KU LEUVEN kulak

Wetenschappelijk rekenen

{`Python`, `Sympy`, `Numpy`, `Matplotlib`}

Groep Wetenschap & Technologie, Kulak

X0B53a – Wetenschappelijk rekenen en schrijven

Overzicht

1. Sessie 1: Functie-analyse
2. Sessie 2: Grafieken
3. Sessie 3: Vectoren en matrices
4. Sessie 4: Programmeerstructuren
5. Sessie 5: Inoefenopdracht (facultatief)
6. Sessie 6: Gequoteerde oefening

Overzicht

Programmeren versus Rekenen

Beginselen van Programmeren	Wetenschappelijk rekenen
Programmeren	Scripten
Werken in 2 fasen: 1. Schrijven 2. Uitvoeren	Interactief werken: Resultaat per commando
Algoritmes implementeren	Concrete berekeningen
Gestructureerde code	Quick & dirty commando's
Algemene principes	Symbolische en numerieke berekeningen, grafieken
`VS Code`	`VS Code Interactive Mode`

Programmeren versus scripten

Programmeren	Scripten
1. Schrijven: `som.py`	Lijn per lijn uitvoeren: `F9`

`## som.py`
`# This program ...`
`#`
`a = 1`
`b = 2`
`print(a+b)`
	`In[1]: a=1`
	`In[2]: a`
	`Out[2]: 1`
	`In[3]: b=2`
	`In[4]: b`
	`Out[4]: 2`
	`In[5]: a+b`
	`Out[5]: 3`

2. Uitvoeren: `F5`	
`> python som.py` `3`	

Python versus Anaconda

$\texttt{Anaconda} = \{\texttt{Python}, \texttt{Sympy}, \texttt{Numpy}, \texttt{Matplotlib}, \ldots\}$

Pakket	Functionaliteit
`IPython`	Interactieve shell
`Sympy`	Symbolisch rekenen
`Numpy`	Numeriek rekenen
`Matplotlib`	Grafieken
`SciKit`	Machine learning
...	

Getallen in `Python`

- `int` – oneindige precisie
- `float` – eindige mantisse
- Nood aan *reële* getallen

```
In[1]: 21**34
Out[1]: 9025183088777951914332401034032563746234570 81
In[2]: type(21**34)
Out[2]: int
In[3]: type(13/21)
Out[3]: float
In[4]: 13/21
Out[4]: 0.6190476190476191
```

Eén commando inlezen uit module

- Cosinus niet gedefinieerd in `Python`
- Wel in module `Sympy`
- Inlezen met `from sympy import cos`

```
In[1]: cos(0)
NameError: name 'cos' is not defined
In[2]: from sympy import cos
In[3]: cos(0)
Out[3]: 1
```

Volledige module inlezen

- Tientallen commando's: sinus, tangens, π, ...
- Simultaan inlezen met `import sympy`
- Prefix nodig!?

```
In[1]: tan(pi)
NameError: name 'tan' is not defined
In[2]: import sympy
In[3]: tan(pi)
NameError: name 'tan' is not defined
In[4]: sympy.tan(sympy.pi)
Out[4]: 0
```

Conflicterende commando's

Zelfde commando's in andere pakketten hebben vaak andere werking:

- `sympy.pi` – object met wiskundige eigenschappen
- `numpy.pi` – numerieke benadering

```
In[1]: import sympy
In[2]: sympy.pi
Out[2]: pi
In[3]: type(sympy.pi)
Out[3]: sympy.core.numbers.Pi
In[4]: import numpy
In[5]: numpy.pi
Out[5]: 3.141592653589793
In[6]: type(numpy.pi)
Out[6]: float
```

Prefixen weglaten

- Enkel als er geen conflicten mogelijk zijn:
  `from sympy import *`

```
In[1]: exp(I*pi)
NameError: name 'exp' is not defined
In[2]: from sympy import *
In[3]: exp(I*pi)
Out[3]: -1
```

Symbolisch rekenen

- Symbolisch rekenen kan met `Sympy`
- Output van `Sympy`-commando's steeds exact
- Niet bij breuken (`float`)!
- Gebruik `Rational()`

```
In[1]: from sympy import *
In[2]: sqrt(8)
Out[2]: 2*sqrt(2)
In[3]: 1/2
Out[3]: 0.5
In[4] Rational(1,2)
Out[4]: 1/2
```

Rekenen met onbekenden

- Wiskundige onbekenden initialiseren met `symbols()`
- Berekeningen met onbekenden resulteren is een *expressie*
- Expressies evalueren met `.subs()`

```
In[1]: from sympy import *
In[2]: p = (1+x)**2
NameError: name 'x' is not defined
In[3]: var('x')
In[4]: p = (x+1)**2
In[5]: expand(p)
Out[6]: x**2 + 2*x + 1
```

Expressies

- Evalueren met methode `.subs()`
- Klassieke functie-evaluatie (*function call*) werkt niet

```
In[1]: p = (x+1)**2
In[2]: p(4)
TypeError: 'Pow' not callable
In[3]: p.subs(x,4)
Out[3]: 25
In[4]: Dp = diff(p,x); Dp
Out[4]: 2*x + 2
In[5]: Dp(4)
TypeError: 'Add' not callable
In[6]: Dp.subs(x,4)
Out[6]: 10
```

Functies

- Definieer functies met `def`
- Functies makkelijk te evalueren (*callable*)
- Is `f` een functie, dan is `f(x)` een expressie

```
In[1]: def f(x): return (x+1)**2; f(4)
Out[1]: 25
In[2]: Df = diff(f(x),x); Df
Out[2]: 2*x + 2
In[3]: Df(4)
TypeError: 'Add' not callable
In[4]: Df.subs(x,4)
Out[4]: 4
In[5]: def Df(x0): return diff(f(x),x).subs(x,x0); Df(0)
Out[5]: 2
```

Functie-analyse

- Limiet
  `limit(expression, variable, value, direction)`
- Afgeleide
  `diff(expression, variable)`
- Primitieve functie
  `integrate(expression, variable)`
- Bepaalde integraal
  `integrate(expression, (variable, value1, value2) )`
- Reeksontwikkeling
  `series(expression, variable, value, degree)`
- Nulpunt zoeken
  `solve(expression, variable)`

Aan de slag

- Ga naar `Toledo`, `[X0B53a]`, `Rekenen`
- Lees hoofdstuk 1 van de handleiding:
  `WetenschappelijkRekenenHandleiding.pdf`
  Zorg dat je alle commando's begrijpt!

Overzicht

Grafieken tekenen

- Computergrafieken zijn in wezen losse punten (x_i, y_i), verbonden door lijnstukken
- Voldoende groot aantal n voor *glad* resultaat
- In wezen betekent grafieken maken intensief rekenen:
  - `Numpy` voor rekenwerk
  - `Matplotlib` voor grafische voorstelling
  - (en `Sympy` om met symbolische functies te rekenen)
  - Daarom: pakketten mét prefix gebruiken

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
```

Puntenlijsten berekenen

- x-waarden meestal equidistant tussen grenzen a en b
- Twee mogelijkheden in `Numpy`:
  - `arange(a,b,d)` – kies stapgrootte `d`
  - `linspace(a,b,n)` – kies aantal `n`
- corresponderende y-waarden componentsgewijs berekenen:
  - automatisch in `Python` en `Numpy`
  - `Sympy`-functies zijn strikt $\mathbb{R} \to \mathbb{R}$ (workaround later)

```
np.arange(0,1,0.1)
xx = np.linspace(0,1,10); xx
yy = xx**2; yy
zz = np.sin(xx); zz
# sp.sin(xx) # werkt niet
```

Grafiekvenster openen en manipuleren

`Matplotlib`-commando's:

- `figure()` – nieuwe grafiek openen
- `plot()` – grafiek toevoegen
- `show()` – grafiekvenster tonen
- `close()` – grafiekvenster sluiten

```
plt.figure()
plt.plot(xx,yy)
plt.plot(xx,zz)
plt.show()
plt.close()
```

Lay-out van grafieken

- `scatter()` voor louter puntenplot
- `lw` – lijndikte
- `label` – invoer voor legende
- Andere opties via een samengestelde code:

`'.'`	punt	`'b'`	blauw	`'-'`	doorlopende lijn
`'o'`	cirkel	`'g'`	groen	`'--'`	streepjeslijn
`'*'`	ster	`'r'`	rood	`'-.'`	punt-streeplijn
`'+'`	plusteken	`'k'`	zwart	`':'`	stippellijn

```
xx = np.linspace(0,2*np.pi,20); plt.figure()
plt.plot(xx,np.sin(xx),'b-',label='Sinus')
plt.plot(xx,np.cos(xx),'g:*',lw=2,label='Cosinus')
plt.plot(xx,np.tan(xx),'r--o',label='Tangens')
plt.scatter(xx,1/np.tan(xx),c='k',label='Cotangens')
```

Grafieken afwerken

- `xlim()` en `ylim()` – bereik instellen
- `xlabel()` en `ylabel()` – aslabels toevoegen
- `suptitle()` – grafiektitel toevoegen
- `legend()` – legende toevoegen

```
plt.ylim([-2,2])
plt.xlim([np.pi/2,3*np.pi/2])
plt.xlabel('x')
plt.ylabel('y')
plt.suptitle('Goniometrische functies')
plt.legend()
```

Kwalitatieve figuren exporteren

- `savefig()` – grafiek exporteren
- Gebruik voldoende plotpunten
  (te veel vergt extra rekenkracht/geheugen)
- Verzorg lay-out (titel, aslabels, lijnstijlen,. . .)
  - Grafiek moet ondubbelzinnig leesbaar zijn
  - Artefacten benadrukken: kies gepast bereik
  - Functioneel: grafieken onderscheiden
  - Denk aan zwart-wit-afdrukken
- Voeg steeds extensie `.pdf` toe!
  - Resulteert in vectoriële afbeelding
  - Standaard is `.png`, rasterafbeelding (vermijden)

```
plt.savefig('figuur.pdf')
```

Grafieken van Sympy-objecten

Sympy-functies bereken slechts één functiewaarde tegelijk

1. Ofwel werken met Numpy-functies
   - Geen symbolische berekeningen mogelijk
2. Ofwel `lambdify()`: converteert naar numerieke functie
   - Aangewezen bij symbolische berekeningen
3. Ofwel het Sympy-smartplot-commando gebruiken
   - Berekent automatisch plotpunten
   - Goed voor snelle resultaten, minder manipuleerbaar

```
xx = np.linspace(0,2*np.pi,200); x = sp.symbols('x')
yy = sp.lambdify(x,sp.sin(x),'numpy')(xx)
plt.plot(xx,np.sin(xx))     # 1.
plt.plot(xx,yy)             # 2.
sp.plotting.plot(sp.sin(x)) # 3.
```

Aan de slag

- Ga naar `Toledo`, `[X0B53a]`, `Rekenen`
- Maak de oefeningen bij hoofdstuk 2: `WetenschappelijkRekenenOefeningen.pdf`
- Lees grondig sectie 2.2: `WetenschappelijkRekenenHandleiding.pdf` Tracht ook de andere grafiektypes te begrijpen. Commando's uit sectie 2.1: zeer interessant maar komen verder niet aan bod binnen dit vak.
- Volgende week vervolg: werk desnoods thuis af

Overzicht

Lijsten

- Opeenvolging van meerdere waarden `a = [...]`
- Indices `0` tot `n-1`: `a[0],...,a[n]`
- Samenvoegen met plusteken, `a[:3]+a[3:]` $\equiv$ `a`

```
In[1]: a = [2,3,5,7,11,13]; type(a)
Out[1]: <class 'list'>
In[2]: a[0]
Out[2]: 2
In[3]: a[6]
IndexError: list index out of range
In[4]: a+a
Out[4]: [2, 3, 5, 7, 11, 13, 2, 3, 5, 7, 11, 13]
In[5]: a[:3]+a[3:]
Out[5]: [2, 3, 5, 7, 11, 13]
```

Speciale lijsten

- `[]` lege lijst
- `range(start,stop,stap)` voor "rekenkundige" lijsten
- `[... for ... in ...]` voor constructieve lijsten
- `[... for ... in ... if ...]` met voorwaarde
- `In` en `Out` in `IPython`

```
In[1]: range(3,10,2)[3]
Out[1]: 9
In[2]: [x**2 for x in range(10)]
Out[2]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
In[3]: [x**2 for x in range(10) if x%2==0]
Out[3]: [0, 4, 16, 36, 64]
In[4]: Out[1]
Out[4]: 9
```

Lijsten manipuleren

- `del()` element verwijderen
- Element overschrijven
- Element toevoegen
- Element tussenvoegen

```
In[1]: b = [17,18,19,20,29]
In[2]: del(b[1]); b
Out[2]: [17,19,20,29]
In[3]: b[1]=20
Out[3]: [17,20,20,29]
In[4]: b += [31]
Out[4]: [17,20,20,29,31]
In[4]: b = b[:2] + [12] + b[2:]
Out[4]: [17,20,12,20,29,31]
```

Symbolische matrices

- `sp.Matrix()` maakt matrix van lijst van lijsten (rijen)
- Lijst van getallen (lijsten) wordt kolomvector
- Lijst van lijst van getallen wordt rijvector

```
In[1]: v = sp.Matrix([1,2,3]); v
Out[1]: Matrix([[1],
                [2],
                [3]])
In[2]: w = sp.Matrix([[4,5,6]]); w
Out[2]: Matrix([[4, 5, 6]])
In[3]: A = sp.Matrix([[1,2,3],[3,5,7],[2,1,3]]); A
Out[3]: Matrix([[1, 2, 3],
                [3, 5, 7],
                [2, 1, 3]])
```

Selecties in symbolische matrices

- Methode `.shape` geeft dimensie van matrix
- Element selecteren met `A[i,j]`
- Kolom of rij selecteren met `A[:,j]` resp. `A[i,:]`

```
In[1]: A.shape
Out[2]: (3, 3)
In[2]: A[1,2]
Out[2]: 7
In[3]: A[1,:]
Out[3]: Matrix([[3, 5, 7]])
In[4]: A[:,2]
Out[4]: Matrix([[3],
                [7],
                [3]])
```

Symbolisch matrixrekenen

- `sp.Matrix()`: enkel strikt wiskundige bewerkingen
- Wiskundige objecten in matrices mogelijk

```
In[1]: 2*v.transpose()+w
Out[1]: Matrix([[6, 9, 12]])
In[2]: v*w
Out[2]: Matrix([[ 4,  5,  6],
                [ 8, 10, 12],
                [12, 15, 18]])
In[3]: sp.var('x y z')
In[4]: A*sp.Matrix([x,y,z])
Out[4]: Matrix([[  x + 2*y + 3*z],
                [3*x + 5*y + 7*z],
                [  2*x + y + 3*z]])
```

Symbolische matrixbewerkingen

- Meeste `sp.Matrix()`-technieken zijn *methodes*
- Symbolisch rekenen typisch traag!

```
In[1]: sp.var('a b c d')
In[2]: B = sp.Matrix([[a,b],[c,d]])
In[3]: B.det()
Out[3]: a*d - b*c
In[4]: B.inv()
Out[4]: Matrix([
    [1/a+b*c/(a**2*(d-b*c/a)),-b/(a*(d-b*c/a))],
    [        -c/(a*(d-b*c/a)),    1/(d-b*c/a)]])
In[5]: A.eigenvects()[1]
Out[5]: (1 + sqrt(6), 1, [Matrix([
    [-sqrt(6)*(-sqrt(6) + 1)/(3*(sqrt(6)/3 + 1))],  ...])])
```

Aan de slag

- Ga naar `Toledo`, `[X0B53a]`, `Rekenen`
- Maak de oefeningen bij hoofdstuk 3:
  `WetenschappelijkRekenenOefeningen.pdf`
- Lees hoofdstuk 3 in de handleiding:
  `WetenschappelijkRekenenHandleiding.pdf`
  Secties 1 en 2 kwamen al aan bod in de les,
  tracht ook sectie 3 (numerieke matrices) te begrijpen.
- Volgende week vervolg: werk desnoods thuis af

Overzicht

Programmeer- en rekentechnieken consolideren

- Mogelijkheden van `Python` (automatisatie) samenvoegen met wetenschappelijke pakketten (symbolisch en numeriek rekenen, grafieken maken)
- Voorbeeld: Functie-analyse automatiseren
  1. Kritieken punten berekenenen (`Sympy`)
  2. Punten overlopen (`for`-lus)
  3. Punten classificeren (`if`-statement)
  4. Resultaat tekenen (`Matplotlib`)
  5. Geheel in een `Python`-procedure gieten

Programmastructuur

```
##
# Beschrijving van het programma

import module as ...

def main():
   <code>

## Doel van de functie
# @param x uitleg bij eerste argument
# @return uitleg bij de output
def functie(x,y):
   <code>

main()
```

Bijkomende informatie vinden

- Informatie over werking van commando's ($3\times$ identiek)
  - Inline help in console: `?commando`
  - Object inspector: `ctrl+I`
  - Online documentatie: `python.org`, `sympy.org`, ...
- Informatie bij specifieke problemen
  - Zoekmachine: `google`, ...
  - Gespecialiseerde fora: `stackoverflow`, ...
- Tijdens de examenopdracht
  - Verplicht werken op een PC-klas computer: test dit op voorhand!
  - Enkel de ingebouwde help (console of object inspector) en lokale bronnen (`.py` en `.pdf`) toegelaten

Aan de slag

- Ga naar `Toledo`, `[X0B53a]`, `Rekenen`
- Lees hoofdstuk 4 in de handleiding:
  - Herhaal syntax programmeerstructuren (4.1)
  - Lees programmeertips (4.2)
- Maak de oefeningen bij hoofdstuk 4:
  `WetenschappelijkRekenenOefeningen.pdf`
- Volgende week inoefenopdracht:
  bereid dit voor alsof het een examen betreft!

Overzicht

Instructies voor facultatieve en examenoefening

- Maak één `Python`-script, indienen via `Toledo`
- Voorbeeldcode in de functie `main()`
- Voorzie de code van commentaar
- Focus eerst op de algemene structuur van elke procedure:
  procedure hoeft niet te werken om (grotendeels) juist te zijn (kleine) syntaxfouten opsporen vaak tijdrovend
- Wanneer een procedure niet correct is,
  werk er toch mee verder,
  gebruik dummy-uitvoer om te testen
- Vermijd dubbel werk, gebruik liever hulpprocedures

Inoefenopdracht `Python` (facultatief)

- Dit is een inoefenopdracht:
  los zo ernstig mogelijk op, binnen de voorziene tijd, samenwerken mag – in stilte!
- Opgave telt drie bladzijden:
  lees deze aandachtig vooraleer te beginnen
- Enkel de ingebouwde help (console of object inspector) en
  lokale bronnen (`.py` en `.pdf`) toegelaten
- Antwoord indienen via Toledo:
  enkel het `.py`-bestand
- Niemand verlaat het examenlokaal
  vóór het einde van de toets
- Respecteer bestandsnaam en deadline:
  indienen na maximaal 2u!

Overzicht

Gequoteerde oefening `Python` (14u)

- Dit is een examenopdracht:
  ongeoorloofde communicatie = fraude
- Opgave telt drie bladzijden:
  lees deze aandachtig vooraleer te beginnen, in het bijzonder de instructies!
- Gebruik enkel de ingebouwde help (object inspector) en lokale bronnen (`.py` en `.pdf`)
- Antwoord indienen via Toledo:
  enkel het `.py`-bestand
- Niemand verlaat het examenlokaal
  vóór het einde van de toets
- Respecteer bestandsnaam en deadline:
  indienen vóór 16u!

Gequoteerde oefening `Python` (16u)

- Dit is een examenopdracht:
  ongeoorloofde communicatie = fraude
- Opgave telt drie bladzijden:
  lees deze aandachtig vooraleer te beginnen, in het bijzonder de instructies!
- Gebruik enkel de ingebouwde help (object inspector) en lokale bronnen (`.py` en `.pdf`)
- Antwoord indienen via Toledo:
  enkel het `.py`-bestand
- Respecteer bestandsnaam en deadline:
  indienen vóór 18u!