Федеральное государственное автономное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
"ВЫСШАЯ ШКОЛА ЭКОНОМИКИ"

Московский институт электроники и математики имени А. Н. Тихонова
Программа "Прикладная математика"

Нигматуллин Роман Максимович

Лабораторная работа

Решение систем нелинейных уравнений

3 курс, группа БПМ203

Преподаватель:
Брандышев Петр Евгеньевич

Москва, 2021 г.

Содержание

1 Общие функции для ЛР на Python

Здесь реализованы функции поиска градиента, гессиана, якобиана, также метод Ньютона для поиска корней системы нелинейных уравнений и метод Ньютона для оптимизации нелинейной функции нескольких переменных.

```
import numpy as np
from typing import Callable


def grad(f: Callable[[np.array], np.array],
         x: np.array,
         eps: float = 1e-5) -> np.array:
    dim = len(x)
    grad_vector = np.zeros((dim, ), dtype=np.double)
    for i in range(dim):
        delta = np.zeros(dim)
        delta[i] += eps
        grad_vector[i] = (f(x + delta) - f(x - delta)) / (eps * 2)
    return grad_vector


def hessian(f: Callable[[np.array], np.array],
            x: np.array,
            eps: float = 1e-5) -> np.array:
    dim = len(x)
    hess = np.zeros((dim, dim), dtype=np.double)
    for i in range(dim):
        i_d = np.zeros(dim)
        i_d[i] += eps
        for j in range(dim):
            j_d = np.zeros(dim)
            j_d[j] += eps
            hess[i, j] = (f(x - i_d - j_d) - f(x + i_d - j_d)
                          - f(x - i_d + j_d) + f(x + i_d + j_d)
```

```
                          ) / (4 * eps ** 2)
    return hess


def jacobian(f: Callable[[np.array], np.array],
             x: np.array,
             f_cnt: int,
             eps: float = 1e-5) -> np.array:
    jac = np.zeros((f_cnt, len(x)), dtype=np.double)
    for i in range(len(x)):
        delta = np.zeros(len(x))
        delta[i] += eps
        jac[:, i] = (f(x + delta) - f(x - delta)) / (eps * 2)
    return jac


def root(f: Callable[[np.array], np.array],
           initial: np.array,
           eps: float = 1e-6) -> np.array:
    x = initial.astype(np.double)
    iter_cnt = 0
    f_cnt = len(f(initial))
    while np.linalg.norm(f(x)) > eps:
        x -= np.linalg.inv(jacobian(f, x, f_cnt)).dot(f(x))
        iter_cnt += 1
    return x, iter_cnt


def minimize(f: Callable[[np.array], np.array],
             initial: np.array,
             eps: float = 1e-6) -> np.array:
    x = initial.astype(np.double)
    point_grad = grad(f, x)
    iter_cnt = 0
    while np.linalg.norm(point_grad) > eps:
        x -= np.linalg.inv(hessian(f, x)).dot(point_grad)
        point_grad = grad(f, x)
        iter_cnt += 1
    return x, iter_cnt
```

2 Решение системы нелинейных уравнений методом Ньютона (4.1.16)

2.1 Формулировка задачи

Найти с точностью $\epsilon = 10^{-6}$ все корни системы нелинейных уравнений:

$$f_1(x_1, x_2) = 0$$

$$f_2(x_1, x_2) = 0$$

Использовать метод Ньютона для системы нелинейных уравнений, найти корни с помощью встроенного функционала языка для решения уравнений.

1. Локализовать корни графически, преобразовав уравнения к виду $x_2 = g_i(x_1)$
2. Реализовать метод Ньютона с заданной точностью и подсчетом итераций, для решения линейных систем использовать встроенные методы.
3. Вычислить все корни полученным методом
4. Сравнить результаты с встроенным алгоритмом из пакетов языка

$$sin(0.5x_1 + x_2) - 1.2x_1 - 1 = 0$$

$$x_1^2 + x_2^2 - 1 = 0$$

2.2 Код на Python

```
import numpy as np
import scipy.optimize
import matplotlib.pyplot as plt

from optimize import root


def var_f(x: np.array) -> np.array:
    return np.array([np.sin(0.5 * x[0] + x[1]) - 1.2 * x[0] - 1,
                     x[0]**2 + x[1]**2 - 1], dtype=np.double)


custom_solution_f, iter_cnt_f = root(var_f, np.array([-1, 0]))
custom_solution_s, iter_cnt_s = root(var_f, np.array([1, 2]))
scipy_solution_f = scipy.optimize.fsolve(var_f, np.array([-1, 0]))
scipy_solution_s = scipy.optimize.fsolve(var_f, np.array([1, 2]))


print(f'Newton\'s method first solution: {custom_solution_f}')
print(f'Number of iterations: {iter_cnt_f}')
print(f'Newton\'s method second solution: {custom_solution_s}')
print(f'Number of iterations: {iter_cnt_s}')
print(f'Scipy solutions: {scipy_solution_f}, {scipy_solution_s}')


x, y = np.meshgrid(np.arange(-2, 2, 0.005), np.arange(-2, 2, 0.005))
plt.figure(figsize=(6, 5))
plt.contour(x, y, np.sin(0.5 * x + y) - 1.2 * x - 1, [0], colors=['blue'])
plt.contour(x, y, x**2 + y**2 - 1, [0], colors=['green'])
plt.savefig('plots/nonlinear_newton.png', dpi=300)
plt.show()
```

2.3 Результат работы программы

Программа находит верные корни системы, совпадающие с корнями, найденными встроенным пакетом scipy.optimize, а различия обусловлены машинной точностью и заданным ϵ.

```
Newton's method first solution: [-0.93976819  0.34181244]
Number of iterations: 4
Newton's method second solution: [-0.18601448  0.98254746]
Number of iterations: 5
Scipy solutions: [-0.93976819  0.34181244], [-0.18601531  0.98254685]
```

2.4 График двух нелинейных уравнений из системы

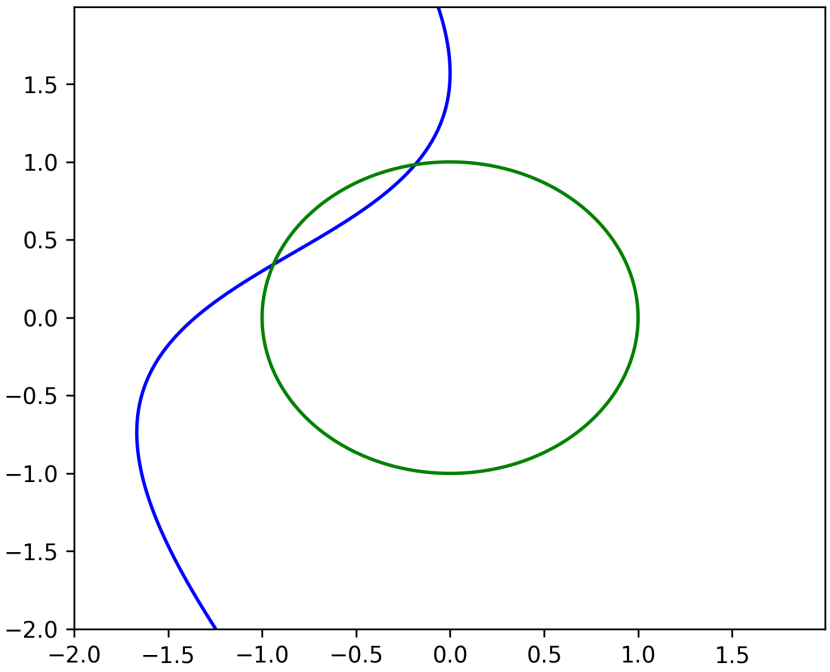

Рис. 1: Два уравнения, пересечения в точках корней системы

3 Ближайшая точка к поверхности (4.5.6)

3.1 Формулировка задачи

Даны координаты точек $P_i, i = 1, 2, 3$ и уравнение поверхности S в пространстве R^3.

1. Определить ближайшую к поверхности точку и наиболее удаленную от поверхности точку.

2. Построить на одном чертеже точечный график поверхности S и заданные точки P_i.

Это можно сделать при помощи минимизации функции H при условии принадлежности к плоскости S, принадлежность которой можно задать через обобщенные координаты.

$$H(x_1, x_2, x_3) = \sum_{k=1}^{3} (x_j - x_j^k)^2$$

$$S : (\frac{x_1}{a_1})^2 + (\frac{x_2}{a_2})^2 + (\frac{x_3}{a_3})^2 = 1$$

$$a_1 = 8.5 - N * 0.25$$

$$a_2 = 2.3 + N * 0.3$$

$$a_3 = 4 - N * 0.1$$

$$P_1 : (14, 8.2, 13.011)$$

$$P_2 : (7.425, 7.532, 9.758)$$

$$P_3 : (13.125, 4.438, 5.75)$$

3.2 Код на Python

```
import numpy as np
import matplotlib.pyplot as plt
from optimize import minimize


def distance_f(p, a1, a2, a3):
    def f(phi):
        z = np.zeros(3, dtype=np.double)
        z[0] = a1 * np.sin(phi[0]) * np.sin(phi[1])
        z[1] = a2 * np.sin(phi[0]) * np.cos(phi[1])
        z[2] = a3 * np.cos(phi[0])
        return np.sum((z - p)**2)
    return f


def inv_transform(a1, a2, a3):
    def inv(phi):
        return np.array([
            a1 * np.sin(phi[0]) * np.sin(phi[1]),
            a2 * np.sin(phi[0]) * np.cos(phi[1]),
            a3 * np.cos(phi[0])
        ])
    return inv
```

```
ps = np.array([
    [14, 8.2, 13.011],
    [7.425, 7.532, 9.758],
    [13.125, 4.438, 5.75]
])
N = 16
a1 = 8.5 - N * 0.25
a2 = 2.3 + N * 0.3
a3 = 4 - N * 0.1
distances = {}

for p in ps:
    solution = None
    res_dist = np.inf
    for i in np.arange(0, np.pi, 0.1):
        angles = np.array([i, i], dtype=np.float16)
        inv = inv_transform(a1, a2, a3)
        dist = distance_f(p, a1, a2, a3)
        solution_angles, iter_cnt = minimize(dist, angles)
        if dist(solution_angles) < res_dist:
            res_dist = dist(solution_angles)
            solution = inv(solution_angles)
    distances[tuple(p)] = (solution, res_dist)


with np.printoptions(precision=4):
    for point, (solution, res_dist) in distances.items():
        print(f'Point {point}:')
        print(f'Distance = {res_dist:.4f}, closest point = {solution}')


fig = plt.figure(figsize=plt.figaspect(1))
ax = fig.add_subplot(projection='3d')

coefs = (a1, a2, a3)
rx, ry, rz = 1 / np.sqrt(coefs)
u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = rx * np.outer(np.cos(u), np.sin(v))
y = ry * np.outer(np.sin(u), np.sin(v))
z = rz * np.outer(np.ones_like(u), np.cos(v))

ax.plot_surface(x, y, z,  rstride=4, cstride=4, color='pink')
ax.scatter(ps.T[0], ps.T[1], ps.T[2], marker='^')
for axis in 'xyz':
    getattr(ax, 'set_{}lim'.format(axis))((-2, 16))
plt.savefig('plots/point_distances.png', dpi=300)
plt.show()
```

3.3 Результат работы программы

Видно, что ближайшей к поверхности оказалась точка P_2, а самой дальней от поверхности - точка P_1.

```
Point (14.0, 8.2, 13.011):
Distance = 277.8194, closest point = [3.3335 3.588  1.0622]
Point (7.425, 7.532, 9.758):
Distance = 105.0394, closest point = [2.6064 4.3222 1.3012]
Point (13.125, 4.438, 5.75):
Distance = 112.4042, closest point = [4.0685 2.3429 0.6515]
```

3.4 Чертеж данной поверхности и точек

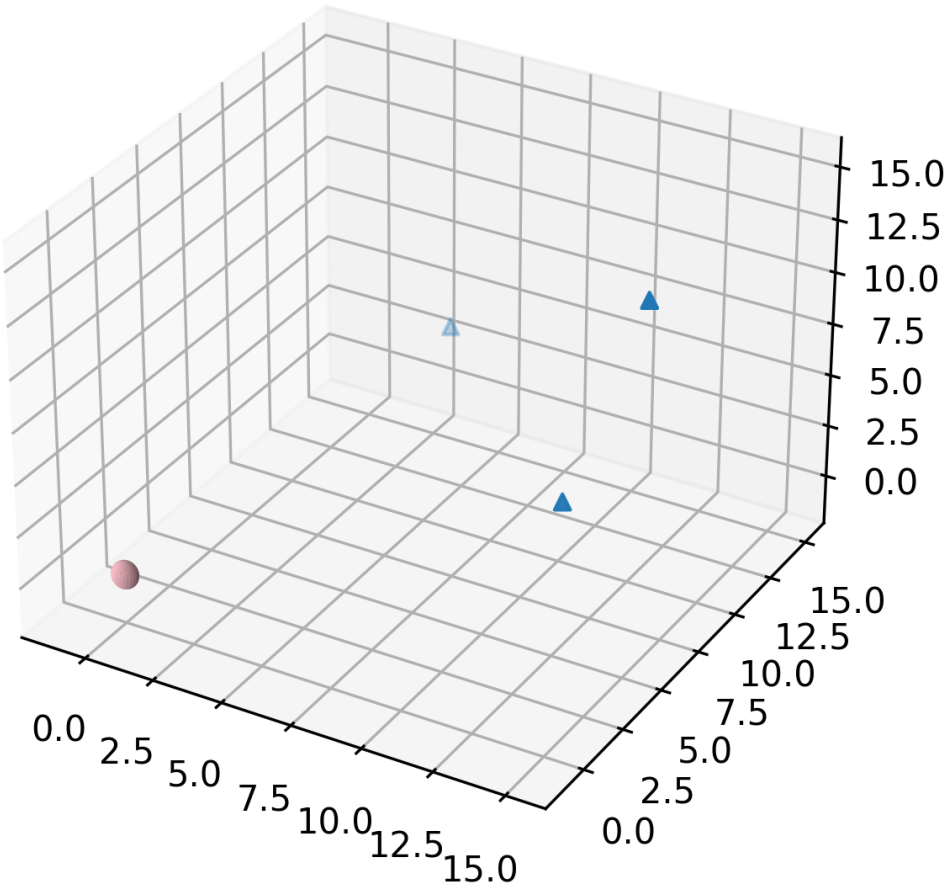

Рис. 2: Розовая - поверхность, синие - точки P_i