Numeričke metode

Aleksandar Maksimović
IRB

ODJ

Opisat ćemo nekoliko najčešćih numeričkih metoda za rješavanje običnih diferencijalnih jednadžbi (skraćeno ODJ) oblika

$$y'(x) = f(x, y(x)), \quad y \in (a, b),$$

uz zadani početni uvjet $y(a) = y_0$ ili uz zadani rubni uvjet $r(y(a), y(b)) = 0$, gdje je r neka zadana funkcija.

Sustav običnih diferencijalnih jednadžbi je općenitiji problem:

$$\begin{aligned} y_1' &= f_1(x, y_1, \ldots, y_n), \\ y_2' &= f_2(x, y_1, \ldots, y_n), \\ &\vdots \\ y_n' &= f_n(x, y_1, \ldots, y_n). \end{aligned}$$

$$\mathbf{y} = [y_1, \ldots, y_n]^T \quad \text{i} \quad \mathbf{f} = [f_1, \ldots, f_n]^T$$

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x)),$$

ODJ

Diferencijalne jednadžbe višeg reda

$$y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)})$$

supstitucijama

$$y_1 = y, \quad y_2 = y', \quad \ldots, \quad y_n = y^{(n-1)}$$

svodimo na sustav jednadžbi prvog reda:

$$\begin{aligned}
y_1' &= y' = y_2, \\
y_2' &= y'' = y_3, \\
&\vdots \\
y_n' &= y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)}) = f(x, y_1, y_2, y_3, \ldots, y_n),
\end{aligned}$$

Eulerova metoda

Eulerova metoda je zasigurno najjednostavnija metoda za rješavanje inicijalnog problema za ODJ oblika

$$y' = f(x, y), \quad y(a) = y_0.$$

Metoda se zasniva na ideji da se y' u gornjoj jednadžbi zamijeni s podijeljenom razlikom

$$y'(x) = \frac{y(x+h) - y(x)}{h} + \mathcal{O}(h),$$

Zanemarivanjem kvadratnog člana u gornjem razvoju dobivamo aproksimaciju

$$y(x+h) \approx y(x) + hf(x, y(x)).$$

$$h = \frac{b-a}{n}, \quad x_i = a + ih,$$

$i = 0, \ldots, n.$

prvo aproksimiramo rješenje u točki $x_1 = a + h$

$$y(x_1) \approx y_1 = y_0 + hf(x_0, y_0).$$

Eulerova metoda

$x_2 = x_1 + h$:

$$y_2 = y_1 + hf(x_1, y_1),$$

Opisani postupak nazivamo Eulerova metoda, i možemo ga kraće zapisati rekurzijom

$$y_{i+1} = y_i + hf(x_i, y_i), \quad i = 1, \ldots, n,$$

RK metode

Koristeći sličnu ideju kao u Eulerovoj metodi, diferencijalnu jednadžbu

$$y' = f(x, y), \quad y(a) = y_0$$

na intervalu $[a, b]$, možemo rješavati tako da podijelimo interval $[a, b]$ na n jednakih podintervala, označivši

$$h = \frac{b-a}{n}, \quad x_i = a + ih, \quad i = 0, \dots, n.$$

Sada y_{i+1}, aproksimaciju rješenja u točki x_{i+1}, računamo iz y_i korištenjem aproksimacije oblika

$$y(x+h) \approx y(x) + h\Phi(x, y(x), h; f),$$

te dobivamo rekurziju:

jednokoračne metode

$$y_{i+1} = y_i + h\Phi(x_i, y_i, h; f), \quad i = 0, \dots, n-1.$$

RK metode

Funkciju Φ nazivamo **funkcija prirasta**, a različit izbor te funkcije definira različite metode. Uočimo da je funkcija f iz diferencijalne jednadžbe (10.3.1) parametar od Φ (tj. Φ zavisi o f). Tako je npr. u Eulerovoj metodi

$$\Phi(x, y, h; f) = f(x, y).$$

O odabiru funkcije Φ ovisi i točnost metode.

$$\Delta(x; h) = \frac{y(x+h) - y(x)}{h},$$

nazivamo **lokalna pogreška diskretizacije.**

Runge–Kuttine metode.

$$\Phi(x, y, h) = \sum_{j=1}^{r} \omega_j k_j(x, y, h),$$

RK metode

a k_j su zadani s

$$k_j(x,y,h) = f\Big(x + c_j h, y + h\sum_{l=1}^{r} a_{jl} k_l(x,y,h,f)\Big), \quad j = 1,\ldots,r.$$

Broj r zovemo broj stadija Runge–Kuttine (RK) metode, i on označava koliko puta moramo računati funkciju f u svakom koraku.

Različit izbor koeficijenata ω_j, c_j i a_{jl} definira različite metode.

k_i nalazi na lijevoj i na desnoj strani jednadžbe, tj. zadan je implicitno govorimo o **implicitnoj** Runge–Kuttinoj metodi.

$a_{jl} = 0$ za $l \geq j$. Tada k_j možemo izračunati preko $k_1, \ldots, k_{j-1}$,

RK metode nazivamo **eksplicitnima**.

RK metode

Primjer odabira koeficijenata prikazat ćemo na RK metodi s dva stadija:

$$\Phi(x,y,h) = \omega_1 k_1(x,y,h) + \omega_2 k_2(x,y,h),$$
$$k_1(x,y,h) = f(x,y),$$
$$k_2(x,y,h) = f(x+ah, y+ahk_1).$$

Razvojem k_2 u Taylorov red po varijabli h dobivamo

$$k_2(x,y,h) = f + h(f_x a + f_y a f) + \frac{h^2}{2}(f_{xx}a^2 + 2f_{xy}a^2 f + f_{yy}a^2 f^2) + \mathcal{O}(h^3),$$

$$y(x+h) = y(x) + hf + \frac{h^2}{2}(f_x + f_y f) + \frac{h^3}{6}[f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_y(f_x + f_y f)] + \mathcal{O}(h^4).$$

Ovdje smo iskoristili da je $y(x)$ rješenja diferencijalne jednadžbe:

$$y'(x) = f(x,y) = f,$$

RK metode

te pravila za deriviranje

$$y''(x) = f_x + f_y f,$$
$$y'''(x) = f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_y(f_x + f_y f).$$

Sada je lokalna pogreška diskretizacije jednaka

$$\frac{y(x+h) - y(x)}{h} - \Phi(x, y(x), h) = \frac{y(x+h) - y(x)}{h} - (\omega_1 k_1(x, y, h) + \omega_2 k_2(x, y, h))$$
$$= (1 - \omega_1 - \omega_2)f + h(f_x + f_y f)\left(\frac{1}{2} - \omega_2 a\right)$$
$$+ h^2\left[(f_{xx} + 2f_{xy}f + f_{yy}f^2) \cdot \left(\frac{1}{6} - \frac{\omega_2 a^2}{2}\right) + \frac{1}{6}f_y(f_x + f_y f)\right]$$
$$+ \mathcal{O}(h^3).$$

RK metode

$$1 - \omega_1 - \omega_2 = 0.$$ 1. red

$$\frac{1}{2} - \omega_2 a = 0$$ 2. red

$$\omega_2 = t \neq 0, \quad \omega_1 = 1 - t, \quad a = \frac{1}{2t}.$$

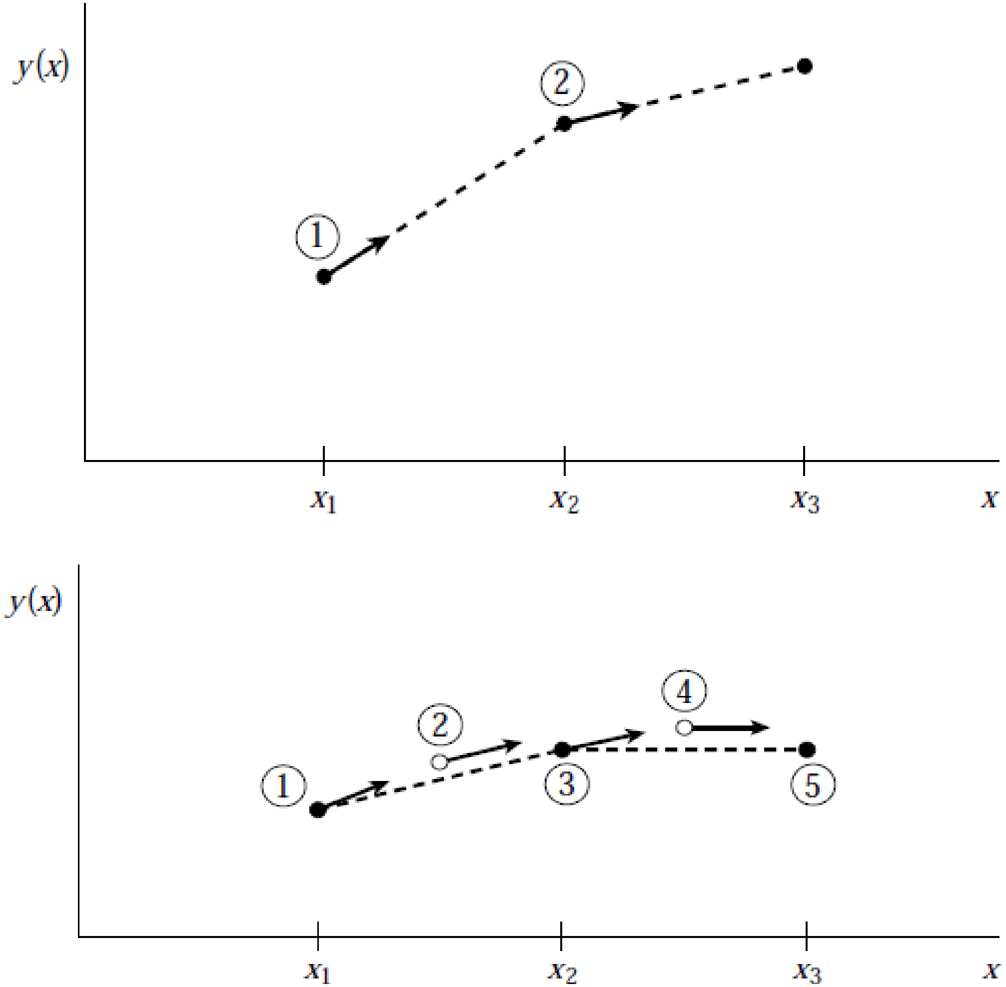
Za $t = 1/2$ dobivamo Heunovu metodu:

$$\Phi = \frac{1}{2}(k_1 + k_2),$$
$$k_1 = f(x, y),$$
$$k_2 = f(x + h, y + hk_1),$$

$t = 1$ dobiva modificirana Eulerova metoda:

$$\Phi = f\left(x + \frac{h}{2}, y + \frac{h}{2} f(x, y)\right).$$

RK metode



Eulerova metoda

Runge Kutta metoda drugog reda ili modificirana Eulerova metoda ili midpoint metoda.
Pogreška prvog reda eliminirana je derivacijama na početku i sredini svakog koraka.

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

$$k_2 = hf\left(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1\right)$$

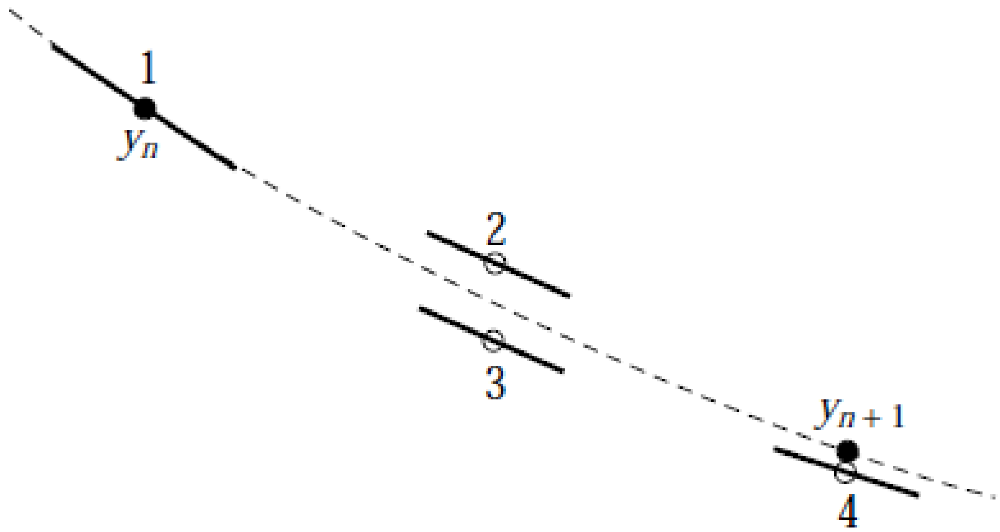
$$y_{n+1} = y_n + k_2 + O(h^3)$$

RK 4

Evo nekoliko primjera RK-4 metoda. Najpopularnija je "klasična" Runge–Kutta metoda, koja se u literaturi najčešće naziva Runge–Kutta ili RK-4 metoda (iako je to samo jedna u nizu Runge–Kutta metoda):

$$
\begin{aligned}
\Phi &= \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),\\
k_1 &= f(x, y),\\
k_2 &= f\left(x + \frac{h}{2}, y + \frac{h}{2}k_1\right),\\
k_3 &= f\left(x + \frac{h}{2}, y + \frac{h}{2}k_2\right),\\
k_4 &= f\left(x + h, y + hk_3\right).
\end{aligned}
$$

RK 4



U svakom koraku 4 puta izračunavamo derivaciju. Pomoću informacija dobivenih derivacijama izračunava se konačna vrijednost funkcije (prikazana punim krugom na slici).

RK 3/8

Spomenimo još i 3/8-sku metodu:

$$\Phi = \frac{1}{8}(k_1 + 3k_2 + 3k_3 + k_4),$$
$$k_1 = f(x, y),$$
$$k_2 = f\left(x + \frac{h}{3}, y + \frac{h}{3}k_1\right),$$
$$k_3 = f\left(x + \frac{2}{3}h, y - \frac{h}{3}k_1 + hk_2\right),$$
$$k_4 = f(x + h, y + h(k_1 - k_2 + k_3))$$

primjer

Svedite na sistem diferencijalnih jednadžbi prvog reda i riješite RK–2 metodom s korakom $h = 0.1$ *diferencijalnu jednadžbu*

$$y'' + 2y' + 3x = 5, \quad y(0) = 1, \quad y'(0) = 2$$

u točki $x = 0.1$.

Označimo s $z = y'$. *Deriviranjem i uvrštavanjem u polaznu jednadžbu dobivamo sistem diferencijalnih jednadžbi*

$$\begin{aligned} y' &= z \\ z' &= 5 - 2z - 3x \end{aligned}$$

uz početne uvjete $y(0) = 1$, $z(0) = 2$. *Rješenje zadatka dobivamo odmah u prvom koraku*

$$\begin{bmatrix} y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ z_0 \end{bmatrix} + \frac{1}{2}\left(\begin{bmatrix} k_{11} \\ k_{12} \end{bmatrix} + \begin{bmatrix} k_{21} \\ k_{22} \end{bmatrix}\right).$$

primjer

Uočimo da je

$$\begin{bmatrix} y' \\ z' \end{bmatrix} = \begin{bmatrix} z \\ 5 - 2z - 3x \end{bmatrix}, \quad \begin{bmatrix} y(0) \\ z(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Odatle, po formuli za RK–2 slijedi

$$\begin{bmatrix} k_{11} \\ k_{12} \end{bmatrix} = 0.1 \begin{bmatrix} 2 \\ 5 - 2 \cdot 2 - 3 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}.$$

Jednako tako, imamo

$$\begin{bmatrix} y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} k_{11} \\ k_{12} \end{bmatrix} = \begin{bmatrix} 1.2 \\ 2.1 \end{bmatrix},$$

odakle izračunavamo k_2

primjer

$$\begin{bmatrix} k_{21} \\ k_{22} \end{bmatrix} = 0.1 \begin{bmatrix} 2.1 \\ 5 - 2 \cdot 2.1 - 3 \cdot 0.1 \end{bmatrix} = \begin{bmatrix} 0.21 \\ 0.05 \end{bmatrix}.$$

Sve zajedno daje

$$\begin{bmatrix} y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \frac{1}{2} \left(\begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.21 \\ 0.05 \end{bmatrix} \right) = \begin{bmatrix} 1.205 \\ 2.075 \end{bmatrix},$$

što znači $y'(0.1) \approx 1.205$ *i* $z(0.1) = y'(0.1) \approx 2.075.$

Adaptivne metode

Iako smo u prošlom potpoglavlju pretpostavili da je korak integracije h konstantan tijekom cijelog postupka rješavanja diferencijalne jednadžbe, očito je da se h može mijenjati u svakom koraku integracije, pa jednokoračnu metodu možemo pisati u obliku:

$$y_{i+1} = y_i + h_i \Phi(x_i, y_i, h_i).$$

Prvo ćemo pokazati kako se određuje duljina koraka h_i tako da bude postignuta neka unaprijed zadana točnost ε.

Neka su s Φ i $\bar{\Phi}$ zadane dvije metode reda p i $p+1$. Tada računamo aproksimacije

$$y_{i+1} = y_i + h_i \Phi(x_i, y_i, h_i),$$
$$\bar{y}_{i+1} = y_i + h_i \bar{\Phi}(x_i, y_i, h_i).$$

Iz (10.3.4) slijedi da je:

$$y(x_i + h_i) = y(x_i) + h_i \Phi(x_i, y(x_i), h_i) + C(x_i) h_i^{p+1} + \mathcal{O}(h_i^{p+2}),$$

RK Fehlberg metode

$$y(x_i + h_i) = y(x_i) + h_i \bar{\Phi}(x_i, y(x_i), h_i) + \mathcal{O}(h_i^{p+2}).$$

Cilj je da pogreška u i-tom koraku bude manja od ε. Stoga ćemo pretpostaviti da je aproksimacija y_i za $y(x_i)$ točna, tj. $y_i = y(x_i)$. Sada oduzimanjem gornje dvije jednadžbe slijedi

$$h_i[\Phi(x_i, y_i, h_i) - \bar{\Phi}(x_i, y_i, h_i)] = C(x_i) h_i^{p+1} + \mathcal{O}(h_i^{p+2}).$$

Iz prve dvije jednakosti oduzimanjem slijedi

$$h_i[\Phi(x_i, y_i, h_i) - \bar{\Phi}(x_i, y_i, h_i)] = \bar{y}_{i+1} - y_{i+1},$$

te uvrštavanjem u (10.3.17) dobivamo

$$y_{i+1} - \bar{y}_{i+1} = C(x_i) h_i^{p+1} + \mathcal{O}(h_i^{p+2}).$$

RK Fehlberg metode

RK 5-tog reda

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

$$k_2 = hf(x_n + a_2 h, y_n + b_{21} k_1)$$

$$\dots$$

$$k_6 = hf(x_n + a_6 h, y_n + b_{61} k_1 + \cdots + b_{65} k_5)$$

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 + c_6 k_6 + O(h^6)$$

RK 4-tog reda

$$y^*_{n+1} = y_n + c^*_1 k_1 + c^*_2 k_2 + c^*_3 k_3 + c^*_4 k_4 + c^*_5 k_5 + c^*_6 k_6 + O(h^5)$$

pogreška

$$\Delta \equiv y_{n+1} - y^*_{n+1} = \sum_{i=1}^{6} (c_i - c^*_i) k_i$$

odnos koraka i pogreške, indeks 0 označava željenu pogrešku

$$h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{0.2}$$

RK Fehlberg metode

Cash-Karp Parameters for Embedded Runga-Kutta Method								
i	a_i	b_{ij}					c_i	c_i^*
1							$\frac{37}{378}$	$\frac{2825}{27648}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0	0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$	$\frac{18575}{48384}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$	$\frac{13525}{55296}$
5	1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0	$\frac{277}{14336}$
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$	$\frac{1}{4}$
$j =$		1	2	3	4	5		

podprogrami

Algoritmi: rk4, rkck, mmid, stoerm

biranje koraka: rkqs, bsstep, stiff

glavni programi: rkdumb i odeint

void rk4(float y[], float dydx[], int n, float x, float h, float yout[], void (*derivs)(float, float [], float []))

Given values for the variables y[1..n] and their derivatives dydx[1..n] known at x, use the fourth-order Runge-Kutta method to advance the solution over an interval h and return the incremented variables as yout[1..n], which need not be a distinct array from y. The user supplies the routine derivs(x,y,dydx), which returns derivatives dydx at x.

```
SUBROUTINE rk4(y,dydx,n,x,h,yout,derivs)
INTEGER n,NMAX
REAL h,x,dydx(n),y(n),yout(n)
EXTERNAL derivs
PARAMETER (NMAX=50) Set to the maximum number of functions.
```

podprogrami

void rkdumb(float vstart[], int nvar, float x1, float x2, int nstep, void (*derivs)(float, float [], float []))

Starting from initial values vstart[1..nvar] known at x1 use fourth-order Runge-Kutta
to advance nstep equal increments to x2. The user-supplied routine derivs(x,v,dvdx)
evaluates derivatives. Results are stored in the global variables y[1..nvar][1..nstep+1]
and xx[1..nstep+1].

SUBROUTINE rkdumb(vstart,nvar,x1,x2,nstep,derivs)

INTEGER nstep,nvar,NMAX,NSTPMX

PARAMETER (NMAX=50,NSTPMX=200) Maximum number of functions and maximum number of values to be stored.

REAL x1,x2,vstart(nvar),xx(NSTPMX),y(NMAX,NSTPMX)

EXTERNAL derivs

COMMON /path/ xx,y Storage of results.

C USES rk4

podprogrami

void rkqs(float y[], float dydx[], int n, float *x, float htry, float eps, float yscal[], float *hdid, float *hnext, void (*derivs)(float, float [], float []))

Fifth-order Runge-Kutta step with monitoring of local truncation error to ensure accuracy and adjust stepsize. Input are the dependent variable vector y[1..n] and its derivative dydx[1..n] at the starting value of the independent variable x. Also input are the stepsize to be attempted htry, the required accuracy eps, and the vector yscal[1..n] against which the error is scaled. On output, y and x are replaced by their new values, hdid is the stepsize that was actually accomplished, and hnext is the estimated next stepsize. derivs is the user-supplied routine that computes the right-hand side derivatives.

```
SUBROUTINE rkqs(y,dydx,n,x,htry,eps,yscal,hdid,hnext,derivs)
INTEGER n,NMAX
REAL eps,hdid,hnext,htry,x,dydx(n),y(n),yscal(n)
EXTERNAL derivs
PARAMETER (NMAX=50) Maximum number of equations.
C USES derivs,rkck
```

podprogrami

```
extern int kmax,kount;
extern float *xp,**yp,dxsav;
```

User storage for intermediate results. Preset kmax and dxsav in the calling program. If kmax != 0 results are stored at approximate intervals dxsav in the arrays xp[1..kount], yp[1..nvar] [1..kount], where kount is output by odeint. Dening declarations for these variables, with memory allocations xp[1..kmax] and yp[1..nvar][1..kmax] for the arrays, should be in the calling program.

```
void odeint(float ystart[], int nvar, float x1, float x2, float eps, float h1,  float hmin, int *nok, int *nbad,
        void (*derivs)(float, float [], float []),  void (*rkqs)(float [], float [], int, float *, float, float, float [],
        float *, float *, void (*)(float, float [], float [])))
```

Runge-Kutta driver with adaptive stepsize control. Integrate starting values ystart[1..nvar] from x1 to x2 with accuracy eps, storing intermediate results in global variables. h1 should be set as a guessed first stepsize, hmin as the minimum allowed stepsize (can be zero). On output nok and nbad are the number of good and bad (but retried and fixed) steps taken, and ystart is replaced by values at the end of the integration interval. derivs is the user-supplied routine for calculating the right-hand side derivative, while rkqs is the name of the stepper routine to be used.

podprogrami

SUBROUTINE odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs,rkqs)
INTEGER nbad,nok,nvar,KMAXX,MAXSTP,NMAX
REAL eps,h1,hmin,x1,x2,ystart(nvar),TINY
EXTERNAL derivs,rkqs
PARAMETER (MAXSTP=10000,NMAX=50,KMAXX=200,TINY=1.e-30)
Runge-Kutta driver with adaptive stepsize control. Integrate the starting values ystart(1:nvar) from x1 to x2 with accuracy eps, storing intermediate results in the common block /path/.
h1 should be set as a guessed first stepsize, hmin as the minimum allowed stepsize (can be zero). On output nok and nbad are the number of good and bad (but retried and fixed) steps taken, and ystart is replaced by values at the end of the integration interval. derivs is the user-supplied subroutine for calculating the right-hand side derivative, while rkqs is the name of the stepper routine to be used. /path/ contains its own information about how often an intermediate value is to be stored.
COMMON /path/ kmax,kount,dxsav,xp,yp
User storage for intermediate results. Preset dxsav and kmax.

NR primjeri

```
gcc -o Cxrk4 xrk4.c rk4.c -I nrutils/ nrutils/nrutil.c bessj0.c bessj.c bessj1.c  -lm
f77 -o Frk4 xrk4.for rk4.for bessj.for bessj0.for bessj1.for
```

$$x' = -y$$

$$y' = x - \frac{1}{x} y$$

$$z' = y - \frac{2}{x} z$$

$$w' = z - \frac{3}{x} w$$

```c
void derivs(float x,float y[],float dydx[])
{       dydx[1] = -y[2];
        dydx[2]=y[1]-(1.0/x)*y[2];
        dydx[3]=y[2]-(2.0/x)*y[3];
        dydx[4]=y[3]-(3.0/x)*y[4];
}
```

```fortran
SUBROUTINE derivs(x,y,dydx)
    REAL x,y(*),dydx(*)
    dydx(1)=-y(2)
    dydx(2)=y(1)-(1.0/x)*y(2)
    dydx(3)=y(2)-(2.0/x)*y(3)
    dydx(4)=y(3)-(3.0/x)*y(4)
    return
    END
```

Bessel Function:	J0	J1	J3	J4
For a step size of: 0.20				
RK4:	0.671133	0.498290	0.159351	0.032869
Actual:	0.671133	0.498289	0.159349	0.032874
For a step size of: 0.40				
RK4:	0.566879	0.541971	0.207395	0.050358
Actual:	0.566855	0.541948	0.207356	0.050498

NR primjeri

```
f77 -o Frkdumb xrkdumb.for rkdumb.for rk4.for bessj.for bessj0.for bessj1.for
gcc -o Crkdumb xrkdumb.c rkdumb.c rk4.c -I nrutils/ nrutils/nrutil.c bessj0.c bessj.c
        bessj1.c  -lm
```

Source F77:

```
COMMON /path/ x,y
    EXTERNAL derivs
    x1=1.0
    vstart(1)=bessj0(x1)
    vstart(2)=bessj1(x1)
    vstart(3)=bessj(2,x1)
    vstart(4)=bessj(3,x1)
    x2=20.0
    call rkdumb(vstart,NVAR,x1,x2,NSTEP,derivs)
```

Source C:

```
float x1=1.0,x2=20.0,*vstart;
        vstart=vector(1,NVAR);
        xx=vector(1,NSTEP+1);
        y=matrix(1,NVAR,1,NSTEP+1);
        vstart[1]=bessj0(x1);
        vstart[2]=bessj1(x1);
        vstart[3]=bessj(2,x1);
        vstart[4]=bessj(3,x1);
        rkdumb(vstart,NVAR,x1,x2,NSTEP,derivs);
```

primjer

Lorenzov atraktor:

gcc -o lorenz lorenz.c rkdumb.c rk4.c nrutils/nrutil.c -I nrutils/

```
void derivs(float x, float y[], float dydx[]) {
    dydx[1] = -sigma*y[1] + sigma*y[2] ;
    dydx[2] = -y[1]*y[3] + r*y[1] - y[2] ;
    dydx[3] = y[1]*y[2] - b*y[3] ;
}
```

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = rx - y - xz$$
$$\dot{z} = xy - bz.$$

$$\sigma = 10,\ b = \tfrac{8}{3},\ r = 28.$$

y[1]=x, y[2]=y,y[3]=z.

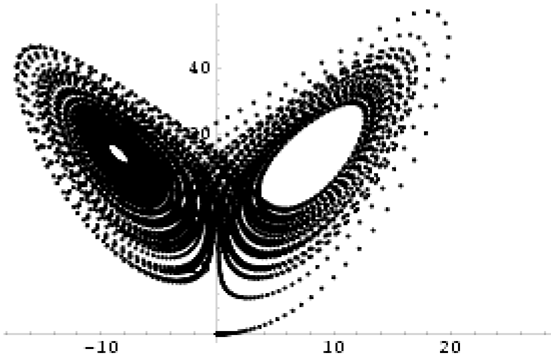
f77 -g -o Florenz lorenz.for rk4.for

rkdumb podprogram nalazi se u lorenz.for

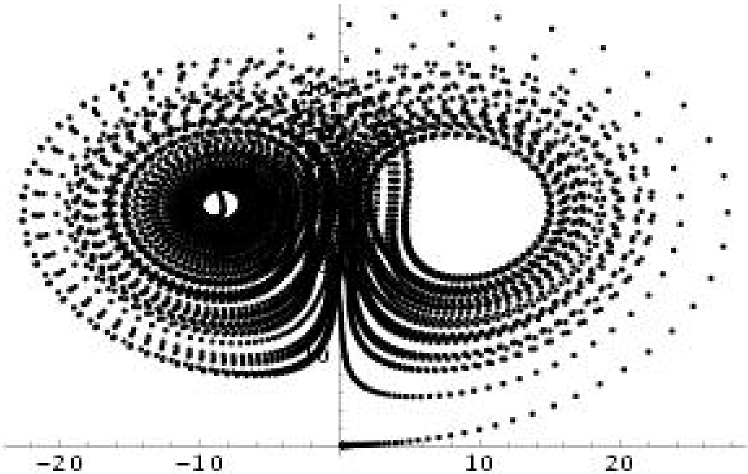
- zbog velike duljine niza 6000 promjenjeni su parametri u podprogramu
- maksimalna duljina niza 200 promjenjena je na 6000, a NMAX 50->3.

```
SUBROUTINE rkdumb(vstart,nvar,x1,x2,nstep,derivs)
  INTEGER nstep,nvar,NMAX,NSTPMX
  PARAMETER (NMAX=3,NSTPMX=6000)
```

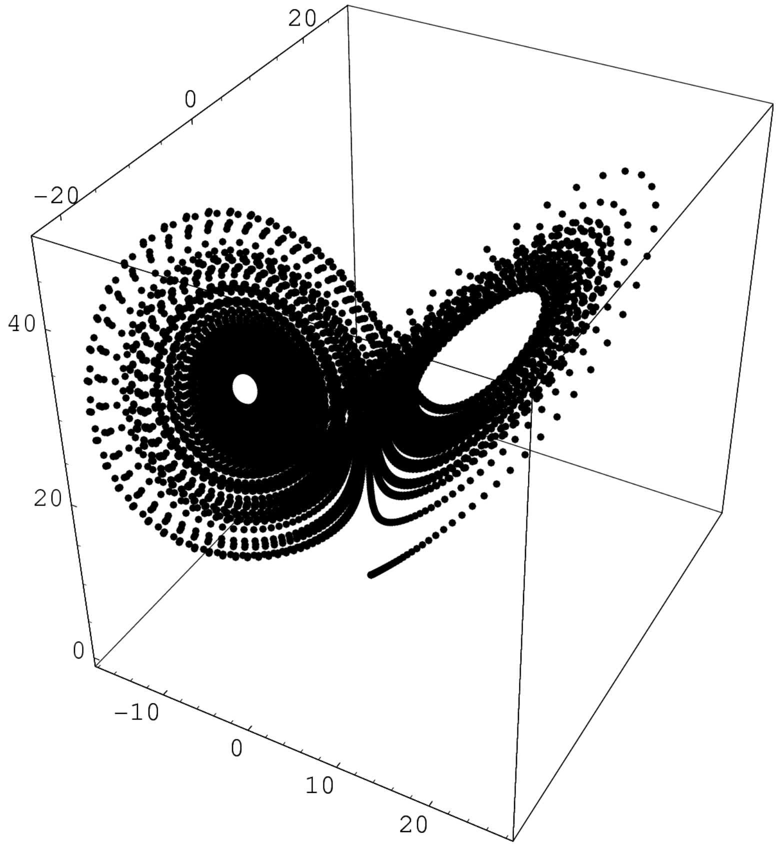
primjer



x-y ravnina



y-z ravnina



primjer

van der Pol jednadžba

$$y_1'' - \mu(1 - y_1^2)\, y_1' + y_1 = 0$$

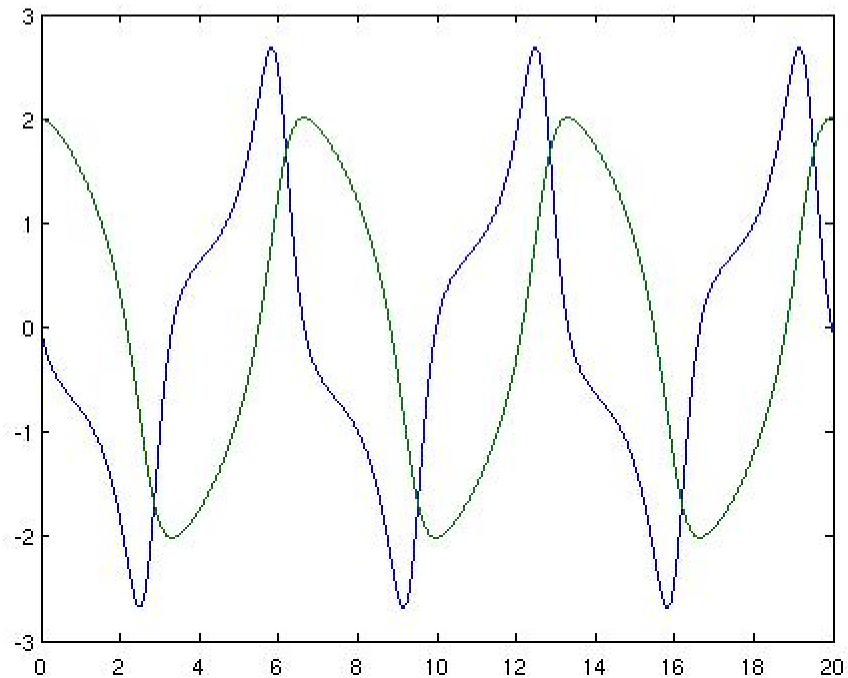
Pretvori u sustav prvog reda

$$y_1' = y_2$$

$$y_2' = \mu(1 - y_1^2) y_2 - y_1$$

Početni uvjet y(1)=2, y(2)=0

parametar μ=1




```
gcc -o Cvdpol vdpol.c rkdumb.c  rk4.c  nrutils/nrutil.c  -I nrutils/
f77 -o Fvdpol vdpol.for rk4.for
gcc -o CvdpolOde vdpol1.c odeint.c rkqs.c rkck.c nrutils/nrutil.c  -I nrutils/ -lm
f77 -o Fvdpol1 vdpol1.for odeint.for  rkqs.for rkck.for
```

RKDUMB
fiksan korak
ODEINT varijabilan

Krute (stiff) diferencijalne jednadžbe

Najvažnija upotreba višekoračnih metoda je rješavanje (sistema) krutih diferencijalnih jednadžbi. Najpoznatija metoda poznata je kao Gearova metoda (po Williamu C. Gear-u) i sastoji se od Adams prediktora i korektora varijabilnog reda i varijabilnog koraka.

Za diferencijalnu jednadžbu reći ćemo da je kruta, ako mala perturbacija početnih uvjeta dovede do velike perturbacije u rješenju problema.

Zadana je diferencijalna jednadžba

$$y' = 10(y - x) - 9, \quad y(0) = 1.$$

Opće rješenje ove diferencijalne jednadžbe je

$$y(x) = ce^{10x} + x + 1.$$

Partikularno rješenje za ovaj početni uvjet je $y = x + 1.$

Krute (stiff) diferencijalne jednadžbe

Ako malo perturbiramo početni uvjet na $y(0) = 1 + \varepsilon$, onda je partikularno rješenje te jednadžbe

$$y = \varepsilon e^{10x} + x + 1.$$

2. Primjer
van der Pol jednadžba

Početni uvjeti y(1)=2, y(2)=0
parametar μ=1000
granice intervala t=[0,3000]

```
gcc -g -o Cvdpol1stiff vdpol1stiff.c odeint1.c rkqs.c rkck.c nrutils/nrutil.c  -I nrutils/ -lm
```

promjenjena je vrijednost parametra MAXSTP 10000 na 1500000, file odeint1.c

```
f77 -g -o Fvdpol1stiff vdpol1stiff.for rkqs.for rkck.for
```

vdpol1stiff sadrži odeint podprogram jer smo matricama promjenili dimenzije

parametri: KMAXX=9000 i u programu kmax=KMAXX, inače podrazumjeva vrijednost 200

Krute (stiff) diferencijalne jednadžbe

Rosenbrock metoda

gcc -g -o Cvdpol1Ros vdpol1Ros.c odeint1.c nrutils/nrutil.c -I nrutils/ -lm lubksb.c ludcmp.c stiff.c

f77 -o Fvdpol1Ros vdpol1Ros.for stiff.for lubksb.for ludcmp.for

Vremenski interval [0,3000] predstavlja problem "stiff" podprogramima .

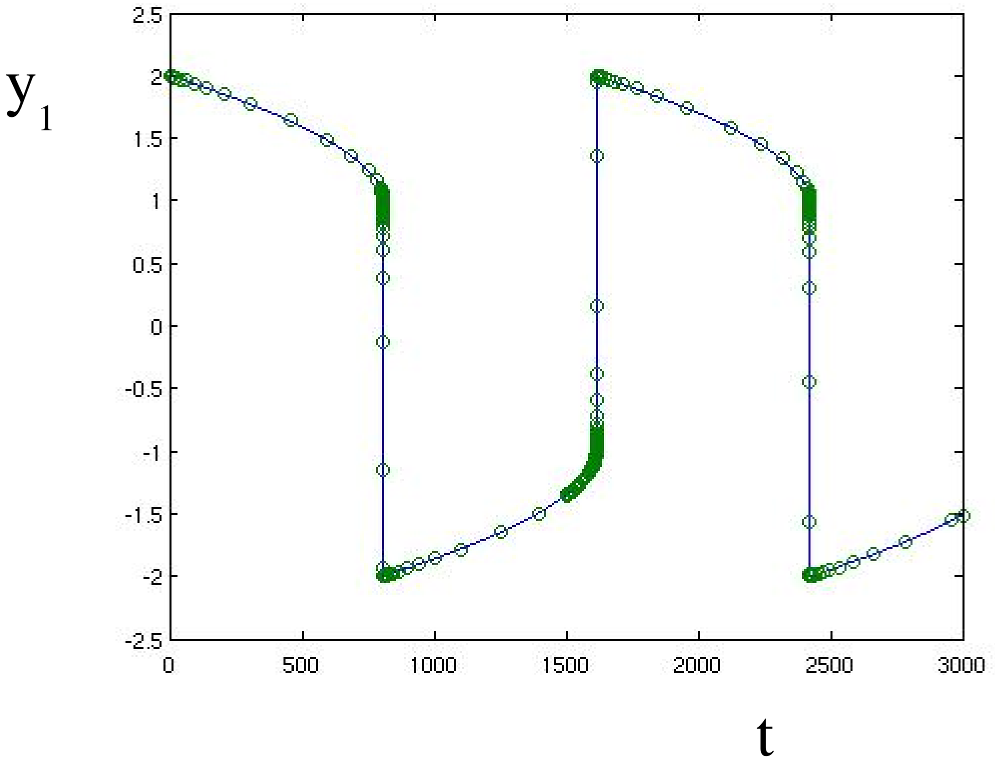
Diferencijalna jednadžba ne ovisi eksplicitno o vremenu, rastavljamo interval na dva.

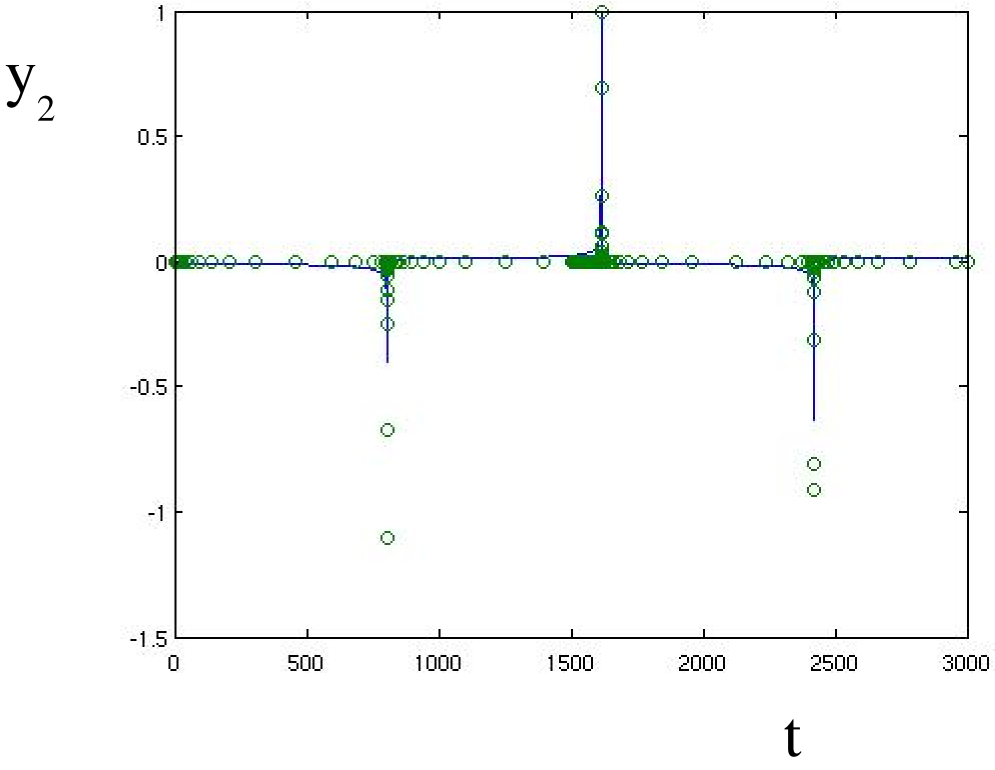
Manje od 600 točaka potrebno za riješenje problema, faktor 10 u odnosu na RK4.

```
 x2=1500.0;
  for (j=1;j<=2;j++)   {
odeint(vstart,NVAR,x1,x2,eps,h1,hmin,...,derivs,stiff)
;
...............
           x1=0; x2=x1+1500.0;
...........
}
```

```
x2=1500.0
 do 9 j=1,2
 call odeint(vstart,NVAR,x1,x2,...bad,derivs,stiff)
...................
x1=0.0
x2=x1+1500.0
```

van der Pol





μ=1000

o - Rosenbrock

-- - RK4

Zadatak za praktikum

Neka je $\theta(t)$ kut njihala s vertikalom u vremenu t kao na slici. Početni uvjeti su $\theta(0)=\pi/4$ i $\theta'(0)=0$. Pozicija njihala određena je diferencijalnom jednadžbom

$$\theta''(t) - \frac{g}{l}\sin(\theta(t)) = 0,$$

gdje je $g=9.8$ *m/sec*^{2} akceleracija zbog gravitacije i $l=0.5$ *m* je duljina njihala.

1. Napiši problem kao sustav diferencijalnih jednadžbi prvog reda
2. Napiši funkciju derivs koja predstavlja diferencijalnu jednadžbu njihala
3. Riješi problem u vremenu t=0 do 5 za 400 koraka. Koristi podprograme rkdumb ili odeint. Procijeni **period** njihala.
4. Pošaljite rezultat i source na mail Aleksandar.Maksimovic@irb.hr

Zadatak za praktikum

Riješi sustav diferencijalnih jednadžbi prvog reda pomoću Rosenbrock metode i pomoću neke druge metode koja nije za "krute" diferencijalne jednadžbe.

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} -0.1 & -199.9 \\ 0 & -200 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

vremenski interval riješenja je [0,0.1].

1. Usporedi efikasnost ove dvije metode, koliko je koraka bilo potrebno?
2. Odaberi jedno numeričko riješenje i usporedi ga s analitičkim riješenjem

Analitičko riješenje nacrtaj funkcijom g1=Plot[...], numeričko g2=ListPlot[] nakon učitavanja podataka s Import[..] funkcijom. Show[g1,g2] prikazuje slike zajedno.

4. Pošaljite rezultat i source na mail Aleksandar.Maksimovic@irb.hr

Analitičko riješenje: $y_1(t) = \exp(-0.1\,t) + \exp(-200\,t) \quad y_2(t) = \exp(-200\,t)$

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - Numerical Recipes in C
  - Numerical Recipes in Fortran
- L. F. Shampine, R. C. Allen, Jr., S. Pruess: FUNDAMENTALS OF NUMERICAL COMPUTING, John Wiley & Sons, Inc. (1997)