

Solving a System of First-order Ordinary Differential Equations with a Runge-Kutta-Fehlberg Method

This *Numerit* program (demodeq2) demonstrates how to solve a system of first-order vector differential equations using a *variable step size* Runge-Kutta-Fehlberg method. The function that solves this problem is called `rkf78`.

This function solves a system of first-order vector differential equations of the form

$$\frac{d\vec{y}}{dt} = f(\vec{y}, t) \quad (1)$$

Equation (1) indicates that the system of differential equations may be a function of the independent vector \vec{y} and time t .

The syntax of the `rkf78` function is as follows:

```
function rkf78 (neq, ti, tf, h, tetol, x, xout)
` solve first order system of differential equations
` Runge-Kutta-Fehlberg 7(8) method
` input
` neq = number of differential equations
` ti = initial simulation time
` tf = final simulation time
` h = initial guess for integration step size
` tetol = truncation error tolerance (non-dimensional)
` x = integration vector at time = ti
` output
` xout = integration vector at time = tf
```

The function `rkf78` requires a user-coded *Numerit* function that evaluates the system of vector differential equations. The correct syntax for this function is as follows:

```
function orbeqm (t, y, ydot)
` first order equations of orbital motion
` cowell's method
` input
` t = simulation time (seconds)
` y = state vector
` output
` ydot = integration vector
```

Numerical Analysis with Numerit

In the parameter list, t is the current integration time, y is the vector of independent variables, and $ydot$ is the vector of first-order derivatives. Prior to calling the `rkf78` function the user needs to *redirect* the actual name of the function to solve with a statement similar to `orbeqm -> eqm1`. The user must also remember to initialize the `rkf78` function by including the statement `rkcoef = 1` in the main program and making this flag available by also including the statement `common rkcoef` in the main program.

The differential equations function for this example is

```
function eqm1(t, y, ydot)
  ` first order equations of orbital motion
  ` Keplerian motion - no perturbations
  ` input
  ` t = simulation time
  ` y = state vector at time = t
  ` output
  ` ydot = integration vector at time = t
  ` Numerical Analysis with Numerit
  .....
  ` first order equations of orbital motion
  ` input
  ` t = simulation time (seconds)
  ` y = state vector
  ` output
  ` ydot = integration vector
  ` Numerical Analysis with Numerit
  .....
mu = 398600.5

r2 = y[1] * y[1] + y[2] * y[2] + y[3] * y[3]
r1 = sqrt(r2)
r3 = r2 * r1
  ` integration vector
ydot[1] = y[4]
ydot[2] = y[5]
```

Numerical Analysis with Numerit

```
ydot[3] = y[6]
ydot[4] = -mu * y[1] / r3
ydot[5] = -mu * y[2] / r3
ydot[6] = -mu * y[3] / r3
```

The demonstration program solves the system of six vector differential equations that describe the "unperturbed" or *Keplerian* motion of an Earth-orbiting spacecraft. The program requires an initial time, final time, initial step size and truncation error tolerance.

Typical initial conditions for the position and velocity of the spacecraft are "hardwired" within the program. The position vector is in units of feet and the velocity vector input is feet per second. Each vector is converted to metric units before the integration is performed. The demonstration program will print the results of the numerical integration in metric units.

Typical position (feet) and velocity vectors (feet per second) for the Space Shuttle are as follows:

```
-19472500.3, 6587457.02, 7367882.5
-4687.10293, -23436.308, 8566.3774
```

The system of three second-order vector differential equations of *Keplerian* or unperturbed orbital motion are given by:

$$\begin{aligned}\frac{d^2 r_x}{dt^2} &= -\mu \frac{r_x}{R^3} \\ \frac{d^2 r_y}{dt^2} &= -\mu \frac{r_y}{R^3} \\ \frac{d^2 r_z}{dt^2} &= -\mu \frac{r_z}{R^3}\end{aligned}\tag{2}$$

where

$r_x, r_y, r_z =$ position vector of the spacecraft
 $R = \sqrt{r_x^2 + r_y^2 + r_z^2} =$ position magnitude
 $\mu =$ gravitational constant of the Earth

We can use a technique called *order reduction* to create an equivalent first-order system of differential equations. With state variable substitutions defined by

$$\begin{aligned}y_1 &= r_x & y_2 &= r_y & y_3 &= r_z \\ y_4 &= v_x & y_5 &= v_y & y_6 &= v_z\end{aligned}\tag{3}$$

Numerical Analysis with Numerit

we have the following system of equivalent *first-order* vector differential equations:

$$\begin{aligned} \dot{y}_1 &= v_x & \dot{y}_2 &= v_y & \dot{y}_3 &= v_z \\ \dot{y}_4 &= \dot{v}_x & \dot{y}_5 &= \dot{v}_y & \dot{y}_6 &= \dot{v}_z \end{aligned} \quad (4)$$

and the equivalent initial conditions given by:

$$\begin{aligned} y_1(t_0) &= r_x(t_0) & y_2(t_0) &= r_y(t_0) & y_3(t_0) &= r_z(t_0) \\ y_4(t_0) &= v_x(t_0) & y_5(t_0) &= v_y(t_0) & y_6(t_0) &= v_z(t_0) \end{aligned} \quad (5)$$

The orbital period is the time required for the Space Shuttle to complete one orbit around the Earth. For this example the orbital period is 5404.135 seconds. If we numerically integrate the equations of motion for exactly one orbital period the Space Shuttle should return very close to the initial conditions given above.

This software can be used to assess the effect of different truncation error tolerances on how well the orbit returns to the initial conditions by integrating for exactly one or more orbital periods. This type of analysis can help you evaluate the "closure" errors and "calibrate" the integrator for any particular problem.

The propagation of an orbit forward or backward from a set of initial conditions is also called the orbital initial value problem (IVP).

The following is a typical draft output created with this computer program.

```
program demodeq2
< solution of differential equations with RKF78 method >
truncation error tolerance 1e-08
position vector error (meters)
0.00448080299975118 0.0396239993278868 -0.0128639730974101
velocity vector error (meters/second)
-4.27685571402492e-05 1.68226019781059e-05 1.55433865778321e-05
orbital period 5404.13510403513 seconds
```

For this example the truncation error tolerance was 1.0e-8, the initial step size guess was 10 seconds and the integration period was exactly one orbital period.