

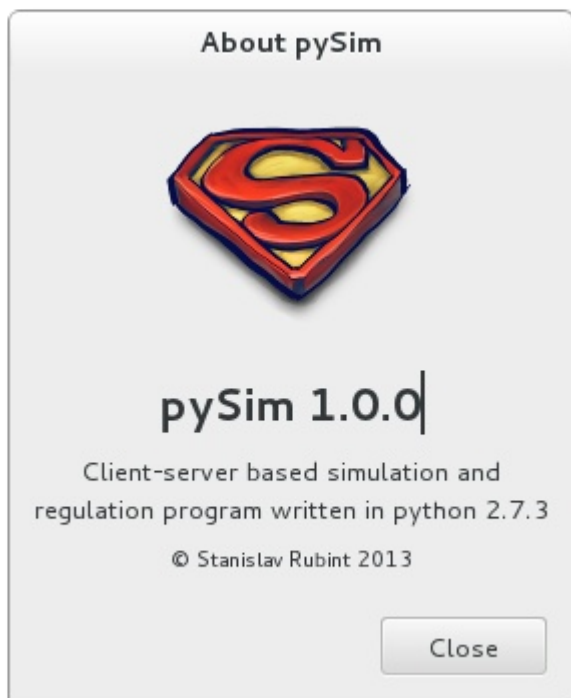
# **Telematika a riadenie**

Inverzné kyvadlo, riadenie stavovým regulátorom po linearizácii

Vypracoval

Stanislav Rubint  
Robotika, I.Sem  
25.12.2013

**Anotácia** - Inverzné kyvadlo, stavový regulátor, linearizácia, stabilita, opis systému diferenciálnymi rovnicami, vzdialené ovládanie, telekomunikácia a riadenie, klient-server architektúra, objektovo orientované programovanie, numerické riešenie diferenciálnych rovníc, grafické užívateľské prostredie



**pySim** – simulátor systému inverzného kyvadla na pohyblivej podstave pohybujúcej sa v jednej osi s možnosťou rotácie ramena ľubovoľne v rovine ktorú tvorí táto os a os gravitačného zrýchlenia a jeho riadenia za pomoci sily aplikovanej v zmysle osi pohybu podstavy.

**Úvodom** – program, ktorý pozostáva z troch častí, server, klient a animátor, slúži na simulovanie dynamiky skutočného systému inverzného kyvadla a zároveň jeho riadenie pomocou metódy vsádzania pólov na linearizovaný model tejto sústavy a reguláciou proporcionálno-integračným regulátorom. Simulácia a v skutočnosti numerický výpočet diferenciálnych rovníc opisujúcich tento systém prebieha na serveri, klient sa stará o

počiatočné nastavenia, poskytuje rozhranie s používateľom a samostatný modul animácie, implementovaný do tlačidla klientskej aplikácie, sa stará o analýzu, prepočet a vykreslenie odsimulovaných dát.

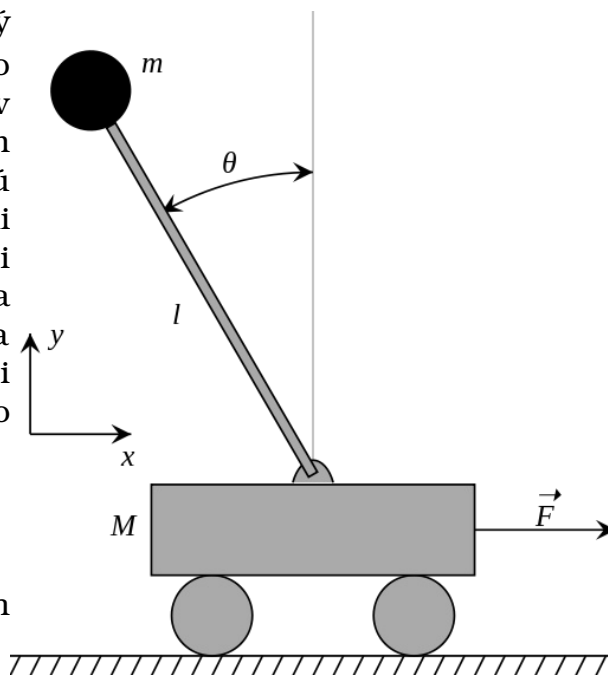
**Inverzné kyvadlo** – je dynamický systém, ktorý pozostáva z matematického kyvadla s hmotnosťou  $m$  koncentrovanej v jednom bode tohto kyvadla, s druhým bodom pripevneným k pohyblivej podstave. Celú sústavu vieme opísať dvomi súradnicami alebo zjednodušene, nakoľko medzi objektmi sa nachádzajú väzby, posunutím vozíka a uhlom ktorý zvierá kyvadlo s osou kolmou na smer pohybu podložky (zvislicou), akéhosi vozíka. Lagrangeova funkcia takéhoto zjednodušeného systému sa dá zapísať ako

$$L = \frac{1}{2} M v_1^2 + \frac{1}{2} m v_2^2 - m g l \cos \theta$$

kde  $v_1$  a  $v_2$  vieme vyjadriť pomocou stavových premenných a to nasledovne

$$L = \frac{1}{2} (M + m) \dot{x}^2 - m l \dot{x} \dot{\theta} \cos \theta + \frac{1}{2} m l^2 \dot{\theta}^2 - m g l \cos \theta$$

Diferenciálne rovnice dostaneme postupným derivovaním Lagrangeovej funkcie a to pomocou nasledujúcej rovnosti



$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0 \quad \text{a} \quad \frac{d}{dt} \frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = F$$

pričom dostaneme dve rovnice pre polohu vozíka

$$(M + m) \ddot{x} - m\ell \ddot{\theta} \cos \theta + m\ell \dot{\theta}^2 \sin \theta = F$$

a

$$\ell \ddot{\theta} - g \sin \theta = \ddot{x} \cos \theta$$

pre uhol ktorý zviaza kyvadlo so zvislicou. Z rovníc je jasne vidieť, že tieto dve statové premenné sú priamo závislé jedna od druhej a taktiež že je to silne nelineárny systém. Nakoľko cieľom riadenia je linearizovať tento systém na okolie pracovného bodu kde  $\theta \rightarrow 0$  a teda  $\sin(\theta)=0$  a  $\cos(\theta)=1$ . Simulovaný systém je avšak trochu zložitejší a opisuje reálny systém pomocou nasledujúcich rovníc

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \beta(x_3) \left( a_{23} \sin x_3 \cos x_3 + a_{22} x_2 + a_{24} \cos x_3 x_4 + a_{25} \sin x_3 x_4^2 + b_2 F \right) \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= \beta(x_3) \left( a_{43} \sin x_3 + a_{42} \cos x_3 x_2 + a_{44} x_4 + a_{45} \cos x_3 \sin x_3 x_4^2 + b_4 \cos x_3 F \right) \end{aligned}$$

kde  $x_1$  reprezentuje polohu vozíka a  $x_3$  uhol ktorý zviaza kyvadlo so zvislicou. Sila  $F$  ktorou musíme pôsobiť na vozík vypočítame nasledovne

$$k_{xx} := -0.01313482817 \quad x_1(t) - 10.10509022 \quad x_2(t) - 46.76016035 \quad x_3(t) - 4.218843951 \quad x_4(t)$$

pričom sila  $F$  ktorú aplikujeme na vozík vypočítame  $F = P(x_{10} - k_{xx}) + I(x_{10} - x_1)$ .

**Server.py** – na nasledujúcich pár riadkoch je gro simulácie servera

```

11 def func(x, t): #tu sa pocita diferencialna cast
12
13     X1,X2,X3,X4=x[0],x[1],x[2],x[3]
14     beta3=beta(X3)
15     s3=sin(X3)
16     c3=cos(X3)
17
18     fa=(.01313482817*X1 +10.10509022*X2 + 46.76016035*X3 + 4.218843951*X4)*kP
19     fa+=X1*(-kI)*kP
20
21     x1=X2
22     x2=beta3*(a23*s3*c3+a22*X2+a24*c3*X4+a25*s3*X4*X4+ b2*fa)
23     x3=X4
24     x4=beta3*(a43*s3+a42*c3*X2+a44*X4+a45*c3*s3*X4*X4+ b4*c3*fa)
25
26     return [x1,x2,x3,x4,fa]
27

```

táto funkcia *func* sa používa ako argument vo funckii *odeint* ktorá integruje ODE algoritmom (podobne ako matlab) na riadku

```

70 data=odeint(func, x0, t)

```

Počiatkové hodnoty, teda vektor  $x$  ktorý je na vstupe funkcie, časový vektor  $t$  na ktorom sa integruje a aj začiatok simulácie sa začína diaľkovo, povelom cez socket ktorý je naviazaný na *Streamovanom* sockete (TCP).

```
28 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
29 for i in range(20000,30000):
30     try:
31         s.bind(("", i)) #listen to all on port from input
32         print "waiting for connection on port %d"%i
33     except: pass
34 s.listen(1) #wait for one connection
35 conn, addr = s.accept() #connection accepted
36 print 'Connected by', addr
37 while 1:
38     data = conn.recv(1024)
39
40     if data[0:4]=="exec":
41         exec(data[4:])
42         conn.sendall("ack")
43         print ">> Variables loaded"
44     elif data[0:4]=="exit":
45         conn.close()
46         exit(0)
```

V skratke: socket je vytvorený na riadku 28, na r.29-33 sa pokúša pripojiť na prvý voľný lokálny port od 20000 do 30000, r.34 počúva jedno prichádzajúce spojenie. Toto je tzv. *Blocking call* a teda program čaká kým sa niekto nepripojí (r.35). V nekonečnej slučke prijíma jeden kilobajt dát (pokiaľ si program nevyžiada iný počet), z ktorých vždy prvé 4 bajty sú príznakové ktoré ovládajú beh programu (r.40++) alebo neprijme reťazec “*exit*”. Za zmienku stoja nasledujúce riadky kódu

```
77 f = conn.makefile('wb', len(data)+1024 )
78 pickle.dump(data, f, pickle.HIGHEST_PROTOCOL)
79 f.close()
```

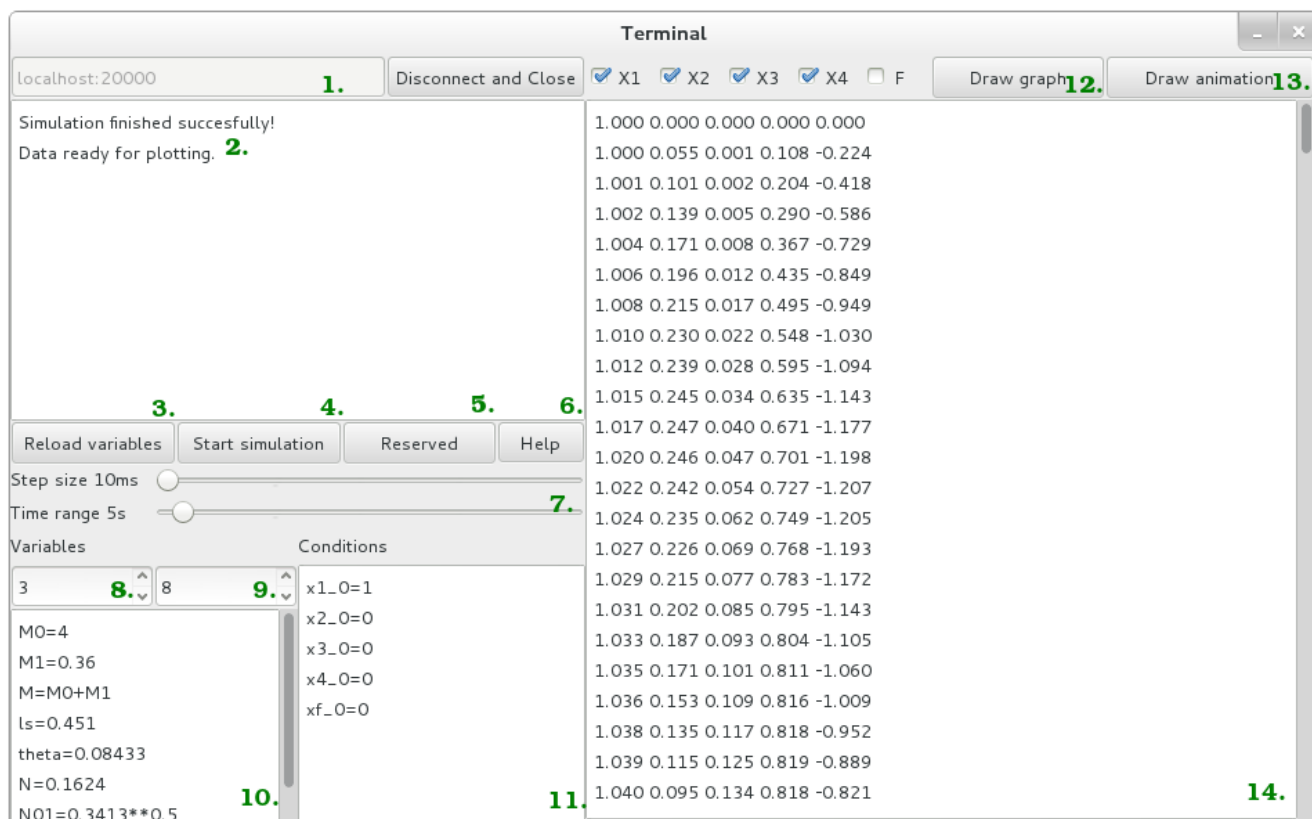
balík *pickle* slúži na ukladanie (dumping) premenných alebo prostredia do súboru alebo do reťazca. Python stavia a je písaný v jazyku C, ktorý využíva (podľa unixovho princípu pristupovať ku všetkému ako k súboru) sockety taktiež ako súbory, na riadku 77 si takýto súbor otvoríme módom “*wb*” (write on byte level) teda zápis na binárnej úrovni a nie reťazcovej, zapíšeme do súboru nasimulované dáta (r.78) a zavrieme.

Celý kód servera vďaka jednoduchosti jazyka python aj s medzerami medzi odstavcami zaberá len 80 riadkov.

**Klient.py** – aplikácia na strane klienta využíva na interakciu s učiteľom *GUI wx* – základný modul windows x systému dostupný pomaly v každom jazyku. Tento

kód spolu s modulom animácie je dlhý až 380 riadkov, z čoho bez mála 240 riadkov je len vykreslenie a obsluha grafickej časti programu.

Grafické prostredie je rozdelené logicky vertikálnou pomyslenou čiarou na časť vstupu informácií od užívateľa, vľavo a poskytnutie informácií užívateľovi, vpravo.



Pole adresy (1) slúži na zadanie IP adresy servera a port na ktorom počúva. Ak klient nevie port môže nahradiť číslo portu frázou *all* kedy sa program pokúsi vyhľadať port na ktorom server počúva. Priebeh diania je pravidelne aktualizovaný v okne monológu (2). Tlačidlá nižšie slúžia na načítanie časových nastavení simulácie z bežcov (7), načítanie konštánt P (8) a I (9), podmienok, konštánt (10) a počiatočných podmienok systému (11). Tieto údaje sú načítané a zaslané serveru prvým tlačidlom (3). Simulácia je odštartovaná tlačidlom *Start simulation* (4) po ktorej sa nasimulované dáta prenesú do dátového poľa (14). Tlačidlom *Help* môžeme vyvolať informácie o programe, v novších verziách pribudne možno návod a popřípade oživenie rezervovaného neaktívneho tlačidla (5). Zaškrtávacími políčkami si môžeme vybrať ktoré premenné chceme zobrazíť a to sa stane po stlačení tlačidla *Draw graph* (12). Tlačidlom animácie (13) vyvoláme spustenie aplikácie, ktorá dáta, ktoré boli prijaté zo strany servera a uložené automaticky na disk, rozpohybuje v animácii pre lepšiu predstavu o tom ako systém reaguje na riadenie.

Prijatie zapísaných dát zo strany servera je realizované, podobne ako na strane servera čítaním zo "súbora" socketu, čo ilustrujú nasledujúce riadky (276-278)

```
276 f = self.s.makefile('rb', int(data[4:])+1024 )
277 data = pickle.load(f)
278 f.close()
```

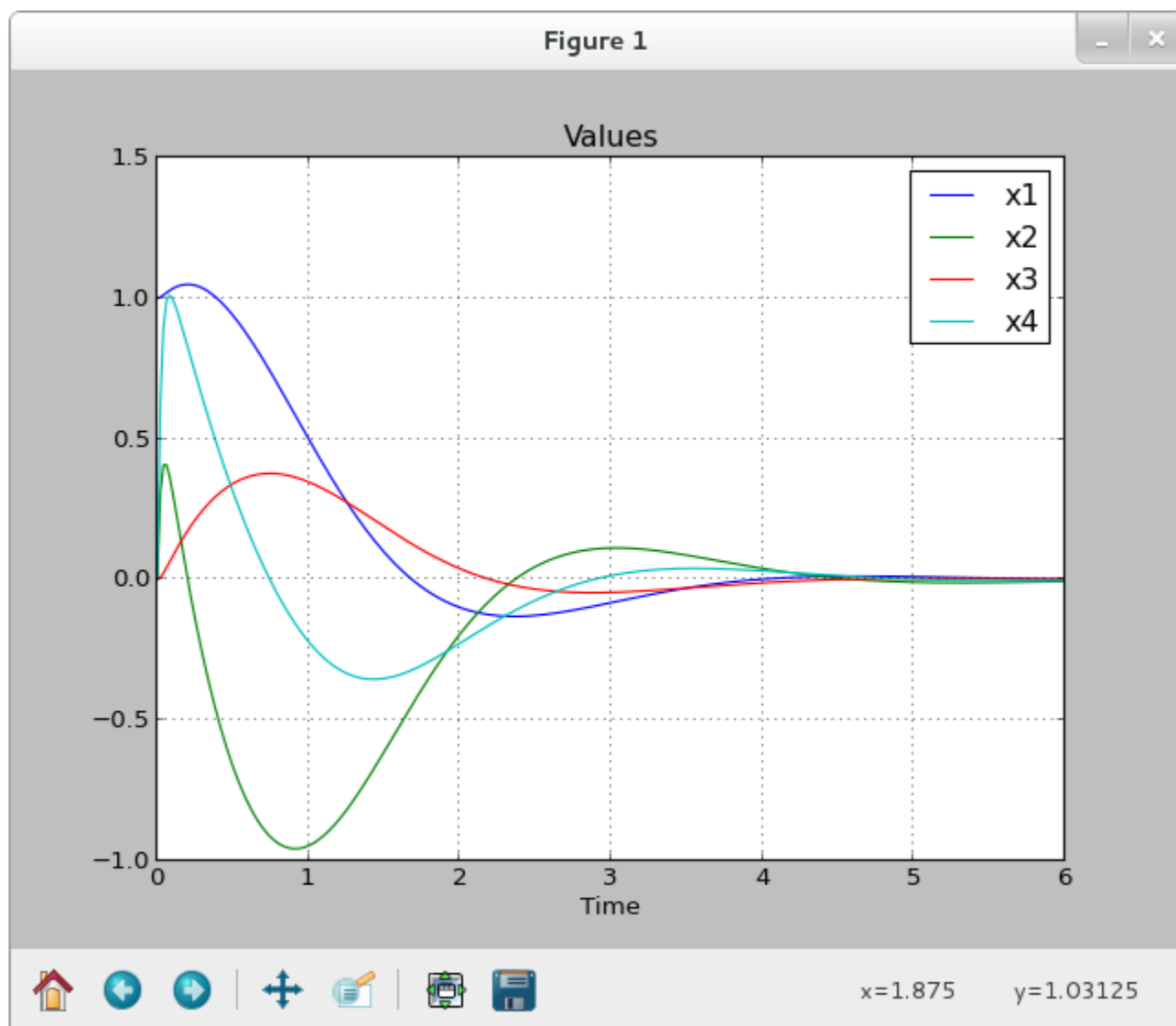
Aplikácia klienta využíva multiparadigmovosť jazyka python, a to znamená že zatiaľ čo server sa uspokojí s dvomi funkciami bez OOP (objektovo orientované programovanie), klient je riešený objektovo pomocou jednej triedy s 15-timi funkciami. Nadviazanie spojenia so serverom preto vyzerá už trochu inak

```
197     def connect(self, adress):
198         try:
199             self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
200             HOST,PORT=adress.split(":")
201             if PORT=="all": #vyskusa vsetky porty od 20000 do 30000
202                 for i in range(20000,30000):
203                     try:
204                         self.s.connect((HOST, int(i)))
205                         PORT=i
206                         break
207                     except:pass
208             if PORT=="all":
209                 self.richTextCtrl1.SetValue("No host available")
210                 raise socket.error
211         else:
212             self.s.connect((HOST, int(PORT)))
```

kde sa klient snaží pripojiť na zadanú adresu a port na ktorom počúva server, v prípade zadania frázy "all" sa pokúsi nájsť port na ktorom server počúva. V prípade akejkolvek chyby alebo nenájdení servera, túto skutočnosť patrične ohlásí užívateľovi v monológovom okne. V ďalších riadkoch sa program stará o acyklické odosielanie dát servru, pričom komunikácia a ovládanie pozostáva z príznaku, tj. 4 bajtov reťazca ktorý identifikuje príkaz a dát v dĺžke do 1kB. V prípade dát s variabilnou dĺžkou o tom najprv sever alebo klient oznámi a prijme/pošle sa daný počet znakov (r. 260-265)

```
259         self.richTextCtrl1.SetValue("Sending time frame")
260         self.t=[t/1000.0*vstep for t in range(int(vrange*1000/vstep))]
261
262         self.s.sendall(str(len(self.t)+128))
263         while self.s.recv(128)[0:3]!="ack": pass
264
265         self.s.sendall("t=[t/1000.0*%d for t in range(int(%d*1000.0/%d))]"%(vstep, v
266
267         while self.s.recv(1024)[0:3]!="gmx": pass
```

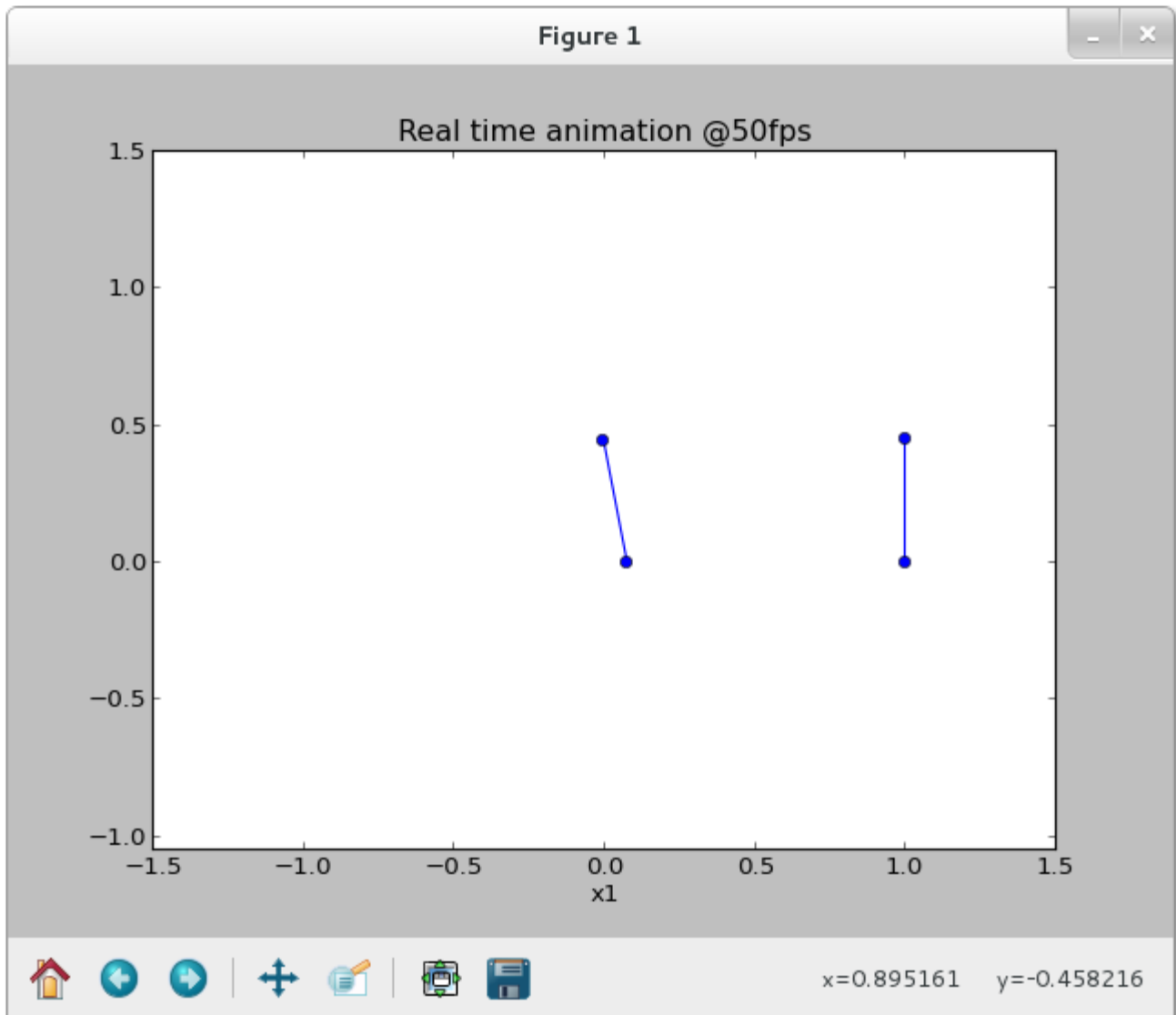
Grafická odozva na dáta je reprezentovaná grafom ktorý je plne interaktívny (zaujímavosť: presne tento modul grafického zobrazovania sa využíval na zobrazenie dát z Mars-Rovera)



z grafu je možné si vyselektovať jednotlivé veličiny, priblížiť si ho, uložiť obrázkovú reprezentáciu údajov v populárnych formátoch popřípadе aj nepopulárnych.



Popřípade animáciou kyvadla, ktorá je tiež riešená formou animovaného grafu, kde jednotlivé stavové premenné sú prevedené na uhly a zobrazené v rovine, pre presnú reprezentáciu údajov, ktorá je navyše vykresľovaná real-time, čo značí že presne tak rýchlo by sa reálny systém s danými počiatočnými podmienkami pohyboval, čo sa tiež využíva napríklad v diaľkovom riadení s veľmi dlhým prenosovým oneskorením.



Za túto animáciu je zodpovedná funkcia v module **animacia.py**, ktorá prepočítava pohybujúce sa kyvadlo (priamku) na základe nasimulovaných údajov

```
9     def update_line(self, n, data, line, ls):
10         x=float(data[n].split(" ")[0])
11         fi=float(data[n].split(" ")[2])
12         line.set_data([x, x-ls*sin(fi)], [0, ls*cos(fi)])
13         return line,
```

**Záver** – Program je napísaný v jazyku Python, ktorý je interpretovaný a multiparadigmaticý, čo znamená že sa dá programovať v ňom rôznymi spôsobmi (od “assemblerovského” štýlu, cez “Céčkovský” štýl až po “OOP” ako v Jave) a je za behu prekladaný, čo šetrí čas pri vývoji, nakoľko ho netreba zakaždým kompilovať. Odladený bol na verzii 2.7.3. K chodu využíva knižnice: ***numpy, matplotlib, scipy, wx, math, pickle, socket, os, sys, time***. Tučným písmom sú zvýraznené tie ktoré nie sú štandardnou súčasťou interpretera a treba ich doinštalovať. Nakoľko je python interpretovaný jazyk, nezáleží na akej platforme je program spúšťaný, programovaný a odladený bol avšak na Linuxe konkrétne distribúcií Fedora 19.



## server.py – zdrojový kód

```
1. from scipy.integrate import odeint
2. from thread import start_new_thread as new
3. import numpy as np
4. from math import exp, sin, cos
5. from time import sleep, time, gmtime
6. import sys, pickle, socket
7.
8. def beta(x3):
9.     return abs(1+((N**2)*(sin(x3)**2)/(N01**2)))*(-1)
10.
11. def func(x, t): #tu sa pocita diferencialna cast
12.
13.     X1,X2,X3,X4=x[0],x[1],x[2],x[3]
14.     beta3=beta(X3)
15.     s3=sin(X3)
16.     c3=cos(X3)
17.
18.     fa=(.01313482817*X1 +10.10509022*X2 + 46.76016035*X3 + 4.218843951*X4)*kP
19.     fa+=X1*(-kl)*kP
20.
21.     x1=X2
22.     x2=beta3*(a23*s3*c3+a22*X2+a24*c3*X4+a25*s3*X4*X4+ b2*fa)
23.     x3=X4
24.     x4=beta3*(a43*s3+a42*c3*X2+a44*X4+a45*c3*s3*X4*X4+ b4*c3*fa)
25.
26.     return [x1,x2,x3,x4,fa]
27.
28. s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
29. for i in range(20000,30000):
30.     try:
31.         s.bind(("", i)) #listen to all on port from input
32.         print "waiting for connection on port %d"%i
33.     except: pass
34. s.listen(1) #wait for one connection
35. conn, addr = s.accept() #connection accepted
36. print 'Connected by', addr
37. while 1:
38.     data = conn.recv(1024)
39.
40.     if data[0:4]=="exec":
41.         exec(data[4:])
```

```

42.     conn.sendall("ack")
43.     print ">> Variables loaded"
44. elif data[0:4]=="exit":
45.     conn.close()
46.     exit(0)
47. elif data[0:4]=="simu":
48.     timestamp=time()
49.     medzicas=timestamp
50.     gmt=gmtime(time())
51.     hodina, minuta, sekunda=gmt.tm_hour, gmt.tm_min, gmt.tm_sec
52.     print ">> Start of simulation at time %d:%d:%d"%(hodina, minuta, sekunda)
53.     conn.sendall("gmt")
54.     data=conn.recv(256)
55.     length=int(data)
56.     print ">> Length of data field:",length
57.     print ">> ...acknowledged"
58.     conn.sendall("ack")
59.     data=conn.recv(length)
60.     exec(data)
61.     print ">> Time frame loaded in %.1fms."%(time()*1000-medzicas*1000)
62.     medzicas=time()
63.     conn.sendall("gmx")
64.     data=conn.recv(1024)
65.     exec(data)
66.     print ">> default values: ", x0
67.     print ">> Simulating..."
68.     #Numericke prepocitanie diferencialnych rovníc
69.
70.     data=odeint(func, x0, t)
71.
72.     print ">> Simulated succesfully in %.1fms."%(time()*1000-medzicas*1000)
73.     medzicas=time()
74.     print ">> Length of simulated data to be transmitted: %dkB"%(int(len(data))*5/1024)
75.     conn.sendall("data%d"%len(data))
76.
77.     f = conn.makefile('wb', len(data)+1024 )
78.     pickle.dump(data, f, pickle.HIGHEST_PROTOCOL)
79.     f.close()
80.     print ">> Simulated and transmitted in %.1fms.\n"%((time()-timestamp)*1000)
81.
82. else:
83.     exit()

```

## *klient.py – zdrojový kód*

```
1. #Boa:Frame:Mainframe
2.
3. import wx, wx.stc, wx.grid, pickle, wx.richtext, socket, sys, os
4. import matplotlib.pyplot as plt
5. from matplotlib import animation
6. import numpy as np
7. from math import sin,cos
8. from animacia import animuj
9.
10. def create(parent):
11.     return Mainframe(parent)
12.
13. [wxID_MAINFRAME, wxID_MAINFRAMEADRESA, wxID_MAINFRAMEANIME_BTN,
14. wxID_MAINFRAMECONDT_LBL, wxID_MAINFRAMECONDT_TXTCTRL,
15. wxID_MAINFRAMECONNECT_BTN, wxID_MAINFRAMEDATABAZA, wxID_MAINFRAMEDRAW_BTN,
16. wxID_MAINFRAMEHELP_BTN, wxID_MAINFRAMEKI_CTRL, wxID_MAINFRAMEKP_CTRL,
17. wxID_MAINFRAMELOAD_VARS_BTN, wxID_MAINFRAMEPLOT_BTN,
18. wxID_MAINFRAMERICHTEXTCTRL1, wxID_MAINFRAMESHOW_F, wxID_MAINFRAMESHOW_X1,
19. wxID_MAINFRAMESHOW_X2, wxID_MAINFRAMESHOW_X3, wxID_MAINFRAMESHOW_X4,
20. wxID_MAINFRAMESTART_BTN, wxID_MAINFRAMESTEP_LBL, wxID_MAINFRAMESTEP_SLDR,
21. wxID_MAINFRAMETIME_RANGE_LBL, wxID_MAINFRAMETIME_SLDR,
22. wxID_MAINFRAMEVAR1_LBL, wxID_MAINFRAMEVARIABLES_TEXTCTRL,
23.] = [wx.NewId() for _init_ctrls in range(26)]
24.
25. class Mainframe(wx.Frame):
26.     def _init_ctrls(self, prnt):
27.         # generated method, don't edit
28.         wx.Frame.__init__(self, id=wxID_MAINFRAME, name=u'Mainframe',
29.             parent=prnt, pos=wx.Point(392, 181), size=wx.Size(946, 554),
30.             style=wx.DEFAULT_FRAME_STYLE, title=u'Terminal')
31.         self.SetClientSize(wx.Size(946, 554))
32.         self.SetToolTipString(u'Main window')
33.         self.SetIcon(wx.Icon(u'/home/stanke/Documents/s.ico',
34.             wx.BITMAP_TYPE_ICO))
35.
36.         self.adresa = wx.TextCtrl(id=wxID_MAINFRAMEADRESA, name=u'adresa',
37.             parent=self, pos=wx.Point(0, 0), size=wx.Size(272, 31), style=0,
38.             value=u'localhost:all')
39.         self.adresa.SetToolTipString(u'Adress and port of listening server')
40.         self.adresa.SetEditable(True)
41.         self.adresa.Enable(True)
```

```
42. self.adresa.Bind(wx.EVT_CHAR, self.adress_key_enter)
43.
44. self.connect_btn = wx.Button(id=wxID_MAINFRAMECONNECT_BTN,
45.     label=u'Connect', name=u'connect_btn', parent=self,
46.     pos=wx.Point(272, 0), size=wx.Size(144, 31), style=0)
47. self.connect_btn.SetToolTipString(u'Connect to server')
48. self.connect_btn.Bind(wx.EVT_BUTTON, self.connect_to_server,
49.     id=wxID_MAINFRAMECONNECT_BTN)
50.
51. self.richTextCtrl1 = wx.richtext.RichTextCtrl(id=wxID_MAINFRAMERICHTEXTCTRL1,
52.     parent=self, pos=wx.Point(0, 32), size=wx.Size(416, 232),
53.     style=wx.richtext.RE_MULTILINE, value=u'Succesfully started')
54. self.richTextCtrl1.SetLabel(u'')
55. self.richTextCtrl1.SetToolTipString(u'Server-related informations')
56. self.richTextCtrl1.SetEditable(False)
57. self.richTextCtrl1.SetHelpText(u'')
58. self.richTextCtrl1.Enable(False)
59.
60. self.load_vars_btn = wx.Button(id=wxID_MAINFRAMELOAD_VARS_BTN,
61.     label=u'Load variables', name=u'load_vars_btn', parent=self,
62.     pos=wx.Point(0, 264), size=wx.Size(120, 32), style=0)
63. self.load_vars_btn.SetToolTipString(u'Re/Load variables from param file')
64. self.load_vars_btn.Enable(False)
65. self.load_vars_btn.Bind(wx.EVT_BUTTON, self.load_variables,
66.     id=wxID_MAINFRAMELOAD_VARS_BTN)
67.
68. self.start_btn = wx.Button(id=wxID_MAINFRAMESTART_BTN,
69.     label=u'Start simulation', name=u'start_btn', parent=self,
70.     pos=wx.Point(120, 264), size=wx.Size(120, 32), style=0)
71. self.start_btn.SetToolTipString(u'Start simulation on server')
72. self.start_btn.Enable(False)
73. self.start_btn.Bind(wx.EVT_BUTTON, self.start_simulation,
74.     id=wxID_MAINFRAMESTART_BTN)
75.
76. self.plot_btn = wx.Button(id=wxID_MAINFRAMEPLOT_BTN,
77.     label=u'Draw graph', name=u'plot_btn', parent=self,
78.     pos=wx.Point(666, 0), size=wx.Size(126, 32), style=0)
79. self.plot_btn.SetToolTipString(u'Plot all data received from server')
80. self.plot_btn.Enable(False)
81. self.plot_btn.Bind(wx.EVT_BUTTON, self.plot_values,
82.     id=wxID_MAINFRAMEPLOT_BTN)
83.
84. self.help_btn = wx.Button(id=wxID_MAINFRAMEHELP_BTN, label=u'Help',
```

```

85.         name=u'help_btn', parent=self, pos=wx.Point(352, 264),
86.         size=wx.Size(64, 31), style=0)
87.     self.help_btn.SetToolTipString(u'Help me please, I am stuck!')
88.     self.help_btn.Enable(True)
89.     self.help_btn.Bind(wx.EVT_BUTTON, self.get_help,
90.         id=wxID_MAINFRAMEHELP_BTN)
91.
92.     self.step_slidr = wx.Slider(id=wxID_MAINFRAMESTEP_SLDR, maxValue=100,
93.        minValue=10, name=u'step_slidr', parent=self, pos=wx.Point(104,
94.        296), size=wx.Size(312, 24), style=wx.SL_HORIZONTAL, value=20)
95.     self.step_slidr.SetLabel(u'')
96.     self.step_slidr.Bind(wx.EVT_COMMAND_SCROLL_THUMBTRACK,
97.        self.step_size_modified, id=wxID_MAINFRAMESTEP_SLDR)
98.
99.     self.step_lbl = wx.StaticText(id=wxID_MAINFRAMESTEP_LBL,
100.        label=u'Step size 20ms', name=u'step_lbl', parent=self,
101.        pos=wx.Point(0, 296), size=wx.Size(104, 19), style=0)
102.
103.     self.time_range_lbl = wx.StaticText(id=wxID_MAINFRAMETIME_RANGE_LBL,
104.        label=u'Time range 10s', name=u'time_range_lbl', parent=self,
105.        pos=wx.Point(0, 320), size=wx.Size(104, 19), style=0)
106.
107.     self.time_slidr = wx.Slider(id=wxID_MAINFRAMETIME_SLDR, maxValue=100,
108.        minValue=1, name=u'time_slidr', parent=self, pos=wx.Point(104,
109.        320), size=wx.Size(312, 21), style=wx.SL_HORIZONTAL, value=10)
110.     self.time_slidr.Bind(wx.EVT_COMMAND_SCROLL_THUMBTRACK,
111.        self.time_range_modified, id=wxID_MAINFRAMETIME_SLDR)
112.
113.     self.variables_textctrl = wx.richtext.RichTextCtrl(id=wxID_MAINFRAMEVARIABLES_TEXTCTRL,
114.        parent=self, pos=wx.Point(0, 400), size=wx.Size(208, 184),
115.        style=wx.richtext.RE_MULTILINE,
116.
    value=u'M0=4\nM1=0.36\nM=M0+M1\nls=0.451\ntheta=0.08433\nN=0.1624\nN01=0.3413**0.5\nFr=10\n
    C=0.00145\nng=9.81\nF=0')
117.     self.variables_textctrl.SetToolTipString(u'Conditions and variables')
118.     self.variables_textctrl.SetLabel(u'')
119.     self.variables_textctrl.SetName(u'variables_textctrl')
120.
121.     self.var1_lbl = wx.StaticText(id=wxID_MAINFRAMEVAR1_LBL,
122.        label=u'Variables', name=u'var1_lbl', parent=self, pos=wx.Point(0,
123.        344), size=wx.Size(208, 24), style=0)
124.
125.     self.condt_txtctrl = wx.richtext.RichTextCtrl(id=wxID_MAINFRAMECONDT_TXTCTRL,

```

```
126.         parent=self, pos=wx.Point(208, 368), size=wx.Size(208, 224),
127.         style=wx.richtext.RE_MULTILINE,
128.         value=u'x1_0=1\nx2_0=0\nx3_0=0\nx4_0=0\nxf_0=0')
129. self.condt_txtctrl.SetLabel(u'')
130. self.condt_txtctrl.SetName(u'condt_txtctrl')
131.
132. self.condt_lbl = wx.StaticText(id=wxID_MAINFRAMECONDT_LBL,
133.     label=u'Conditions', name=u'condt_lbl', parent=self,
134.     pos=wx.Point(208, 344), size=wx.Size(208, 24), style=0)
135.
136. self.databaza = wx.richtext.RichTextCtrl(id=wxID_MAINFRAMEDATABAZA,
137.     parent=self, pos=wx.Point(416, 32), size=wx.Size(528, 520),
138.     style=wx.richtext.RE_MULTILINE, value=u'')
139. self.databaza.SetLabel(u'')
140. self.databaza.SetToolTipString(u'Database')
141. self.databaza.SetName(u'databaza')
142.
143. self.show_x1 = wx.CheckBox(id=wxID_MAINFRAMESHOW_X1, label=u'X1',
144.     name=u'show_x1', parent=self, pos=wx.Point(416, 0),
145.     size=wx.Size(50, 32), style=0)
146. self.show_x1.SetValue(True)
147.
148. self.show_x2 = wx.CheckBox(id=wxID_MAINFRAMESHOW_X2, label=u'X2',
149.     name=u'show_x2', parent=self, pos=wx.Point(466, 0),
150.     size=wx.Size(50, 32), style=0)
151. self.show_x2.SetValue(True)
152.
153. self.show_x3 = wx.CheckBox(id=wxID_MAINFRAMESHOW_X3, label=u'X3',
154.     name=u'show_x3', parent=self, pos=wx.Point(516, 0),
155.     size=wx.Size(50, 32), style=0)
156. self.show_x3.SetValue(True)
157.
158. self.show_x4 = wx.CheckBox(id=wxID_MAINFRAMESHOW_X4, label=u'X4',
159.     name=u'show_x4', parent=self, pos=wx.Point(566, 0),
160.     size=wx.Size(50, 32), style=0)
161. self.show_x4.SetValue(True)
162.
163. self.show_f = wx.CheckBox(id=wxID_MAINFRAMESHOW_F, label=u'F',
164.     name=u'show_f', parent=self, pos=wx.Point(616, 0),
165.     size=wx.Size(50, 32), style=0)
166. self.show_f.SetValue(True)
167.
168. self.anime_btn = wx.Button(id=wxID_MAINFRAMEANIME_BTN,
```



```

169.         label=u'Reserved', name=u'anime_btn', parent=self,
170.         pos=wx.Point(240, 264), size=wx.Size(112, 32), style=0)
171.     self.anime_btn.SetToolTipString(u"Show result")
172.     self.anime_btn.Enable(False)
173.
174.     self.draw_btn = wx.Button(id=wxID_MAINFRAMEDRAW_BTN,
175.         label=u'Draw animation', name=u'draw_btn', parent=self,
176.         pos=wx.Point(792, 0), size=wx.Size(152, 32), style=0)
177.     self.draw_btn.SetToolTipString(u'Show animation of process')
178.     self.draw_btn.Bind(wx.EVT_BUTTON, self.draw_anim,
179.         id=wxID_MAINFRAMEDRAW_BTN)
180.
181.     self.ki_ctrl = wx.SpinCtrl(id=wxID_MAINFRAMEKI_CTRL, initial=10,
182.         max=1000, min=0, name=u'ki_ctrl', parent=self, pos=wx.Point(104,
183.         368), size=wx.Size(104, 31), style=wx.SP_ARROW_KEYS)
184.     self.ki_ctrl.SetToolTipString(u'Regulator speed multiplier')
185.
186.     self.kp_ctrl = wx.SpinCtrl(id=wxID_MAINFRAMEKP_CTRL, initial=10,
187.         max=1000, min=0, name=u'kp_ctrl', parent=self, pos=wx.Point(0,
188.         368), size=wx.Size(104, 31), style=wx.SP_ARROW_KEYS)
189.     self.kp_ctrl.SetToolTipString(u'Stabilisor response multiplier')
190.
191.     def __init__(self, parent):
192.         self._init_ctrls(parent)
193.         #own variables
194.         self.connected=False
195.
196.
197.     def connect(self, adress):
198.         try:
199.             self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
200.             HOST,PORT=adress.split(":")
201.             if PORT=="all": #vyskusa vsetky porty od 20000 do 30000
202.                 for i in range(20000,30000):
203.                     try:
204.                         self.s.connect((HOST, int(i)))
205.                         PORT=i
206.                         break
207.                     except:pass
208.             if PORT=="all":
209.                 self.richTextCtrl1.SetValue("No host available")
210.                 raise socket.error
211.         else:

```

```
212.         self.s.connect((HOST, int(PORT)))
213.
214.         self.richTextCtrl1.SetValue("Connected to %s:%s"%(HOST, PORT))
215.         self.adresa.SetValue("%s:%s"%(str(HOST), str(PORT)))
216.         self.adresa.Disable()
217.         self.connect_btn.SetLabel("Disconnect and Close")
218.         self.connect_btn.SetToolTipString(u'Terminate communication with host server and exit program')
219.         self.connected=True
220.         self.load_vars_btn.Enable()
221.
222.     except socket.error:
223.         self.richTextCtrl1.SetValue("Connection refused (no host listening?)")
224.     except socket.gaierror:
225.         self.richTextCtrl1.SetValue("Connection refused (unknown host?)")
226.     except ValueError:
227.         self.richTextCtrl1.SetValue("Connection refused (bad port?)")
228.
229. def connect_to_server(self, event=None):
230.     if self.connected:
231.         self.richTextCtrl1.SetValue("Exiting")
232.         self.s.sendall("exit")
233.         self.s.close()
234.         quit()
235.     else:
236.         self.connect(self.adresa.GetValue())
237.         self.load_vars_btn.SetFocus()
238.
239. def step_size_modified(self, event):
240.     self.step_lbl.SetLabel(u'Step size %dms'%self.step_slidr.GetValue())
241.     self.load_vars_btn.SetFocus()
242.
243. def time_range_modified(self, event):
244.     self.time_range_lbl.SetLabel(u'Time range %ds'%self.time_slidr.GetValue())
245.     self.load_vars_btn.SetFocus()
246.
247. def adress_key_enter(self, event):
248.     if event.GetKeyCode()==13:
249.         self.connect_to_server()
250.     else:
251.         event.Skip()
252.
253. def start_simulation(self, event):
254.     vrange=int(self.time_slidr.GetValue())
```

```

255.     vstep=int(self.step_slider.GetValue())
256.     self.s.sendall("simu")
257.     while self.s.recv(128)[0:3]!="gmt": pass
258.
259.     self.richTextCtrl1.SetValue("Sending time frame")
260.     self.t=[t/1000.0*vstep for t in range(int(vrange*1000/vstep))]
261.
262.     self.s.sendall(str(len(self.t)+128))
263.     while self.s.recv(128)[0:3]!="ack": pass
264.
265.     self.s.sendall("t=[t/1000.0*%d for t in range(int(%d*1000.0/%d))]"%(vstep, vrange, vstep))
266.
267.     while self.s.recv(1024)[0:3]!="gmx": pass
268.     self.richTextCtrl1.SetValue("Sending default values")
269.     self.s.sendall("x0=[%f,%f,%f,%f,%f]\nK=%f"%
        (self.x1_0,self.x2_0,self.x3_0,self.x4_0, self.xf_0,self.xf_0))
270.
271.     data=self.s.recv(1024)
272.     while data[0:4] != "data": data=self.s.recv(1024)[0:4]
273.
274.     self.richTextCtrl1.SetValue("Receiving values")
275.
276.     f = self.s.makefile('rb', int(data[4:])+1024 )
277.     data = pickle.load(f)
278.     f.close()
279.
280.     text=""
281.     for i in data:
282.         for j in i:
283.             text+="%.3f "%j
284.             text+="\n"
285.     self.databaza.SetValue(text)
286.
287.     self.sol=data
288.     self.plot_btn.Enable()
289.     self.anime_btn.Enable()
290.     self.richTextCtrl1.SetValue("Simulation finished succesfully!\nData ready for plotting.")
291.
292.     subor=file("simulation.dat", "w")
293.     subor.write(str(self.step_slider.GetValue())+"\n"+self.databaza.GetValue())
294.     subor.close()
295.
296.     self.plot_btn.SetFocus()

```

```

297.
298. def plot_values(self, event):
299.     plt.ion()
300.     plt.figure()
301.
302.     if self.show_x1.GetValue(): plt.plot(self.t,self.sol[:,0],label="x1")
303.     if self.show_x2.GetValue(): plt.plot(self.t,self.sol[:,1],label="x2")
304.     if self.show_x3.GetValue(): plt.plot(self.t,self.sol[:,2],label="x3")
305.     if self.show_x4.GetValue(): plt.plot(self.t,self.sol[:,3],label="x4")
306.     if self.show_f.GetValue(): plt.plot(self.t,self.sol[:,4],label="F")
307.
308.     plt.grid()
309.     plt.xlabel('Time')
310.     plt.ylabel('')
311.     plt.title('Values')
312.     plt.legend(loc=0)
313.
314. def get_help(self, event):
315.     info = wx.AboutDialogInfo()
316.     info.Name = 'pySim'
317.     info.Version = '1.0.0'
318.     info.Copyright = '(C) Stanislav Rubint 2013'
319.     info.Description = 'Client-server based simulation and regulation program written in python 2.7.3'
320.     wx.AboutBox(info)
321.
322. def draw_anim(self, event):
323.     os.system("python animacia.py")
324.
325. def load_variables(self, event): #nasleduje vypočet bulharských konstant
326.     vars="kP=%f\nkI=%f\n"%(self.kp_ctrl.GetValue(),self.ki_ctrl.GetValue())
327.     +self.variables_textctrl.GetValue()+"\nb2=-theta/(N01**2)\nb4=-N/(N01**2)\na22=-theta*Fr/
328.     (N01**2)\na23=-N**2*g/(N01**2)\na24=N*C/(N01**2)\na25=theta*N/(N01**2)\na42=N*Fr/
329.     (N01**2)\na43=M*N*g/(N01**2)\na44=-M*C/(N01**2)\na45=-N**2/(N01**2)"
330.
331.     self.s.sendall("exec"+vars)
332.     while self.s.recv(16)[0:3]!="ack": pass
333.     self.richTextCtrl1.SetValue("Variables transmitted to server")
334.     self.load_vars_btn.SetLabel("Reload variables")
335.     exec(self.condt_txtctrl.GetValue().replace("x","self.x"))
336.     self.start_btn.Enable()
337.     self.start_btn.SetFocus()

```

```
337. if __name__ == '__main__':  
338.     app = wx.PySimpleApp()  
339.     frame = create(None)  
340.     frame.Show()  
341.     #app.MainLoop()
```

## *animacia.py – zdrojový kód*

```
1. import numpy as np
2. from math import sin, cos
3. import matplotlib.pyplot as plt
4. import matplotlib.animation as animation
5.
6. class Animacia:
7.     def __init__(self):
8.         self.ls=0.451
9.     def update_line(self,n, data, line, ls):
10.         x=float(data[n].split(" ")[0])
11.         fi=float(data[n].split(" ")[2])
12.         line.set_data([x,x-ls*sin(fi)],[0,ls*cos(fi)])
13.         return line,
14.
15.     def rozpohybuj(self):
16.         fig1 = plt.figure()
17.         subor=file("simulation.dat","r")
18.         data=subor.read().split("\n")
19.         cas=data[0]
20.         data=data[1:]
21.         subor.close()
22.         a=[]
23.         for i in data:
24.             a.append(abs(float(i.split(" ")[0])))
25.
26.         l, = plt.plot([], [], '-o')
27.         a=max(a)
28.         a=max([a,self.ls])
29.         plt.xlim(-a-self.ls, a+self.ls)
30.         plt.ylim(-a, a+self.ls)
31.         plt.xlabel('x1')
32.         plt.title('Real time animation @%dfps'%(1000/int(cas)))
33.
34.         line_ani = animation.FuncAnimation(fig1, self.update_line, np.arange(0,len(data)), fargs=(data, l,self.ls),
35.             interval=int(cas), blit=True)
36.
37.         plt.show()
38.
39.     def animuj(event=None):
40.         anim1=Animacia()
41.         anim1.rozpohybuj()
```

42.

43. `if __name__ == "__main__":`

44.  `animuj()`