4 O. Dane jest macierz $\mathbf{A} \in \mathbb{R}^{128\times128}$ o następującej strukturze

$$\mathbf{A} = \begin{bmatrix} 4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 1 & 4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \dots \\ 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 & 1 \\ \dots & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ \dots & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 0 \\ \dots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 \\ \dots & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 \end{bmatrix} \qquad (4)$$

Rozwiązać równanie $\mathbf{Ax} = \mathbf{e}$, gdzie $\mathbf{A}$ jest macierzą (4), natomiast $\mathbf{e}$ jest wektorem, którego wszystkie składowe są równe 1, za pomocą

(a) metody Gaussa-Seidela,

(b) metody gradientów sprzężonych.

Algorytmy **muszą** uwzględniać strukturę macierzy (4) — w przeciwnym razie zadanie nie będzie zaliczone!
Oba algorytmy proszę zastartować z tego samego przybliżenia początkowego. Porównać graficznie tempo zbieżności tych metod, to znaczy jak zmieniaję się normy $\|\mathbf{x}_k - \mathbf{x}_{k-1}\|$, gdzie $\mathbf{x}_k$ oznacza k-ty iterat. Porównać efektywną złożoność obliczeniową ze złożonością obliczeniową rozkładu Cholesky'ego dla tej macierzy.

Metoda Gaussa-Seidela

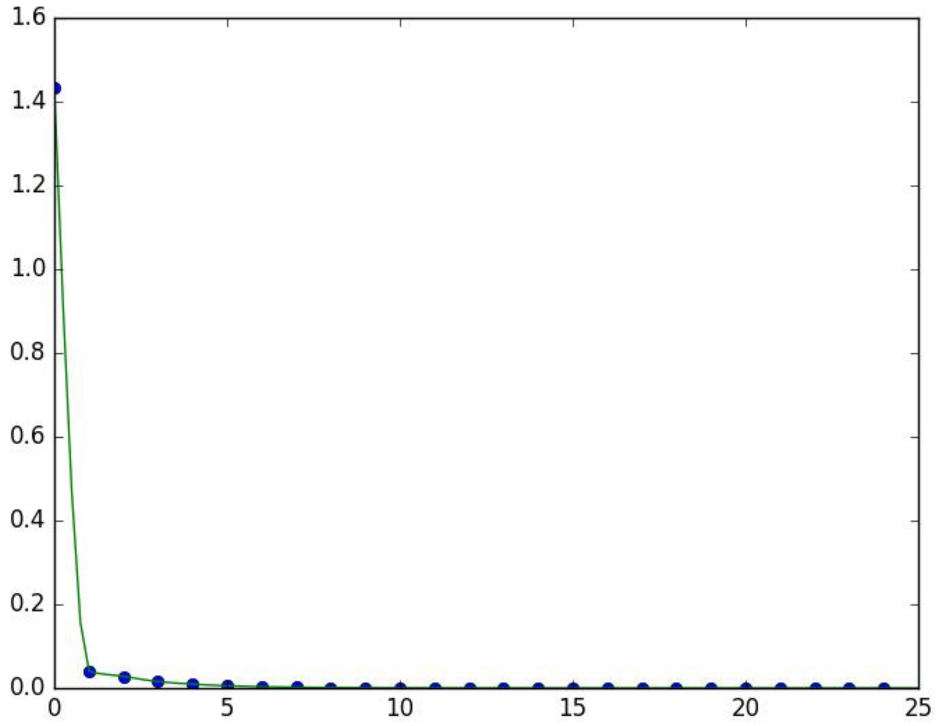
```
import numpy as np
import sys as s
import math as m
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline
n=128
A=np.diag([1.0]*(n-4),-4)+np.diag([1.0]*(n-1),-1)+np.diag([4.0]*n,0)+np.diag([1.0]*(n-1),1)
+np.diag([1.0]*(n-4),4)
b=np.ones(n)
x = np.zeros(n)
x_prev = np.zeros(n)
p=25
fx_p=np.zeros(p)
for k in range(0,p):
    for i in range(0,n):
        s1 = 0.
        s2 = 0.
        if(i==0):
```

```
            s2=(A[i,i+1]*x_prev[i+1])+(A[i,i+4]*x_prev[i+4])
        elif(i>0 and i<4):
            s1=A[i,i-1]*x[i-1]
            s2 = (A[i, i + 1] * x_prev[i + 1]) + (A[i, i + 4] * x_prev[i + 4])
        elif(i>=4 and i<n-4):
            s1=A[i,i-4]*x[i-4]+A[i,i-1]*x[i-1]
            s2 = (A[i, i + 1] * x_prev[i + 1]) + (A[i, i + 4] * x_prev[i + 4])
        elif(i>=n-4 and i<(n-1)):
            s1 = A[i, i - 4] * x[i - 4] + A[i, i - 1] * x[i - 1]
            s2 = A[i, i + 1] * x_prev[i + 1]
        elif(i==(n-1)):
            s1 = A[i, i - 4] * x[i - 4] + A[i, i - 1] * x[i - 1]
        x[i]=(b[i]-s1-s2)/A[i,i]
    norm=0
    norm=np.linalg.norm(x-x_prev)
    fx_p[k]=norm
    x_prev=x.copy()
for i in range(x.size):
    s.stdout.write("x%d="%(i+1))
    print("%.5f" % x[i])

arg=np.arange(0,p,1)
cs=CubicSpline(arg,fx_p)
arg_new=np.linspace(0,p,100)

plt.plot(arg,fx_p,'o')
plt.plot(arg_new,cs(arg_new))
plt.show()
```

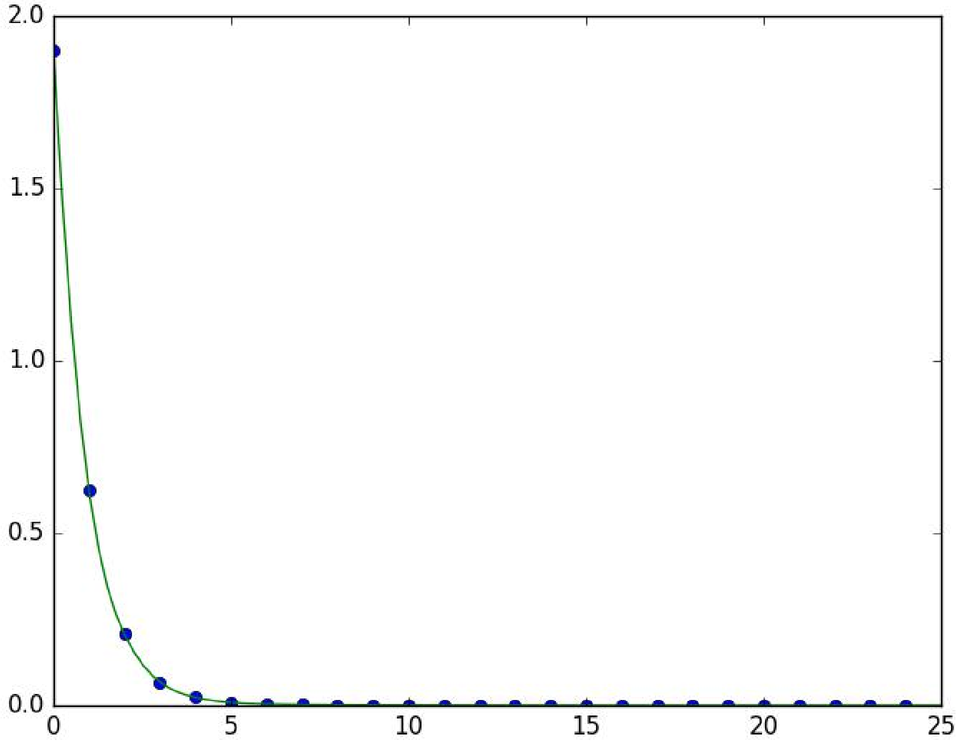
Metoda gradientów sprzężonych

```
import numpy as np
from scipy.sparse.linalg import cg
from scipy.sparse import dia_matrix
import math as m
import matplotlib.pyplot as plt
import sys as s
from scipy.interpolate import PchipInterpolator as Pch
norm= np.zeros(25)
def conjugate_grad(A, b):
    n=len(b)
    x = np.zeros(n)
    r = b-A.dot(x)
    w = - r
    z=A.dot(w)
    a = r.dot(w)/w.dot(z)
    x = x + (a * w)
    x_prev=np.zeros(n)
    for i in range(25):
```

```
        r=r-a*z
        B=(r.dot(z))/(w.dot(z))
        w=-r+(B*w)
        z=A.dot(w)
        a=(r.dot(w))/(w.dot(z))
        x=x+(a*w)
        norm[i]=np.linalg.norm(x-x_prev)
        x_prev=x.copy()
    print(norm)
    return x
n=128
A=dia_matrix(np.diag([1.0]*(n-4),-4)+np.diag([1.0]*(n-1),-1)+np.diag([4.0]*n,0)+np.diag([1.
0]*(n-1),1)+np.diag([1.0]*(n-4),4))
b=np.ones(n)
x=conjugate_grad(A,b)
for i in range(x.size):
    s.stdout.write("x%d="%(i+1))
    print("%.5f" % x[i])
arg=np.arange(0,25,1)
cs=Pch(arg,norm)
arg_new=np.linspace(0,25,100)
plt.plot(arg,norm,'o')
plt.plot(arg_new,cs(arg_new))
plt.show()
```
Wykres norm metody gradientów sprzężonych



Wykres norm metody Gaussa-Seidela



Rozwiązania równania dla obu algorytmów:

x1=0.19428

x2=0.13093

x3=0.14679

x4=0.16231

x5=0.09196

x6=0.13521

x7=0.11958

x8=0.11200

x9=0.14035

x10=0.11670

x11=0.12768

x12=0.12977

x13=0.11793

x14=0.12996

x15=0.12322

x16=0.12332

x17=0.12821

x18=0.12232

x19=0.12617

x20=0.12551

x21=0.12357

x22=0.12638

x23=0.12428

x24=0.12491

x25=0.12562
x26=0.12432
x27=0.12542
x28=0.12497
x29=0.12475
x30=0.12533
x31=0.12477
x32=0.12505
x33=0.12510
x34=0.12484
x35=0.12512
x36=0.12496
x37=0.12497
x38=0.12507
x39=0.12494
x40=0.12503
x41=0.12501
x42=0.12497
x43=0.12503
x44=0.12498
x45=0.12500
x46=0.12501
x47=0.12498
x48=0.12501
x49=0.12500
x50=0.12499
x51=0.12501
x52=0.12499
x53=0.12500
x54=0.12500
x55=0.12500
x56=0.12500
x57=0.12500
x58=0.12500
x59=0.12500
x60=0.12500
x61=0.12500
x62=0.12500
x63=0.12500
x64=0.12500
x65=0.12500
x66=0.12500
x67=0.12500
x68=0.12500
x69=0.12500
x70=0.12500
x71=0.12500
x72=0.12500
x73=0.12500

x74=0.12500
x75=0.12500
x76=0.12500
x77=0.12499
x78=0.12501
x79=0.12499
x80=0.12500
x81=0.12501
x82=0.12498
x83=0.12501
x84=0.12500
x85=0.12498
x86=0.12503
x87=0.12497
x88=0.12501
x89=0.12503
x90=0.12494
x91=0.12507
x92=0.12497
x93=0.12496
x94=0.12512
x95=0.12484
x96=0.12510
x97=0.12505
x98=0.12477
x99=0.12533
x100=0.12475
x101=0.12497
x102=0.12542
x103=0.12432
x104=0.12562
x105=0.12491
x106=0.12428
x107=0.12638
x108=0.12357
x109=0.12551
x110=0.12617
x111=0.12232
x112=0.12821
x113=0.12332
x114=0.12322
x115=0.12996
x116=0.11793
x117=0.12977
x118=0.12768
x119=0.11670
x120=0.14035
x121=0.11200
x122=0.11958

x123=0.13521
x124=0.09196
x125=0.16231
x126=0.14679
x127=0.13093
x128=0.19428