Objektno programiranje
⟨ Numerički Python ⟩

Tihomir Žilić, Mario Essert, Vladimir Milić

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

Zagreb, 2020./2021.

Sadržaj

1. N-dimenzionalna polja: `numpy.array`
2. Linearna algebra: `scipy.linalg`
3. Interpolacija: `scipy.interpolate`
4. Obične diferencijalne jednadžbe
   - Simulacije dinamičkih sustava
5. Optimizacija: `scipy.optimize` i `CVXOPT`

N-dimenzionalna polja: `numpy.array`

Objekt polje (engl. array)

Može se promatrati kao fleksibilnija i numerički efikasnija lista sa sljedećim pretpostavkama i svojstvima:

- svi elementi moraju biti istog tipa (cijeli, realni ili kompleksni brojevi),
- polja nisu dio standardnog Python-a, nego su dio dodatnog paketa NumPy (engl. *Numerical Python*),
- naredba za učitavanje paketa:

```
>>> from numpy import*
```

- NumPy sadrži veliki broj matematičkih operacija koje se mogu primijeniti na polja, pa se tzv. vektorizacijom može ubrzati izvršavanje Python programa.

Kreiranje polja

```
>>> from numpy import*
>>> n = 4
>>> a = zeros(n) # 1-dimenzionalno polje (vektor)
>>> a
array([ 0., 0., 0., 0.])

>>> p = q = 2
>>> a = zeros((p,q,3)) # 3-dimenzionalno polje (tenzor)
>>> a
array([[[ 0.,  0.,  0.],
        [ 0.,  0.,  0.]],

       [[ 0.,  0.,  0.],
        [ 0.,  0.,  0.]]])
```

```
>>> a = arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = arange(1., 9., 2)   # sintaksa: start, stop, korak
# ne ukljucuje posljednjeg
>>> b
array([ 1., 3., 5., 7.])

>>> c = linspace(0, 1, 6)
>>> c
array([ 0. , 0.2, 0.4, 0.6, 0.8, 1. ])
```

Pretvaranje liste u polje

```
>>> l1 = [0, 1.2, 4, -9.1, 5, 8]
>>> a = array(l1)
>>> a
array([ 0. ,  1.2,  4. , -9.1,  5. ,  8. ])
```

Pretvaranje ugnježđene liste u 2-dimenzionalno polje:

```
>>> x = [0, 0.5, 1]; y = [-6.1, -2, 1.2] # Python liste
>>> a = array([x, y]) # polje sa x i y kao retcima
>>> a
array([[ 0. ,  0.5,  1. ],
       [-6.1, -2. ,  1.2]])
```

Ako Python lista ima samo cjelobrojne elemente (`integer`), tada će i `array` kao rezultat imati polje sa cjelobrojnim elementima

```
>>> z = array([1, 2, 3])
>>> print(z)
[1 2 3]

>>> z = array([1, 2, 3], float)
>>> print(z)
[ 1. 2. 3.]
```

Indeksiranje elemenata polja

```
>>> a = arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[0]
0
>>> a[1]
1
>>> a[2]
2

>>> a = diag(arange(5)) # 2-dimenzionalno polje (matrica)
>>> a
array([[0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 2, 0, 0],
       [0, 0, 0, 3, 0],
       [0, 0, 0, 0, 4]])
>>> a[1,1]
1
```

```
>>> a[2,1] = 10 # treci redak, drugi stupac
>>> a
array([[0, 0, 0, 0, 0],
       [ 0, 1, 0, 0, 0],
       [ 0, 10, 2, 0, 0],
       [ 0, 0, 0, 3, 0],
       [ 0, 0, 0, 0, 4]])
>>> a[1]
array([0, 1, 0, 0, 0])
```

Slicing

```
>>> a = arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2:9:3] # sintaksa: [start:kraj:korak]
# zadnji indeks nije ukljucen)
array([2, 5, 8])

>>> a[:4]
array([0, 1, 2, 3])

>>> a[1:3]
array([1, 2])
>>> a[::2]
array([0, 2, 4, 6, 8])
>>> a[3:]
array([3, 4, 5, 6, 7, 8, 9])
```

Sve tri komponente slice-a nisu potrebne: predodređene vrijednosti su *start*=0, *kraj*=zadnji, *korak*=1.

```
>>> a = eye(5)
>>> a
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
>>> a[2:4,:3] # treci i cetvrti redak, prva tri stupca
array([[0., 0., 1.],
       [0., 0., 0.]])
```

Vektorizacija

Pretpostavimo da imamo funkciju $f(x)$ koju želimo računati u n točaka x_0, x_1, $\ldots$, x_n.

```
>>> def f(x):
... return x**3
...

>>> x2 = linspace(0, 1, 5)
>>> y2 = array([f(xi) for xi in x2])

>>> y2 = f(x2)
>>> y2
array([ 0.  , 0.015625, 0.125  , 0.421875, 1. ])
```

Linearna algebra: `scipy.linalg`

Kreiranje matrica

Matrica $\mathbf{A}$ s n redaka, m stupaca i s elementima a_{ij} zapisuje se kao

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix} = [a_{ij}].$$

- U primjerima koji slijede podrazumijeva se učitavanje paketa:

```
>>> from scipy import*
>>> from scipy import linalg
```

```
A = mat('[1 3 5; 2 5 1; 2 3 8]') #slicno kao u MATLAB-u

    matrix([[1, 3, 5],
            [2, 5, 1],
            [2, 3, 8]])

B = matrix([[2,4,6], [8,10,12], [14,16,18]])

    matrix([[ 2,  4,  6],
            [ 8, 10, 12],
            [14, 16, 18]])

nul_matrica = mat(zeros((5,5))) #nul-matrica

matrica_jedinica = mat(ones((3,2))) #matrica jedinica

jedinicna_matrica = mat(identity(5)) #jedinicna matrica ili
jedinicna_matrica = mat(eye(5))

dijagonalna_matrica = mat(diag([1.,3,5]))
```

Jednakost matrica

Matrica $\mathbf{A}$ dimenzije $n \times m$ je jednaka matrici $\mathbf{B}$ dimenzije $p \times q$ ako vrijedi $n = p$ i $m = q$ i

$$a_{ij} = b_{ij}, \text{ za sve } 1 \leq i \leq n,\ 1 \leq j \leq m$$

U tom slučaju pišemo

$$\mathbf{A} = \mathbf{B}.$$

U Python-u vrlo jednostavno:

```
A == B

    matrix([[False, False, False],
            [False, False, False],
            [False, False, False]], dtype=bool)
```

Zbrajanje matrica

Neka je $\mathbf{A} \in \mathbb{R}^{n \times m}$ i $\mathbf{B} \in \mathbb{R}^{p \times q}$. Ako je $n = p$ i $m = q$ onda matricu $\mathbf{C} \in \mathbb{R}^{n \times m}$ s elementima

$$c_{ij} = a_{ij} + b_{ij}, \quad 1 \leq i \leq n, \;\; 1 \leq j \leq m,$$

zovemo zbrojem ili sumom matrica $\mathbf{A}$ i $\mathbf{B}$ i pišemo

$$\mathbf{C} = \mathbf{A} + \mathbf{B}.$$

U Python-u vrlo jednostavno:

```
C = A + B

    matrix([[ 3,  7, 11],
            [10, 15, 13],
            [16, 19, 26]])
```

Umnožak matrice sa skalarom

Ako je $\mathbf{A} \in \mathbb{R}^{n\times m}$ i $c \in \mathbb{R}$, onda matricu $\mathbf{D} \in \mathbb{R}^{n\times m}$ s elementima

$$d_{ij} = c\, a_{ij}, \quad 1 \leq i \leq n, \ \ 1 \leq j \leq m,$$

zovemo umnožak matrice $\mathbf{A}$ s skalarom c i označavamo

$$\mathbf{D} = c\,\mathbf{A}.$$

U Python-u vrlo jednostavno:

```
c = 3
D = c*A

    matrix([[ 3,  9, 15],
            [ 6, 15,  3],
            [ 6,  9, 24]])
```

Umnožak matrica

Neka je $\mathbf{A} \in \mathbb{R}^{n\times m}$ i $\mathbf{B} \in \mathbb{R}^{m\times p}$, umnožak ili produkt matrica $\mathbf{A}$ i $\mathbf{B}$ je matrica

$$\mathbf{C} = \mathbf{A}\,\mathbf{B} \in \mathbb{R}^{n\times p}$$

čiji elementi su određeni formulom

$$c_{ij} = a_{i1}\,b_{1j} + a_{i2}\,b_{2j} + a_{i3}\,b_{3j} + \ldots + a_{im}\,b_{mj} = \sum_{k=1}^{m} a_{ik}\,b_{kj},\ 1 \leq i \leq n,\ \ 1 \leq j \leq p,$$

U Python-u vrlo jednostavno:

```
C = A*B

    matrix([[ 96, 114, 132],
            [ 58,  74,  90],
            [140, 166, 192]])
```

Transponirana matrica

Neka je $\mathbf{A} \in \mathbb{R}^{n \times m}$. Matrica $\mathbf{A}^\mathsf{T} \in \mathbb{R}^{m \times n}$ se naziva transponirana matrica $\mathbf{A}$, ako je svaki redak od $\mathbf{A}^\mathsf{T}$ jednak odgovarajućem stupcu matrice $\mathbf{A}$.

U Python-u vrlo jednostavno:

```
A.T
```

```
    matrix([[1, 2, 2],
            [3, 5, 3],
            [5, 1, 8]])
```

Determinanta matrice

Neka su a_{ij} elementi kvadratne matrice i neka je $M_{ij} = |\mathbf{A}_{ij}|$ determinanta matrice pomicanjem u lijevo i-tog retka i j-tog stupca. Tada za bilo koji redak i

$$|\mathbf{A}| = \sum_j (-1)^{i+j} a_{ij} \, M_{ij}$$

U Python-u vrlo jednostavno:

```
linalg.det(A)
```

```
    -25.0
```

Inverz matrice

Inverz kvadratne matrice $\mathbf{A}$ je matrica $\mathbf{A}^{-1}$ takva da je $\mathbf{A}\,\mathbf{A}^{-1} = \mathbf{I}$ gdje je $\mathbf{I}$ jedinična matrica.

U Python-u vrlo jednostavno:

```
linalg.inv(A)

    array([[-1.48,  0.36,  0.88],
           [ 0.56,  0.08, -0.36],
           [ 0.16, -0.12,  0.04]])
```

```
A.I

    matrix([[-1.48,  0.36,  0.88],
            [ 0.56,  0.08, -0.36],
            [ 0.16, -0.12,  0.04]])
```

Sustavi linearnih algebarskih jednadžbi

Primjer:

$$
\begin{aligned}
x_1 + 3x_2 + 5x_3 &= 10, \\
2x_1 + 5x_2 + x_3 &= 8, \\
2x_1 + 3x_2 + 8x_3 &= 3.
\end{aligned}
$$

U Python-u vrlo jednostavno:

```
A = mat('[1 3 5; 2 5 1; 2 3 8]')
b = mat('[10;8;3]')
linalg.solve(A,b)

    array([[-9.28],
           [ 5.16],
           [ 0.76]])
A.I*b

    matrix([[-9.28],
            [ 5.16],
            [ 0.76]])
```

Svojstvene (vlastite) vrijednosti i vektori

Problem svojstvenih vrijednosti (vektora):

$$\mathbf{A}\,\mathbf{v} = \lambda\,\mathbf{v}.$$

Svojstvene vrijednosti su rješenja karakterističnog polinoma

$$|\mathbf{A} - \lambda\,\mathbf{I}| = 0.$$

```
A = mat('[1 5 2; 2 4 1; 3 6 2]')
la,v = linalg.eig(A)
l1,l2,l3 = la

>>> la
array([7.95791620+0.j, -1.25766471+0.j,  0.29974850+0.j])

>>> v
array([[-0.5297175 , -0.90730751,  0.28380519],
       [-0.44941741,  0.28662547, -0.39012063],
       [-0.71932146,  0.30763439,  0.87593408]])

>>> sum(abs(v**2),axis=0)
array([1.,  1.,  1.])
```

Interpolacija: `scipy.interpolate`

Linearna interpolacija

Problem interpolacije: izračunavanje vrijednosti neke funkcije koja je zadana samo na diskretnom skupu točaka $(x_k,\ f_k)$, $k = 0,\ 1, \ldots,\ n$, a približne vrijednosti želimo naći u nekim točkama koje nisu iz zadanog skupa. Ako funkcija ϕ ima svojstvo $\phi(x_k) = f_k$, onda kažemo da ϕ interpolira zadane podatke.

Linearna interpolacija se često koristi za aproksimaciju vrijednosti neke funkcije f_k pomoću dvije poznate vrijednosti te funkcije u dvije različite točke. Pogreška ove interpolacije definirana je sa

$$R_T = f_k(x) - p_k(x), \tag{1}$$

gdje je $p_k(x)$ interpolacijski polinom prvog reda

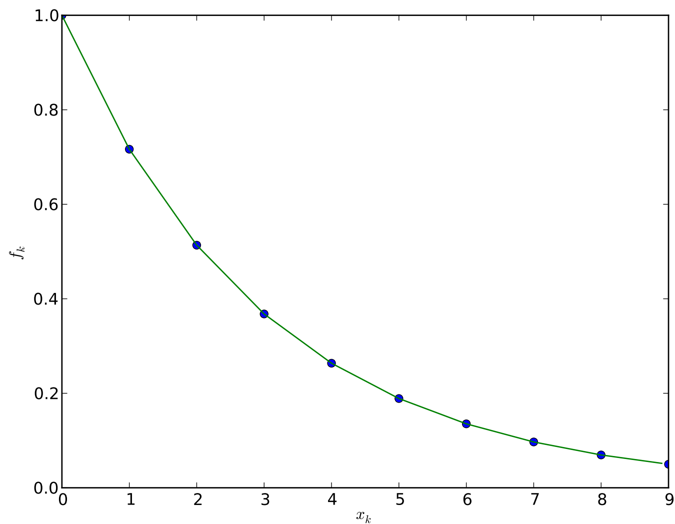
$$p_k(x) = c_{0,k} + c_{1,k}(x - x_{k-1}), \quad x \in [x_{k-1},\ x_k], \quad k = 1,\ \ldots,\ n, \tag{2}$$

gdje su

$$\begin{aligned} c_{0,k} &= f_{k-1}, \\ c_{1,k} &= \frac{f_k - f_{k-1}}{x_k - x_{k-1}}, \quad k = 1,\ \ldots,\ n. \end{aligned} \tag{3}$$

Primjer 1.

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

xk = np.arange(0,10)
fk = np.exp(-xk/3.0)
pk = interpolate.interp1d(xk, fk)

x = np.arange(0,9,0.1)
plt.plot(xk, fk, 'o', x, pk(x),'-')
plt.ylabel('$f_k$')
plt.xlabel('$x_k$')
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
f_k
0
1
2
3
4
5
6
7
8
9
x_k

Splajn (engl. spline) interpolacija

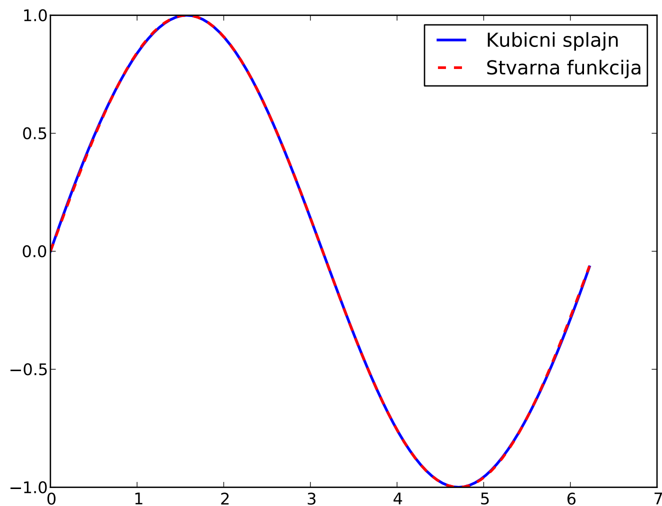
Splajn funkcija reda n je oblika

$$S(x) = \begin{cases} S_0(x) & x \in [x_0,\ x_1] \\ S_1(x) & x \in [x_1,\ x_2] \\ \vdots & \vdots \\ S_{n-1}(x) & x \in [x_{n-1},\ x_n] \end{cases} \tag{4}$$

za $x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n$, gdje su $S_i(x)$ polinomi.

Primjer 2.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate

x = np.arange(0, 2*np.pi+np.pi/4, 2*np.pi/8)
y = np.sin(x)

tck = interpolate.splrep(x, y)
xnew = np.arange(0, 2*np.pi, np.pi/50)
ynew = interpolate.splev(xnew, tck)

plt.figure()
plt.plot(xnew, ynew, xnew, np.sin(xnew), '--r', lw=2)
plt.legend(['Kubicni splajn', 'Stvarna funkcija'])
plt.show()
```


Kubicni splajn
Stvarna funkcija
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6
7

Interpolacija polinomima

Za zadane točke $(x_k,\ y_k)$, $k = 0,\ 1,\ \ldots,\ n$ gdje su $x_i \neq x_j$ za $i \neq j$, postoji interpolacijski polinom $p(x)$ reda n

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n, \tag{5}$$

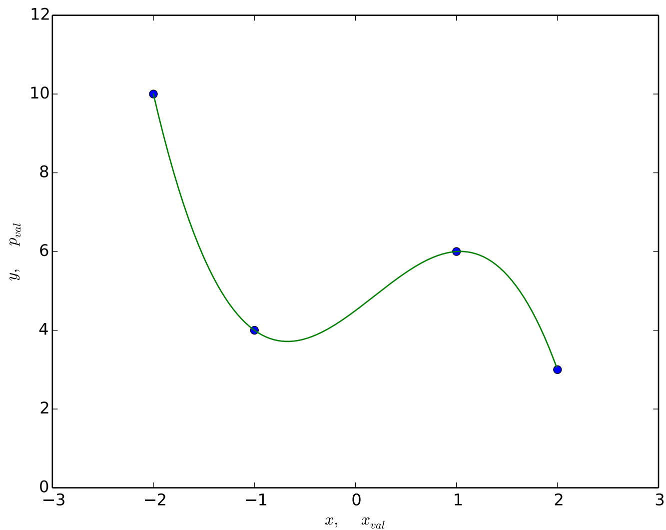
za koji vrijedi

$$p(x_k) = y_k,\ \ k = 0,\ 1,\ \ldots,\ n. \tag{6}$$

Izraz (6) predstavlja sustav od $(n+1)$ linearnih jednadžbi s $(n+1)$ nepoznanica $c_0,\ c_1,\ \ldots, c_n$.

Primjer 3.

```
import numpy as np
from scipy import mat, linalg
import matplotlib.pyplot as plt

x = np.array([-2., -1., 1., 2.]); y = np.array([10., 4., 6., 3.]);

V = mat([x**0, x, x**2, x**3]).T
f = mat('[10;4;6;3]')
c = linalg.lstsq(V, f)

c0 = c[0][0,0]; c1 = c[0][1,0]; c2 = c[0][2,0]; c3 = c[0][3,0];

x_val = np.arange(-2,2,0.01)

p_val = np.polyval([c3, c2, c1, c0], x_val)

plt.plot(x, y, 'o', x_val, p_val)
plt.axis([-3, 3, 0, 12])
plt.xlabel('$x, \;\; x_{val}$'); plt.ylabel('$y, \;\; p_{val}$');
plt.show()
```


12
10
8
6
4
2
0
−3
−2
−1
0
1
2
3
x, x_{val}
y, p_{val}

Obične diferencijalne jednadžbe

Primjer 1.

Običnu linearnu diferencijalnu jednadžbu drugog reda

$$\ddot{x}(t) + \dot{x}(t) + 10\,x(t) = 0,$$

možemo zapisati kao sustav od dvije diferencijalne jednadžbe prvog reda

$$\begin{aligned}\dot{x}_1 &= x_2,\\ \dot{x}_2 &= -10\,x_1 - x_2,\end{aligned}$$

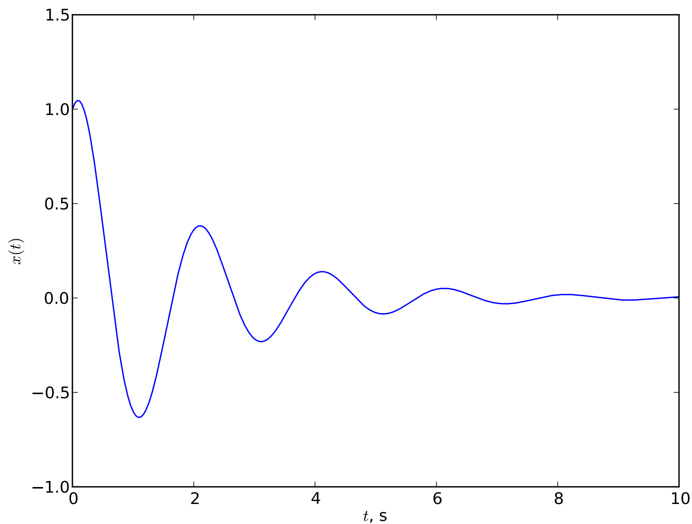
gdje su $x_1 = x$ i $x_2 = \dot{x}$. U matričnom zapisu gornji sustav je sljedećeg oblika

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -10 & -1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}.$$

Rješenje je oblika:

$$\mathbf{x}(t) = e^{t\,\mathbf{A}}\,\mathbf{x}(0)$$

```
from scipy.linalg import expm
import matplotlib.pyplot as plt

A = mat('[0 1;-10 -1]')
x0 = mat('[1;1]')
xp = mat('[1 0]')
t_val = arange(0,10,0.01)
x = [xp*expm(t*A)*x0 for t in t_val]

plt.plot(t_val,array(x).reshape(len(t_val),))
plt.xlabel('$t, s$')
plt.ylabel('$x(t)$')
plt.show()
```


1.5
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
$x(t)$
t, s

Simulacije dinamičkih sustava

- Razmatramo Pythonove ugrađene funkcije (engl. *built-in*) s numeričkim metodama za aproksimaciju rješenja običnih diferencijalnih jednadžbi prvog reda oblika

$$\dot{x}(t) = f(x, t), \quad x(t_0) = x_0, \tag{7}$$

  na vremenskom intervalu $t \in [t_0,\ t_f]$.

- Općenitiji problem je sustav od n običnih diferencijalnih jednadžbi prvog reda

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, x_2, \dots x_n, t), \quad x_1(t_0) = x_{10}, \\ \dot{x}_2 &= f_2(x_1, x_2, \dots x_n, t), \quad x_2(t_0) = x_{20}, \\ &\vdots \\ \dot{x}_n &= f_n(x_1, x_2, \dots x_n, t), \quad x_n(t_0) = x_{n0}, \end{aligned} \tag{8}$$

  za n nepoznatih realnih funkcija $x_i(t)$, $i = 1, 2, \dots, n$. Sustav jednadžbi (8) možemo napisati u obliku analognom izrazu (7) koristeći vektorsku notaciju

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{9}$$

  gdje su

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix}, \quad \mathbf{f}(\mathbf{x}, t) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n, t) \\ f_2(x_1, x_2, \dots, x_n, t) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n, t) \end{bmatrix}, \quad \mathbf{x}_0 = \begin{bmatrix} x_{10} \\ x_{20} \\ \vdots \\ x_{n0} \end{bmatrix}.$$

- U matematičkom opisu dinamičkih sustava pojavljuju se diferencijalne jednadžbe višeg reda oblika

$$x^{(n)} = f\left(x, \dot{x}, \ddot{x}, \ldots, x^{(n-1)}, t\right), \quad x^{(i)}(t_0) = x_{i0}, \;\; i = 0, 1, 2, \ldots, n-1, \tag{10}$$

koje se svode na sustav diferencijalnih jednadžbi prvog reda. Da bi se to postiglo potrebno je uvesti dodatne funkcije

$$\begin{aligned} x_1(t) &:= x(t), \\ x_2(t) &:= \dot{x}(t), \\ x_3(t) &:= \ddot{x}(t), \\ &\vdots \\ x_n(t) &:= x^{(n-1)}(t), \end{aligned}$$

čime se diferencijalna jednadžba (10) transformira u ekvivalentni sustav diferencijalnih jednadžbi prvog reda u vektorskom obliku

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_n \\ f(x_1, x_2, \ldots, x_n, t) \end{bmatrix}, \tag{11}$$

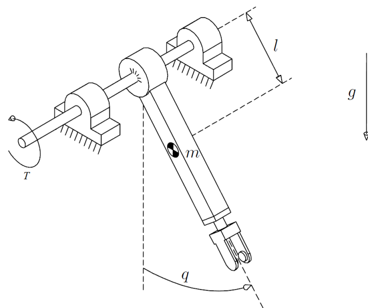
pa možemo koristiti numeričke metode za rješavanje diferencijalne jednadžbe (7).

Mehanički sustav s jednim stupnjem slobode gibanja pogonjen istosmjernim električnim motorom

- Gibanje mehaničkog sustava prikazanog na slici 1 opisano je sljedećom diferencijalnom jednadžbom

$$J\,\ddot{q}(t) + B\,\dot{q}(t) + m\,g\,l\,\sin\left(q(t)\right) = T, \tag{12}$$

gdje su q kut zakreta članka [rad], T ulazni moment [Nm], J ukupni moment tromosti oko osi koja prolazi zglobom [kgm^2], B koeficijent viskoznog prigušenja [Nms/rad], masa članka [kg], l udaljenost težišta od osi zgloba [m], g akceleracija sile teže [m/s^2].



Slika: Preuzeto iz: R. Kelly, V. Santibanez and A. Loria: *Control of Robot Manipulators in Joint Space*, Springer-Verlag, London, 2005.

- Dinamika el. motora prikazanog na slici 2 opisana je sljedećom diferencijalnom jednadžbom:

$$R_a\, i_a(t) + L_a\, \frac{di_a(t)}{dt} + e = u_a, \tag{13}$$

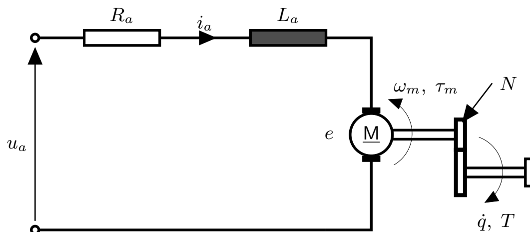
gdje su u_a napon armature [V], i_a struja armature [A], R_a ukupni radni otpor armaturnog kruga [Ω], L_a ukupni induktivitet armaturnog kruga [H] i e protuelektromotorna sila [V].

- Nadalje imamo,

$$e = K_e\, \Phi_n\, \omega_m = K_v\, \omega_m \longrightarrow \text{protuelektromotorna sila}, \tag{14}$$

$$\tau_m = K_t\, i_a \longrightarrow \text{moment motora uz } \Phi_n = const., \tag{15}$$

pri čemu su K_e konstrukcijska konstanta motora, Φ_n nazivna vrijednost glavnog magnetskog toka po polu [Vs], K_v naponska konstanta motora [Vs], ω_m mehanička brzina vrtnje [s^{-1}] i K_t momentna konstanta motora [Nm/A].



Slika: Istosmjerni motor s nezavisnom i konstantnom uzbudom.

- Uzimajući u obzir prijenosni omjer N, imamo relacije

$$T = N\,\tau_m = N\,K_t\,i_a, \tag{16}$$
$$\omega_m = N\,\dot{q}, \tag{17}$$

pa je ukupni sustav opisan sljedećim diferencijalnim jednadžbama

$$J\,\ddot{q} + B\,\dot{q} + m\,g\,l\,\sin(q) = N\,K_t\,i_a, \tag{18}$$
$$L_a \frac{di_a}{dt} + R_a\,i_a + K_v\,N\,\dot{q} = u_a. \tag{19}$$

- Uvodimo nove zavisne varijable definirane na sljedeći način

$$x_1(t) = q(t), \quad x_2(t) = \dot{q}(t), \quad x_3(t) = i_a(t). \tag{20}$$

- Deriviranjem varijabli x_1, x_2, x_3 po vremenu t i na osnovu sustava (18)-(19) dobivamo sustav diferencijalnih jednadžbi prvog reda

$$\dot{x}_1 = x_2, \tag{21}$$
$$\dot{x}_2 = -\frac{m\,g\,l}{J}\,\sin(x_1) - \frac{B}{J}\,x_2 + \frac{N\,K_t}{J}\,x_3, \tag{22}$$
$$\dot{x}_3 = -\frac{K_v\,N}{L_a}\,x_2 - \frac{R_a}{L_a}\,x_3 + \frac{1}{L_a}\,u_a. \tag{23}$$

Simulacija u Pythonu

Optimizacija: `scipy.optimize` i CVXOPT

Matematička formulacija problema optimizacije

Problem optimizacije definiran je na sljedeći način

$$\begin{aligned} &\min_{\mathbf{x}\in\mathcal{R}^n} f(\mathbf{x}) \\ &\text{s. t. } \mathbf{h}(\mathbf{x}) = 0, \quad \mathbf{g}(\mathbf{x}) \geq 0 \end{aligned} \tag{24}$$

Modul scipy.optimize:

- `minimize(func, x0[, args=(), method='BFGS', ...])`
  Minimizira skalarnu funkciju cilja jedne ili više varijabli primjenom metoda: BFGS, Nelder-Mead simplex, Newton Conjugate Gradient, COBYLA, SLSQP.
- `fmin(func, x0[, args=(), xtol, ftol, ...])`
  Minimizira funkciju cilja primjenom simplex metode.
- `fmin_powell(func, x0[, args=(), xtol, ftol, ...])`
  Minimizira funkciju cilja primjenom modificirane Powell-ove metode.
- `fmin_cg((f, x0[, fprime, args=(), ...])`
  Minimizira funkciju cilja primjenom nelinearne metode konjugiranog gradijenta.
- `fmin_bfgs(f, x0[, fprime, args=(), ...])`
  Minimizira funkciju cilja primjenom Broyden–Fletcher–Goldfarb–Shanno (BFGS) metode.

...

Primjer 1.

Potrebno je riješiti sljedeći optimizacijski problem:

$$\begin{aligned} \min \quad & x_1^2 + 2x_2^2 \\ \text{s.t.} \quad & 4x_1 + x_2 \leq 6, \\ & x_1 + x_2 = 3, \\ & x_1 \geq 0, \; x_2 \geq 0. \end{aligned} \tag{25}$$

```
import numpy as np
from scipy.optimize import minimize

def f(x, sign=1.0):
    x1 = x[0]; x2 = x[1];
    return sign*(x1**2 + 2.0*x2**2)

def dfdx(x, sign=1.0):
    x1 = x[0]; x2 = x[1]
    dfdx1 = 2.0*x1
    dfdx2 = 4.0*x2
    return np.array([dfdx1, dfdx2])

const=({'type': 'ineq', 'fun': lambda x: np.array([-4.0*x[0]-x[1]+6.0])},
         {'type': 'eq', 'fun': lambda x: np.array([x[0] + x[1]-3.0])},
         {'type': 'ineq', 'fun': lambda x: np.array([x[0]])},
         {'type': 'ineq', 'fun': lambda x: np.array([x[1]])})

res = minimize(f, [0.0, 0.0], jac=dfdx, constraints=const,
               method='SLSQP', options={'disp': True})
print(res.x)
```

```
>>>
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 9.0
            Iterations: 3
            Function evaluations: 4
            Gradient evaluations: 3
[ 1.  2.]
>>>
```

Primjer 2.

Kemijski proces u reaktoru opisan je sljedećim diferencijalnim jednadžbama

$$\begin{aligned}\dot{x}_1(t) &= k_1 u(t) - k_3 x_1(t), \quad x_1(0) = 0,\\ \dot{x}_2(t) &= 2k_2 u(t)^2, \quad x_2(0) = 0,\\ \dot{x}_3(t) &= k_3 x_1(t), \quad x_3(0) = 0.\end{aligned} \tag{26}$$

Potrebno je odrediti varijablu $u(t)$ koja minimizira funkciju cilja

$$J = -x_1(t_f) + x_2(t_f) + x_3(t_f), \tag{27}$$

gdje je $t_f = \ln(2)/k_3$. Analitičko rješenje problema je:

$$\begin{aligned}u(t) &= \frac{k_1}{4k_2}\left(e^{k_3 t} - 1\right),\\ x_1(t) &= \frac{k_1^2}{8k_2k_3}\left(e^{k_3 t} + e^{-k_3 t} - 2\right),\\ x_2(t) &= \frac{k_1^2}{16k_2k_3}\left(e^{2k_3 t} - 4e^{k_3 t} + 2k_3 t + 3\right),\\ x_3(t) &= \frac{k_1^2}{8k_2k_3}\left(e^{k_3 t} - e^{-k_3 t} - 2k_3 t\right).\end{aligned} \tag{28}$$

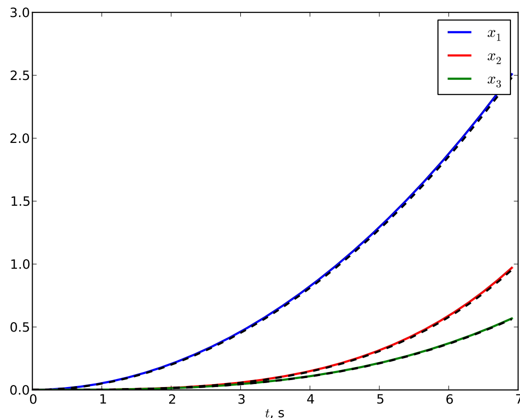
```
from numpy import*
import matplotlib.pyplot as plt
from scipy.optimize import fmin_bfgs
from math import exp

k1r = 2.0; k2r = 1.0; k3r = 0.1; # vrijednosti parametara reaktora

n = 3; # red sustava
nu = 1; # broj upravljackih varijabli

t0 = 0; # pocetno vrijeme
T = log(2)/k3r; # konacno vrijeme
N = 300; # broj vremenskih intervala
tau = (T-t0)/N; # korak integracije

U0 = zeros((nu, N))
```

```
def my_euler(u, h, n_i):
    x=zeros((n,N))
    for i in range(n_i):
        for j in range(n):
            f=reactor(x,u,i)
            x[j,i+1]=x[j,i]+tau*f[j]
    return x.T

def reactor(x, u, i):
    return array([ k1r*u[i]-k3r*x[0,i],
                   2*k2r*u[i]*u[i],
                   k3r*x[0,i] ])

def cost_function(x):
    xout = my_euler(x,tau,N-1)
    J = -xout[N-1,0] + xout[N-1,1] + xout[N-1,2]
    return J

def control(u):
    uout = fmin_bfgs(cost_function,u)
    return uout
```

```
uopt=control(U0)

Xopt=my_euler(uopt, tau, N-1)

t_val=arange(t0, T, tau) #vremenski interval

x1,x2,x3=Xopt.T

ua = [k1r*(exp(k3r*t_vec)-1)/4/k2r for t_vec in t_val]
x1a = [k1r**2*(exp(k3r*t_vec)+exp(-k3r*t_vec)-2)/8/k2r/k3r...
for t_vec in t_val]
x2a = [k1r**2*(exp(2*k3r*t_vec)-4*exp(k3r*t_vec)+2*k3r*t_vec+3)...
/16/k2r/k3r for t_vec in t_val]
x3a = [k1r**2*(exp(k3r*t_vec)-exp(-k3r*t_vec)-2*k3r*t_vec)...
/8/k2r/k3r for t_vec in t_val]
```

```
plt.plot(t_val, x1, t_val, x2, 'r', t_val, x3, 'g',...
t_val,x1a,'--k', t_val, x2a, '--k', t_val, x3a, '--k', lw=2)
plt.legend(['$x_1$', '$x_2$', '$x_3$'])
plt.xlabel('$t$, s')
plt.show()
```

```
>>>
Optimization terminated successfully.
         Current function value: -0.970405
         Iterations: 2
         Function evaluations: 1208
         Gradient evaluations: 4
```



CVXOPT

```
from cvxopt import matrix

A = matrix([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], (2,3))
print A
    [ 1.00e+000  3.00e+000  5.00e+000]
    [ 2.00e+000  4.00e+000  6.00e+000]

B = matrix([ [1.0, 2.0], [3.0, 4.0] ])
print B
    [ 1.00e+00  3.00e+00]
    [ 2.00e+00  4.00e+00]
```

Primjer 3.

Definicija problema linearnog programiranja:

$$\begin{aligned} \min \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{G}\mathbf{x} + \mathbf{s} = \mathbf{h} \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{s} \succeq 0 \end{aligned}$$

Primjer:

$$\begin{aligned} \min \quad & -4x_1 - 5x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \leq 3 \\ & x_1 + 2x_2 \leq 3 \\ & x_1 \geq 0, \quad x_2 \geq 0 \end{aligned}$$

```
from cvxopt import matrix, solvers

c = matrix([-4., -5.])
G = matrix([[2., 1., -1., 0.], [1., 2., 0., -1.]])
h = matrix([3., 3., 0., 0.])
sol = solvers.lp(c, G, h)

     pcost       dcost       gap    pres   dres   k/t
 0: -8.1000e+000 -1.8300e+001  4e+000 0e+000 8e-001 1e+000
 1: -8.8055e+000 -9.4357e+000  2e-001 7e-017 4e-002 3e-002
 2: -8.9981e+000 -9.0049e+000  2e-003 2e-016 5e-004 4e-004
 3: -9.0000e+000 -9.0000e+000  2e-005 1e-016 5e-006 4e-006
 4: -9.0000e+000 -9.0000e+000  2e-007 5e-016 5e-008 4e-008
Optimal solution found.

print sol['x']

[ 1.00e+000]
[ 1.00e+000]
```