NUMERIČKE METODE I MATEMATIČKO MODELIRANJE

5. PREDAVANJE

PRIMJENA NUMERIČKIH METODA U RJEŠAVANJU DIFERENCIJALNIH JEDNADŽBI

(dio II)

RUNGE-KUTTA METODE

- Runge-Kutta metode su zasnovane na Taylorovom razvoju, bolji algoritam od Eulerove metode za rješavanje diferencijalnih jednadžbi

- uključuje međukorak u izračunu y_{i+1}

- Treba riješiti diferencijalnu jednadžbu $\frac{dy}{dt} = f(t, y)$

$$y(t) = \int f(t, y)dt$$

$$y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y)dt$$

RUNGE-KUTTA METODA DRUGOG REDA (RK2)

- prva aproksimacija uključujeTaylorov razvoj f(t,y) oko središta intervala za integraciju, tj. $t_i + h/2$

$$y(t_i + h/2) = y_{i+1/2}$$

$$t_i + h/2 = t_{i+1/2}$$

$$\int_{t_i}^{t_{i+1}} f(t,y)dt \approx hf(t_{i+1/2}, y_{i+1/2}) + O(h^3)$$

$$\Rightarrow \quad y_{i+1} = y_i + hf(t_{i+1/2}, y_{i+1/2}) + O(h^3)$$

- za određivanje vrijednosti $y_{i+1/2}$ primjenjuje se Eulerova metoda

$$y_{(i+1/2)} = y_i + \frac{h}{2}\frac{dy}{dt} = y(t_i) + \frac{h}{2}f(t_i, y_i)$$

RUNGE-KUTTA METODA DRUGOG REDA (RK2)

- algoritam RK2 metode:

$$k_1 = hf(t_i, y_i)$$

$$k_2 = hf(t_{i+1/2}, y_i + k_1/2)$$

$$y_{i+i} \approx y_i + k_2 + O(h^3)$$

- za razliku od metoda sa jednim korakom, Runge-Kutta metoda ima međukorak u proračunu, $t_i + h/2 = t_{(i+1/2)}$

- RK metode zahtjevaju izvođenje više operacija, no daju veću stabilnost rješenja

RUNGE-KUTTA METODA ČETVRTOG REDA (RK4)

$$k_1 = hf(t_i, y_i)$$

$$k_2 = hf(t_i + h/2, y_i + k_1/2)$$

$$k_3 = hf(t_i + h/2, y_i + k_2/2)$$

$$k_4 = hf(t_i + h, y_i + k_3)$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

- prvo se računa k_1 sa t_i, y_1 i f
- zatim se uveća korak za h/2 i računa k_2, k_3, i k_4
- na kraju se određuje nova vrijednost varijable y

ZADATAK 5

Primjenom Runge-Kutta metode četvrtog reda, treba riješiti problem nelinearnog oscilatora sa gušenjem i vanjskom periodičnom silom opisan Duffingovom jednadžbom :

$$\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} - \omega^2 x + bx^3 = f cos(\Omega t)$$

Zadani su parametri jednadžbe $\gamma = 0.02, b = 0.2, \omega = 0.9, \Omega = 1.7$

Treba podesiti amplitudu vanjske periodične sile (f) tako da se dobije kaotično rješenje (za proizvoljne početne uvjete) i nacrtati odgovarajuće grafove putanje x(t), faznog prostora v(x), i Poincareovu mapu (sadrži presjeke faznog prostora u vremenskim koracima t=0,T,2T,3T,… gdje je T=2π/Ω period vanjske sile).

Poincareovu mapu treba računati dovoljno dugo u vremenu tako da se postigne dovoljna gustoća točaka na mapi.

PRIMJER PROGRAMA SA RUNGE-KUTTA METODOM ZA H.O.

```
// /*    This program solves Newton's equation for a block
      sliding on a horizontal frictionless surface. The block
      is tied  to a wall with a spring, and Newton's equation
      takes the form
           m d^2x/dt^2 =-kx
      with k the spring tension and m the mass of the block.
      The angular frequency is omega^2 = k/m and we set it equal
      1 in this example program.

      Newton's equation is rewritten as two coupled differential
      equations, one for the position x  and one for the velocity v
           dx/dt = v    and
           dv/dt = -x   when we set k/m=1

      We use therefore a two-dimensional array to represent x and v
      as functions of t
      y[0] == x
      y[1] == v
      dy[0]/dt = v
      dy[1]/dt = -x

      The derivatives are calculated by the user defined function
      derivatives.

      The user has to specify the initial velocity (usually v_0=0)
      the number of steps and the initial position. In the programme
      below we fix the time interval [a,b] to [0,2*pi].

*/
```

```
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "lib.h"
using namespace  std;
// output file as global variable
ofstream ofile;
// function declarations
void derivatives(double, double *, double *);
void initialise ( double&, double&, int&);
void output( double, double *, double);
void runge_kutta_4(double *, double *, int, double, double,
                   double *, void (*)(double, double *, double *));

int main(int argc, char* argv[])
{
//  declarations of variables
  double *y, *dydt, *yout, t, h, tmax, E0;
  double initial_x, initial_v;
  int i, number_of_steps, n;
  char *outfilename;
  // Read in output file, abort if there are too few command-line arguments
  if( argc <= 1 ){
    cout << "Bad Usage: " << argv[0] <<
      " read also output file on same line" << endl;
    exit(1);
  }
  else{
    outfilename=argv[1];
  }
  ofile.open(outfilename);
  //  this is the number of differential equations
  n = 2;
```

```
//  allocate space in memory for the arrays containing the derivatives
  dydt = new double[n];
  y = new double[n];
  yout = new double[n];
  // read in the initial position, velocity and number of steps
  initialise (initial_x, initial_v, number_of_steps);
  //  setting initial values, step size and max time tmax
  h = 4.*acos(-1.)/( (double) number_of_steps);   // the step size
  tmax = h*number_of_steps;                       // the final time
  y[0] = initial_x;                               // initial position
  y[1] = initial_v;                               // initial velocity
  t=0.;                                           // initial time
  E0 = 0.5*y[0]*y[0]+0.5*y[1]*y[1];               // the initial total energy
  // now we start solving the differential equations using the RK4 method
  while (t <= tmax){
    derivatives(t, y, dydt);   // initial derivatives
    runge_kutta_4(y, dydt, n, t, h, yout, derivatives);
    for (i = 0; i < n; i++) {
      y[i] = yout[i];
    }
    t += h;
    output(t, y, E0);   // write to file
  }
  delete [] y; delete [] dydt; delete [] yout;
  ofile.close();  // close output file
  return 0;
}   //  End of main function
```

```
//     Read in from screen the number of steps,
//     initial position and initial speed
void initialise (double& initial_x, double& initial_v, int& number_of_steps)
{
 cout << "Initial position = ";
 cin >> initial_x;
 cout << "Initial speed = ";
 cin >> initial_v;
 cout << "Number of steps = ";
 cin >> number_of_steps;
}  // end of function initialise


//   this function sets up the derivatives for this special case
void derivatives(double t, double *y, double *dydt)
{
  dydt[0]=y[1];    // derivative of x
  dydt[1]=-y[0]; // derivative of v
} // end of function derivatives
```

```
//    function to write out the final results
void output(double t, double *y, double E0)
{
  ofile << setiosflags(ios::showpoint | ios::uppercase);
  ofile << setw(15) << setprecision(8) << t;
  ofile << setw(15) << setprecision(8) << y[0];
  ofile << setw(15) << setprecision(8) << y[1];
  ofile << setw(15) << setprecision(8) << cos(t);
  ofile << setw(15) << setprecision(8) <<
    0.5*y[0]*y[0]+0.5*y[1]*y[1]-E0 << endl;
}  // end of function output
```

```
/*   This function upgrades a function y (input as a pointer)
     and returns the result yout, also as a pointer. Note that
     these variables are declared as arrays.  It also receives as
     input the starting value for the derivatives in the pointer
     dydx. It receives also the variable n which represents the
     number of differential equations, the step size h and
     the initial value of x. It receives also the name of the
     function *derivs where the given derivative is computed
*/
void runge_kutta_4(double *y, double *dydx, int n, double x, double h,
                   double *yout, void (*derivs)(double, double *, double *))
{
  int i;
  double      xh,hh,h6;
  double *dym, *dyt, *yt;
  //   allocate space for local vectors
  dym = new double [n];
  dyt =  new double [n];
  yt =  new double [n];
  hh = h*0.5;
  h6 = h/6.;
  xh = x+hh;
```

```
  for (i = 0; i < n; i++) {
    yt[i] = y[i]+hh*dydx[i];
  }
  (*derivs)(xh,yt,dyt);     // computation of k2, eq. 3.60
  for (i = 0; i < n; i++) {
    yt[i] = y[i]+hh*dyt[i];
  }
  (*derivs)(xh,yt,dym); //  computation of k3, eq. 3.61
  for (i=0; i < n; i++) {
    yt[i] = y[i]+h*dym[i];
    dym[i] += dyt[i];
  }
  (*derivs)(x+h,yt,dyt);    // computation of k4, eq. 3.62
  //      now we upgrade y in the array yout
  for (i = 0; i < n; i++){
    yout[i] = y[i]+h6*(dydx[i]+dyt[i]+2.0*dym[i]);
  }
  delete []dym;
  delete [] dyt;
  delete [] yt;
}       //  end of function Runge-kutta 4
```