Wstęp do SciPy

Janusz Szwabiński

szwabin@ift.uni.wroc.pl

Wstęp do SciPy

- Kilka uwag ogólnych
- Macierze
- Układy równań liniowych
- Równania nieliniowe
- Interpolacja
- Całkowanie numeryczne
- Przykłady (z list zadań)

Wymagania

- gcc/g77
- Python (2.1.x, 2.2x lub 2.3.x) z obsługą biblioteki zlib
- Numerical Python ($\geq$ 23.5)
- biblioteki Atlas i FFTW (wydajność!)
- WxPython, PyGist, Gnuplot
- F2PY

Pierwsze kroki

- zasoby w sieci
  `http://www.scipy.org`
- sposoby importowania

```
>>> from scipy import *
>>> import scipy
```

- pomoc

```
>>> scipy.info(scipy.optimize.fmin)
```

- kod źródłowy funkcji

```
>>> scipy.source(scipy.optimize.fmin)
```

Moduły dostępne w pakiecie

cluster	algorytmy klasteryzacji danych
cow	pakiet do programowania równoległego
fftpack	domyślny pakiet FFT
fftw	pakiet do FFT oparty o bibliotekę FFTW
ga	algorytmy genetyczne
gplt	pakiet do wykresów w Gnuplocie
integrate	całkowanie numeryczne
interpolate	interpolacja
io	funkcje wejścia/wyjścia
linalg	algebra liniowa
optimize	zagadnienia optymalizacji
plt	pakiet do wykresów w WxPython

signal	przetwarzanie sygnałów
special	funkcje specjalne
stats	funkcje statystyczne
xplt	pakiet do wykresów w Gist

Macierze

Tworzenie macierzy

```
>>> A1=scipy.mat('[1 3 5; 2 5 1; 2 3 8]')
>>> A1
Matrix([[1, 3, 5],
        [2, 5, 1],
        [2, 3, 8]])
>>> A2=scipy.mat('[5 3 5; 2 3 1; 4 3 1]')
```

Dodawanie macierzy

```
>>> A1+A2
Matrix([[ 6,  6, 10],
        [ 4,  8,  2],
        [ 6,  6,  9]])
```

Mnożenie przez skalar

```
>>> 2*A1
Matrix([[ 2,  6, 10],
        [ 4, 10,  2],
        [ 4,  6, 16]])
```

Mnożenie macierzy

```
>>> A1*A2
Matrix([[31, 27, 13],
        [24, 24, 16],
        [48, 39, 21]])
```

Macierz odwrotna

```
>>> A = scipy.mat('[1 3 5; 2 5 1; 2 3 8]')
>>> B = A.I
>>> A*B
Matrix([[  1.00000000e+00,   5.55111512e-17,   4.85722573e-17],
        [ -1.38777878e-16,   1.00000000e+00,   7.63278329e-17],
        [ -2.22044605e-16,   5.55111512e-17,   1.00000000e+00]])
>>> C=scipy.linalg.inv(A)
>>> A*C
Matrix([[  1.00000000e-00,  -6.93889390e-17,   1.04083409e-16],
        [ -2.77555756e-16,   1.00000000e+00,   1.31838984e-16],
        [ -3.33066907e-16,   0.00000000e+00,   1.00000000e+00]])
>>> scipy.round(A*C)
array([[ 1.,  -0.,   0.],
       [-0.,   1.,   0.],
       [-0.,   0.,   1.]])
```

Wyznacznik macierzy

```
>>> A = scipy.mat('[1 3 5; 2 5 1; 2 3 8]')
>>> scipy.linalg.det(A)
-25.000000000000004
```

Norma macierzy

```
>>> A = scipy.mat('[1 5 2; 2 4 1; 3 6 2]')
>>> scipy.linalg.norm(A,2)
9.918391609146008
>>> scipy.linalg.norm(A,1)
15
>>> scipy.linalg.norm(A,scipy.inf)
11
```

Wartości i wektory własne

```
>>> A = scipy.mat('[1 5 2; 2 4 1; 3 6 2]')
>>> la,v = scipy.linalg.eig(A)
>>> l1,l2,l3 = la
>>> print l1,l2,l3
(7.95791620491+0j) (-1.25766470568+0j) (0.299748500767+0j)
>>> print v[:,0]
[-0.5297175  -0.44941741  -0.71932146]
>>> print v[:,1]
[-0.90730751  0.28662547  0.30763439]
>>> print v[:,2]
[ 0.28380519 -0.39012063  0.87593408]
```

Rozkład *LU*

```
>>> A = scipy.mat('[1 3 5; 2 5 1; 2 3 8]')
>>> P,L,U = scipy.linalg.lu(A)
>>> P
array([[ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 1.,  0.,  0.]])
>>> L
array([[ 1.  ,  0.  ,  0.  ],
       [ 1.  ,  1.  ,  0.  ],
       [ 0.5 , -0.25,  1.  ]])
>>> U
array([[ 2.  ,  5.  ,  1.  ],
       [ 0.  , -2.  ,  7.  ],
       [ 0.  ,  0.  ,  6.25]])
```

Układy równań liniowych

```
>>> A = scipy.mat('[1 3 5; 2 5 1; 2 3 8]')
>>> b = scipy.mat('[10;8;3]')
>>> A.I*b
Matrix([[-9.28],
        [ 5.16],
        [ 0.76]])
>>> scipy.linalg.solve(A,b) # lepsza metoda
array([[-9.28],
       [ 5.16],
       [ 0.76]])
```

Równania nieliniowe

```
>>> def fun(x):
...     return x**2 - 2
...
>>> scipy.optimize.fsolve(fun,1)
1.4142135623730947
```

Interpolacja

```
>>> x = scipy.arange(0,2*scipy.pi+scipy.pi/4,2*scipy.pi/8)
>>> y = scipy.sin(x)
>>> tck = scipy.interpolate.splrep(x,y,s=0)
>>> xnew = scipy.arange(0,2*scipy.pi,scipy.pi/50)
>>> ynew = scipy.interpolate.splev(xnew,tck,der=0)
>>> scipy.gplt.plot(xnew,ynew,'w l',x,y,'w p')
```

Całkowanie numeryczne

```
>>> def fun(x):
...     return scipy.exp(-x*x)
...
>>> wynik = scipy.integrate.quad(fun,-scipy.Inf,scipy.Inf)[0]
>>> dokladny_wynik = scipy.sqrt(scipy.pi)
>>> wynik-dokladny_wynik
-2.2204460492503131e-16
```

Przykłady (z list zadań)

- Znajdź rozkład LU macierzy

$$\begin{pmatrix} 0 & 1 & 2 & -1 \\ 1 & 2 & 3 & 1 \\ 2 & 4 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{pmatrix}$$

```
>>> import scipy
>>> A=scipy.mat('[0 1 2 -1; 1 2 3 1; 2 4 1 0; -1 1 0 1]')
>>> P,L,U = scipy.linalg.lu(A)
>>> P
array([[ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.],
       [ 0.,  1.,  0.,  0.],
       [ 1.,  0.,  0.,  0.]])
>>> L
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [-0.5       ,  1.        ,  0.        ,  0.        ],
       [ 0.5       ,  0.        ,  1.        ,  0.        ],
       [ 0.        ,  0.33333333,  0.73333333,  1.        ]])
```

```
>>> U
array([[ 2.        ,  4.        ,  1.        ,  0.        ],
       [ 0.        ,  3.        ,  0.5       ,  1.        ],
       [ 0.        ,  0.        ,  2.5       ,  1.        ],
       [ 0.        ,  0.        ,  0.        , -2.06666667]])
```

- Rozwiąż układ równań

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \\ 1 & 16 & 81 & 256 \end{pmatrix} x = \begin{pmatrix} 2 \\ 10 \\ 44 \\ 190 \end{pmatrix}.$$

Oblicz $\|b - \mathbf{A}\tilde{x}\|_\infty$, gdzie $\tilde{x}$ jest rozwiązaniem numerycznym

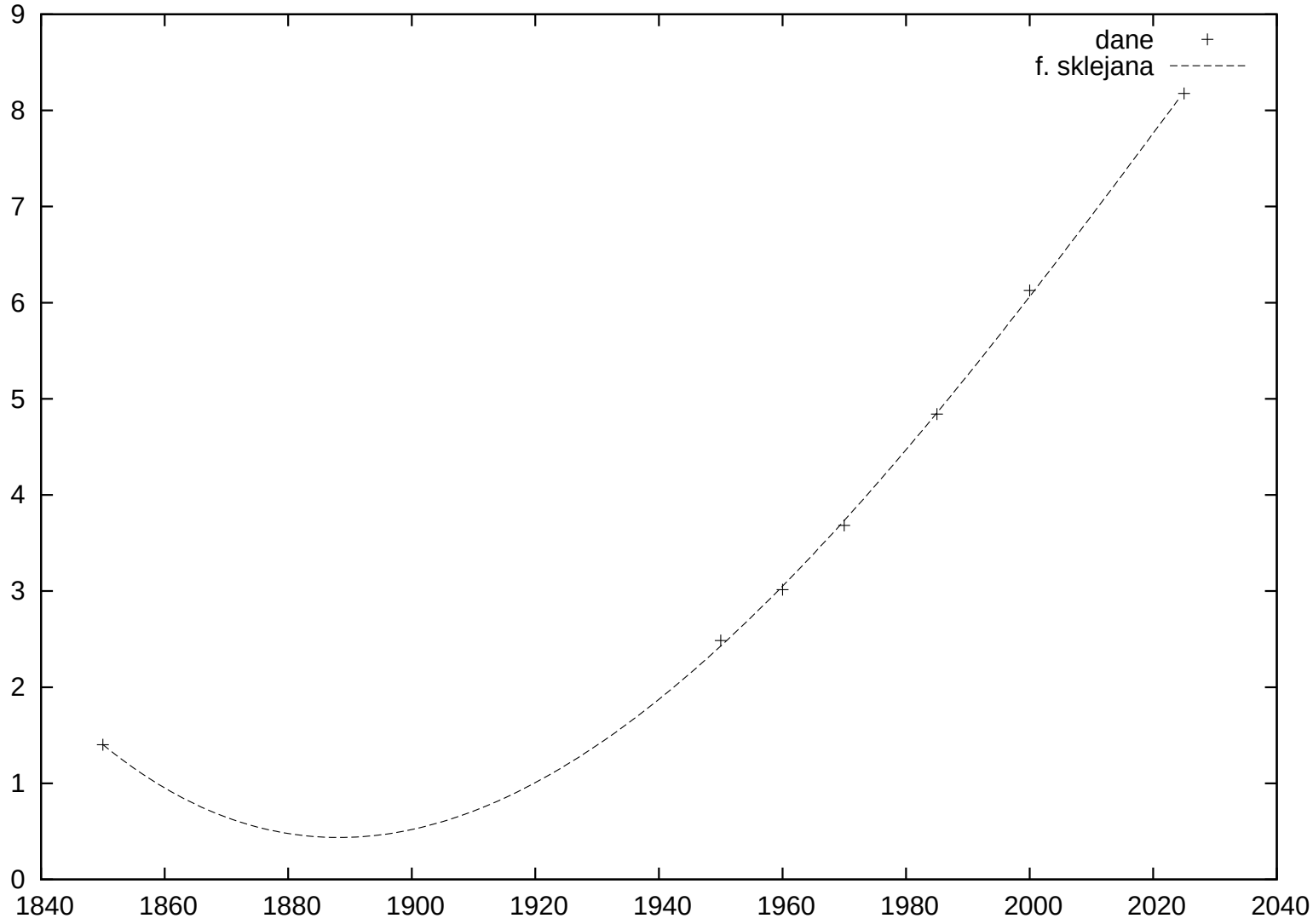
```
>>> import scipy
>>> A = scipy.mat('[1 2 3 4; 1 4 9 16; 1 8 27 64; 1 16 81 256]')
>>> b = scipy.mat('[2; 10; 44; 190]')
>>> res = scipy.linalg.solve(A,b)
>>> res
array([[-1.],
       [ 1.],
       [-1.],
       [ 1.]])
>>> norma = scipy.linalg.norm(b-A*res,scipy.inf)
>>> norma
0.0
```

- W tabeli podana jest liczba ludności na Ziemi w latach 1850-2025 (szacowanie):

rok	1850	1950	1960	1970	1985	2000	2025
ludność (mld)	1,402	2,486	3,014	3,683	4,842	6,127	8,177

Wyznacz interpolującą funkcję sklejaną

```
>>> import scipy
>>> from MyData import *   #patrz python_intro.pdf
>>> dane=Data()
>>> dane.fread('lud.dat')
>>> spline = scipy.interpolate.splrep(dane.x,dane.y)
>>> xnew = scipy.arange(1850,2025,1)
>>> ynew = scipy.interpolate.splev(xnew,spline)
>>> scipy.gplt.plot(xnew,ynew,,dane.x,dane.y)
```


dane
f. sklejana
0
1
2
3
4
5
6
7
8
9
1840
1860
1880
1900
1920
1940
1960
1980
2000
2020
2040

- Oblicz całkę

$$\frac{1}{2\pi}\int_0^{2\pi} \exp\left\{2^{-1/2}\sin(x)\right\}\mathrm{d}x$$

```
>>> import scipy
>>> def fun(x):
...     return scipy.exp(scipy.sin(x)/scipy.sqrt(2))/(2*scipy.pi)
...
>>> calka = scipy.integrate.quad(fun,0,2*scipy.pi)
>>> calka
(1.1289609294541902, 7.4290708898647612e-09)
```

- Ustal punkt, w którym przecinają się krzywe $y = 3x$ oraz $y = e^x$

```
>>> import scipy
>>> def fun(x):
...     return scipy.exp(x)-3*x
...
>>> pp = scipy.optimize.fsolve(fun,1)
>>> pp
0.61906128673594507
>>> fun(pp)
0.0
```

- Znajdź pierwiastki rzeczywiste równania

$$9x^4 + 18x^3 + 38x^2 - 57x + 14 = 0$$

```
>>> import scipy
>>> wiel = scipy.poly1d([9,18,38,-57,14])
>>> print wiel
   4       3       2
9 x + 18 x + 38 x - 57 x + 14
>>> for x in wiel.r:  #wiel.r zawiera wszystkie pierwiastki
...     if x.imag==0: print x
...
(0.666666666667+0j)
(0.333333333333+0j)
```

- Jest dana macierz hermitowska

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

Znajdź wartości i wektory własne macierzy $\mathbf{A}$

```
>>> import scipy
>>> A = scipy.mat('[1 1-1*j 1-2*j 1-3*j; 1+1*j 2 2-1*j 2-2*j;
      1+2*j 2+1*j 3 3-1*j; 1+3*j 2+2*j 3+1*j 4]')
>>> la,v=scipy.linalg.eig(A)
>>> for l in la: print l
...
(10.3993569323-1.29665879217e-17j)
(-1.22156553583+1.36699815578e-16j)
(0.518667170807+1.4608912657e-16j)
(0.303541432686-4.07914215526e-17j)
```