

Numerické řešení ODE

Numerické řešení diferenciálních rovnic

Tomáš Kasalický, 2014

Popis problému

Úkolem je vyřešit soustavu obyčejných diferenciálních rovnic

$$y' = f(t, y), \quad y(0) = y_0,$$

kde $y = (y_1, \dots, y_n)$ je vektor n neznámých funkcí, $y' = (y'_1, \dots, y'_n)$ je vektor jejich derivací a $f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ je vektorová funkce pravých stran. Nezávisle proměnná t má význam času a $y_0 = (y_{10}, \dots, y_{n0}) \in \mathbb{R}^n$ je vektor předepsaných počátečních podmínek v čase 0. Tento problém se z očividných důvodů také někdy nazývá "počáteční úloha". Protože se v celém problému vyskytují funkce pouze jedné proměnné, času t , mluvíme také o tomto problému jako o soustavě obyčejných diferenciálních rovnic (*ordinary differential equations*, ODE).

V tomto notebooku se pokusíme ukázat několik nejjednodušších numerických způsobů řešení zadaného problému.

Eulerova metoda

Abychom mohli aproximovat analytické (přesné) řešení původního problému je nejprve nutné diskretizovat nezávisle proměnnou t . Přibližné řešení y proto hledáme v diskrétních časových bodech $t_k = \Delta k$, $k = 0, 1, 2, \dots$, kde Δ představuje délku časového kroku. Sofistikovanější metody mohou délku tohoto kroku pružně měnit. My se zde zaměříme na nejjednodušší metody s konstantním časovým krokem. Hledané hodnoty závisle proměnné (funkce) y v diskrétních časech t_k si označíme odpovídajícím způsobem jako $y_k = y(t_k)$, $k = 0, 1, 2, \dots$. Všimněme si, že první hodnota, $y_0 = y(0)$ je známa, je to předepsaná počáteční podmínka.

Jakým způsobem se zbavit derivace v diferenciální rovnici? Místo tečny (derivaci) přejdeme k sečně (přímka procházející dvěma sousedními body). Nahradíme proto derivaci (dopřednou) diferencí a tu pak hodnotou pravé strany v předcházejícím okamžiku,

$$\frac{y_{k+1} - y_k}{\Delta} \approx y'(t_k) \approx f(t_k, y_k), \quad k = 0, 1, 2, \dots,$$

nebo jinak zapsáno

$$y_{k+1} = y_k + \Delta f(t_k, y_k), \quad k = 0, 1, 2, \dots,$$

$$y_0 = y(0)$$

Jedná se o rekurentně zadanou posloupnost, kterou můžeme libovolně dlouho iterovat. Přesnost aproximace zřejmě závisí na velikosti časového kroku Δ . Čím bude tento krok menší, tím bude aproximace lepší (aspoň na krátkém časovém intervalu, vizte níže). Eulerova metoda se však v praxi nepoužívá. Nedává příliš přesné výsledky, jak bude patrné z následujících příkladů.

Jako první příklad uvažme soustavu popisující jednorozměrný harmonický oscilátor (těleso na pružině),

$$y' = -\omega^2 x,$$

$$x' = y,$$

$$x(0) = x_0, \quad y(0) = y_0.$$

Zde x představuje vzdálenost od rovnovážné polohy, y rychlost tělesa a $\omega > 0$ frekvenci kmitání. Tento problém můžeme vyřešit explicitně. Prostým dosazením si čtenář snadno ověří, že funkce

$$x(t) = x_0 \cos(\omega t) + \frac{y_0}{\omega} \sin(\omega t),$$

$$y(t) = -\omega x_0 \sin(\omega t) + y_0 \cos(\omega t),$$

řeší zadaný problém. Toto exaktní (přesné) řešení použijeme k porovnání s numerickým (nepřesným) řešením. Získáme tak představu o kvalitě numerického řešení získaného pomocí Eulerovy metody.

```
def g_plot(x0,y0,delta,omg,tmax):
    graph = Graphics()
    graph_params = dict(ymin = -5, ymax = 5)
    graph += plot( x0*cos(omg*x)+y0/omg*sin(omg*x) , (x, 0,
tmax), color="red", axes_labels=['t$', '$x$'])






    data=[(0,(x0,y0))]
    err=[]
    points=[]
    for i in range(floor(tmax/delta)):
        xi = data[-1]
        data.append((xi[0] + delta,(xi[1][0]+delta*xi[1][1],xi[1]
[1]-omg^2*delta*xi[1][0])))
        err.append((xi[0],1/2*y0^2 + 1/2*omg^2*x0^2 - (1/2*(xi[1]
[1])^2 + 1/2*omg^2*(xi[1][0])^2)))
        points.append([xi[0],xi[1][0]])

    graph += list_plot(points)
    print('Body představují numerickou aproximaci, červená křivka
představuje přesné řešení (vynášíme hodnoty polohy).')
    show (graph,**graph_params)
    print('Na spodním grafu je změna energie, od počáteční
```

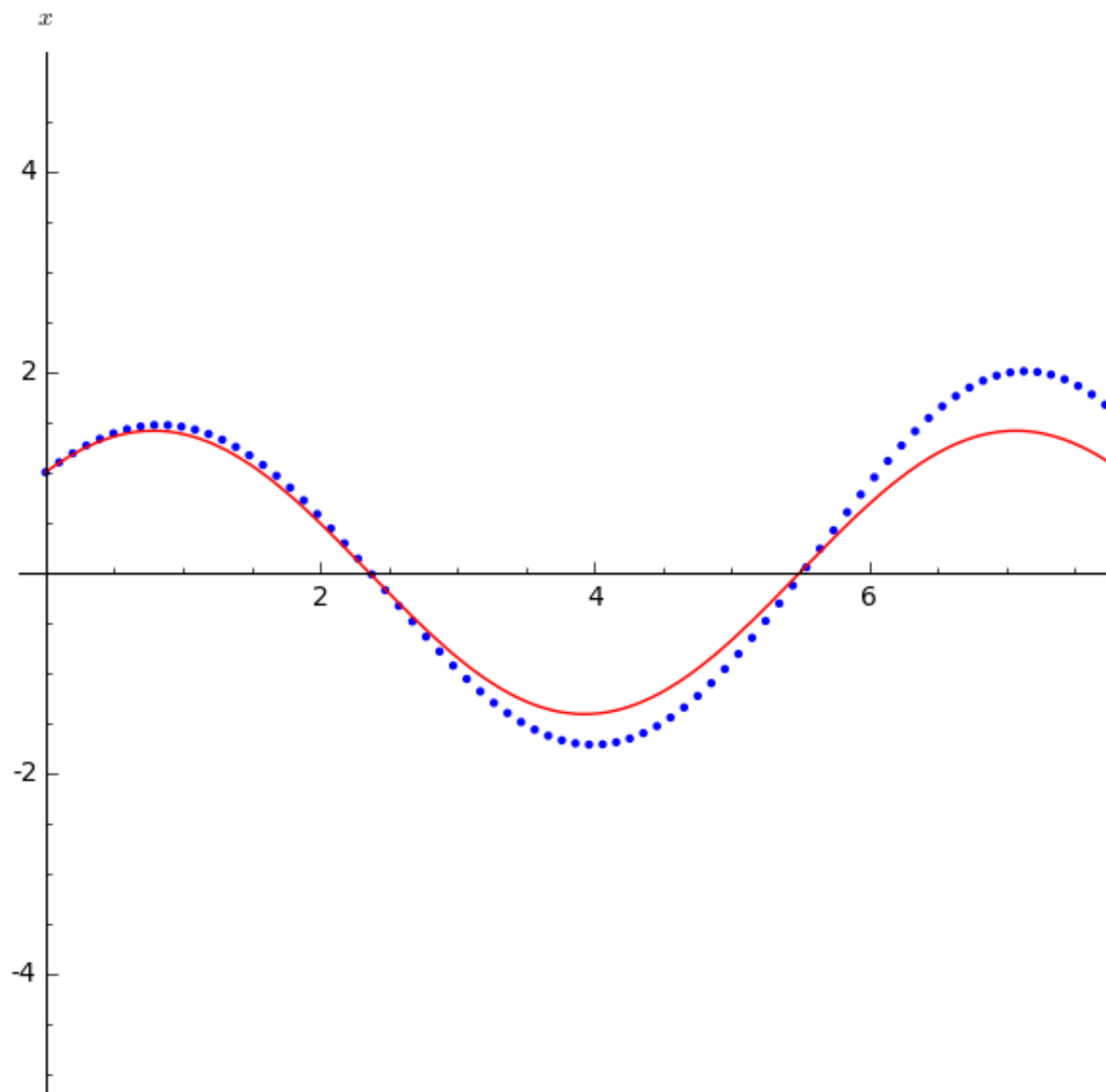
```

hodnoty. \nEnergie musí být v realitě konstantní (neuvažujeme-li
tření). \nVýsledek je tedy špatný.')
    show(list_plot(err, rgbcolor="red", axes_labels=
['$t$', '$E_{err}$']))
#####
# make Interaction
#####
@interact
def _( x0= slider(srange(-3,3.1,.1),default = 1,label="$x_0$"),
      y0= slider(srange(-3,3.1,.1),default = 1,label="$y_0$"),
      omg= slider(srange(.1,4.1,.1),default = 1,label="$\omega$"),
      delta= slider(srange(.001,1.001,.001),default =
.1,label="$\Delta$"),
      tmax= slider(srange(1,20.1,1),default = 10,label="$t_{max}$")
):
    g_plot(x0,y0,delta,omg,tmax)

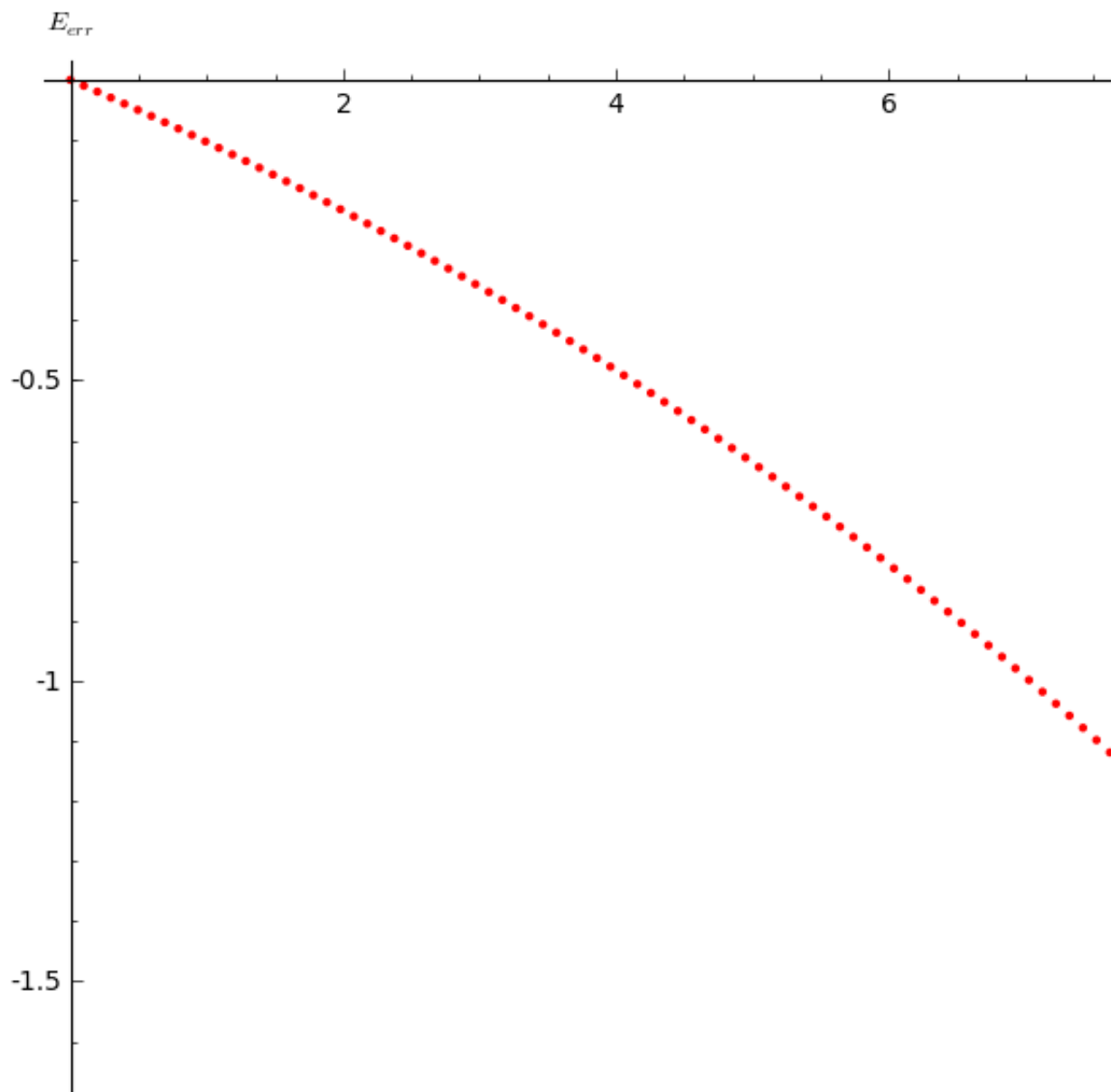
```

x_0		1.0000000000000000
y_0		1.0000000000000000
ω		1.0000000000000000
Δ		0.09900000000000001
t_{max}		10.000000000000000

Body představují numerickou aproximaci, červená křivka představuje přesné řešení (vynášíme hodnoty polohy).



Na spodním grafu je změna energie, od počáteční hodnoty.
Energie musí být v realitě konstantní (neuvažujeme-li tření).
Výsledek je tedy špatný.



Centrální Eulerova metoda

Centrální Eulerova metoda se od eulerovy metody liší pouze tím, kdy vyhodnocuje pravé strany. Časový okamžik volí jako průměr mezi časovými kroky (proto centrální), konkrétně

$$\frac{y_{k+1}-y_k}{\Delta} \approx y'(t_k) \approx f\left(\frac{t_{k+1}+t_k}{2}, \frac{y_{k+1}+y_k}{2}\right), \quad k = 1, 2, 3, \dots$$

Nyní je situace komplikovanější. Ihned totiž nezískáváme explicitní vyjádření y_{k+1} , tato rovnice ho pouze implicitně zadává. Uvážíme-li stejný příklad jako dříve, pak řešíme soustavu:

```
xk1,xk,yk1,yk,delta,omega = var('xk1,xk,yk1,yk,delta,omega')
show(solve( [1/delta*(xk1 - xk) == ( yk1 + yk )/2,
            1/delta*(yk1 - yk) == -omega^2*(xk1 + xk)/
            2], [xk1, yk1]))
```

$$\left[\begin{aligned} x_{k_1} &= -\frac{(\delta^2 \omega^2 - 4) x_k - 4 \delta y_k}{\delta^2 \omega^2 + 4}, & y_{k_1} &= -\frac{4 \delta \omega^2 x_k + (\delta^2 \omega^2 - 4)}{\delta^2 \omega^2 + 4} \end{aligned} \right.$$

Podívejme se, jaké dává toto numerické schéma výsledky.

```
def g_plot(x0,y0,delta,omg,tmax):
    graph = Graphics()
    graph_params = dict(
        ymin = -5, ymax = 5)
    graph += plot( x0*cos(omg*x)+y0/omg*sin(omg*x) , (x, 0,
tmax), color="red", axes_labels=['t$', '$x$'])

    data=[(0,(x0,y0))]
    err=[]
    points=[]
    for i in range(floor(tmax/delta)):
        xi = data[-1]
        data.append((xi[0] + delta,(((4-delta^2*omg^2)*xi[1]
[0]+4*delta*xi[1][1])/(4+delta^2*omg^2),((4-delta^2*omg^2)*xi[1]
[1]-4*delta*omg^2*xi[1][0])/(4+delta^2*omg^2))))
        err.append((xi[0],1/2*y0^2 + 1/2*omg^2*x0^2 - (1/2*(xi[1]
[1])^2 + 1/2*omg^2*(xi[1][0])^2)))
        points.append([xi[0],xi[1][0]])

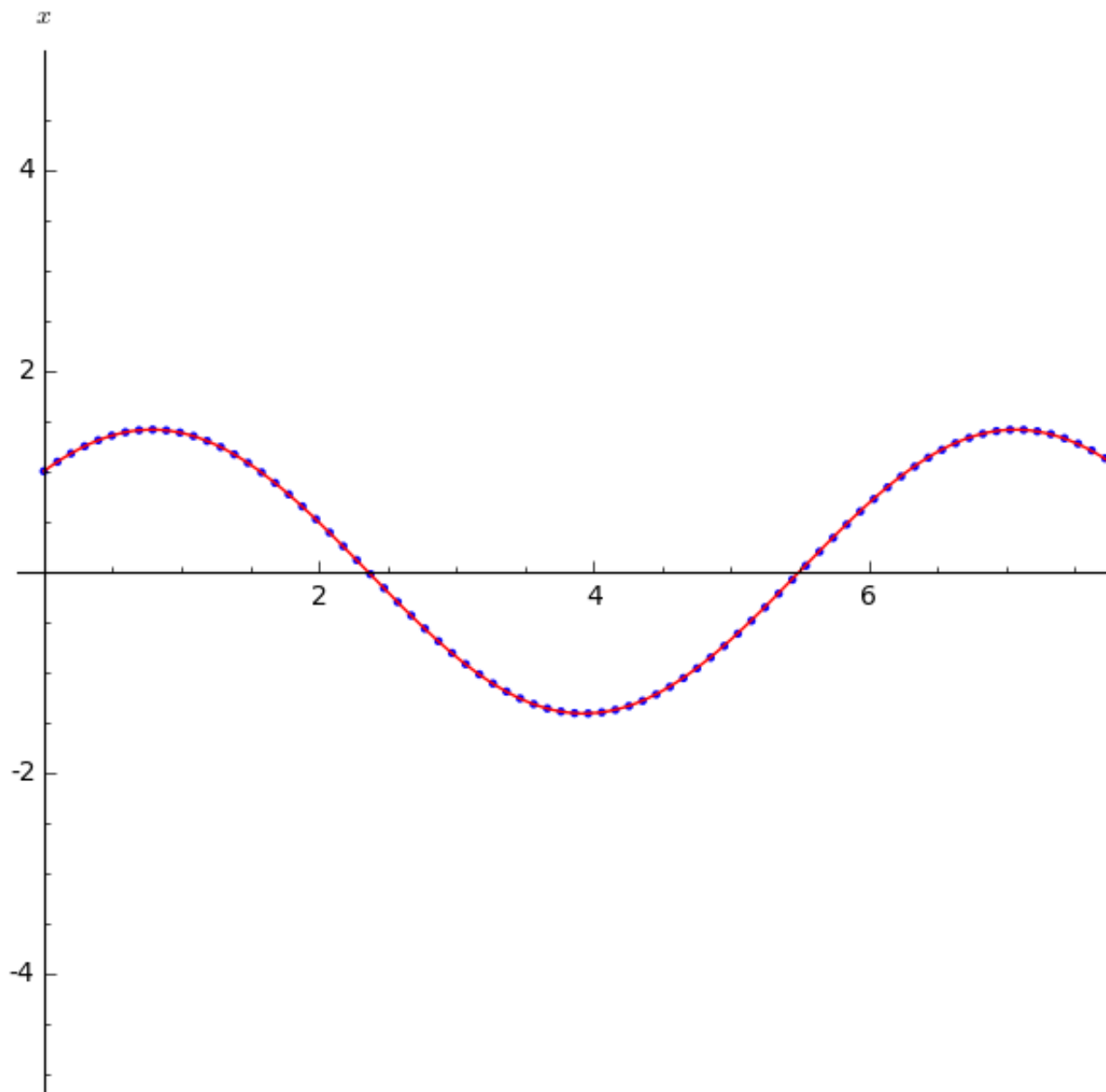
    graph += list_plot(points)
    print('Body představují numerickou aproximaci, červená křivka
představuje přesné řešení.')
    show (graph ,**graph_params)
    print('Na spodním grafu je změna energie, od počáteční
hodnoty.\nEnergie musí být konstantní. V tomto případě to
prakticky platí.')
    show(list_plot(err, rgbcolor="red",axes_labels=
['t$', '$E_{err}$']))
#####
# make Interaction
#####
@interact
def _( x0= slider(srange(-3,3.1,.1),default = 1,label="$x_0$"),
    y0= slider(srange(-3,3.1,.1),default = 1,label="$y_0$"),
    omg= slider(srange(.1,4.1,.1),default = 1,label="$\omega$"),
    delta= slider(srange(.001,1.001,.001),default =
.1,label="$\Delta$"),
    tmax= slider(srange(1,20.1,1),default = 10,label="$t_{max}$")
):
    g_plot(x0,y0,delta,omg,tmax)
```

x_0

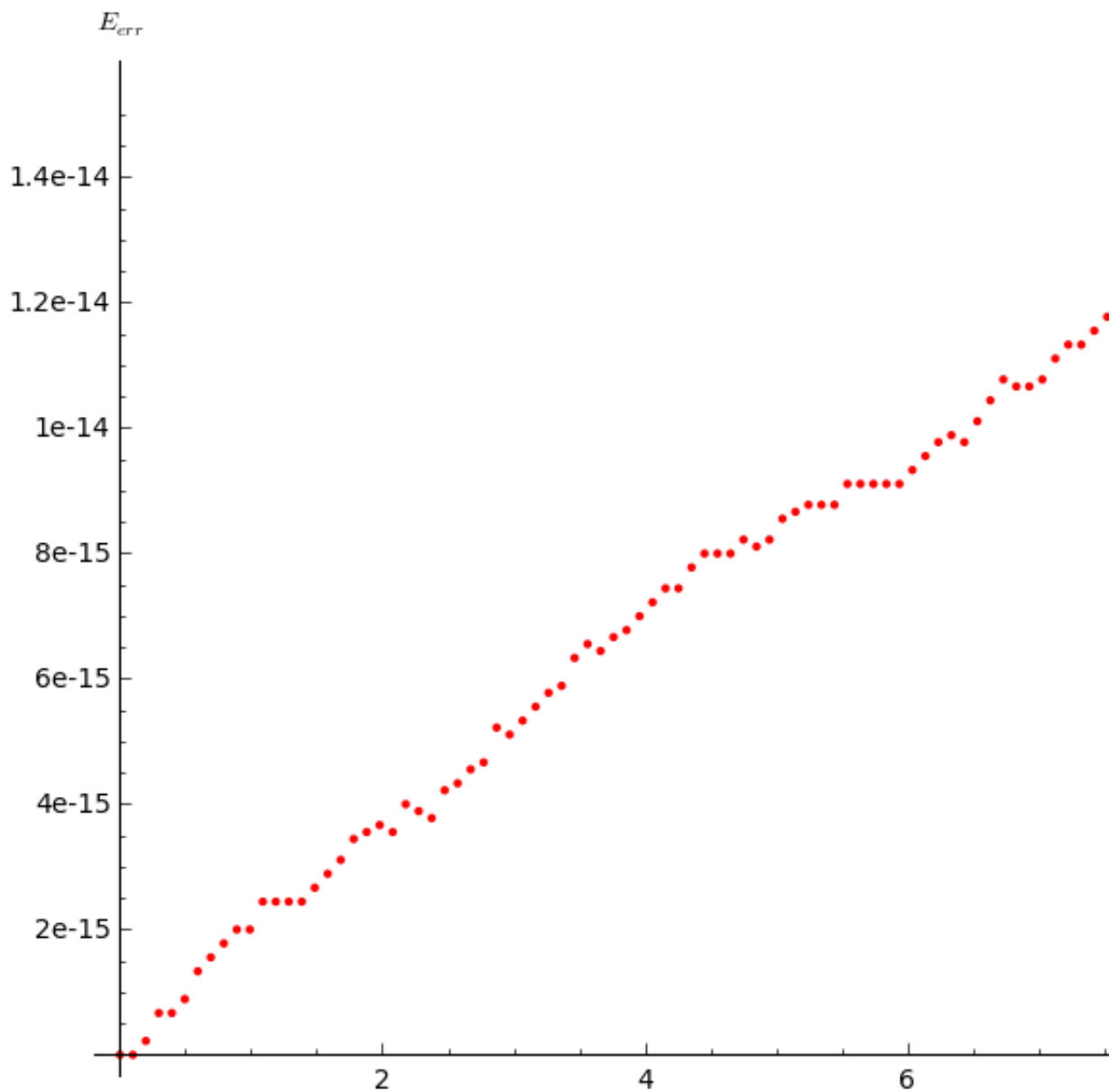
 y_0

ω	<input type="text" value="1.0000000000000000"/>	1.0000000000000000
Δ	<input type="text" value="0.09900000000000001"/>	0.09900000000000001
t_{max}	<input type="text" value="10.000000000000000"/>	10.000000000000000

Body představují numerickou aproximaci, červená křivka představuje přesné řešení.



Na spodním grafu je změna energie, od počáteční hodnoty. Energie musí být konstantní. V tomto případě to prakticky platí.



Runge-Kutta metoda (RK4)

Runge-Kuttova metoda (RK4) je explicitní metoda 4. řádu. Příslušné numerické schéma nebudeme podrobně odvozovat, pouze zmíníme jeho tvar:

$$\begin{aligned}
 k_1 &= f(t_k, y_k), \\
 k_2 &= f\left(t_k + \frac{\Delta}{2}, y_k + \frac{\Delta}{2} k_1\right), \\
 k_3 &= f\left(t_k + \frac{\Delta}{2}, y_k + \frac{\Delta}{2} k_2\right), \\
 k_4 &= f\left(t_k + \Delta, y_k + \Delta k_3\right), \\
 y_{k+1} &= y_k + \frac{1}{6} \Delta (k_1 + 2k_2 + 2k_3 + k_4)
 \end{aligned}$$

```
def rungeStep(tk,yk,delta,omg):
    f = matrix([[0, 1], [-omg^2, 0]]);
    k1 = f*yk
    k2 = f*(yk + delta/2*k1)
```



```

k3 = f*(yk + delta/2*k2)
k4 = f*(yk + delta*k3)
return ([tk + delta, yk + delta/6*(k1 + 2*k2 + 2*k3 + k4)])

def g_plot(x0,y0,delta,omg,tmax):
    graph = Graphics()
    graph_params = dict(
        ymin = -5, ymax = 5)
    graph += plot( x0*cos(omg*x)+y0/omg*sin(omg*x) , (x, 0,
tmax), color="red", axes_labels=['t$', '$x$'])

    data=[(0,vector([x0,y0]))]
    err=[]
    points=[]
    for i in range(floor(tmax/delta)):
        xi = data[-1]
        data.append(rungeStep(xi[0],xi[1],delta,omg))
        err.append((xi[0],1/2*y0^2 + 1/2*omg^2*x0^2 - (1/2*(xi[1]
[1])^2 + 1/2*omg^2*(xi[1][0])^2)))
        points.append([xi[0],xi[1][0]])

    graph += list_plot(points)

    print('Body představují numerickou aproximaci, červená křivka
představuje přesné řešení.')
    show (graph ,**graph_params)
    print('Na spodním grafu je změna energie, od počáteční
hodnoty. Energie musí být konstantní.')
    show(list_plot(err, rgbcolor="red",axes_labels=
['t$', '$E_{err}$']))
#####
# make Interaction
#####
@interact
def _( x0= slider(srange(-3,3.1,.1),default = 1,label="$x_0$"),
    y0= slider(srange(-3,3.1,.1),default = 1,label="$y_0$"),
    omg= slider(srange(.1,4.1,.1),default = 1,label="$\omega$"),
    delta= slider(srange(.001,1.001,.001),default =
.1,label="$\Delta$"),
    tmax= slider(srange(1,20.1,1),default = 10,label="$t_{max}$")
):
    g_plot(x0,y0,delta,omg,tmax)

```

x_0	<input type="text"/>	1.0000000000000000
y_0	<input type="text"/>	1.0000000000000000
ω	<input type="text"/>	1.0000000000000000
Δ	<input type="text"/>	0.09900000000000001
t_{max}	<input type="text"/>	10.000000000000000



Závěr

Na jednoduchém fyzikálním problému jsme porovnali výsledky napočítané pomocí Eulerovy, Centrální Eulerovy a Runge-Kutta 4 metody s výsledky Sage. Celkem lze říci, že Eulerova metoda byla vyloženě špatná. Runge-Kutta 4 dává dobré výsledky pro většinu voleb parametrů, ale pro některé rozsahy už nestačí a chyby v předpovědi vývoje energie (musí být konstantní - Zákon zachování energie) jsou podstatné. Nejlepší výsledky v tomto případě dává Centrální Eulerova metoda. To není náhoda, jedná se totiž o symplektickou metodu, která zachovává některé fundamentální vlastnosti původního klasického mechanického systému.