
- 2) if runs slowly (due to stiffness), try ode15s

### All Matlab odexx routines

see help ode45, or doc ode45

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try.
ode23	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
ode113	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems.
ode15s	Stiff	Low to medium	If ode45 is slow because the problem is stiff.
ode23s	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
ode23t	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

The algorithms used in the ODE solvers vary according to order of accuracy [\[6\]](#) and the type of systems (stiff or nonstiff) they are designed to solve. See [Algorithms](#) for more details.

**Stiff systems of non-linear ODE's:**

Explicit and implicit schemes for non-linear ODE's

$$y'' - \mu(1-y^2)y' + y = 0; \quad y(0)=y_0, y'(0)=v_0$$

**Stiff systems of non-linear ODE's:**

Explicit and implicit schemes for non-linear ODE's

$$y'' - \mu(1-y^2)y' + y = 0; \quad y(0)=y_0, y'(0)=v_0$$