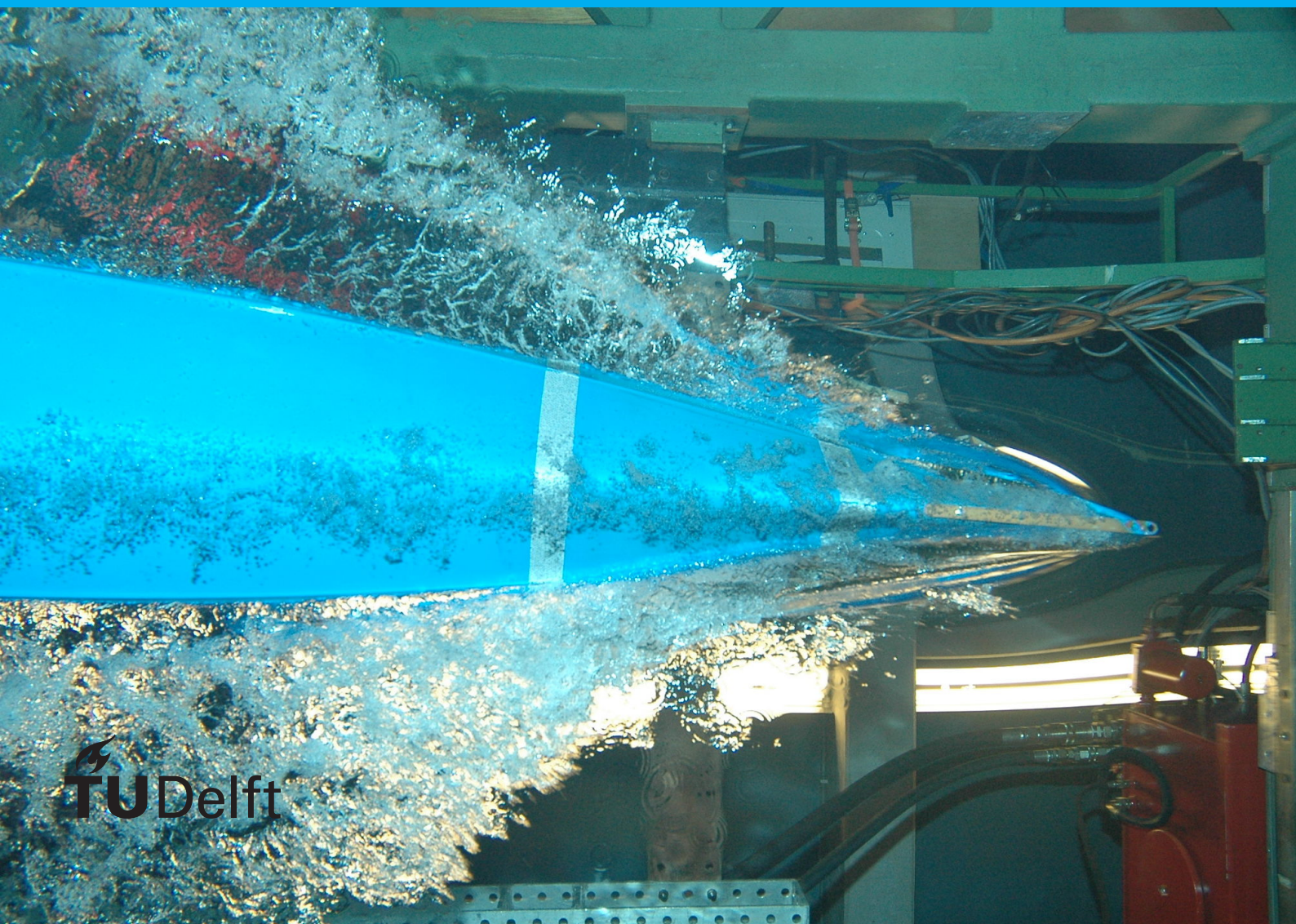
Python Programming

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

1

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 `Nr` af, dat je uit het bestand hebt ingelezen. Druk ook af: het gemiddelde van C_r en het aantal lagers.

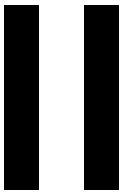
OPDRACHT 2 Plot daarna een histogram, met 10 klassen ('bars', 'bins') voor de C_r-waarde. Plot ook de lijnen voor $min(C_r)$ en $avg(Cr)$ in het histogram. Maak hiervoor het programma af door de `?????` te vervangen door de gevraagde gegevens

1.3. Template-programma

Gebruik template-programma `Bearings.py` voor deze opdracht. Introduceer zelf GEEN NIEUWE VARIABELEN. 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

2

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:

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

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 determanten 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 hoofdfunctie `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 verifieren 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 VARIABELEN. Houd je aan de bestaande namen voor variabelen, functies e.d. Wijzig die niet.

Opgave 3
gematigd moeilijk, 20 min

3

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:

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

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

3.2. Wat moet je doen?

Deze Opgave kent vier opdrachten:

OPDRACHT1 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 waarde $F(a)$ en $F(b = a + h)$.

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 Trapizum 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 VARIABELEN. Houd je aan de bestaande namen voor variabelen, functies e.d. Wijzig die niet.

IV

Opgave 4
moeilijk, 25 min

4

Opgave 4: Signaal analyse

4.1. Intro

Een electronicaonderdeel heeft 2 ingangssignalen:

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

We bestuderen het samengestelde complexe signaal:

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

We willen weten bij welke door ons gekozen omega er wel en wanneer er geen hoekverdraaing (faseverdraaing) 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 $\omega = \frac{k\pi}{2},\ k = 0, 1, 2, 3$

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

4.2. Wat moet je doen?

Deze Opgave kent twee opdrachten:

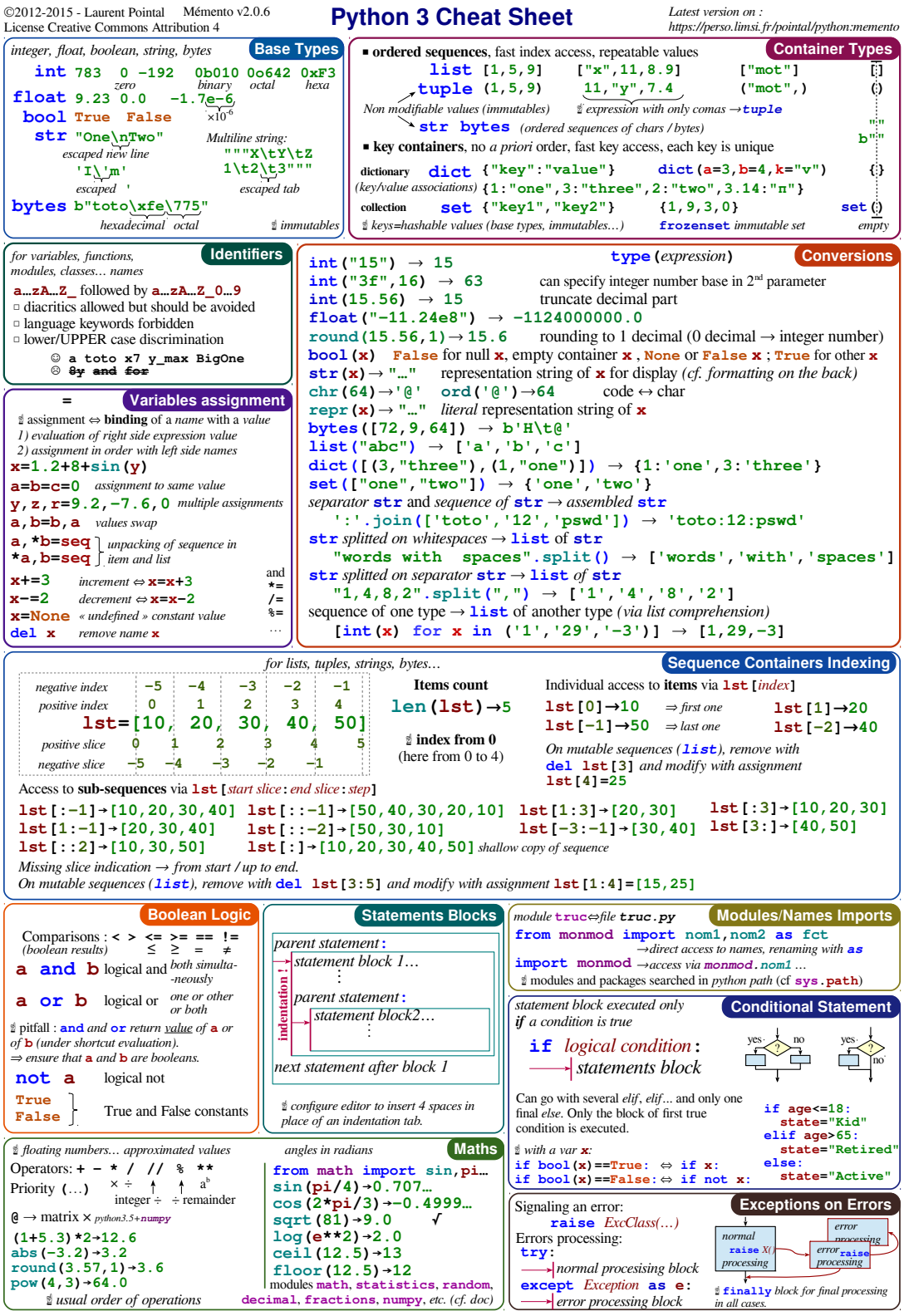
OPDRACHT1 De eerste opdracht is om in het template-programma het complexe signaal $h(\phi)$ in het plot-statement in function `plotSignals()` te definieren (maak het template-programma af)

OPDRACHT 2 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 faseverdraaing 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 VARIABELEN. Houd je aan de bestaande namen voor variabelen, functies e.d. Wijzig die niet.

Appendix A



Figuur 1: PYTHON3 cheat sheet ©2012–2015 see header,
source: https://perso.limsi.fr/pointal/_media/python:cours:mementopython3-english.pdf

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

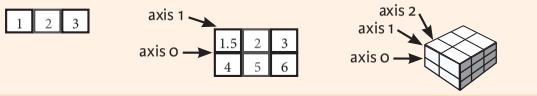
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                        Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                 Multiplication
>>> np.exp(b)                        Exponentiation
>>> np.sqrt(b)                       Square root
>>> np.sin(a)                        Print sines of an array
>>> np.cos(b)                        Element-wise cosine
>>> np.log(a)                        Element-wise natural logarithm
>>> e.dot(f)                         Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)             Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                          Array-wise sum
>>> a.min()                          Array-wise minimum value
>>> b.max(axis=0)                    Maximum value of an array row
>>> b.cumsum(axis=1)                 Cumulative sum of the elements
>>> a.mean()                         Mean
>>> b.median()                       Median
>>> a.corrcoef()                     Correlation coefficient
>>> np.std(b)                        Standard deviation
```

Copying Arrays

```
>>> h = a.view()          Create a view of the array with the same data
>>> np.copy(a)            Create a copy of the array
>>> h = a.copy()          Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()              Sort an array
>>> c.sort(axis=0)        Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                             Select the element at the 2nd index
  3
>>> b[1,2]                           Select the element at row 0 column 2
  6.0                                (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                           Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                         Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                            Select all items at row 0
  array([[1.5, 2., 3.]])             (equivalent to b[0:1, :])
>>> c[1,...]                         Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                        Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                           Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]    Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]    Select a subset of the matrix's rows
  array([[ 4. ,  5. ,  6. ,  4. ],   and columns
         [ 1.5,  2. ,  3. ,  1.5],
         [ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)              Permute array dimensions
>>> i.T                              Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                        Flatten the array
>>> g.reshape(3,-2)                  Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))                  Return a new array with shape (2,6)
>>> np.append(h,g)                   Append items to an array
>>> np.insert(a, 1, 5)               Insert items in an array
>>> np.delete(a,[1])                 Delete items from an array
```

Combining Arrays

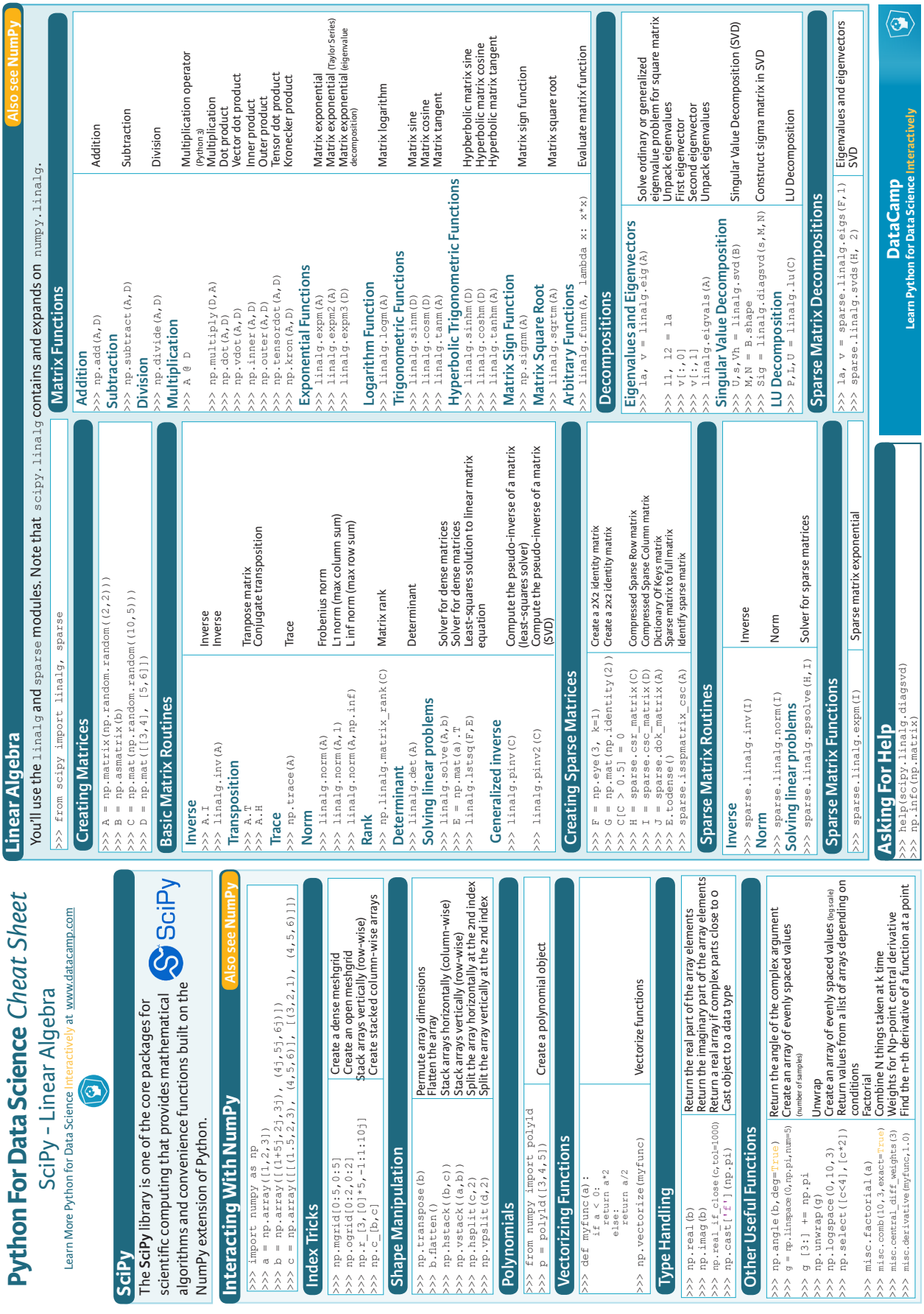
```
>>> np.concatenate((a,d),axis=0)     Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                 Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                       Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                 Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))           Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                       Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                   Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]  index
>>> np.vsplit(c,2)                   Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3. ,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]]])]
```

DataCamp
Learn Python for Data Science Interactively

Figuur 2: PYTHON3 cheat sheet ©2012-2015 see header, source: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf



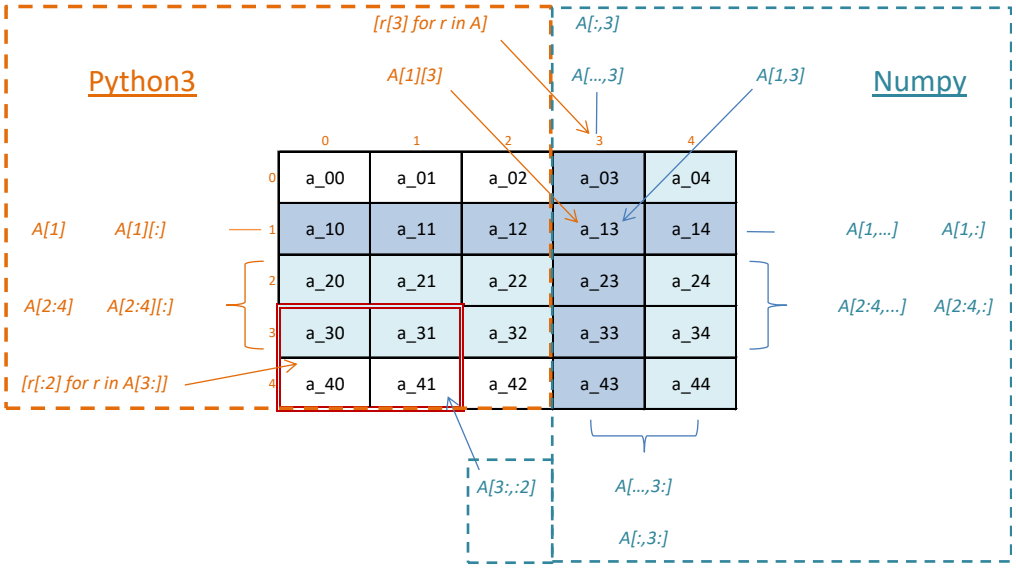
Figuur 3: PYTHON3 cheat sheet ©2012–2015 see header,
source: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_SciPy_Cheat_Sheet_Linear_Algebra.pdf

Appendix D

SPYDER[3] en NUMPY matrix en vector indexing

Maak onderscheid tussen matrix en vector indexering 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 indexering vormt dus een superset van de standaard indexerings
   (c) NUMPY is vooral handiger bij de addressering van kolommen
   (d) NUMPY is vooral handiger bij de addressering 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 indexering en slicing: indexering 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* darvan. Links (oranje) is de standaard PYTHON manier om te indexeeren, 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
`slice(0)`		geen enkel item: voorbeeld matrix A: `[]` (empty matrix)
`slice(1)`		het eerste item: voorbeeld matrix A: `[['a00', 'a01', 'a02', 'a03', 'a04']]`
`slice(2)`		de eerste 2 items: voorbeeld matrix A: `[['a00', 'a01', 'a02', 'a03', 'a04'], ['a10', 'a11', 'a12', 'a13', 'a14']]`
`slice(k)`		de eerste k items: voorbeeld matrix A: `[ [rij 0], [rij 1], ... , [rij k-1]]`
`slice(-1)`		alle items behalve laatste: voorbeeld matrix A: `[['a00', 'a01', 'a02', 'a03', 'a04'], ['a10', 'a11', 'a12', 'a13', 'a14'], ['a20', 'a21', 'a22', 'a23', 'a24'], ['a30', 'a31', 'a32', 'a33', 'a34']]`
`slice(-2)`		alle items behalve de laatste 2: voorbeeld matrix A: `[['a00', 'a01', 'a02', 'a03', 'a04'], ['a10', 'a11', 'a12', 'a13', 'a14'], ['a20', 'a21', 'a22', 'a23', 'a24']]`
`slice(-k)`		alle items behalve de laatste k: voorbeeld matrix A: `[[rij len(A)-(k+1)], [rij len(A)-(k+1)+1], ..., [rij len(A)-1]]`
`slice(-len(A))`		`[]`

Tabel 1: slicing objects reminders

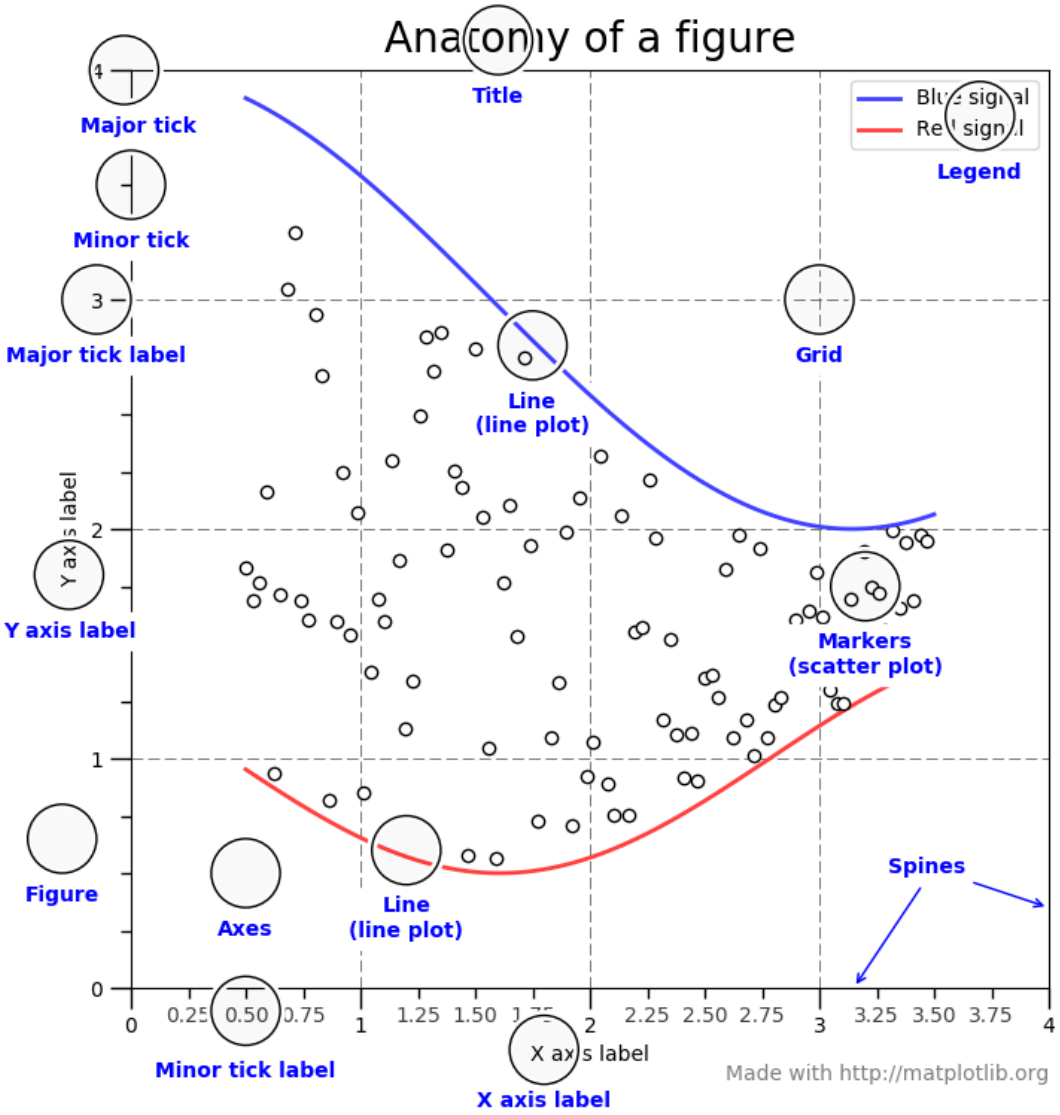
- alles wat je kunt indexeren kan ook worden geindexeerd met een `slice`, dus ook bijvoorbeeld een `tuple` of een `string`

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

MATPLOTLIB **terminology**

In onderstaande figuur[1] toont de begrippen die je bij het maken van plots met behulp van MATPLOTLIB moet weten.

[1] bron: https://matplotlib.org/examples/showcase/anatomy.html



Figuur 5: Terminologie van een MATPLOTLIB plot