Programmeringsteknik: NumPy

Maj Stenmark

maj.stenmark@cs.lth.se

FÖRELÄSNING 2 2024-02-26

Föreläsning 2

* Mera om matrishantering
  * Indexering
  * Linjär algebra
* Polynom
* Beräkning av integraler
* Nollställen, max- och minvärden med `scipy`

* Genomgång temauppgiften i lab 2:
  * `meshgrid`
  * `vectorize`

Repetition — matriser

* Skapa matriser:

```
A = np.array([[1, 1, -1], [2, 1, 1], [4, 3, -1]])
```

* Läsa enskilda element:

```
A[0, 2] #element på rad 0 och kolumn 2, dvs -1
```

* Tilldelning:

```
A[0, 2] = 5
```

* De flesta standardfunktioner är elementvisa, t ex:

  `np.sin(A)` beräknar `sin` av varje enskilt element

* Det finns dedikerade funktioner för matrisberäkningar, t ex:

  `np.matmul(A, B)` för att multiplicera matriser

Indexering i matriser

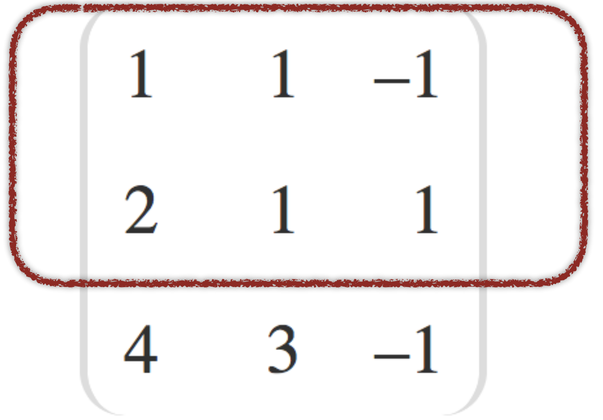
* Vi kan ta ut flera element genom att ange start och slutindex:

```
A[start_row:end_row, start_col:end_col]
```

* Slutindex exkluderas
* Kallas *slicing*

Exempel 1

```
A = np.array([[1, 1, -1], [2, 1, 1], [4, 3, -1]])
print(A[0:2, 0:3])
# skriver ut:
[[ 1  1 -1]
 [ 2  1  1]]
```



Indexering i matriser

* Vi kan också ange steglängd

```
A[start_row:end_row:step, start_col:end_col:step]
```

* Notera att slutindex exkluderas

Exempel 1

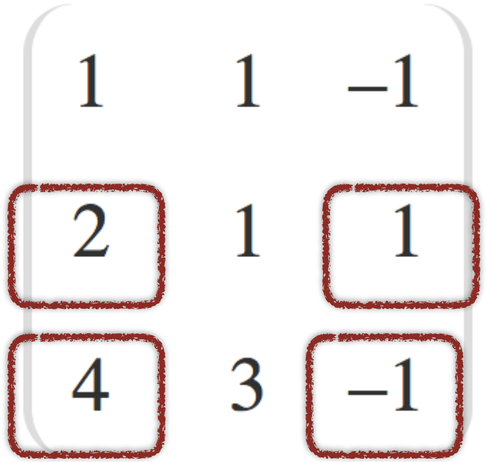
```
A = np.array([[1, 1, -1], [2, 1, 1], [4, 3, -1]])
print(A[1:3:1, 0:3:2])
# skriver ut:
[[ 2  1]
 [ 4 -1]]
```

kolumn 0 och 2

rad 1, 2 (0 exkluderas)

rad 1: element 0 och 2

sista raden: element 0 och 2

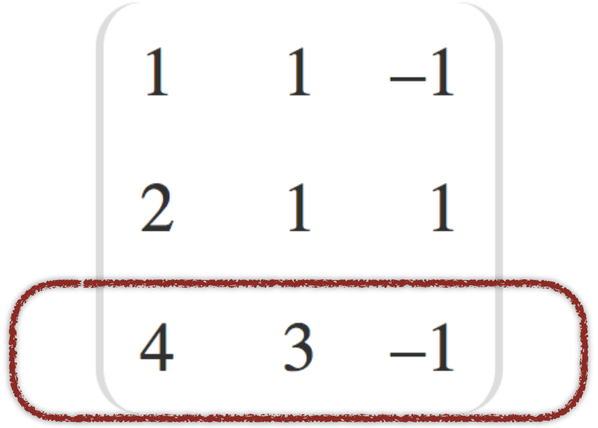


Indexering i matriser

Negativa index räknas från slutet!

Vad ger följande?

```
A = np.array([[1, 1, −1], [2, 1, 1], [4, 3, −1]])
print(A[−1])
```

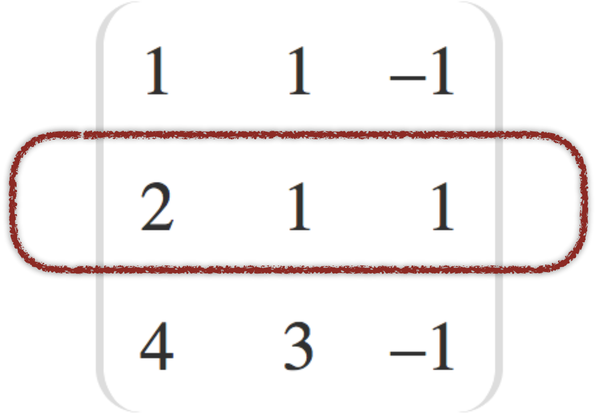


Indexering i matriser

Negativa index räknas från slutet!

Vad ger följande?

```
A = np.array([[1, 1, −1], [2, 1, 1], [4, 3, −1]])
print(A[−2])
```



Indexering i matriser

Hur gör vi om vi vill skriva ut alla rader i omvänd ordning?

Fungerar följande?

```
A = np.array([[1, 1, -1], [2, 1, 1], [4, 3, -1]])
print(A[2:-1:-1])
```

Indexering i matriser

Hur gör vi om vi vill skriva ut alla rader i omvänd ordning?

Fungerar följande?

```
A = np.array([[1, 1, -1], [2, 1, 1], [4, 3, -1]])
print(A[2:-1:-1])
# skriver ut: []
```

Indexering i matriser

Hur gör vi om vi vill skriva ut alla rader i omvänd ordning?

```
A = np.array([[1, 1, -1], [2, 1, 1], [4, 3, -1]])
print(A[2:None:-1])
# skriver ut:
[[ 4  3 -1]
 [ 2  1  1]
 [ 1  1 -1]]
```

None refererar till indexvärdet före första raden

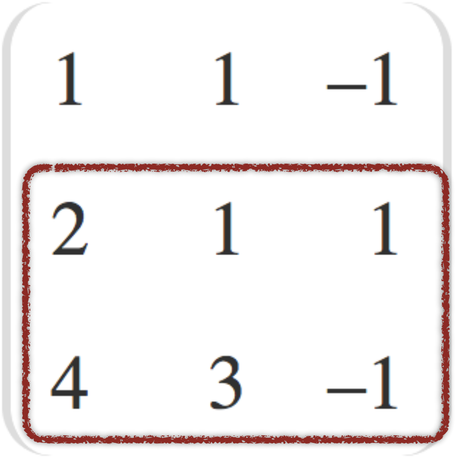
Raderna nerifrån upp

$$\begin{pmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \\ 4 & 3 & -1 \end{pmatrix}$$

Indexering i matriser

* None kan också referera till elementet efter sista

```
print(A[1:None])
#skriver ut
[[ 2  1  1]
 [ 4  3 -1]]
```



* Istället för att skriva ut None kan vi bara lämna platsen tom:

```
print(A[1:])
```

* För att skriva matrisen baklänges:

```
print(A[2::-1])
```

* Vi kan också lämna startindex tomt:

```
print(A[::-1]) eller print(A[::-1, :])
```

Övning

1. Skapa en 5x7-matris med slumpade heltal 0 .. 10 med np.random.randint och skriv ut följande:
   A. Första raden
   B. Andra kolonnen
   C. De fyra hörnen
   D. Det inre av matrisen

Linjär algebra

Det finns många funktioner för linjär algebra i **np.linalg.**

Till exempel för att beräkna determinanten:
`np.linalg.det(A)`

Transponera matrisen: `A.T`

Invertera matriser (om det går): `np.linalg.inv(A)`

Lösa ekvationssystem $Ax = b$: `x = np.linalg.solve(A, b)`

Polynom

* NumPy har polynomrelaterade funktioner.
* Polynom representeras som en vektor av koefficienter:
  polynomet $a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1} + a_nx^n$
  representeras av vektorn [a_0, a_1, a_2, … a_n]

Exempel

$30 - 19x + x^3$ skrivs [30, -19, 0, 1]

Polynomfunktioner

* I modulen `polynomial` finns många polynomfunktioner

* Importera modulen så här:

```
import numpy.polynomial.polynomial as
polynomial
```

Obs, detta är en ny modul, det finns gamla metoder som representerar polynom tvärtom

Polynomfunktioner

* Hitta rötterna (nollställena) till polynomet *p:*

```
roots = polynomial.polyroots(p)
```

* Skapa ett polynom *p* från en lista med rötter (nollställen):

```
p = polynomial.polyfromroots([ x0, x1 …])
```

* Räkna ut värdet av polynomet *p* för x-värdena *x*

```
y = polynomial.polyval(x, p)
```

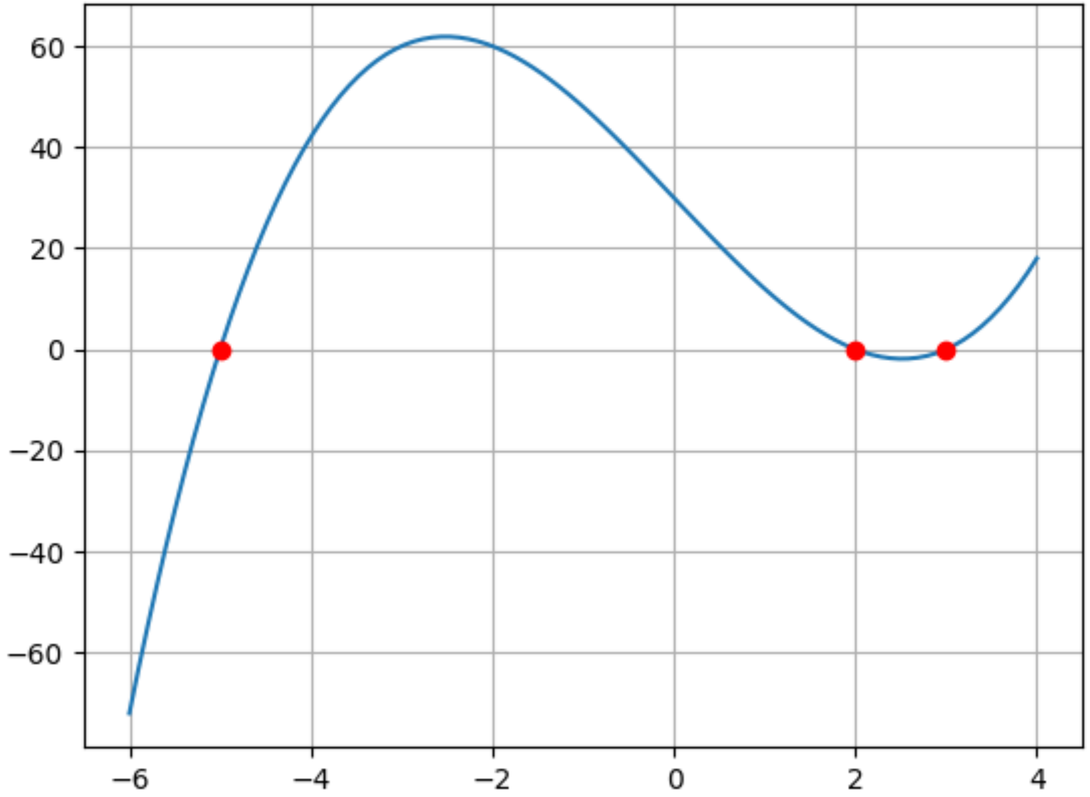
* Anpassa ett *n:te* gradspolynom till punkterna *(x, y)*

```
line = polynomial.polyfit(x, y, n)
```

Exempel `polyroots`

* Hitta rötterna till $30 - 19x + x^3$

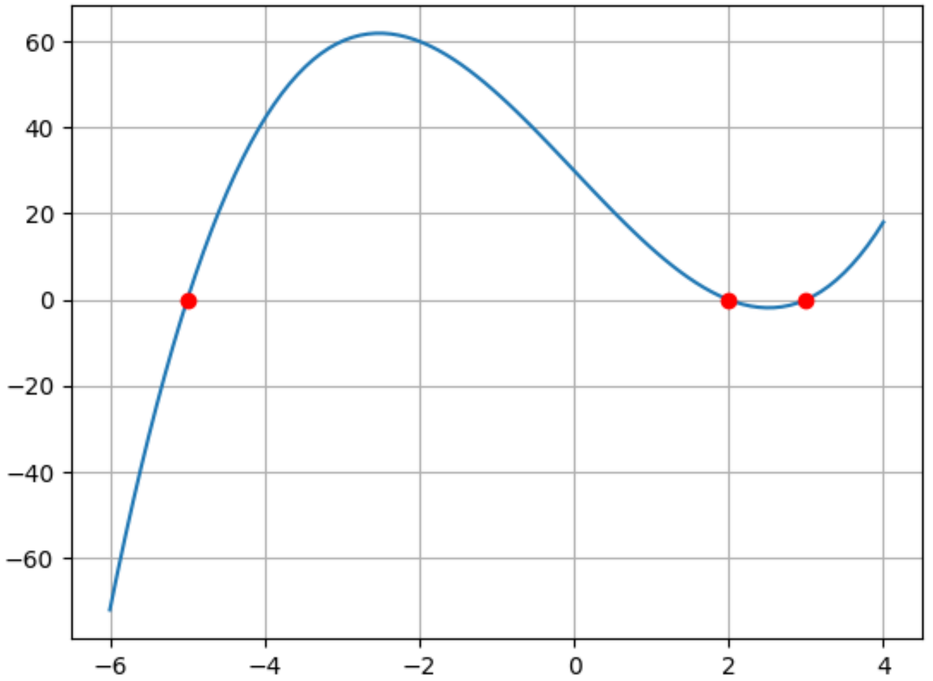
  Hur många rötter finns det?



Exempel `polyroots`

Hitta rötterna till $30 - 19x + x^3$

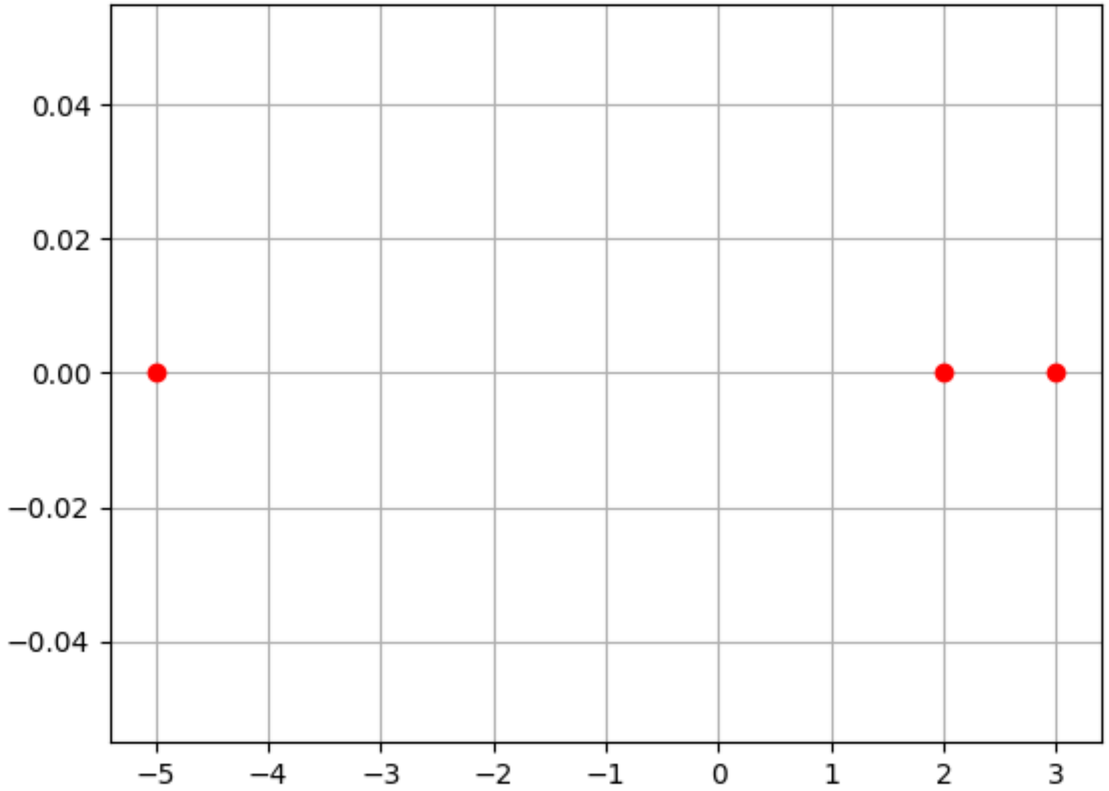
```
p = [30, -19, 0, 1]
roots  = polynomial.polyroots(p)
print(roots)
```

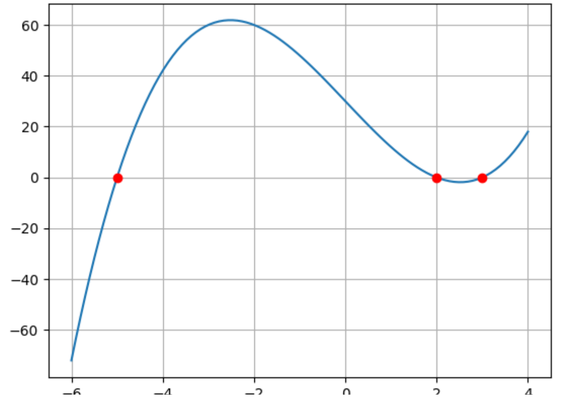


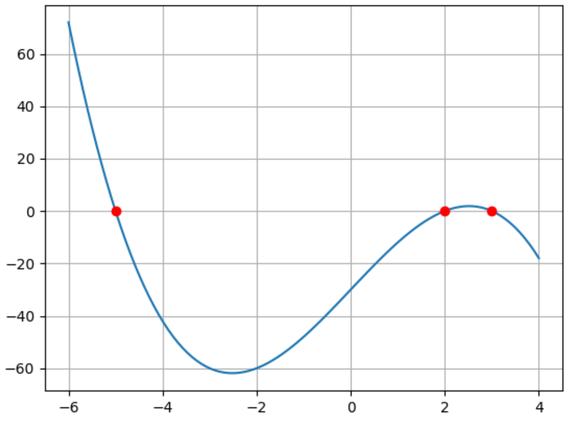
Exempel `polyfromroots`

* Hitta polynomet från rötterna [-5, 2, 3]

  Hur många polynom finns det?



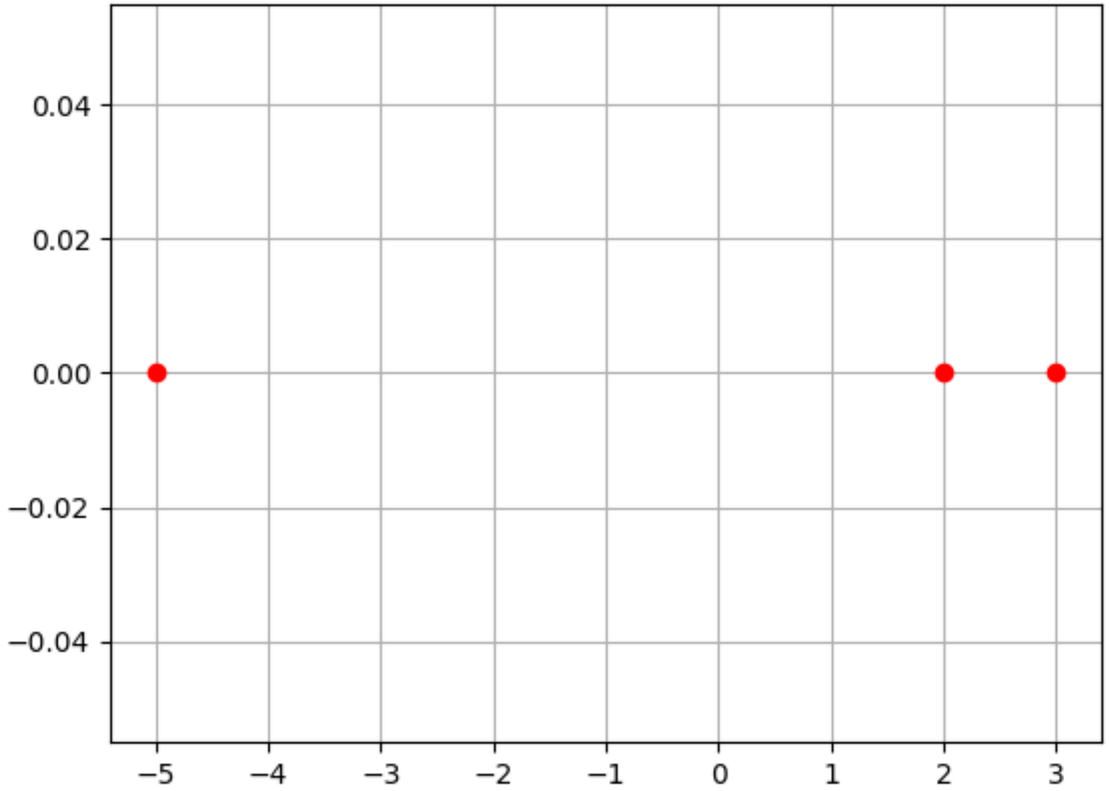


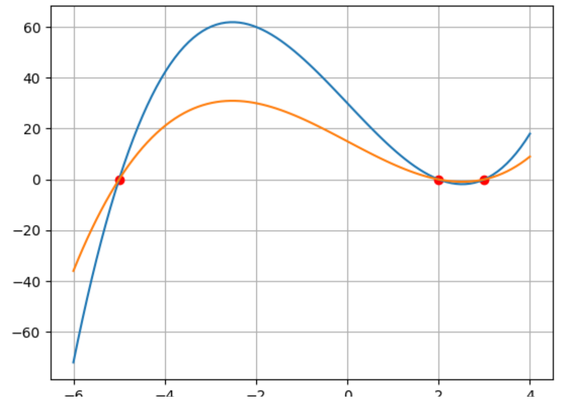


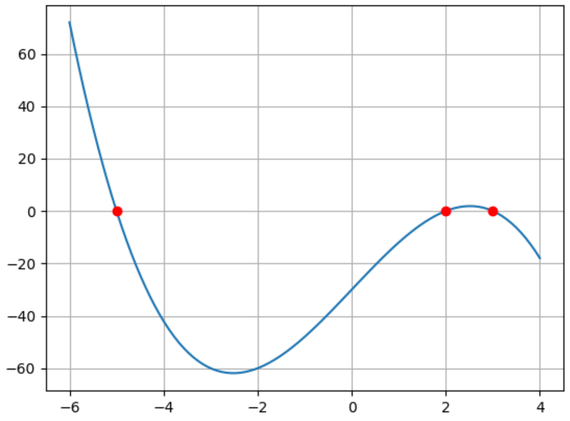
Exempel `polyfromroots`

* Hitta polynomet från rötterna [-5, 2, 3]

  Hur många polynom finns det? Oändligt många, vi kan multiplicera med en konstant.







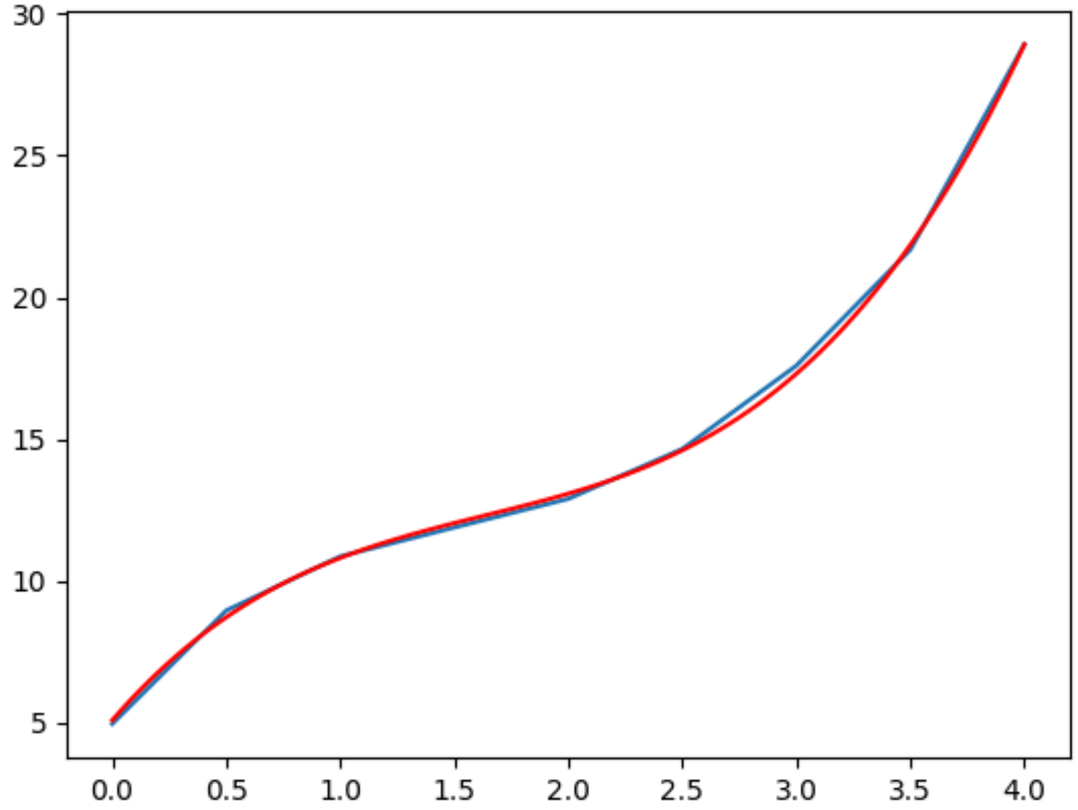
Exempel `polyval`

* För att rita upp polynomet kan vi använda `polyval` och `plot`

```
p = [30, -19, 0, 1]
x = np.linspace(-6, 4, num = 100)
y = polynomial.polyval(x, p)
plt.plot(x, y)
plt.grid()
plt.show()
```

Exempel `polyfit`

* Anpassa ett tredjegradspolynom från mätpunkter



x	y
0,0	4.95
0.5	8.95
1.0	10.86
1.5	11.88
2.0	12.89
2.5	14.65
3.0	17.59
3.5	21.68
4.0	28.93

Läsa och skriva till fil

* Mätdata kommer ofta i en fil.
* Den kan läsas in i ren python eller direkt in i en numpy array.

```
with open('test.txt', 'r') as f:
    lines = f.readlines()
    #gör nåt med lines
```

* Med numpy `loadtxt`:

```
data = np.loadtxt('filnamn.txt')
data = np.loadtxt('filnamn.txt', delimiter =
',', dtype = int)
```

Skriva till fil

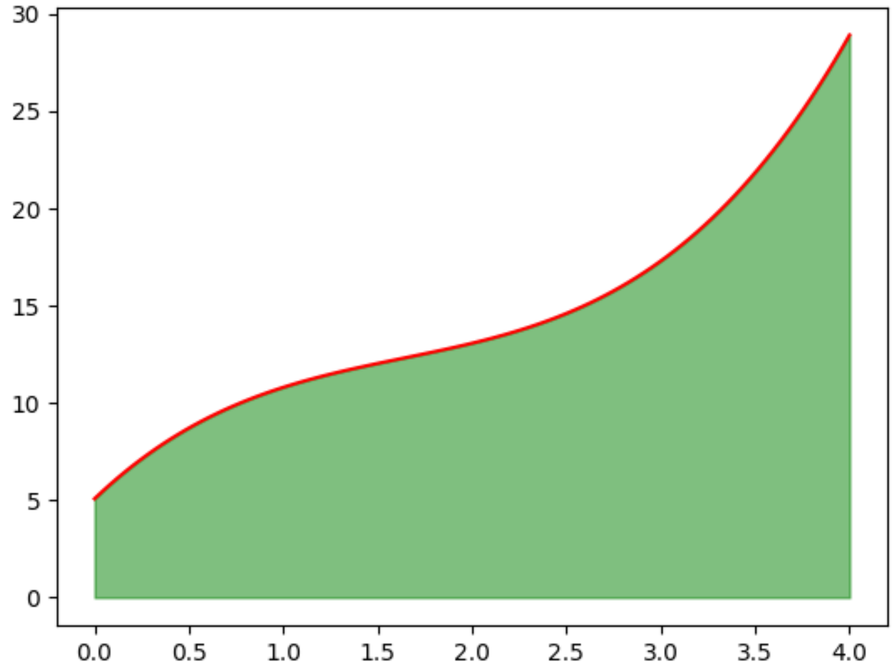
* `np.savetxt('saved.txt', data)` skriver ut numpy/arrayen data till filen. Värdena separeras med ny rad och skriver ut med grundpotensform.

*
```
with open('saved2.txt', 'w') as f:
    f.write('hello')
```
skriver ut en sträng. Argumentet ‘r’ = read och ‘w’ = write

Integralberäkningar

* Integralen av en funktion är “arean under kurvan”.
  T ex, en funktion som beskriver hastigheten för en bil, så är integralen den totala sträckan bilen rört sig.

* Men det finns många färdiga funktioner för det!
  T ex `np.trapezoid(y, x) #beräknar integralen med trapetsmetoden`



scipy

* Biblioteket `scipy` har många funktioner
  för att integrera, hitta nollställen, lösa differentialekvationer
  med mera.

* Scipy-moduler importeras så här:

```
import scipy.integrate as integrate

sol = integrate.quad(f, 1.7, 2.7)
#integrerar funktionen f från x = 1.7 till x
= 2.7
```

scipy

* I `scipy.optimize` finns funktioner för olika optimeringsproblem.

* Importeras med t ex

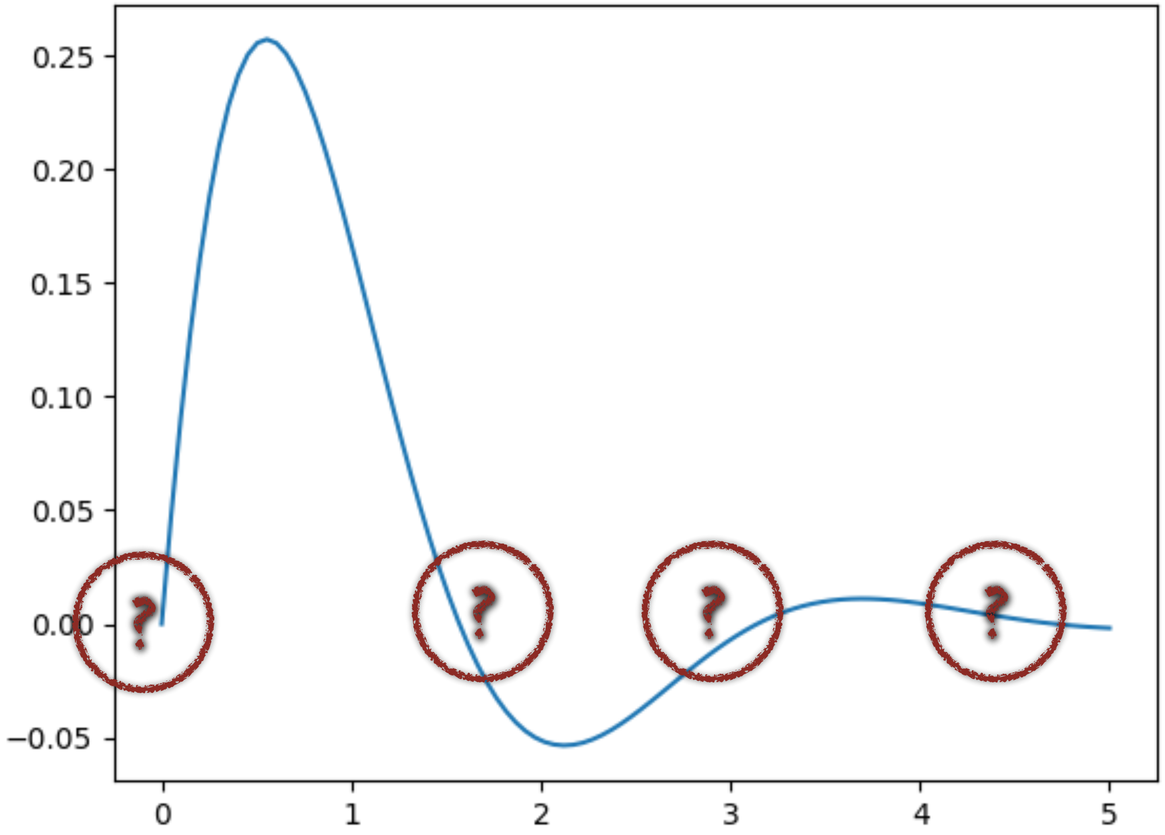
  ```
  import scipy.optimize as optimize
  ```

* `zeros = optimize.fsolve(f, x0)` hittar nollställe till funktionen nära x_0.
* `minval = optimize.minimize(f, x0)` hittar minimivärdet nära x_0
* `minval = optimize.fminbound(f, 2.6, 3)` hittar minimivärdet i intervallet [2.6, 3]

Exempel 1

* Vi tar reda på nollställen för funktionen (i x-intervallet 0..5):

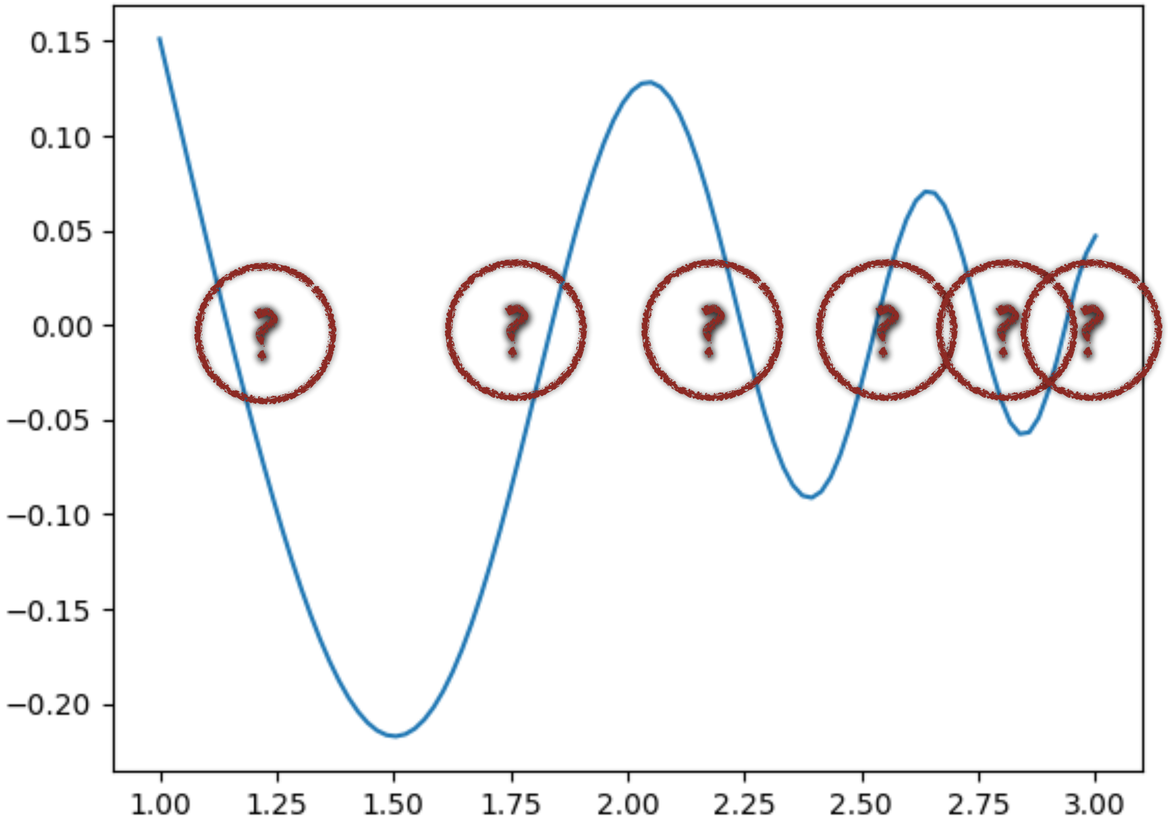
$$f(x) = e^{-x} \sin x \cos x$$



Exempel 2

* Vi tar reda på nollställen för funktionen (i x-intervallet 1..3):

$$f(x) = e^{-x} \sin e^{x}$$

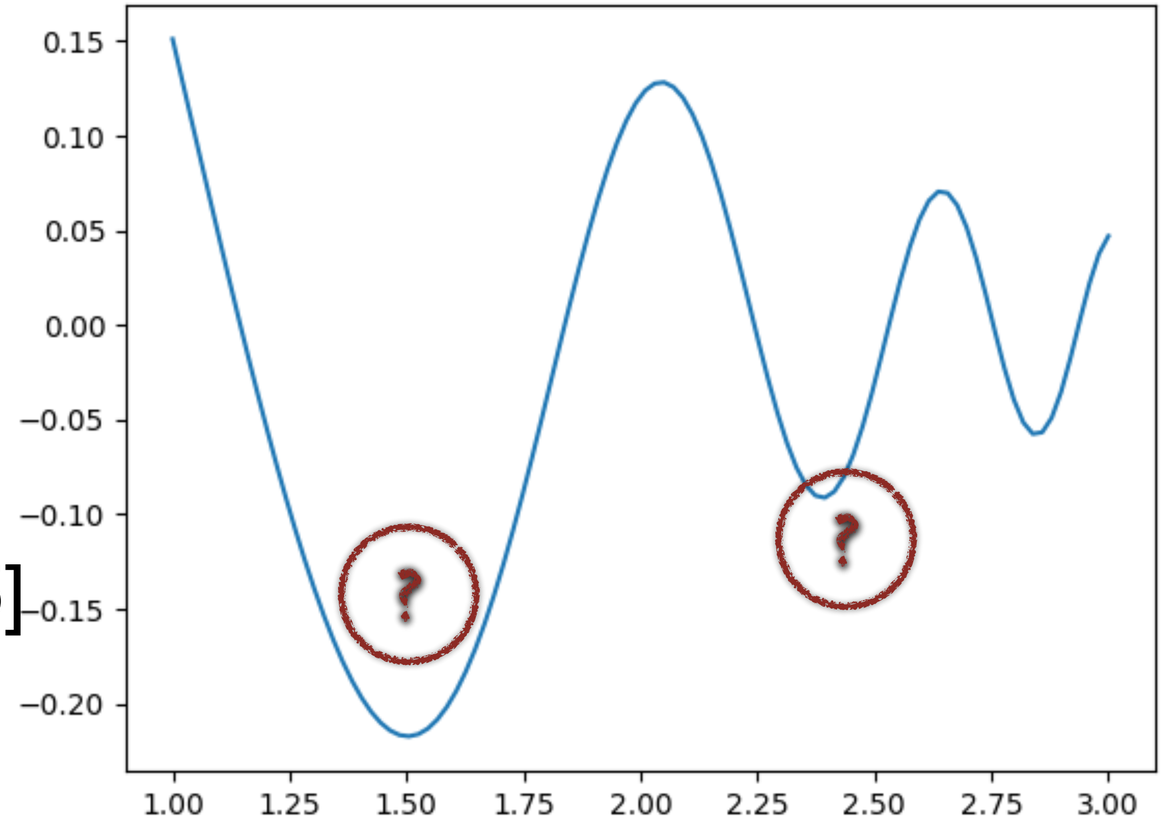


Exempel 3

* Vi tar reda på minvärdet för funktionen (i x-intervallet 1..3):

$$f(x) = e^{-x} \sin e^{x}$$

* Vi vill ha det lokala minvärdet i intervallet [2.1 …2.5]



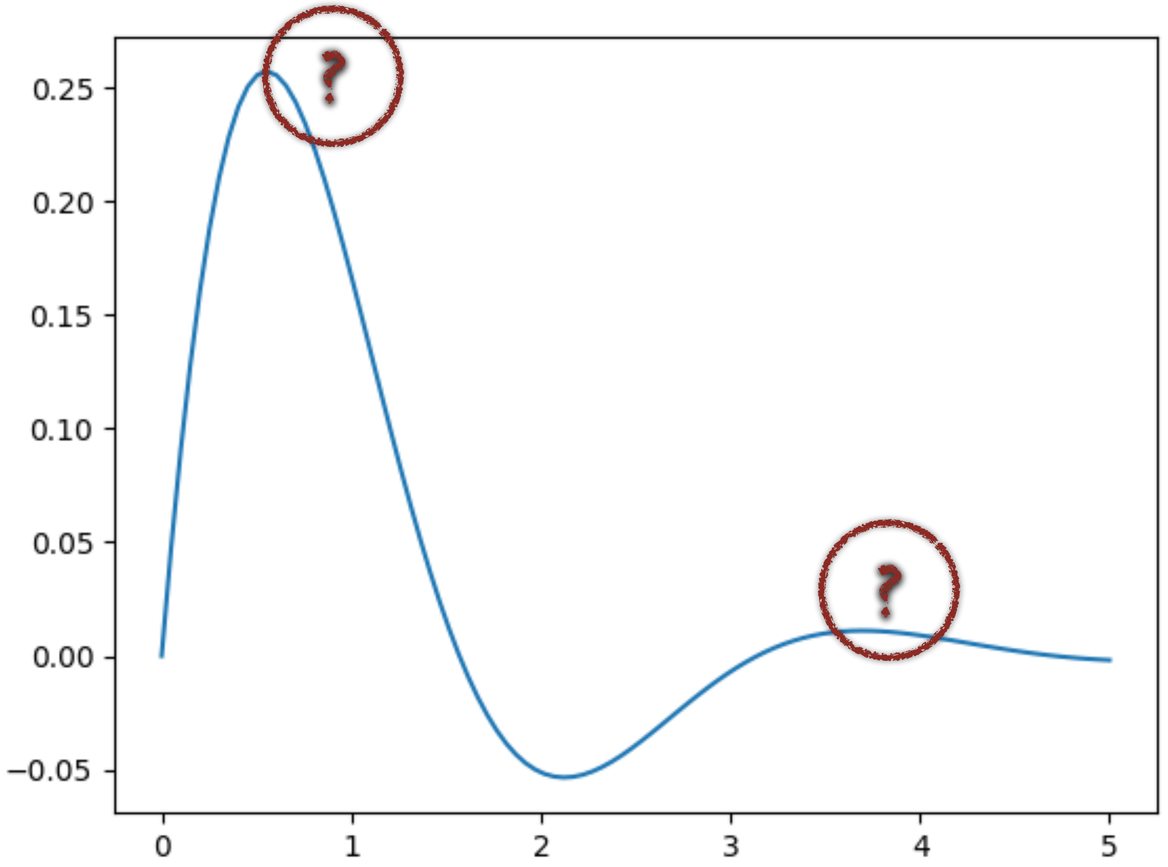
Exempel 4

* Vi tar reda på maxvärdet för funktionen (i x-intervallet 0..5):

$$f(x) = e^{-x} \sin x \cos x$$

* Vi vill ha det lokala maxvärdet i intervallet [3 …5]

Det finns ingen maximize-funktion!
Vad gör vi?

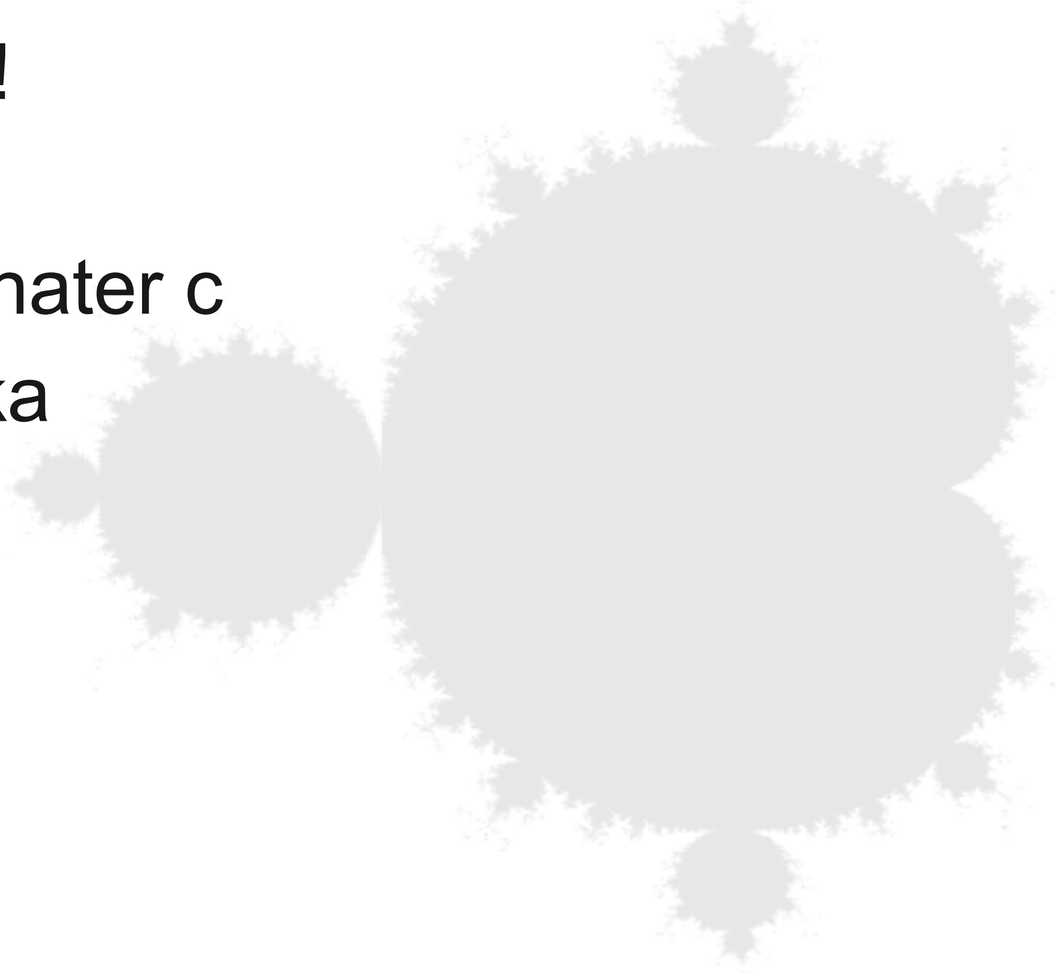


Laboration 2

* Uppfattas ofta som den mest tidskrävande laborationen!
* Börja i god tid!
* För olika komplexa koordinater c beter sig serien nedan olika

$$z_0 = c$$

$$z_{k+1} = z_k^2 + c$$



Laboration 2

1. Skapa matris med komplexa tal `complexmat`
2. Skapa en `converge`-metod som räknar ut om serien konvergerar för *ett* tal. Returnerar k.
3. För varje element i matrisen beräknar vi hur många steg den behöver för att *divergera*

Serien *divergerar* om

$|z_k| > 2$ för $0 \leq k \leq 100$

Om k = 100 konvergerar serien.

$$z_0 = c$$

$$z_{k+1} = z_k^2 + c$$

complexmat

* Hur kan vi skapa en matris för komplexa tal?

```
m = complexmat(5, -2+1j, 1 - 1j)
print(m)
[[-2.00+1.00j  -1.25+1.00j  -0.50+1.00j   0.25+1.00j   1.00+1.00j]
[-2.00+0.50j  -1.25+0.50j  -0.5+0.50j   0.25+0.50j   1.00+0.50j]
[-2.00+0.00j  -1.25+0.00j  -0.5+0.00j   0.25+0.00j   1.00+0.00j]
[-2.00-0.50j  -1.25-0.50j  -0.5-0.50j   0.25-0.50j   1.00-0.50j]
[-2.00-1.00j  -1.25-1.00j  -0.5-1.00j   0.25-1.00j   1.00-1.00j]]
```

* x-värdena går kolumnvis från -2 till 1
* y-värdena går radvis från 1j till -1j
* Matrisen skapas som ett *meshgrid* av två sekvenser.

complexmat

Hur kan vi skapa en matris för komplexa tal?

```
xvals = np.linspace(x_start, x_end, num = N)
yvals = np.linspace(y_start, y_end, num = N)
X, Y = np.meshgrid(xvals, yvals)
print(X)
print(Y)
```

* *Meshgrid* tar in två sekvenser och gör om båda till matriser, där den första har identiska rader och den andra har identiska kolumner.

converge

* Vi ska beräkna antalet iterationer som det behövs för att serien ska konvergera (om abs-värdet är <= 2 efter 100 iterationer).
* En 1000 x 1000- matris för att representera punkter, varje punkt har får olika många iterationer.

converge

* Vi börjar med att definiera en funktion som beräknar antalet iterationer för en punkt (c).

```
def converge(c):
        …
        return iterations
```

* Att loopa över alla punkter tar lååååång tid.
* Vi kan skapa en *vektoriserade funktion* för att kunna skicka in *hela matrisen* som ett argument:

```
vfunc = np.vectorize(converge)
v = vfunc(mat)
```

* v är antalet iterationer i respektive punkt.

Tack för idag!