

Projektet

Teknisk matematik

SF1672

Olof Runborg

Numerisk analys, Matematik, KTH

HT 2021

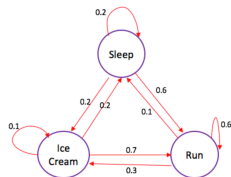
- Projektet görs i grupper om 4-5 studenter.
- 6 projektområden att välja mellan (max 2-3 grupper/område)
- Inom varje område hittar gruppen själva på en konkret frågeställning/tillämpning.
- Självständigt arbete till största del, men tre handledningstillfällen i datorsal.
- Projektet redovisas med en poster.

All info om projektet finns i Canvas

(Modul: Projekt HT21)

- **Markovkedjor**

Tillståndsmodell där nästa tillstånd väljs slumpmässigt bara baserat på nuvarande tillstånd. Massor av tillämpningar.



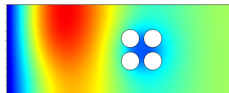
- **Grafteori**

Beskriv grafer med matriser. Hitta strukturer i data med hjälp av matrisoperationer.



- **Temperaturberäkningar**

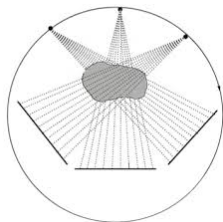
Temperaturen "här" = medelvärdet av temperaturen "i närheten". Ger linjärt ekvationssystem.



Handledare för dessa områden: Simen

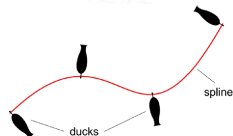
- **Datortomografi**

Generera en bild av kroppens inre från scanning med röntgenstrålar, genom att lösa ett överbestämt linjärt ekvationsystem.



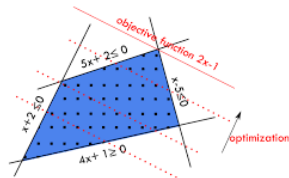
- **Splines**

Skapa mjuka kurvor genom givna punkter.



- **Linjärprogrammering**

Minimera kostnadsfunktion under linjära bivillkor.



Handledare för dessa områden: Olof

1 Val av projektområde

Välj område och vilka du vill arbeta med. Du gör detta genom att gå med i en av de Canvas-grupper som skapats för de olika projektområdena:

SF1672 - CTMAT - PROJEKT <PROJEKTOMRÅDE> <NR>

Ni som arbetar ihop ska vara med i samma Canvas-grupp.

2 Val av problem

Hitta tillsammans på en egen problemställning/tillämpning inom området, gärna problem där datorn verkligen behövs som hjälpmedel. Var kreativa!

3 Projektsammanfattning

Skriv en kort (1-2 sidor) översikt av den bakomliggande teorin och den frågeställning/tillämpning ni valt. Skickas in via Canvas.

4 Projektarbete

5 Poster

Gör poster till projektredovisningen. Skickas in via Canvas ett par dagar före redovisningen.

6 Redovisning. Postermässa. Alla i gruppen ska vara med.

- **16/11:** **Deadline** för val av projektområde. (Och att gå med i en grupp.)
Kontakta Olof för att få fullständig text om området.
- **17/11:** Labtillfälle 1
- **19/11:** **Deadline** för projektsammanfattningen.
- **22/11:** Labtillfälle 2
- **29/11:** Labtillfälle 3, utkast på poster klart
- **7/12:** **Deadline** för inlämning av poster.
- **10/12:** Postermässa kl 10-12.

Beräkningar med Python

- Behöver ofta kunna göra beräkningar med **stora matriser**.
- Modulerna `numpy` ("numerical python") och `scipy` ("scientific python") ger verktyg för detta och andra numeriska beräkningar i Python, med liknande funktionalitet som Matlab.
- Överlapp mellan modulerna. Vi fokuserar på `numpy`.
- Exempel på funktionalitet inom linjär algebra:
 - Effektiva matris-operationer som multiplikation $\text{matris} \times \text{vektor}$ och $\text{matris} \times \text{matris}$.
 - Lösning av linjära ekvationssystem.
 - Beräkning av egenvärden.
- Andra numeriska algoritmer:
 - Numerisk integration.
 - Lösning av algebraiska ekvationer.
 - Lösning av differentialekvationer.
- Modulen `matplotlib` ger plot-funktioner, för att tex plotta kurvor och grafer i 3D.

- `numpy.org` innehåller material och resurser om `numpy`. Mycket av det jag går igenom idag återfinns i guiden *NumPy: the absolute basics for beginners*. Länkar till denna och andra lämpliga delar av `numpy.org` finns på Canvas.
- Modulerna måste först importeras

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

- Kommandona i `numpy` och `matplotlib.pyplot` kan sedan användas med prefixen "`np`" och "`plt`".

- Datatypen `np.array` används för att hantera matriser och vektorer. (Finns även `np.matrix`.) Effektivare än `list`, `tuple` och Pythons vanliga `array`.
- En vektor skapas på liknande sätt som en lista.

```
>>> x = np.array([1, 4, 9, -19])
```

motsvarar $x = \begin{pmatrix} 1 \\ 4 \\ 9 \\ -19 \end{pmatrix}.$

- Särskiljer inte rad- och kolumnvektorer. (Till skillnad från Matlab.)
- En matris skapas genom att separera vektorer med komma.

```
>>> A = np.array([[1, 4, 9], [-2, 4, -5], [-8, 18, 0]])
```

motsvarar $A = \begin{pmatrix} 1 & 4 & 9 \\ -2 & 4 & -5 \\ -8 & 18 & 0 \end{pmatrix}$

numpy array – indexing

- Elementen i en array nås via indexing inom hakparenteser, där 0 motsvarar första elementet:

```
>>> print(x[0])  
1  
>>> print(x[3])  
-19
```

$$\text{för } x = \begin{pmatrix} 1 \\ 4 \\ 9 \\ -19 \end{pmatrix}.$$

- För matriser används två index, där det första motsvarar raden.

```
>>> print(A[0,2])  
9  
>>> print(A[2,0])  
-8
```

$$\text{för } A = \begin{pmatrix} 1 & 4 & 9 \\ -2 & 4 & -5 \\ -8 & 18 & 0 \end{pmatrix}$$

- Går också att använda "ranges" som index:

```
>>> print(x[1:3])  
[4 9]
```

```
>>> print(A[1:3,1:3])  
[[ 4 -5]  
 [18  0]]
```

- Ett ensamt kolon ":" betyder "hela raden/kolumnen"

```
>>> print(A[0,:])  
[1 4 9]
```

```
>>> print(A[:,1])  
[ 4  4 18]
```

- Multiplikation av två arrayer av samma storlek medför *elementvis* multiplikation:

```
>>> A = np.array([[1, 4, 9], [-2, 4, -5], [-8, 18, 0]])
>>> B = np.array([[3, -7, 1], [3, 3, -1], [2, -23, 11]]);
>>> print(A*B)
[[  3 -28  9]
 [ -6 12  5]
 [-16 -414 0]]
```

$$\text{dvs } A = \begin{pmatrix} 1 & 4 & 9 \\ -2 & 4 & -5 \\ -8 & 18 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & -7 & 1 \\ 3 & 3 & -1 \\ 2 & -23 & 11 \end{pmatrix}, \quad \text{ger } \begin{pmatrix} 1 \times 3 & 4 \times (-7) & 9 \times 1 \\ -2 \times 3 & 4 \times 3 & -5 \times (-1) \\ -8 \times 2 & 18 \times (-23) & 0 \times 11 \end{pmatrix}$$

- Matrismultiplikation görs med `np.matmul(A, B)` eller `"@"`:

```
>>> print(np.matmul(A, B))
[[ 33 -202  96]
 [ -4 141 -61]
 [ 30 110 -26]]
>>> print(A@B)    # ger samma
```

$$AB = \begin{pmatrix} 33 & -202 & 96 \\ -4 & 141 & -61 \\ 30 & 110 & -26 \end{pmatrix}$$

$$\text{Ex.: } 33 = 1 \times 3 + 4 \times 3 + 9 \times 2$$

numpy array – multiplikation

- Matris-vektor-multiplikation görs också med `np.matmul(A, y)` eller "@":

```
>>> y = np.array([-1, 0, 3]);  
>>> print(np.matmul(A, y))  
[ 26 -13   8]
```

$$\begin{pmatrix} 1 & 4 & 9 \\ -2 & 4 & -5 \\ -8 & 18 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 26 \\ -13 \\ 8 \end{pmatrix}$$

- OBS! `np.matmul` kräver att matriser/vektorer har giltiga dimensioner:

```
>>> ykort = np.array([-1, 0]);  
>>> print(np.matmul(A, ykort))  
Traceback (most recent call last):  
  File "<pyshell#221>", line 1, in <module>  
    print(np.matmul(A, ykort))  
ValueError: matmul: Input operand 1 has a mismatch...
```

$$\begin{pmatrix} 1 & 4 & 9 \\ -2 & 4 & -5 \\ -8 & 18 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \end{pmatrix} = ???$$

- Notera specialbeteende för "vanlig" multiplikation av matris och vektor med giltig dimension. Ger en *matris*

```
>>> print(A*y)  
[[ -1   0  27]  
 [  2   0 -15]  
 [  8   0  0]]
```

$$\text{ger } \begin{pmatrix} 1 \times 1 & 4 \times 0 & 9 \times 3 \\ -2 \times 1 & 4 \times 0 & -5 \times 3 \\ -8 \times 1 & 18 \times 0 & 0 \times 3 \end{pmatrix}$$

- För att lösa linjära ekvationssystem $A\mathbf{x} = \mathbf{b}$ används

`np.linalg.solve(A,b):`

```
>>> A = np.array([[1, 4, 9], [-2, 4, -5], [-8, 18, 0]])
>>> b = np.array([5, -10, 2]);
>>> x = np.linalg.solve(A,b)
>>> print(x)
[-5.99065421 -2.55140187  2.35514019]
>>> print(np.matmul(A,x))
[  5. -10.   2.]
```

$$\text{dvs} \quad \begin{pmatrix} 1 & 4 & 9 \\ -2 & 4 & -5 \\ -8 & 18 & 0 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 5 \\ -10 \\ 2 \end{pmatrix} \Rightarrow \mathbf{x} = \begin{pmatrix} -5.99065421 \\ -2.55140187 \\ 2.35514019 \end{pmatrix}$$

- Finns också en funktion för att beräkna inversen, men den är inte effektiv/stabil för stora system. Rekommenderas ej.

```
>>> Ai = np.linalg.inv(A)
>>> print(Ai)
[[ 0.42056075  0.75700935 -0.26168224]
 [ 0.18691589  0.3364486  -0.06074766]
 [-0.01869159 -0.23364486  0.05607477]]
```

```
>>> print(np.matmul(Ai,b))
[-5.99065421 -2.55140187  2.35514019]
```

$$A^{-1}\mathbf{b} = \mathbf{x}.$$

Andra praktiska funktioner för matriser

- Transponering $A \mapsto A^T$

```
>>> At = np.transpose(A)    (eller >>> At = A.transpose() )
```

- Storleken på matris/vektor

```
>>> s = np.shape(A)    - ger vektorn s=[a,b] där a är antal rader och  
                        b antal kolumner i A
```

- Skapa speciella matriser

```
>>> A = np.empty((4,5))    - 4x5-matris med oinitialiserade element  
>>> A = np.eye(3)         - 3x3 identitetsmatris  
>>> A = np.ones((3,2))    - 3x2-matris med alla element = 1  
>>> A = np.zeros((3,2))   - 3x2-matris med alla element = 0  
>>> A = np.diagflat([-2,2,3]) - 3x3 diagonal matris med elementen  
                             [-2,2,3] på diagonalen
```

numpy – andra praktiska funktioner

- Beräkna egenvärden och egenvektorer

```
>>> lam, S = np.linalg.eig(A)    - lam är en vektor med alla egenvärden
                                   - S är en matris vars kolumner är
                                   motsvarande egenvektorer
```

- Determinant, $\det A$, och norm, $\|x\|$

```
>>> np.linalg.det(A)             - determinanten till A
>>> np.linalg.norm(x)            - euklidiska längden av x
```

- Om man vill applicera en funktion på alla element i en matris samtidigt kan man använda inbyggda "universal functions".

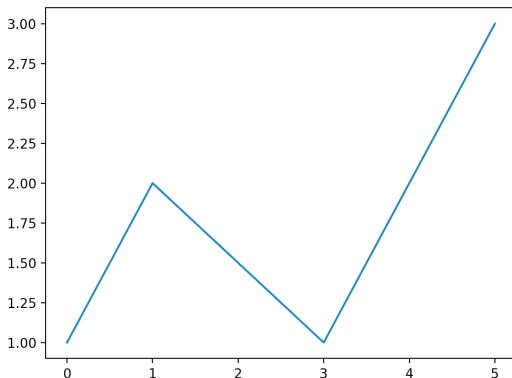
```
>>> import math
>>> C = math.sin(A)               <--- funkar inte
>>> C = np.sin(A)                 <--- använd istället "universal function" i numpy
```

- Egna funktioner kan bli konverterade till "universal functions"

```
>>> C = myfun(A)                  <--- funkar inte
>>> myfunv = np.vectorize(myfun)
>>> C = myfunv(A)                 <--- ok
```

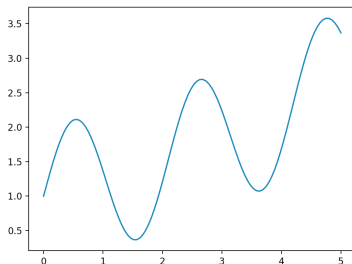
- `plt.plot(x, y)` plottar värdena i vektorn `x` mot värdena i vektorn `y`, dvs en linje mellan punkterna med koordinaterna $(x[0], y[0])$, $(x[1], y[1])$, ...

```
>>> plt.plot([0, 1, 3, 5], [1, 2, 1, 3])
```



- `plt.plot(x, y)` används typiskt tillsammans med `np.linspace` och "universal functions"

```
>>> x = np.linspace(0,5,100)    - skapar en vektor x med 100  
                                  jämnt fördelade värden i  
                                  intervallet [0,5]  
>>> x = np.arange(0,5,0.1)      - alternativ, liknande range  
>>> y = np.sin(x*3)+np.exp(x/5) - evaluerar funktionen  
                                  sin(x*3)+exp(x/5) i alla  
                                  x-punkter och skapar  
                                  motsvarande vektor y  
>>> plt.plot(x,y)              - plottar kurvan
```



- `matplotlib` kan användas i vanlig eller interaktiv mod:
 - **Vanlig:** Alla plottar visas först när man skriver `plt.show()`. Programmet stannar och väntar till man manuellt stänger plot-fönstren.
 - **Interaktiv:** Plottarna visas direkt efter man gjort `plt.plot()`. Programmet stannar inte. Plottarna stängs när programmet kört klart, eller när man gjort `plt.close()` vid prompten.
 - `plt.ion()` och `plt.ioff()` slår på/av interaktiv mod.
- Många, många plot-typer och formatterings-möjligheter. (Se tutorial.)
- Arrayer kan plottas som bilder med `plt.imshow(X)`. (Se dokumentation.)
- Arrayer kan plottas som ytor (grafnen av en funktion av 2 variabler). Lite mer komplicerat. Kräver även modulen `mpl_toolkits.mplot3d`. (Se exempelfil.)

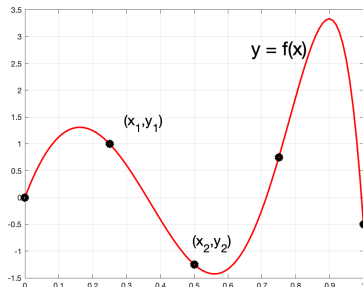
Vi vill hitta koefficienterna c_0, c_1, \dots så att funktionen

$$f(x) = c_0 + c_1 e^x + c_2 e^{2x} + c_3 e^{3x} + c_4 e^{4x}$$

interpolerar punkterna i tabellen

x	0	0.25	0.5	0.75	1
y	0	1	-1.25	0.75	-0.5

Olika c_j ger olika kurvor. För ett speciellt val går funktionen genom alla punkterna.



- Kalla punkterna x_j och y_j , med $j = 0, \dots, 4$. Vi söker alltså c_j så att

$$f(x_j) = y_j, \quad j = 0, \dots, 4.$$

- Vi kan skriva ut det som

$$c_0 + c_1 e^{x_0} + c_2 e^{2x_0} + c_3 e^{3x_0} + c_4 e^{4x_0} = y_0,$$

$$c_0 + c_1 e^{x_1} + c_2 e^{2x_1} + c_3 e^{3x_1} + c_4 e^{4x_1} = y_1,$$

$$c_0 + c_1 e^{x_2} + c_2 e^{2x_2} + c_3 e^{3x_2} + c_4 e^{4x_2} = y_2,$$

$$c_0 + c_1 e^{x_3} + c_2 e^{2x_3} + c_3 e^{3x_3} + c_4 e^{4x_3} = y_3,$$

$$c_0 + c_1 e^{x_4} + c_2 e^{2x_4} + c_3 e^{3x_4} + c_4 e^{4x_4} = y_4.$$

- Detta är ett linjärt ekvationssystem!

$$\underbrace{\begin{pmatrix} 1 & e^{x_0} & e^{2x_0} & e^{3x_0} & e^{4x_0} \\ 1 & e^{x_1} & e^{2x_1} & e^{3x_1} & e^{4x_1} \\ 1 & e^{x_2} & e^{2x_2} & e^{3x_2} & e^{4x_2} \\ 1 & e^{x_3} & e^{2x_3} & e^{3x_3} & e^{4x_3} \\ 1 & e^{x_4} & e^{2x_4} & e^{3x_4} & e^{4x_4} \end{pmatrix}}_A \underbrace{\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}}_c = \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}}_y.$$