Pusselbitar i Python - Linjär algebra

Denna post innehåller exempel på Pythonkod som har med hantering av vektorer och matriser att göra. Tillhörande övningsuppgifter finns i sektionen Uppgifter.

Vektorer och matriser

Det finns två bibliotek som är viktiga att importera i samband med nästan allt arbete med linjär algebra i Python: `numpy` och `matplotlib`. Dessa importeras med

```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

Dessa båda bibliotek innehåller funktioner som hanterar t ex matrisberäkningar (`numpy`) respektive grafisk representation av t ex funktioner och vektorer (`matplotlib`). Dessa rader behöver stå med i början av varje program; när de är inlästa så finns funktionerna tillgängliga med prefixen `np` respektive `plt`.

I detta inlägg kommer jag att fokusera på några av funktionerna i `numpy`, ett annat inlägg kommer att ge exempel på hur man kan visualisera funktioner och matriser.

Skapa matriser och vektorer

En matris kan skapas med

```
1 A = np.array([[1, 2], [3, 4]]) # Skapar en 2x2-matris
```

En vektor ser vi ofta som en matris med en kolonn (kolonnvektorer), en sådan kan skapas enligt

```
1 v = np.array([[-1],[2]])
```

Naturligtvis går det också att skapa radvektorer, då är elementen enbart tal och inte listor (där respektive lista innehåller ett enda tal).

Multiplicera matriser med varandra

Vektorn `v` ovan går att multiplicera med matrisen `A` enligt

```
Av = A@v
# Av är nu en ny vektor, kan skrivas ut med print(Av)
# där resultatet är:
# array([[3],
#        [5]])
```

Med samma metod går det även att multiplicera två (kompatibla) matriser med varandra.

Att komma åt enskilda element i en matris

I många fall behöver man komma åt de enskilda elementen i en vektor eller matris, och det kan göras enligt följande:

```
a12 = A[0, 1] # Första raden, andra kolonnen i A
av2 = Av[1, 0] # Andra raden, första kolonnen i v
```

Efter dessa tilldelningar kommer variabeln `a12` att innehålla talet 2 och `av2` kommer att innehålla talet 5. Observera att Pythons räkning av elementens ordningstal börjar på 0, det är vanligt i programmeringsspråk.

Att komma åt enskilda rader och kolonner i en matris

I vissa andra fall kan man behöva komma åt en hel rad eller kolonn i en matris:

```
row_1 = A[0] # Första raden i A
col_1 = A[:,0] # Första kolonnen i A
```

Radoperationer

Det är enkelt att byta plats på två rader i en matris:

```
A_swap = A[[0, 1]] = A[[1, 0]]
```

Variabeln (matrisen) `A_swap` är nu matrisen `A` där rad 1 och 2 har bytt plats.

En annan radoperation är att multiplicera en rad med ett tal, det görs på följande sätt:

```
A[0] = 2 * A[0] # Multiplicerar första raden med 2
```

Ytterligare en radoperation är att multiplicera en rad med ett tal och addera den med en annan rad, det görs helt enkelt med:

```
2 * A[0] + A[1]
```

vilket multiplicerar första raden med 2 och sedan adderar den till den andra raden.

Radreduktion med `sympy`

Det finns faktiskt också möjlighet att göra en fullständig Gauss-Jordan elimination på matriser, men för detta måste ytterligare ett bibliotek importeras:

```
import sympy as sp # Bibliotek för symbolisk hantering
B = np.array([[1,2,3],[3,4,5]])
sol = sp.Matrix(B).rref()
```

`rref` betyder Reduced Row Echelon Form (reducerad trappstegsform), och är den matris som erhålls efter en Gauss-Jordan elimination. Variabeln `sol` kommer nu att innehålla

```
[1, 0, -1],
[0, 1,  2]]), (0, 1))
```

vilket ska förstås som `x = -1` och `y = 2`, dvs lösningen till ekvationssystemet

```
 x + 2y = 3
3x + 4y = 5
```

som definierades av matrisen `B` (läs den som en utökad koefficientmatris).

Invertering av matriser

Det går även att invertera matriser på ett enkelt sätt:

```
A = [[1, 3], [3, 4]]
inv_A = np.linalg.inv(A)
```

Matrisen `inv_A` kommer nu att vara inversen av `A` som den definierades ovan. Kom ihåg att det enbart går att invertera kvadratiska matriser som är icke-singulära.

Lösning av ekvationssystem

Även om det går att lösa ekvationssystem genom att både göra rref-form av matriser och genom att invertera matriser, så finns det en egen funktion för att lösa dem också. Vi prövar att lösa systemet

```
 x + 2y = 3
3x + 4y = 5
```

genom

```
A = np.array([[1, 2], [3, 4]])
v = np.array([[3], [5]])
sol = np.linalg.solve(A, v)
```

Efter programkörningen kommer `sol` att vara en vektor som innehåller komponenterna -1 och 2, alltså lösningen till ekvationssystemet.

Determinanter

Det också att bestämma determinanten till en matris med `numpy`. Det görs genom

```
determinant = np.linalg.det(A) # A samma array som förra exemplet
```

Då kommer variabeln `determinant` att innehålla talet -2, som ju är är determinanten till A. (Vad kan sägas om en matris med determinanten 0?)

Skalärprodukt

Skalärprodukten av två vektorer kan bestämmas med funktionen `dot()`. Exempel:

```
u = np.array([1, 2, 3])
v = np.array([2, 3, 4])
dot_prod = np.dot(u, v)
```

Uppgifter

I de följande uppgifterna, låt

$$A = \begin{bmatrix} 2 & 4 & -4 \\ -1 & 3 & 4 \\ 5 & -5 & 6 \end{bmatrix}, \; B = \begin{bmatrix} 2 & 4 & -4 \\ -1 & 3 & 4 \\ 5 & 5 & -12 \end{bmatrix}, \; \mathbf{u} = \begin{bmatrix} 3 & 2 & \frac{48}{17} \end{bmatrix} \text{ och } \mathbf{v} = \begin{bmatrix} 8 \\ 12 \\ -17 \end{bmatrix}$$

Lös uppgifterna nedan i ett Jupyter- eller colab-dokument. **OBS!** Notera att $\mathbf{u}$ är en radvektor och $\mathbf{v}$ är en kolonnvektor.

1. Deklarara ovanstående matriser och vektorer i Python med hjälp av `numpy`. (Under rubriken Skapa matriser och vektorer ovan så finns exempel på hur en kolonnvektor skapas. Kan du komma på hur du skapar en radvektor?)

2. Låt Python addera matriserna A och B.

3. Låt Python addera vektorerna $\mathbf{u}$ och $\mathbf{v}$. Resultat, och varför?

4. Pröva att utföra multiplikationerna AB, BA, $A\mathbf{u}$, $A\mathbf{v}$, $\mathbf{uv}$ och $\mathbf{vu}$. Vilka multiplikationer fungerar (och varför) och vad blir resultatet?

5. Lös ekvationen $A\mathbf{x} = \mathbf{v}$ på fyra olika sätt:
   - med `solve`-funktionen
   - med matrisinvertering
   - med radreduktion med `sympy`
   - genom att låta Python göra de radoperationer (som du anger) som behövs för att lösa ekvationen

6. Försök att lösa ekvationen $B\mathbf{x} = \mathbf{v}$ med samma metoder som i uppgift 5. Hur gick det (med respektive metod)?

7. Beräkna determinanten på A, B och på AB. Resultat?

8. Byt plats på rad ett och rad två i A och beräkna determinanten på denna matris. Jämför resultatet med värdet på determinanten för A.

9. Elementen i en matris som är resultatet av en matrismultiplikation kan ses som skalärprodukter. Elementet $(ab)_{11}$ (det översta elementet till vänster i matrisprodukten AB) är skalärprodukten av den första raden i A och första kolonnen i B. Beräkna på så sätt alla element i AB (se ovan hur rader och

kolonner extraheras ur matriser). Kontrollera svaret med `A@B`.

10. Det går att skapa matriser av en given dimension med slumpade tal på ett enkelt sätt i Python. T ex skapar `np.random.randint(10, size=9).reshape(3,3)` en 3×3-matris med heltalselement mellan 0 och 9. Skapa en sådan matris och förvissa dig om att denna multiplicerat med sin invers ger identitetsmatrisen. Pröva även att utföra BB^{-1}. Varför blir resultatet som det blir?

11. Du känner till begeppet "elementär matris". Om en godtycklig matris A multipliceras med en elementär matris från höger så innebär det att en radoperation har ägt rum på A. Vilken radoperation som ägt rum beror på den elementära matrisen (se kap. 7 i kurskompendiet). I den här uppgiften ska du generera elementära matriser i ett Python-program. Se kodskelett på denna länk.