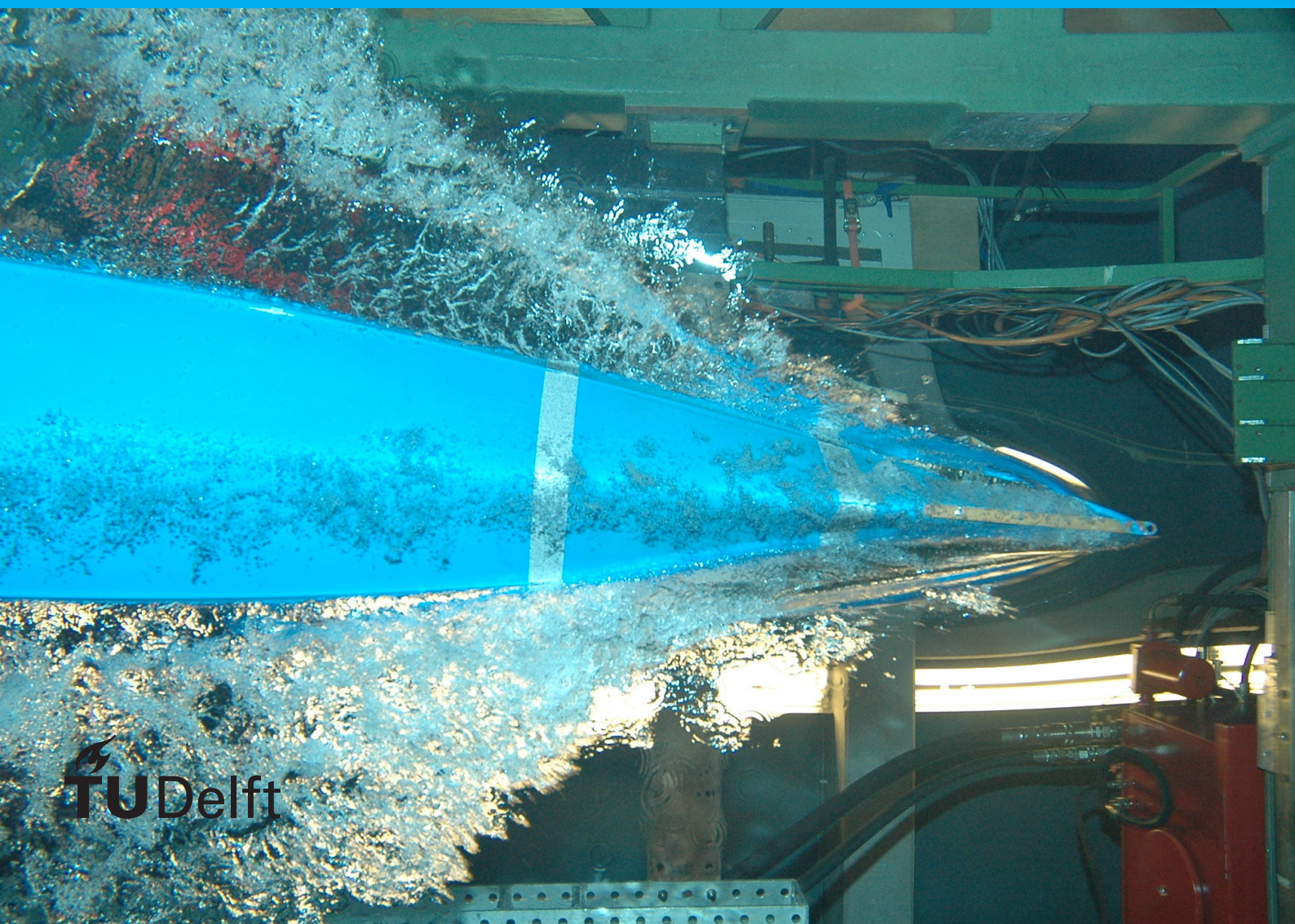


Python Programming

Bart H. M. Gerritsen
Giuseppe Radaelli

Oefentoets

15 December 2017



Aanwijzingen voor de Toets

Maak deze oefentoets zelf, thuis, in maximaal twee uur. Breng je programma's mee naar de gezamenlijke bespreking van de Oefentoets (zie Brightspace) en scoor jezelf zoals aangegeven zal worden tijdens de gezamenlijke bespreking. Dan beoordeel je zelf of je de materie voldoende beheerst voor de eigenlijke Toets.

Deze Oefentoets is bedoeld als voorbereiding, is niet verplicht, en heeft verder geen invloed op je cijfer PYTHON Q2.

*Bart H. M. Gerritsen
Giuseppe Radaelli*

Delft, December 2017



Opgave 1 makkelijk, 15 min

Opgave 1: Zoek het beste lager voor je karretje

1.1. Intro

Je hebt voor je karretje een partij van 109 lagers op de kop getikt en wil nu het lager met de laagste rolweerstandscoefficient zien te vinden. Je wilt ook weten wat het verschil is t.o.v. het lager met de hoogste C_r -waarde. Je hebt alle lagers getest en de gegevens opgeslagen in file `Bearings.csv`. Die lees je in en dan kun je beginnen met de analyse.

Open template-programma `Bearings.py` en kijk door het programma heen. Begin onderin bij de hoofdfunctie `runMainProgram()` en bekijk welke hoofdstappen er uitgevoerd worden.

1.2. Wat moet je doen?

Deze Opgave kent twee opdrachten:

OPDRACHT1 De eerste opdracht is om het lager te zoeken waarvoor de C_r -waarde het laagst is, en het lager waarvoor C_r het hoogst is. Druk van deze lagers het N_r af, dat je uit het bestand hebt ingelezen. Druk ook af: het gemiddelde van C_r en het aantal lagers.

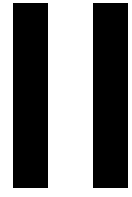
OPDRACHT2 Plot daarna een histogram, met 10 klassen ('bars', 'bins') voor de C_r -waarde. Plot ook de lijnen voor $\min(C_r)$ en $\text{avg}(C_r)$ in het histogram. Maak hiervoor het programma af door de `????` te vervangen door de gevraagde gegevens

1.3. Template-programma

Gebruik template-programma `Bearings.py` voor deze opdracht. Introduceer zelf GEEN NIEUWE VARIABLEN. Houd je aan de bestaande namen voor variabelen, functies e.d. Wijzig die niet.

De data in array 'data' in het template-programma is als volgt georganiseerd:

	nr	Cr-waarde
0	1	0.00099
1	2	0.00191
j
N-1	N	0.00328



Opgave 2

gematigd moeilijk, 20 min

Opgave 2: Controleer de oplossing uit de Reader

2.1. Intro

In de reader, Blok 2 is een toepassingsprobleem gegeven, waarin twee lijnen L_1 en L_2 :

$$\begin{aligned} L_1 \quad 4x + 2y &= -2 \\ L_2 \quad -3x + y &= 1 \end{aligned}$$

elkaar snijden in het punt $Q = -\frac{1}{5}(2, 1)$. We gaan in deze Opgave het resultaat van deze berekening controleren door gebruik te maken van de Regel van Cramer.

2.2. Toelichting

We schrijven: $\mathbf{AX} = \mathbf{b}$, ofwel:

$$\mathbf{A} = \begin{pmatrix} 4 & 2 \\ -3 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

Je herkent hierin direct \mathbf{A} en vector \mathbf{b} . Kolomvector $\mathbf{X} = (x, y)^t$ is de plaatsvector van het snijpunt Q van L_1 en L_2 . We kunnen dit snijpunt uitrekenen, als volgt:

$$\mathbf{X} = \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{|\mathbf{A}|} \begin{pmatrix} \begin{vmatrix} -2 & 2 \\ 1 & 1 \end{vmatrix} \\ \begin{vmatrix} 4 & -2 \\ -3 & 1 \end{vmatrix} \end{pmatrix}$$

De laatste vorm is de toepassing van de Regel van Cramer. Je ziet dat we x en y vinden door twee maal een determinant uit te rekenen, en die dan te vermenigvuldigen met $|\mathbf{A}|^{-1}$. Determinant $|\mathbf{A}|$ is helder, maar wat zijn de twee andere determinanten precies? Je vindt deze twee determinanten door achtereenvolgens de eerste (boven, x) en de tweede (onder, voor y) kolom vector in \mathbf{A} te vervangen door vector \mathbf{b} , die is gegeven. Van die matrices die je dan krijgt, reken je de determinant uit. In formule:

$$x_i = \frac{\det(\mathbf{A}_i)}{\det(\mathbf{A})}$$

In deze Regel van Cramer staat x_i dus resp. voor x en voor y , en \mathbf{A}_i voor matrix \mathbf{A} met kolomvector \mathbf{A}^i vervangen door vector $\mathbf{b} = (-2, 1)^t$.

Open template-programma [Cramer.py](#) en kijk door het programma heen. Begin onderin bij de hoofd-functie `main()` en bekijk welke hoofdstappen er uitgevoerd worden.

2.3. Wat moet je doen?

Deze Opgave kent drie opdrachten:

OPDRACHT1 De eerste opdracht is om in het template-programma de functie `solve()` af te maken, zodat je met de regel van Cramer kunt verifiëren dat de oplossing van $\mathbf{AX} = \mathbf{b}$ die we in de Reader hebben gevonden (nl. $X = -\frac{1}{5}[2, 1]^T$) correct is.

OPDRACHT 2 Bereken jouw vector \mathbf{B} met $\mathbf{AX} = \mathbf{B}$ die je dus met jouw oplossing uit OPDRACHT 1 voor vector \mathbf{X} vindt. Maak hiervoor het programma af door de `?????` te vervangen door de gevraagde gegevens

OPDRACHT 3 Bepaal: $\mathbf{D} = \mathbf{B} - \mathbf{b}$ (dus het verschil van jouw \mathbf{B} met \mathbf{b} uit de Reader) en maak onderstaande test in het template-programma compleet om te bepalen of de absolute afwijking van elk van de elementen in vector $\mathbf{D} = \mathbf{B} - \mathbf{b}$ kleiner zijn dan de gegeven DELTA. Vul in wat er moet worden afgedrukt voor een correcte melding van de testresultaten.

2.4. Template-programma

Gebruik template-programma [Cramer.py](#) voor deze opdracht. Introduceer zelf GEEN NIEUWE VARIABLEN. Houd je aan de bestaande namen voor variabelen, functies e.d. Wijzig die niet.



Opgave 3

gematigd moeilijk, 20 min

Opgave 3: Integreren met de Riemann Sum

3.1. Intro

De Riemann Som is de eenvoudigste vorm om functie numeriek te kunnen integreren. De zogenaamde *midpoint* Riemann Sum is verwant aan de *trapezium regel*, ook een gemakkelijk te begrijpen en te programmeren vorm voor numerieke integratie. We bekijken beiden nader in deze Opgave. Behalve de midpoint (centrale) Riemann som, is er ook de maximum Riemann Max Som en de minimum Riemann Min Som. Er geldt altijd:

$$RSum_{max} \geq RSum_{mid} \geq RSum_{min}$$

Open template-programma [Riemann.py](#) en kijk door het programma heen. Begin onderin bij de hoofd-functie `main()` en bekijk welke hoofdstappen er uitgevoerd worden.

3.2. Wat moet je doen?

Deze Opgave kent vier opdrachten:

OPDRACHT 1 De eerste opdracht is om in het template-programma de functie `mid()` af te maken, waarmee de midpoint waarde van $F(x)$ wordt berekend. Deze waarde is het gemiddelde van de ingevoerde waarden Fa en Fb .

OPDRACHT 2 Daarna bepaal je de waarde van de numerieke integraal van $F(x)$ met behulp van de trapezium regel, door toepassing van function `numpy.trapz()`. Maak het template-programma af op dit punt

OPDRACHT 3 Druk dan de midpoint Riemann Sum en de Trapezium regel integraal af op een enkele regel, zodat de waarde van beiden kan worden vergeleken (maak het template-programma af)

OPDRACHT 4 Bepaal het percentuele verschil (als percentage van Riemann Min Som `mnRSum`), tussen de Riemann Max Som `mxRSum` en de Riemann Min Som `mnRSum`, waartussen de midpoint waarde moet liggen (maak het template-programma af)

3.3. Template-programma

Gebruik template-programma [Riemann.py](#) voor deze opdracht. Introduceer zelf GEEN NIEUWE VARIABLEN. Houd je aan de bestaande namen voor variabelen, functies e.d. Wijzig die niet.

IV

Opgave 4
moeilijk, 25 min

Opgave 4: Signaal analyse

4.1. Intro

Een electronicaonderdeel heeft 2 ingangsignalen:

1. een sinusvorming ingangssignaal $f(\phi) = \sin(\phi)$
2. een afgeleide daarvan $g(\phi) = \cos(\phi + \omega)$, waarin de faseverdraaiing (hoekverdraaiing) omega kan worden ingesteld door onszelf

We bestuderen het samengestelde complexe signaal:

$$h(\phi) = g(\phi) + j \cdot f(\phi) = \cos(\phi + \omega) + j \cdot \sin(\phi)$$

We willen weten bij welke door ons gekozen omega er wel en wanneer er geen hoekverdraaiing (faseverdraaiing) optreedt in het samengestelde signaal $h(\phi)$. We bestuderen h op het interval: $\phi \in [-\frac{3}{2}\pi, \frac{3}{2}\pi]$ en bestuderen het effect van $\omega = \frac{k\pi}{2}$, $k = 1, 2, 3$

Open template-programma [Signals.py](#) en kijk door het programma heen. Begin onderin bij de hoofd-functie `main()` en bekijk welke hoofdstappen er uitgevoerd worden.

4.2. Wat moet je doen?

Deze Opgave kent twee opdrachten:

OPDRACHT1 De eerste opdracht is om in het template-programma het complexe signaal $h(\phi)$ in het plot-statement in function `plotSignals()` te definiëren (maak het template-programma af)

OPDRACHT2 bestudeer het polar diagram en geef aan voor welke waarde(n) van de door ons gekozen omega (zie functie `main()`), er wel en welke er geen faseverdraaiing optreedt. Druk dat uit in waarden van $k\pi/2$. Maak het template-programma af op dit punt en druk deze regel af

4.3. Template-programma

Gebruik template-programma [Signals.py](#) voor deze opdracht. Introduceer zelf GEEN NIEUWE VARIABLEN. Houd je aan de bestaande namen voor variabelen, functies e.d. Wijzig die niet.

Appendix A

©2012-2015 - Laurent Pointal Memento v2.0.6
License Creative Commons Attribution 4

Python 3 Cheat Sheet

Latest version on :
<https://perso.limsi.fr/pointal/python:memento>

integer, float, boolean, string, bytes

```

int 783 0 -192 0b010 0o642 0xF3
      zero      binary  octal   hexa
float 9.23 0.0 -1.7e-6
bool True False
str "One\nTwo"
      escaped new line
      'I\m'
      escaped '
bytes b"toto\xfe\775"
      hexadecimal octal
    
```

Base Types

ordered sequences, fast index access, repeatable values

```

list [1,5,9] ["x",11,8.9] ["mot"]
tuple (1,5,9) 11,"y",7.4 ("mot",)
    
```

Container Types

Non modifiable values (immutables) \Rightarrow expression with only commas \rightarrow tuple

str bytes (ordered sequences of chars / bytes)

key containers, no a priori order, fast key access, each key is unique

```

dictionary dict {"key": "value"} dict(a=3,b=4,k="v")
(key/value associations) {1:"one",3:"three",2:"two",3.14:"pi"}
collection set {"key1","key2"} {1,9,3,0} set()
      keys=hashable values (base types, immutables...) frozenset immutable set empty
    
```

for variables, functions, modules, classes... names

Identifiers

- a...zA...Z_ followed by a...zA...Z_0...9
- diacritics allowed but should be avoided
- language keywords forbidden
- lower/UPPER case discrimination
- ```

a toto x7 y_max BigOne
@y and fee

```

Variables assignment

$\Rightarrow$  assignment  $\Leftrightarrow$  binding of a name with a value

- evaluation of right side expression value
- assignment in order with left side names

```

x=1.2+8+sin(y)
a=b=c=0 assignment to same value
y,z,r=9.2,-7.6,0 multiple assignments
a,b=b,a values swap
a,*b=seq unpacking of sequence in item and list
x+=3 increment \Leftrightarrow x=x+3 and
x-=2 decrement \Leftrightarrow x=x-2 /=
x=None « undefined » constant value %
del x remove name x ...

```

int("15")  $\rightarrow$  15

int("3f",16)  $\rightarrow$  63 can specify integer number base in 2<sup>nd</sup> parameter

int(15.56)  $\rightarrow$  15 truncate decimal part

float("-11.24e8")  $\rightarrow$  -1124000000.0

round(15.56,1)  $\rightarrow$  15.6 rounding to 1 decimal (0 decimal  $\rightarrow$  integer number)

bool(x) False for null x, empty container x, None or False x; True for other x

str(x)  $\rightarrow$  "..." representation string of x for display (cf. formatting on the back)

chr(64)  $\rightarrow$  '@' ord('@')  $\rightarrow$  64 code  $\leftrightarrow$  char

repr(x)  $\rightarrow$  "..." literal representation string of x

bytes([72,9,64])  $\rightarrow$  b'H\td'

list("abc")  $\rightarrow$  ['a','b','c']

dict([(3,"three"),(1,"one")])  $\rightarrow$  {1:'one',3:'three'}

set(["one","two"])  $\rightarrow$  {'one','two'}

separator str and sequence of str  $\rightarrow$  assembled str

':'.join(['toto','12','pswd'])  $\rightarrow$  'toto:12:pswd'

str splitted on whitespaces  $\rightarrow$  list of str

"words with spaces".split()  $\rightarrow$  ['words','with','spaces']

str splitted on separator str  $\rightarrow$  list of str

"1,4,8,2".split(",")  $\rightarrow$  ['1','4','8','2']

sequence of one type  $\rightarrow$  list of another type (via list comprehension)

[int(x) for x in ('1','29','-3')]  $\rightarrow$  [1,29,-3]

type(expression)

Conversions

for lists, tuples, strings, bytes...

negative index -5 -4 -3 -2 -1

positive index 0 1 2 3 4

lst=[10, 20, 30, 40, 50]

positive slice 0 1 2 3 4 5

negative slice -5 -4 -3 -2 -1

Items count

len(lst)  $\rightarrow$  5

$\Rightarrow$  index from 0 (here from 0 to 4)

Individual access to items via lst[index]

lst[0]  $\rightarrow$  10  $\Rightarrow$  first one lst[1]  $\rightarrow$  20

lst[-1]  $\rightarrow$  50  $\Rightarrow$  last one lst[-2]  $\rightarrow$  40

On mutable sequences (list), remove with del lst[3] and modify with assignment lst[4]=25

Access to sub-sequences via lst[start slice: end slice: step]

lst[: -1]  $\rightarrow$  [10, 20, 30, 40] lst[: -1]  $\rightarrow$  [50, 40, 30, 20, 10] lst[1:3]  $\rightarrow$  [20, 30] lst[:3]  $\rightarrow$  [10, 20, 30]

lst[1: -1]  $\rightarrow$  [20, 30, 40] lst[: -2]  $\rightarrow$  [50, 30, 10] lst[-3: -1]  $\rightarrow$  [30, 40] lst[3:]  $\rightarrow$  [40, 50]

lst[: :2]  $\rightarrow$  [10, 30, 50] lst[:]  $\rightarrow$  [10, 20, 30, 40, 50] shallow copy of sequence

Missing slice indication  $\rightarrow$  from start / up to end.

On mutable sequences (list), remove with del lst[3:5] and modify with assignment lst[1:4]=[15,25]

Sequence Containers Indexing

Comparisons: < > <= >= == != (boolean results)

a and b logical and both simultaneously

a or b logical or one or other or both

$\Rightarrow$  pitfall: and and or return value of a or of b (under shortcut evaluation).

$\Rightarrow$  ensure that a and b are booleans.

not a logical not

True False } True and False constants

Boolean Logic

parent statement:

statement block 1...

parent statement:

statement block2...

next statement after block 1

Statements Blocks

floating numbers... approximated values

Operators: + - \* / // % \*\*

Priority (...)

$\times \div \uparrow \downarrow a^b$

integer  $\div \div$  remainder

@  $\rightarrow$  matrix  $\times$  python3.5+ numpy

(1+5.3)\*2  $\rightarrow$  12.6

abs(-3.2)  $\rightarrow$  3.2

round(3.57,1)  $\rightarrow$  3.6

pow(4,3)  $\rightarrow$  64.0

$\Rightarrow$  usual order of operations

angles in radians

from math import sin, pi...

sin(pi/4)  $\rightarrow$  0.707...

cos(2\*pi/3)  $\rightarrow$  -0.4999...

sqrt(81)  $\rightarrow$  9.0

log(e\*\*2)  $\rightarrow$  2.0

ceil(12.5)  $\rightarrow$  13

floor(12.5)  $\rightarrow$  12

modules math, statistics, random, decimal, fractions, numpy, etc. (cf. doc)

Maths

module truc  $\Leftrightarrow$  file truc.py

from monmod import nom1, nom2 as fct

$\rightarrow$  direct access to names, renaming with as

import monmod  $\rightarrow$  access via monmod.nom1...

$\Rightarrow$  modules and packages searched in python path (cf sys.path)

Modules/Names Imports

statement block executed only if a condition is true

if logical condition:

statements block

Can go with several elif, elif... and only one final else. Only the block of first true condition is executed.

if with a var x:

if bool(x)==True:  $\Leftrightarrow$  if x:

if bool(x)==False:  $\Leftrightarrow$  if not x:

Conditional Statement

Signaling an error:

raise ExceClass(...)

Errors processing:

try:

normal processing block

except Exception as e:

error processing block

Exceptions on Errors

Figuur 1: PYTHON3 cheat sheet ©2012–2015 see header,

source: [https://perso.limsi.fr/pointal/\\_media/python:cours:mementopython3-english.pdf](https://perso.limsi.fr/pointal/_media/python:cours:mementopython3-english.pdf)



## Python For Data Science Cheat Sheet

### SciPy - Linear Algebra

Learn More Python for Data Science [Interactively](https://www.datacamp.com) at [www.datacamp.com](https://www.datacamp.com)

**SciPy**  
The SciPy library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

**Interacting With NumPy** Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3)], (4j,5j,6j))
>>> c = np.array([[(1,5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

**Index Tricks**

```
>>> np.mgrid[0:5,0:5]
>>> np.ogrid[0:2,0:2]
>>> np.F_[3,0]*5,-1:1:10:]
>>> np.c_[b,c]
```

**Shape Manipulation**

```
>>> np.transpose(b)
>>> b.flatten()
>>> np.hstack((b,c))
>>> np.vstack((a,b))
>>> np.split(c,2)
>>> np.vsplit(d,2)
```

**Polynomials**

```
>>> from numpy import polyval
>>> p = polyval([3,4,5])
Create a polynomial object
```

**Vectorizing Functions**

```
>>> def myfunc(a):
... if a < 0:
... return a*2
... else:
... return a/2
>>> np.vectorize(myfunc)
```

**Type Handling**

```
>>> np.real(b)
>>> np.imag(b)
>>> np.real_if_close(c,tol=1000)
>>> np.cast['f'](np.pi)
```

**Other Useful Functions**

```
>>> np.angle(b,deg=True)
>>> g = np.linspace(0,np.pi,num=5)
>>> g[3:] += np.pi
>>> np.unwrap(g)
>>> np.logspace(0,10,3)
>>> np.select([c<4],[c*2])
>>> misc.factorial(a)
>>> misc.comb(10,3,exact=True)
>>> misc.central_diff_weights(3)
>>> misc.derivative(myfunc,1,0)
```

## Linear Algebra

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

**Creating Matrices**

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4],[5,6]])
```

**Basic Matrix Routines**

**Inverse**

```
>>> A.I
>>> linalg.inv(A)
```

**Transposition**

```
>>> A.T
>>> A.H
```

**Trace**

```
>>> np.trace(A)
```

**Norm**

```
>>> linalg.norm(A)
>>> linalg.norm(A,1)
>>> linalg.norm(A,np.inf)
```

**Rank**

```
>>> np.linalg.matrix_rank(C)
```

**Determinant**

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

**Solving linear problems**

```
>>> linalg.solve(A,b)
>>> E = np.mat(a).T
>>> linalg.lstsq(F,E)
```

**Generalized inverse**

```
>>> linalg.pinv(C)
>>> linalg.pinv2(C)
```

**Creating Sparse Matrices**

```
>>> F = np.eye(3, k=1)
>>> G = np.mat(np.identity(2))
>>> C[C > 0,5] = 0
>>> H = sparse.csr_matrix(C)
>>> I = sparse.csc_matrix(D)
>>> J = sparse.dok_matrix(A)
>>> E.todense()
>>> sparse.linalg.msc.csc(A)
```

**Sparse Matrix Routines**

**Inverse**

```
>>> sparse.linalg.inv(I)
```

**Norm**

```
>>> sparse.linalg.norm(I)
```

**Solving linear problems**

```
>>> sparse.linalg.spsolve(H,I)
```

**Sparse Matrix Functions**

```
>>> sparse.linalg.expm(I)
```

**Asking For Help**

```
>>> help(scipy.linalg,diagsvd)
>>> np.info(np.matrix)
```

## Matrix Functions

**Addition**

```
>>> np.add(A,D)
```

**Subtraction**

```
>>> np.subtract(A,D)
```

**Division**

```
>>> np.divide(A,D)
>>> A @ D
```

**Multiplication**

```
>>> np.multiply(D,A)
>>> np.dot(A,D)
>>> np.vdot(A,D)
>>> np.inner(A,D)
>>> np.outer(A,D)
>>> np.tensordot(A,D)
>>> np.kron(A,D)
```

**Exponential Functions**

```
>>> linalg.expm(A)
>>> linalg.expm2(A)
>>> linalg.expm3(D)
```

**Logarithm Function**

```
>>> linalg.logm(A)
```

**Trigonometric Functions**

```
>>> linalg.sinc(D)
>>> linalg.cosm(D)
>>> linalg.cism(A)
```

**Hyperbolic Trigonometric Functions**

```
>>> linalg.sinhm(D)
>>> linalg.coshm(D)
>>> linalg.tanhm(A)
```

**Matrix Sign Function**

```
>>> np.signm(A)
```

**Matrix Square Root**

```
>>> linalg.sqrtn(A)
```

**Arbitrary Functions**

```
>>> linalg.funm(A, lambda x: x*x)
```

**Evaluate matrix function**

**Eigenvalues and Eigenvectors**

```
>>> la, v = linalg.eig(A)
Solve ordinary or generalized eigenvalue problem for square matrix
Unpack eigenvalues
First eigenvector
Second eigenvector
Unpack eigenvalues
```

**Singular Value Decomposition (SVD)**

```
>>> U,s,Vh = linalg.svd(B)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)
Construct sigma matrix in SVD
```

**LU Decomposition**

```
>>> P,U = linalg.lu(C)
LU Decomposition
```

**Sparse Matrix Decompositions**

```
>>> la, v = sparse.linalg.eigs(F,1)
Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H,2)
SVD
```

Figuur 3: PYTHON3 cheat sheet ©2012–2015 see header, source: [https://s3.amazonaws.com/assets.datacamp.com/blog\\_assets/Python\\_SciPy\\_Cheat\\_Sheet\\_Linear\\_Algebra.pdf](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_SciPy_Cheat_Sheet_Linear_Algebra.pdf)

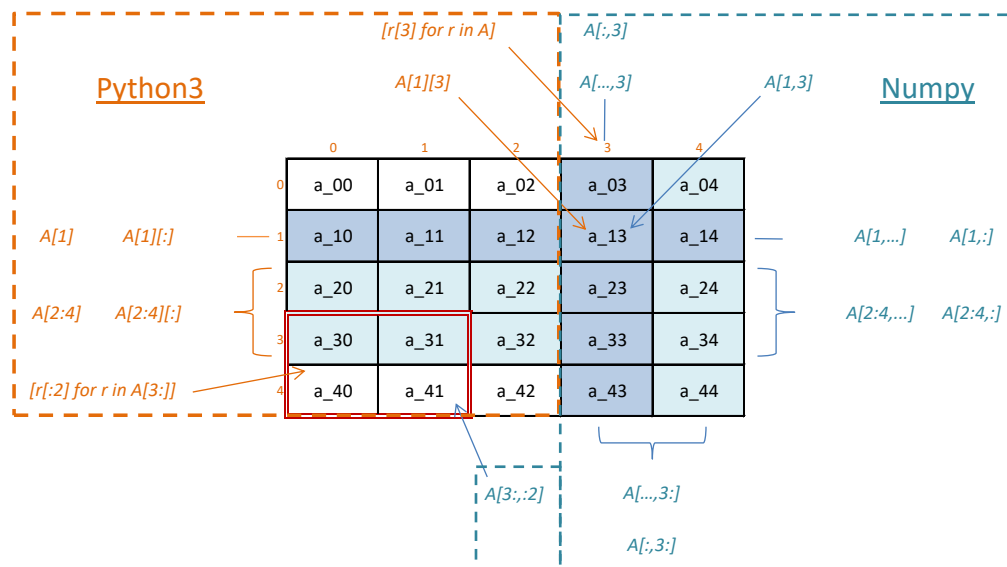


# Appendix D

## SPYDER<sup>3</sup> en NUMPY matrix en vector indexing

Maak onderscheid tussen matrix en vector indexing in de standaard PYTHON code en in een NUMPY omgeving. Daarbij geldt:

1. PYTHON en NUMPY hebben niet dezelfde manier van indexeren helaas. Je kunt zeggen:
  - (a) alles wat in PYTHON werkt, werkt over het algemeen ook in NUMPY-context
  - (b) NUMPY indexing vormt dus een superset van de standaard indexering
  - (c) NUMPY is vooral handiger bij de adressering van kolommen
  - (d) NUMPY is vooral handiger bij de adressering van multi-dimensionale arrays
2. beide vormen van indexering kunnen worden gemixed in NUMPY maar niet andersom
3. beide vormen van indexering worden ook ondersteund met `slice` objecten, zeg maar ranges van indexen; zie onder
4. PYTHON en NUMPY kennen twee-richtingen indexing: positieve indexen voor oplopende indexeringen en negatieve indexering voor aflopende indexen



Figuur 4: matrix indexing en slicing: indexing is het door middel van een index verwijzen naar een element of een groep van elementen in een data structuur zoals een matrix, een vector, een tuple of een list. Een slice is een deel uit zo'n samengestelde data structuur, langs een van de assen daarvan. Links (oranje) is de standaard PYTHON manier om te indexeren, rechts: de door NUMPY daaraan toegevoegde indexing en slicing. Wat in standaard PYTHON werkt, werkt ook in NUMPY. Met name de aanduiding van kolommen zijn wat makkelijker in NUMPY in vergelijking met PYTHON

## indexeren met een slice

Indexing met behulp van een `slice` object:

```
...
print even rows 0,2,4 from 5x5 matrix A ...
print(A[0:5:2]) # slice(start,stop,step), stop not included
evenRows = slice(0,5,2)
print(A[evenRows])
...
```

Index en het corresponderende `slice`-object gebruiken dezelfde `start, stop, step` sequence, de een gebruikt een `:` separator, de ander een comma. Een paar voorbeelden:

- format `slice(start, stop, step)`, met `start` included, `stop` excluded (!), `step` een positieven of negatieve integer
- `A[slice(start, end)]` is equivalent aan `A[start:end]` geeft: alle items `start .. end-1` (de default `step` is gelijk aan 1)

| slice                       | geeft | deze slice                                                                                                                                                                                              |
|-----------------------------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>slice(0)</code>       |       | geen enkel item: voorbeeld matrix A:<br>[] (empty matrix)                                                                                                                                               |
| <code>slice(1)</code>       |       | het eerste item: voorbeeld matrix A:<br>[['a00', 'a01', 'a02', 'a03', 'a04']]                                                                                                                           |
| <code>slice(2)</code>       |       | de eerste 2 items: voorbeeld matrix A:<br>[['a00', 'a01', 'a02', 'a03', 'a04'], ['a10', 'a11', 'a12', 'a13', 'a14']]                                                                                    |
| <code>slice(k)</code>       |       | de eerste k items: voorbeeld matrix A:<br>[ [rij 0], [rij 1], ... , [rij k-1]]                                                                                                                          |
| <code>slice(-1)</code>      |       | alle items behalve laatste: voorbeeld matrix A:<br>[['a00', 'a01', 'a02', 'a03', 'a04'], ['a10', 'a11', 'a12', 'a13', 'a14'], ['a20', 'a21', 'a22', 'a23', 'a24'], ['a30', 'a31', 'a32', 'a33', 'a34']] |
| <code>slice(-2)</code>      |       | alle items behalve de laatste 2: voorbeeld matrix A:<br>[['a00', 'a01', 'a02', 'a03', 'a04'], ['a10', 'a11', 'a12', 'a13', 'a14'], ['a20', 'a21', 'a22', 'a23', 'a24']]                                 |
| <code>slice(-k)</code>      |       | alle items behalve de laatste k: voorbeeld matrix A:<br>[[rij len(A)-(k+1)], [rij len(A)-(k+1)+1], ..., [rij len(A)-1]]                                                                                 |
| <code>slice(-len(A))</code> |       | []                                                                                                                                                                                                      |

Tabel 1: slicing objects reminders

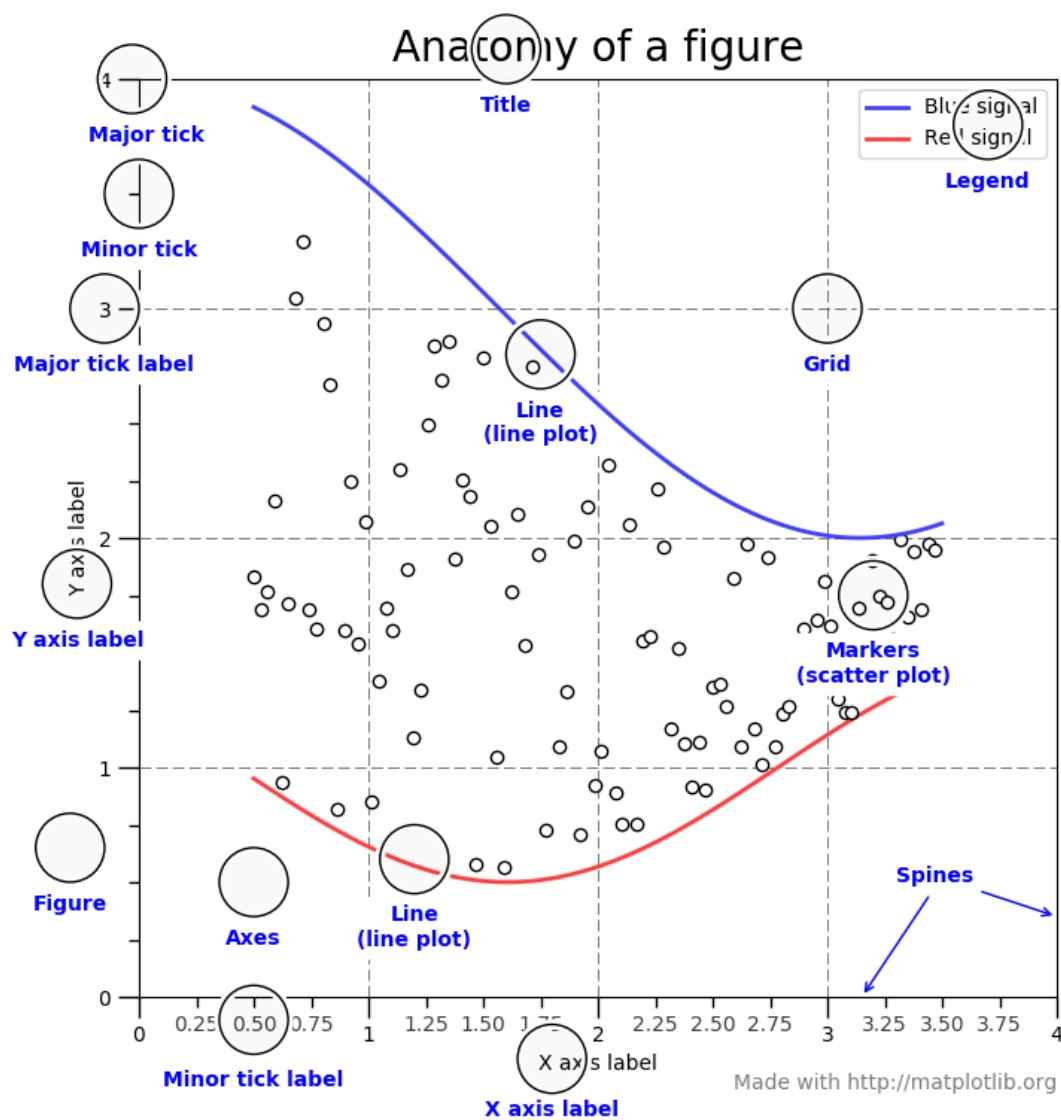
- alles wat je kunt indexeren kan ook worden geïndexeerd met een `slice`, dus ook bijvoorbeeld een tuple of een string

Meer details: zie <http://www.pythoncentral.io/how-to-slice-listsarrays-and-tuples-in-python/>

## MATPLOTLIB terminology

In onderstaande figuur<sup>1</sup> toont de begrippen die je bij het maken van plots met behulp van MATPLOTLIB moet weten.

<sup>1</sup>bron: <https://matplotlib.org/examples/showcase/anatomy.html>



Figuur 5: Terminologie van een MATPLOTLIB plot