Politechnika Świętokrzyska

Laboratorium

Metody numeryczne

Ćwiczenie 1

Wprowadzenie do obliczeń w języku Python

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie studentów z językiem programowania Python i i bibliotekami pozwalającymi na efektywne implementowanie obliczeń numerycznych. Poznanie funkcji pozwalających na wyświetlanie wyników w postaci graficznej.

dr inż Robert Kazała

W języku Python należy zaimportować wybrany moduł. W przypadku korzystania z programu Spider domyślnie importowane są do zadanych przestrzeni nazw następujące moduły:
import numpy as np, import scipy as sp, import matplotlib as mpl, import matplotlib.pyplot as plt.

Podstawy języka Python

Praca interaktywna

```
# dodawanie dwóch liczb
>>> 1 + 1
2
# tworzenie zmiennych
>>> a = 1
>>> a
1
# sprawdzanie typów
>>> type(a)
<type 'int'>
# typ integer o dowolnej precyzji
>>> a = 1203405503201
>>> a
1203405503201L
>>> type(a)
<type 'long'>

# liczby rzeczywiste
>>> b = 1.2 + 3.1
>>> b
4.2999999999999998
>>> type(b)
<type 'float'>

# liczby zespolone
>>> c = 2+1.5j
>>> c
(2+1.5j)
```

Pyton posiada cztery typy numeryczne, dla architektury 32-bitowej są one następujące:

- integer (4 bajty)
- long integer (dowolna precyzja)
- float (8 bajtów jak typ double w C)
- complex (16 bajtów)

Moduł Numeric umożliwia wykorzystanie większej ilości typów numerycznych.

Tworzenie ciągów znaków

```
# z wykorzystaniem podwójnego lub pojedynczego cudzysłowu
>>> s = “hello world”
>>> print s
hello world
>>> s = ‘hello world’
>>> print s
hello world
```

Formatowanie ciągów znaków

```
>>> s = “some numbers:”
>>> x = 1.34
>>> y = 2
>>> s = “%s %f, %d” % (s,x,y)
>>> print s
some numbers: 1.34, 2
```

Tworzenie list

```
>>> l = [10,11,12,13,14]
>>> print l
[10, 11, 12, 13, 14]
```

Tworzenie list z wykorzystaniem zakresów - range(start, stop, step)

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(2,7)
[2, 3, 4, 5, 6]
>>> range(2,7,2)
[2, 4, 6]
```

Odczyt pojedynczych elementów

```
>>> list = [10,11,12,13,14]
>>> list[0]
10
```

Tworzenie list o różnym typie elementów

```
>>> list = [10,’eleven’,[12,13]]
>>> list[1]
‘eleven’
>>> list[2]
[12, 13]
```

Odczyt zagnieżdżonych elementów listy

```
>>> list[2][0]
12
```

Usuwanie elementów listy

```
>>> del l[2]
>>> l
[10,’eleven’]
```

Odczyt zakresów listy

```
>>> l = [10,11,12,13,14]
>>> l[1:3]
[11, 12]
>>> l[1:-2]
[11, 12]
# wykorzystanie ujemnych indeksów
>>> l[-4:3]
[11, 12]

# odczyt pierwszych trzech elementów
>>> l[:3]
[10,11,12]
# odczyt ostatnich dwóch elementów
>>> l[-2:]
[13,14]
```

Wykonywanie operacji na listach

```
>>> l = [10,21,23,11,24]
# dodanie elementu
>>> l.append(11)
>>> print l
[10,21,23,11,24,11]
# zliczanie wystąpień wartości 11
>>> l.count(11)
2
# pierwsze wystąpienie 11
>>> l.index(11)
3
# usunięcie pierwszej 11
>>> l.remove(11)
>>> print l
[10,21,23,24,11]
# sortowanie listy
>>> l.sort()
>>> print l
[10,11,21,23,24]
# zmiana kolejności elementów listy
>>> l.reverse()
>>> print l
[24,23,21,11,10]
```

Słowniki

```
# utworzenie pustego słownika i dodawanie elementów
>>> record = {}
>>> record[‘first’] = ‘James’
>>> record[‘last’] = ‘Maxwell’
>>> record[‘born’] = 1831
>>> print record
{'first': 'James', 'born': 1831, 'last': 'Maxwell'}
```

Niemodyfikowalne sekwencje obiektów (tuples)

```
>>> t = (1,’two’)
>>> print t
(1,‘two’)
>>> t[0]
1
```

Wyrażenie warunkowe

```
>>> x = 10
>>> if x > 0:
... print 1
... elif x == 0:
... print 0
... else:
... print –1
... < hit return >
1
```

Pętla for – iteracja po sekwencji obiektów

```
>>> for i in range(5):
... print i,
... < hit return >
0 1 2 3 4

>>> for i in ‘abcde’:
... print i,
... < hit return >
a b c d e

>>> l=[‘dogs’,’cats’,’bears’]
>>> accum = ‘’
>>> for item in l:
... accum = accum + item
... accum = accum + ‘ ‘
... < hit return >
>>> print accum
dogs cats bears
```

Pętla while

```
>>> lst = range(3)
>>> while lst:
... print lst
... lst = lst[1:]
... < hit return >
[0, 1, 2]
[1, 2]
[2]
```

Tworzenie funkcji

```
>>> def add(x,y):
... a = x + y
... return a

# wywołanie funkcji z liczbami
>>> x = 2
>>> y = 3
>>> add(x,y)
5

# wywołanie funkcji z ciągami
>>> x = 'foo'
>>> y = 'bar'
>>> add(x,y)
'foobar'
```

Moduł NumPy

```
>>> import numpy as np
>>> x = np.float32(1.0)
>>> x
1.0
>>> y = np.int_([1,2,4])
>>> y
array([1, 2, 4])
>>> z = np.arange(3, dtype=np.uint8)
>>> z
array([0, 1, 2], dtype=uint8)
```

Tworzenie tablic

```
>>> a = array([1,2,3,4])
>>> b = array([2,3,4,5])
>>> a + b
array([3, 5, 7, 9])
>>> a[0]
1
>>> a[0] = 10
>>> a
[10, 2, 3, 4]
```

```
# Utworzenie wektora od 0 do 10
>>> x = arange(11.)
>>> a = (2*pi)/10.
>>> a
0.628318530718
>>> a*x
array([ 0.,0.628,…,6.283])
>>> y = sin(a*x)
```

Tablice wielowymiarowe

```
>>> a = array([[ 0, 1, 2, 3],
[10,11,12,13]])
>>> a
array([[ 0, 1, 2, 3],
[10,11,12,13]])

>>> a[1,3]
13

>>> a[1]
array([10, 11, 12, 13])

>>> a[0,3:5]
array([3, 4])
>>> a = array([0,1.,2,3],'f')
>>> a.dtype()
'f'
```

Typ danych	Opis
bool	Boolean (True or False) stored as a byte
int	Platform integer (normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float	Shorthand for float64.
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex	Shorthand for complex128.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Funkcje do tworzenia tablic

arange(start,stop=None,step=1,typecode=None)

```
>>> arange(0,2*pi,pi/4)
array([ 0.000, 0.785, 1.571, 2.356, 3.142,
3.927, 4.712, 5.497])

ones(shape,typecode=None,savespace=0)
zeros(shape,typecode=None,savespace=0)

>>> ones((2,3),typecode=Float32)
array([[ 1., 1., 1.],
[ 1., 1., 1.]],'f')
```

Operatory i funkcje matematyczne

a + b	add(a,b)
a - b	subtract(a,b)
a % b	remainder(a,b)
a * b	multiply(a,b)
a / b	divide(a,b)
a ** b	power(a,b)

Operatory porównania i logiczne

equal	(==)
greater_equal	(>=)
logical_and	(and)
logical_not	(not)
not_equal	(!=)
less	(<)
logical_or	(or)
greater	(>)
less_equal	(<=)
logical_xor	

```
>>> a = array(((1,2,3,4),(2,3,4,5)))
>>> b = array(((1,2,5,4),(1,3,4,5)))
>>> a == b
array([[1, 1, 0, 1],
[0, 1, 1, 1]])
# funkcjonalny odpowiednik
>>> equal(a,b)
array([[1, 1, 0, 1],
[0, 1, 1, 1]])
```

Operatory bitowe

bitwise_and	(&)
bitwise_or	(|)
invert	(~)

right_shift(a,shifts)
left_shift(a,shifts)
bitwise_xor

Funkcje trygonometryczne

sin(x) sinh(x)
cos(x) cosh(x)
arccos(x) arccosh(x)
arctan(x) arctanh(x)
arcsin(x) arcsinh(x)
arctan2(x,y)

Inne funkcje

exp(x)	log(x)
log10(x)	sqrt(x)
absolute(x)	conjugate(x)
negative(x)	ceil(x)
floor(x)	fabs(x)
hypot(x,y)	fmod(x,y)
maximum(x,y)	minimum(x,y)

Biblioteka Matplotlib

```
import pylab
```

lub

```
from pylab import *
```

Przykład 1

```
from scipy import *
from pylab import *
x = r_[0:101]
y01 = sin(2*pi*x/100)
y02 = cos(2*pi*x/100)
plot(x,y01,linewidth=5.0)
hold(True)
plot(x,y02,linewidth=5.0)
xlabel('x');ylabel('y)
title('Plottingsin(x) & cos(x)');
legend(('sin(x)','cos(x)'));
grid(True)
```

Przykład 2

```
from pylab import *
from scipy import *
from scipy.fftpackimport fftshift
x=r_[0:101]
y01=sin(2*pi*x/100)
y02=cos(2*pi*x/100)
y03 = randn(100);
y04 = sinc(2*pi*x/100);
Y04 = abs(fftshift(fft(y04)))
y05 = sinc(1.5*pi*x/100);
y06 = sinc(2.5*pi*x/100);
Y06 = abs(fftshift(fft(y06)))
subplot(2,2,1);plot(x,y01,linewidth=3);hold(True);
plot(x,y02,'r',linewidth=3)
grid(True);title('sin(x) & cos(x)');
subplot(2,2,2);plot(y03,linewidth=2);grid(True);title('Random
Numbers');
subplot(2,2,3);plot(x,y04,'k',linewidth=3);grid(True);title('sinc(
x)');
hold(True);
subplot(2,2,3);plot(x,y05,'--',linewidth=3);
subplot(2,2,3);plot(x,y06,'r',linewidth=2);
subplot(2,2,4);plot(Y04,linewidth=2.5);grid(True);title('FFT of
sinc(x)');
show()
```

Przykład 3

```
from scipy import *
from scipy.fftpackimport fftshift
from pylab import *
x,y= meshgrid(r_[-3:3:100j], r_[-3:3:100j]);
z = 3*(1-x)**2*exp(-x**2-(y+1)**2) -10*(x/5-x**3-y**5)*exp(-x**2-
y**2) -(1/3)*exp(-(x+1)**2-y**2);
wav = io.read_array('wavdata');
x1 = r_[1:wav.size+1]
subplot(2,2,1);contour(x,y,z,25);grid(True);title('Simple 2D
Contour Plot');
subplot(2,2,2);contourf(x,y,z);title('Filled Contour');
subplot(2,2,3);h=specgram(wav);title('Spectrogram');
subplot(2,2,4);hist(randn(100));title('Histogram');
show();
```

Literatura

The Python Tutorial, https://docs.python.org/3/tutorial/index.html

Python Tutorial, https://www.w3schools.com/python/

Numpy tutorial, https://numpy.org/devdocs/user/quickstart.html

Scipy tutorial, https://docs.scipy.org/doc/scipy/reference/

Matplotlib, https://matplotlib.org/index.html

The Python Standard Library, https://docs.python.org/3/library/index.html

The Python Language Reference, https://docs.python.org/3/reference/index.html

Zadania

1. Zapoznać się podstawami języka Python, uruchomić i zmodyfikować przykłady z instrukcji.
2. Napisać programy pokazujące działanie operatorów.
3. Uruchomić przykłady 1, 2 , 3 znajdujące się w instrukcji.
4. Napisać program wyznaczający wyznacznik macierzy drugiego i trzeciego stopnia.