```python
from numpy import linspace, zeros, empty, exp, linalg, dot
from matplotlib.pyplot import style, figure, axes

# Функция подготавливает матрицу СЛАУ
def system_matrix(N,x,h,N_s,s,l_y,l_z) :
    A = zeros((3*N_s,N+1))
    for j in range(0,3*N_s,3) :
        A[j,0] = 1/2*(s[int(j/3)] - x[0])*h/((s[int(j/3)] - x[0])**2 + l_y**2 + l_z**2)**3/2
        for n in range(1,N) :
            A[j,n] = (s[int(j/3)] - x[n])*h/((s[int(j/3)] - x[n])**2 + l_y**2 + l_z**2)**3/2
        A[j,N] = 1/2*(s[int(j/3)] - x[N])*h/((s[int(j/3)] - x[N])**2 + l_y**2 + l_z**2)**3/2
    for j in range(1,3*N_s,3) :
        A[j,0] = 1/2*l_y*h/((s[int((j-1)/3)] - x[0])**2 + l_y**2 + l_z**2)**3/2
        for n in range(1,N) :
            A[j,n] = l_y*h/((s[int((j-1)/3)] - x[n])**2 + l_y**2 + l_z**2)**3/2
        A[j,N] = 1/2*l_y*h/((s[int((j-1)/3)] - x[N])**2 + l_y**2 + l_z**2)**3/2
    for j in range(2,3*N_s,3) :
        A[j,0] = 1/2*l_z*h/((s[int((j-2)/3)] - x[0])**2 + l_y**2 + l_z**2)**3/2
        for n in range(1,N) :
            A[j,n] = l_z*h/((s[int((j-2)/3)] - x[n])**2 + l_y**2 + l_z**2)**3/2
        A[j,N] = 1/2*l_z*h/((s[int((j-2)/3)] - x[N])**2 + l_y**2 + l_z**2)**3/2
    return A

def conjugate_gradient_method(A,M,N,b,x,alpha) :
    # Выделение памяти под вспомогательные массивы
    r = empty(N); p = empty(N); q = empty(N)
    # Инициализация счётчика итераций
    iter = 1
    # Реализация метода сопряжённых градиентов :
    p = 0.
    while iter <= N :
        # Вычисление невязки r
        if iter == 1 :
            r = dot(A.T,dot(A,x) - b) + alpha*x
        else :
            # При iter >= 2 невязка вычисляется по рекурпентной формуле
            ScalP = dot(p,q)
            r = r - 1/ScalP*q
        # Вычисление вспомогательного вектора p
        ScalP = dot(r,r)
        p = p + 1/ScalP*r
        # Вычисление вектора q (основная операция)
        q = dot(A.T,dot(A,p)) + alpha*p
        # Вычисление нового приближения для решения
        ScalP = dot(p,q)
        x = x - 1/ScalP*p
        # Увеличение счётчика итераций на единицу
        iter = iter + 1
    return x

def rho_model(x) :
    return 2*exp(-(x - 0.382)**2/0.009) + 1.2*exp(-(x - 0.618)**2/0.018)

a = 0.; b = 1.

N_model  = 100

h_model = (b - a)/N_model
x_model = linspace(a,b,N_model+1)

X_model = zeros(N_model+1)
for n in range(N_model+1) :
    X_model[n] = rho_model(x_model[n])

c = -1.; d = 2.
l_y = 0.8 ; l_z = 0.2
```

```
N_s = 200
x_s = linspace(c,d,N_s)

A = system_matrix(N_model,x_model,h_model,N_s,x_s,l_y,l_z)

B = dot(A,X_model)

N  = 200
h = (b - a)/N
x = linspace(a,b,N+1)

A = system_matrix(N,x,h,N_s,x_s,l_y,l_z)

alpha = 10**(-13)
X_inv_cgm = zeros(N+1)
X_inv_cgm = conjugate_gradient_method(A,3*N_s,N+1,B,X_inv_cgm,alpha)

# Отрисовка решения
style.use('dark_background')

fig = figure()
ax = axes(xlim=(a,b), ylim=(-0.5,2.5))
ax.set_xlabel('x'); ax.set_ylabel('rho')
ax.plot(x_model,X_model,'-g',lw=2)
ax.plot(x,X_inv_cgm,'-y',lw=2)

# Листинг програмы, реализущей приближённое решение
# интегрального уравнения Фредгольма 1-го рода
# на примере решения обратной задачи электростатики
# (решение ищется с помощью минимизаци целевого функционала
# методом сопряжённых градиентов)
# (с регуляризацией)
```