

Lecturer: *D. J. Ivers*

Assignment 2 — Initial Value Problems for Systems of ODE's

Due 5pm Thursday 2nd June.

Read the course messages for general information on assignments, the form your Assignment 2 file must take and how to submit it. This assignment is worth 4 marks.

1. Attached is a listing of the file `$mc3nm/ass2.inc` which contains an incomplete main program implementing the classical fourth-order Runge-Kutta for solving the second-order initial value problem

$$My'' + ry' + Mg = 0, \quad y(0) = H, \quad y'(0) = 0,$$

for an object of mass M dropped from a vertical height H in a uniform gravitational field g and subject to an air resistance proportional to the velocity. Assume $M = 10$ kg, $r = 0$, $g = 10 \text{ m}^2 \text{ s}^{-1}$ and $H = 10$ m. In `$mc3nm/ass2.inc` a question mark `?` indicates a missing single character, three horizontal dots `...` indicates a missing part or the missing whole of a line and three vertical dots indicates a missing block of code. Copy the file to your directory and complete the program to have the following features:

- (a) The problem parameters (`mass` (for M), `r` and `g` in Question 1; `mu` and `nut` in Question 2) are input from a file called `parameters.dat` using a namelist `parameters` and echoed to the screen. Copy the file `$mc3nm/parameters.dat` to `parameters.dat`.
- (b) The arrays `y`, `yrk4`, `k1`, `k2`, `k3`, `k4` are declared and dimensioned in the main program.
- (c) The initial time `t0`, the final time `tn`, the number of steps `n`, the output frequency `nprint` and the vector `y0` of initial values are input from a file using a namelist `run_data` and echoed to the screen. Use the array `y` for the vector `y0` of initial values. The name of the data file is input from the workstation with a prompt.
- (d) The values of `t` and `y` are written into a file with appropriate headers. An implied do-loop is not used for `y`. The name of the output file is input from the workstation with a prompt.
- (e) The step-size `h` is set by `t0`, `tn` and `n`.
- (f) Values of `t` and `y` are printed every `nprint` steps with format descriptors `F14.5` and `E14.5`, respectively.

- (g) The first-order system of ode's is given by the array-valued internal function `f`. Local arrays `y` and `f_result` are declared and dimensioned in `f`.
- (h) The fourth-order Runge-Kutta subroutine is `rk4` from Set 5.

2. Consider the following Earth-Moon-spaceship problem,

$$x'' = 2y' + x - \frac{\tilde{\mu}(x + \mu)}{r_1^3} - \frac{\mu(x - \tilde{\mu})}{r_2^3},$$

$$y'' = -2x' + y - \frac{\tilde{\mu}y}{r_1^3} - \frac{\mu y}{r_2^3},$$

where $\tilde{\mu} = 1 - \mu$, $r_1^2 = (x + \mu)^2 + y^2$ and $r_2^2 = (x - \tilde{\mu})^2 + y^2$. The coordinates of the spaceship are (x, y) in a reference frame rotating with the Earth and the Moon, which are at $(-\mu, 0)$ and $(\tilde{\mu}, 0)$, respectively. The equations have been scaled so that distance is measured in units of the Earth-Moon distance R , time in units of Ω^{-1} , where Ω is the angular speed of the Moon about the Earth and velocity in units of $R\Omega$. Since the mass of the spaceship is negligible compared to the masses of the Earth and Moon, the Earth-Moon-spaceship problem is a case of the restricted three-body problem.

Modify your program from Question 1 to solve the Earth-Moon-spaceship problem. Use the following initial values of the dependent variables

$$x(0) = 0.994, \quad y(0) = 0, \quad x'(0) = 0, \quad y'(0) = -2.0015851,$$

for $\mu = 0.012277471$. Take `t0=0`, `tn=17.5`, `n=20000` and `nprint=50`.

- (a) Show numerically that the orbit is periodic and find the period as accurately as possible.
- (b) Find the minimum radius from the Moon to the orbit. Use a block-if in the `t_loop` to determine the latest minimum radius `minr2` and the corresponding time `t`, position (x, y) and velocity (x', y') [use an array `ymin`]. Output the results to the screen with appropriate headers after completing the `t_loop`. From your output file (output t, x, y, x', y', r_2) determine the time(s), position(s) and velocity(ies) of any other points of the orbit, where the minimum radius occurs.
- (c) Use `funplot` to plot the orbit.

Read the course messages for details on what to hand in.

```

program rk4sysmain
!-----
! This program solves the initial value problem for a system
! of first-order differential equations using the classical
! fourth-order Runge-Kutta method.
!-----
! Declare problem size:
integer, parameter :: m=?
! Declare program variables:
real :: mass,...
integer :: n, ...
real :: t0, ...
real, dimension(m) :: y, ...
character(len=20) :: datafile, ??????
namelist /??????????/ mass,...
????????? /run_data/ t0,...,y
!...Open and enter parameters from file parameters.dat:
open (unit=7, file='parameters.dat', status='old')
read (7,nml=parameters)
close(7)
!...Echo parameters to terminal:
write (*,nml=parameters)
!...Name, open and enter run_data from data_file:
write (*,*) ' Enter run_data file name:'
read (*,10) ??????????
10 format (a)
open (?????=7, file=datafile, status='old')
read (7,nml=...)
close(7)
!...Echo run_data to terminal:
write (*,???=run_data)
! Name, open and head output file:
write (*,*) ' Enter output file name:'
read (*,10) output
open (unit=8, file=???????, status='unknown')
write (8,30)
30 format(' t y y'' ') !Q1 (adjust spacing if needed)
!30 format(' t x ... !Q2
! Initialise values for D0-loop (y initialised directly by input):
t = ...
h = ...
! Initialise minr2 for Q2 only:
! r2min = ... !Q2
t_loop: do j= 1, ?
! 4th order Runge-Kutta subroutine:
call rk4
y = ???
t = ...

```

```

! Print solutions every nprint steps:
if ( mod(j,nprint)==0 ) then
!   r2 = ... !Q2
!   write (8,40) t, ..., r2 !Q2
!   write (8,40) t, ...
40  format (?????, ??????)
endif

! The following code is for Question 2 only:
!r2 = ...
!Calculate the minimum of r2:
!if (r2min > r2) then
!  r2min= ...
!  .
!endif

enddo t_loop

! The following ... is for Question 2 only:
! Output the minimum of r2:
!  .
!  .

stop

?????????
!-----
function f(t,y) result(f_result)
real, intent(in):: t
real, dimension(m), intent(in):: y
real, dimension(m) :: f_result
f_result(1) = ...
!  .
!  .
!  .

end ?????????? ?
!-----
subroutine rk4
! Purpose: To implement the classical fourth-order Runge-Kutta
! method for the system of M first-order differential equations
!  $y'=f(t,y)$  over one interval of width h. Thus subroutine rk4
! approximates  $y(t+h)$  with yrk4, given  $y=y(t)$ .

k1 = h*f(t,y)
k2 = h*f(t + h/2.,y + k1/2.)
k3 = h*f(t + h/2.,y + k2/2.)
k4 = h*f(t+h,y+k3)
yrk4 = y + (k1 + 2.*k2 + 2.*k3 + k4)/6.

return
end subroutine rk4
!-----
end ...

```