

Runge-Kutta methods

introduction

- recall, we are solving initial value problems of the form

$$\frac{du}{dt} = f(t, u), \quad u(0) = u_0$$

using numerical methods which determine approximations $u_k \approx u(t_k)$ for some numerical grid $t_0 < t_1 < \dots < t_n$

- Runge-Kutta (RK) methods are one-step methods with

$$u_{k+1} = u_k + h\phi(t_k, u_k)$$

- are defined by a special class of functions

$$\phi : [0, T] \times \mathbb{R}^d \rightarrow \mathbb{R}^d$$

$t \quad u$

- The determination of the value of ϕ has two stages:
 1. In the *first stage*, approximations U_k^j of $u(s_j)$ for $s_j \in [t_k, t_{k+1}]$ and $j = 1, \dots, m$ are determined
 2. In the *second stage*, these values are used to compute the value of ϕ by the formula

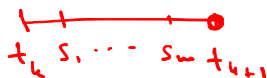
$$\phi(t_k, u_k) = \sum_{j=1}^m c_j f(s_j, U_k^j)$$

(Handwritten red annotations: a bracket groups the sum, and an arrow points from the label k_j to the term $f(s_j, U_k^j)$)

- note that in the RK literature you can often find $k_j = f(s_j, U_k^j)$ (k_j has nothing to do with the index k !)

first stage of RK

U_k^j



- recall that for all $s_j \in [t_k, t_{k+1}]$ the exact solution $u(s_j)$ satisfies

$$u(s_j) = u(t_k) + \int_{t_k}^{s_j} f(r, u(r)) dr$$

- replacing the integral by numerical quadrature leads to

$$U_k^j = u_k + h \sum_{i=1}^{m, j-1} b_{ji} f(s_i, U_k^i), \quad j = 1, \dots, m$$

explicit

where b_{ji} are again quadrature weights

- note however, that the b_{ji} are not necessarily standard quadrature weights and require the solution of systems of polynomial equations

- ▶ for explicit methods, one chooses the matrix $B = [b_{ji}]$ to be lower diagonal with zero diagonal elements such that $b_{ji} = 0$ for $i \geq j$
- ▶ the equation is solved by substitution
- ▶ for the first stage often variants of Euler or the midpoint rules are used
- ▶ note that the points $s_i \in [t_k, t_{k+1}]$ may contain the same point multiple times so that one may have several approximations of $U_k^i \approx u(s_i)$ with different errors

second stage of RK

- ▶ in this stage we determine $u_{k+1} \approx u(t_{k+1})$ from the U_k^i
- ▶ the exact value is

$$u(t_{k+1}) = u(t_k) + \int_{\underline{t_k}}^{t_{k+1}} f(s, u(s)) ds$$

so that the formula for ϕ can be interpreted as a quadrature rule for the integral with quadrature points s_j and weights c_j but note that unlike for usual quadrature rules one may have duplicate quadrature points

- ▶ for given points s_j the weights c_j are determined such that
 - ▶ the quadrature rule is highly accurate
 - ▶ lower order (larger) errors in U_k^j are cancelled

design of Runge-Kutta methods

- ▶ the determination of the coefficients b_{ij} and c_j and the points $s_i = t_k + a_i h$ defining the Runge-Kutta method are an advanced topic which cannot be covered in more detail here
 - ▶ one considers approximation and stability aspects
- ▶ Runge-Kutta methods are probably the most widely used solution methods for ODE initial value problems and are also used for initial value problems for PDEs (partial differential equations)

references

- ▶ Peter Henrici, *Discrete Variable Methods in Ordinary Differential Equations*, John Wiley, 1961
- ▶ John C. Butcher, *The Numerical Analysis of Ordinary Differential Equations*, John Wiley, 1987
- ▶ A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*, Cambridge Uni Press 1996
- ▶ E. Hairer, S.P. Norsett, G. Wanner, *Solving Ordinary Differential Equations I – Nonstiff Problems*, Springer 2008, 2nd rev. ed

Butcher tableau

- ▶ often the coefficients a_i , b_{ij} and c_j are included in a block matrix

$$T = \begin{vmatrix} a & B \\ 0 & c^T \end{vmatrix}$$

which is called the *Butcher tableau* after one of the major contributors John Butcher from Auckland

- ▶ note that $a_1 = 0$ and $b_{0j} = 0$ and are sometimes omitted from the tableau

example 1: Euler's method

- ▶ many classical one-step methods actually are Runge-Kutta methods
- ▶ Stage 1 is trivial for Euler
 - ▶ $m = 1$ and $s_1 = t_k$ so that



$$\underline{U_k^1} = u_k$$

- ▶ Stage 2

$$\phi(t_k, u_k) = \underline{f(s_1, U_k^1)} = f(t_k, u_k)$$

- ▶ rectangle rule

example 2: Heun's method

- ▶ Stage 1

- ▶ $m = 2$, $s_1 = t_k$ and $s_2 = t_{k+1} = t_k + h$



$$U_k^1 = u_k$$

$$U_k^2 = u_k + hf(s_1, U_k^1) \quad \text{Euler}$$

- ▶ U_k^2 uses rectangle rule

- ▶ Stage 2

$$\phi(t_k, u_k) = 0.5f(s_1, \underline{U_k^1}) + 0.5f(s_2, \underline{U_k^2})$$

- ▶ the associated quadrature rule is the trapezoidal rule

example 3: midpoint method

- ▶ Stage 1

- ▶ $m = 2$, $s_1 = t_k$ and $s_2 = t_k + h/2$



$$U_k^1 = u_k$$

$$U_k^2 = u_k + 0.5hf(s_1, U_k^1)$$

- ▶ U_k^2 uses rectangle rule

- ▶ Stage 2

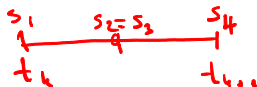
$$\phi(t_k, u_k) = f(s_2, U_k^2)$$

- ▶ the associated quadrature rule is the midpoint rule

example 4: fourth order Runge Kutta method

► Stage 1

- $m = 4$, $s_1 = t_k$, $s_2 = s_3 = t_k + h/2$ and $s_4 = t_k + h$



$$U_k^1 = u_k$$

$$U_k^2 = u_k + 0.5hf(s_1, U_k^1) \quad \text{Euler}$$

$$U_k^3 = u_k + 0.5hf(s_2, U_k^2)$$

$$U_k^4 = u_k + hf(s_3, U_k^3)$$

- U_k^2 uses rectangle rule
- U_k^3 uses right-handed rectangle rule
- U_k^4 uses midpoint rule

► Stage 2

$$\phi(t_k, u_k) = \frac{1}{6}(f(s_1, U_k^1) + 2f(s_2, U_k^2) + 2f(s_3, U_k^3) + f(s_4, U_k^4))$$

- the associated quadrature rule is Simpson's rule

example 5: 3/8-rule fourth-order method

► stage 1

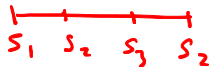
- $m = 4$, $s_1 = t_k$, $s_2 = t_k + h/3$, $s_3 = t_k + 2h/3$ and $s_4 = t_k + h$

$$U_k^1 = u_k$$

$$U_k^2 = u_k + h/3 f(s_1, U_k^1)$$

$$U_k^3 = u_k + -h/3 f(s_1, U_k^1) + hf(s_2, U_k^2)$$

$$U_k^4 = u_k + hf(s_1, U_k^1) - hf(s_2, U_k^2) + hf(s_3, U_k^3)$$



- U_k^2 uses rectangle rule
- U_k^3 and U_k^4 do not use standard rules

► stage 2

$$\phi(t_k, u_k) = \frac{1}{8} (f(s_1, U_k^1) + 3f(s_2, U_k^2) + 3f(s_3, U_k^3) + f(s_4, U_k^4))$$

- the associated quadrature rule is Simpson's 3/8 rule

Computational exploration

Example

Consider the IVP

$$\frac{du(t)}{dt} = -4t(1+t^2)u(t)^2, \quad u(0) = 1.$$

You can check that the exact solution is

$$u(t) = \frac{1}{(t^2 + 1)^2}.$$

example run of library RK routine DOPRI5 from odepack, s

set the ode and integrator

`f = lambda t,u : -4*t*(1.0+t**2)*u**2` *# rhs of ODE*

`solver = scint.ode(f)`

`solver.set_integrator('dopri5')`

initial conditions

`t0 = 0.0; u0 = 1.0`

`solver.set_initial_value(u0, t0)`

compute solution

`n = 20`

`tn = linspace(0,1.0,n)`

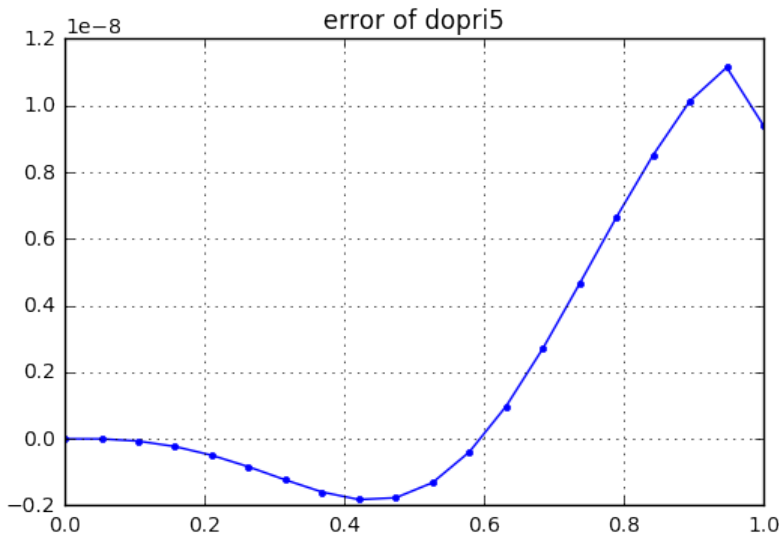
`uex = 1.0/(tn**2+1.0)**2` *# exact solution*

`unum = [u0,]`

`for t in tn[1:]:`

`unum.append(solver.integrate(t)[0])`


```
plt.title('error of dopri5')  
plt.plot(tn, unum-uex, '-.');  
plt.grid('on')
```



study accuracy vs cost (number of integration steps) of dopri5

```
# set the ode and integrator
```

```
f = lambda t,u : -4*t*(1.0+t**2)*u**2
```

```
def solout(t,u):  
    ts.append(t)
```

```
t0 = 0.0; u0 = 1.0    # initial conditions
```

```
tn = 1.0
```

```
uex = 1.0/(tn**2+1.0)**2    # exact solution
```

```
for itol in range(5,15):  
    atol = 10**(-itol); solver = scint.ode(f)  
    solver.set_integrator('dopri5', atol=atol, rtol=0)  
    ts = []; solver.set_solout(solout) # also done at every  
    solver.set_initial_value(u0, t0)  
    unum = solver.integrate(t)[0]  
    print(len(ts), atol, (unum-uex)/atol)
```

```
11 1e-05 0.519207076399  
14 1e-06 0.33133525329  
19 1e-07 0.297607787858  
27 1e-08 0.270854844109  
39 1e-09 0.246938580695  
60 1e-10 0.210362283148  
92 1e-11 0.198763228099  
144 1e-12 0.198230321047  
226 1e-13 0.194289029309  
355 1e-14 0.210942374679
```

documentation for the ode solvers

```
help(solver)
```

Help on ode in module scipy.integrate._ode object:

```
class ode(builtins.object)
|   A generic interface class to numeric integrators.
|
|   Solve an equation system :math:`y'(t) = f(t,y)` with (c
|
|   *Note*: The first two arguments of ``f(t, y, ...)`` are
|   opposite order of the arguments in the system definition
|   by `scipy.integrate.odeint`.
|
|   Parameters
|   -----
|   f : callable ``f(t, y, *f_args)``
|       Right-hand side of the differential equation. t is
|       ``y.shape == (n,)``
```

Reference

A. C. Hindmarsh, *ODEPACK, A Systematized Collection of ODE Solvers*, in vol. 1 of IMACS Transactions on Scientific Computation), pp. 55-64, 1983.