Python. Pochodne i całki

B.Kowal

1 lutego 2022

Wagi formuły na różnice centralne

```python
from scipy.misc import central_diff_weights
import numpy as np

print("Wagi dla drugiej pochodnej:")
for i in range(1,4):
  n=2*i+1
  w=central_diff_weights(n,ndiv=2)
  print("Liczba wezlow:",n)
  print("Wagi:",'\n',np.round(w,8),'\n')
```

```
Wagi dla drugiej pochodnej:
Liczba wezlow: 3
Wagi:
 [ 1. -2.  1.]

Liczba wezlow: 5
Wagi:
 [-0.08333333  1.33333333 -2.5         1.33333333 -0.08333333]

Liczba wezlow: 7
Wagi:
 [ 0.01111111 -0.15        1.5        -2.72222222  1.5        -0.15
  0.01111111]
```

Wagi formuły na różnice centralne

-4h	-3h	-2h	-h	0	h	2h	3h	4h
			-1/2	0	1/2			
		1/12	-8/12	0	8/12	-1/12		
	-1/60	9/60	-45/60	0	45/60	-9/60	1/60	
			1	-2	1			
		-1/12	16/12	-30/12	16/12	-1/12		
	-2/180	-27/180	270/180	-490/180	270/180	-27 /180	2/180	
		-1	2	0	-2	1/2		
	1/8	-8/8	13/8	0	-13/8	8/8	-1/8	
-7/240	72/240	-338/240	488/240	0	-488/240	338/240	-72/240	7/240

Analiza błędów formuł na różnice centralne.

Pierwsza pochodna:

▸ 3 węzły ▸ 5 węzłów ▸ 7 węzłów

Druga pochodna:

▸ 3 węzły ▸ 5 węzłów ▸ 7 węzłów

Trzecia pochodna:

▸ 5 węzłów ▸ 7 węzłów ▸ 9 węzłów

numpy.poly1d - wielomian jednej zmiennej, pochodne

```
import numpy as np
#wielomian o współczynnikach 1,3,2,1
f = np.poly1d([1,3,2,1])
print("Wielomian: \n"+ str(f))
fp=f.deriv()
print("Pochodna wielomianu: \n"+ str(fp)+" i jej wartosc w 0:
↪  "+str(fp(0.0)))
fpp=f.deriv(2)
print("Druga pochodna wielomianu: \n"+ str(fpp)+" i jej wartosc w 0:
↪  "+str(fpp(0.0)))
```

```
Wielomian:
   3     2
1 x + 3 x + 2 x + 1
Pochodna wielomianu:
   2
3 x + 6 x + 2 i jej wartosc w 0: 2.0
Druga pochodna wielomianu:

6 x + 6 i jej wartosc w 0: 6.0
```

Metoda całkowania Simpsona

```python
import numpy as np
from scipy.integrate import simps

def f(x):
  return np.sin(x**2+1)
#zakres calkowania:
x1=0.0
x2=1.0
#liczba wezlow:
n=5
x=np.linspace(x1, x2, num=n, endpoint=True)
print("Wezly:",x)
res=simps(f(x),x)
print("Wynik calkowania:",res)
```

```
Wezly: [0.   0.25 0.5  0.75 1.  ]
Wynik calkowania: 0.9285749778105026
```

Współczynniki całkowania metodą Gaussa-Legendre'a

```python
import numpy as np

for n in range(2,6):
  x=np.polynomial.legendre.leggauss(n)[0]
  w=np.polynomial.legendre.leggauss(n)[1]
  print("Liczba wezlow:",n)
  print("Punkty w przedziale [-1,1]:",x)
  print("Wagi:",w,'\n')
```

```
Liczba wezlow: 2
Punkty w przedziale [-1,1]: [-0.57735027  0.57735027]
Wagi: [1. 1.]

Liczba wezlow: 3
Punkty w przedziale [-1,1]: [-0.77459667  0.          0.77459667]
Wagi: [0.55555556 0.88888889 0.55555556]

Liczba wezlow: 4
Punkty w przedziale [-1,1]: [-0.86113631 -0.33998104  0.33998104  0.86113631]
Wagi: [0.34785485 0.65214515 0.65214515 0.34785485]

Liczba wezlow: 5
Punkty w przedziale [-1,1]: [-0.90617985 -0.53846931  0.          0.53846931  0.90617985]
Wagi: [0.23692689 0.47862867 0.56888889 0.47862867 0.23692689]
```

Całka pojedyncza scipy.integrate.quad

```python
import scipy integrate as integrate
import numpy as np

def f(x):
  return np.sin(x)/x

#zakres calkowania x:(0,3)
res= integrate.quad(f, 0.0, 3.0)
print(res)

#zakres calkowania x:(-3,3) bez punktu 0
res2= integrate.quad(f, -3.0, 3.0,points=(0))
print(res2)
```

```
(1.8486525279994683, 2.0524166011165116e-14)
(3.6973050559989367, 4.104833202233023e-14)
```

Całka podwójna scipy.integrate.dblquad

```python
import scipy.integrate as integrate
import numpy as np

def h(x,y):
  return np.cos(x)
def f(x,y):
  return y*x**2

#zakres calkowania y:(0,3) (drugi argument całkowanej funkcji), x:(0,pi/2) (pierwszy argument całkowanej funkcji)
print(integrate.dblquad(h, 0.0, 3.0, 0.0, np.pi/2),'\n')

#zakres calkowania y:(0,1), x:(0,3)
print(integrate.dblquad(f, 0.0, 1.0, 0.0, 3.0))
```

```
(2.9999999999999996, 3.330669073875469e-14)

(4.5, 9.970310392422172e-14)
```

Całka podwójna scipy.integrate.dblquad

```python
import scipy.integrate as integrate
import numpy as np

def h(x,y):
  return np.cos(x)
def f(x,y):
  return y*x**2

#zakres calkowania x:(0,pi/2) (pierwszy argument całkowanej funkcji), y:(0,3) (drugi argument całkowanej funkcji)
print(integrate.dblquad(h, 0.0, 3.0, 0.0, np.pi/2),'\n')

#zakres calkowania x:(0,3), y:(0,1)
print(integrate.dblquad(f, 0.0, 1.0, 0.0, 3.0))
```

```
(2.9999999999999996, 3.330669073875469e-14)

(4.5, 9.970310392422172e-14)
```

Całka podwójna scipy.integrate.dblquad

```
import scipy.integrate as integrate
import numpy as np

def h(x,y):
  return np.cos(x)
def f(x,y):
  return y*x**2

#zakres calkowania y:(0,3) (drugi argument całkowanej funkcji),
↪  x:(0,pi/2) (pierwszy argument całkowanej funkcji)
print(integrate.dblquad(h, 0.0, 3.0, 0.0, np.pi/2),'\n')

#zakres calkowania y:(0,1), x:(0,3)
print(integrate.dblquad(f, 0.0, 1.0, 0.0, 3.0))
```

```
(2.9999999999999996, 3.330669073875469e-14)

(4.5, 9.970310392422172e-14)
```

Całka podwójna scipy.integrate.dblquad

```python
import scipy.integrate as integrate
import numpy as np

def f(x,y):
  return y*x**2

#zakres calkowania
#x:(0,3*y) (pierwszy argument całkowanej funkcji),
#y:(0,1) (drugi argument całkowanej funkcji)

print(integrate.dblquad(f, 0.0, 1.0, 0.0, lambda a : 3*a))
```

```
(1.8000000000000003, 9.905502173200625e-14)
```

Całka podwójna scipy.integrate.dblquad

```
import scipy.integrate as integrate
import numpy as np

def f(x,y):
  return y*x**2

#zakres calkowania
#x:(0,3*y) (pierwszy argument całkowanej funkcji),
#y:(0,1) (drugi argument całkowanej funkcji)

print(integrate.dblquad(f, 0.0, 1.0, 0.0, lambda a : 3*a))
```

```
(1.8000000000000003, 9.905502173200625e-14)
```