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 линейных уравнений итерационными методами

3 курс, группа БПМ203

Преподаватель:
Брандышев Петр Евгеньевич

Москва, 2021 г.

Содержание

1 Общие функции для ЛР на Python **1**

2 Решение СЛАУ методом Гаусса - Зейделя (5.1.16) **2**
2.1 Формулировка задачи 2
2.2 Код на Python 3
2.3 Результат работы программы 4

3 Метод простой итерации (5.3) **4**
3.1 Формулировка задачи 4
3.2 Теоретическая оценка 4
3.3 Код на Python 5
3.4 Результат работы программы 5

4 Метод Зейделя для разреженной матрицы (5.6.4) **5**
4.1 Формулировка задачи 5
4.2 Структура данных разреженной матрицы 6
4.3 Код на Python 9
4.4 Результат работы программы 9

1 Общие функции для ЛР на Python

Здесь реализованы функции решения СЛАУ при помощи метода Гаусса - Зейделя и метода простой итерации.

```python
from typing import Optional, Union
import numpy as np
import numpy.typing as npt

from sparse_matrix import SparseMatrix


def seidel(a: Union[npt.NDArray, SparseMatrix],
           b: npt.NDArray,
           x: Optional[npt.NDArray] = None,
           eps: float = 1e-9,
           max_iter: Optional[int] = None) -> tuple[npt.NDArray, int]:
    """
    Seidel solver for system of linear equations.

    :param a: np array or sparse matrix, matrix form of the system
    :param b: np array, right part vector of the system
    :param x: np array, initial point for solution
    :param eps: float, precision by norm
    :param max_iter: int, number of max iterations
    """
    n = len(a)
    x = x if x is not None else np.zeros(n)
    converge = False
    iter_cnt = 0
```

```
    while not converge:
        x_new = np.copy(x)
        for i in range(n):
            s1 = sum(a[i, j] * x_new[j] for j in range(i))
            s2 = sum(a[i, j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / a[i, i]
        iter_cnt += 1
        converge = np.linalg.norm(x_new - x, ord=np.inf) <= eps
        if max_iter:
            converge = iter_cnt >= max_iter
        x = x_new
    return x, iter_cnt


def simple_iterative(a: Union[npt.NDArray, SparseMatrix],
                     b: npt.NDArray,
                     x: Optional[npt.NDArray] = None,
                     eps: float = 1e-9,
                     max_iter: Optional[int] = None) -> tuple[npt.NDArray, int]:
    x = x if x is not None else np.zeros(len(a))
    L = np.tril(a, -1)
    U = np.triu(a, 1)
    D = np.diag(np.diag(a))
    B = np.linalg.inv(L + D).dot(-U)
    c = np.linalg.inv(L + D).dot(b)
    x_new = x.copy()
    for i in range(max_iter):
        x_new = x.copy()
        x_new = B.dot(x_new) + c
        if np.linalg.norm(x - x_new, ord=np.inf) <= eps:
            return x_new, i
        x = x_new
    return x_new, max_iter
```

2 Решение СЛАУ методом Гаусса - Зейделя (5.1.16)

2.1 Формулировка задачи

Дана система уравнений $Ax = b$. Найти решение системы с помощью метода Гаусса. Выполнить 10 итераций по методу Зейделя. Принимая решение, полученное с помощью метода Гаусса за точное, найти величину абсолютной погрешности итерационного решения.

1. Задать матрицу системы A и вектор правой части b. Используя встроенную функцию решения пакетов языка, найти решение системы $Ax = b$ с помощью метода Гаусса.

2. Преобразовать систему $Ax = b$ к виду $x = Bx + c$, удобному для итераций. Проверить выполнение достаточного условия сходимости итерационных методов $||B||_\infty < 1$.

3. Написать программу, решающую систему линейных уравнений с помощью метода Зейделя, выполнить 10 итераций по методу Зейделя; взять любое начальное приближение. Предварительно проверить условие сходимости для метода Зейделя. Принимая решение, полученное в п. 1 за точное, найти величину абсолютной погрешности итерационного решения (использовать норму $||...||_\infty$.).

4. Взять другое начальное приближение. Объяснить полученные результаты.

$$b = \begin{pmatrix} -468.1 & 122.3 & -257.2 & -223.6 & 35.9 \end{pmatrix}$$

$$A = \begin{pmatrix} 79.2 & 0 & 35 & 19.8 & 24 \\ 39.6 & 85 & 0 & 19.8 & 25 \\ 19.8 & -15 & 45 & 0 & 10 \\ 49.5 & 18 & 20 & 89.1 & 0 \\ 9.9 & 15 & 20 & -49.5 & 95 \end{pmatrix}$$

2.2 Код на Python

```python
import numpy as np
from solve import seidel

A = np.array([
    [79.2, 0, 35, 19.8, 24],
    [39.6, 85, 0, 19.8, 25],
    [19.8, -15, 45, 0, 10],
    [49.5, 18, 20, 89.1, 0],
    [9.9, 15, 20, -49.5, 95],
])
b = np.array([-468.1, 122.3, -257.2, -223.6, 35.9])
n = 5

L = np.tril(A, -1)
U = np.triu(A, 1)
D = np.diag(np.diag(A))
B = np.linalg.inv(L + D).dot(-U)

np_solution = np.linalg.solve(A, b)
print(f"Numpy solution: {np_solution}\n")

converge = np.linalg.norm(B, ord=np.inf) < 1
print(f"System convergence with Seidel iterative method: {converge}")
if converge:
    seidel_solution_1, _ = seidel(A, b, x=np.zeros(n), max_iter=10)
    seidel_solution_2, _ = seidel(A, b, x=np.ones(n), max_iter=10)
    eps_1 = np.linalg.norm(np_solution - seidel_solution_1, ord=np.inf)
    eps_2 = np.linalg.norm(np_solution - seidel_solution_2, ord=np.inf)

    print(f"x = {np.zeros(n)}")
    print("Seidel method solution:")
    print(seidel_solution_1)
```

```
print(f"Solution precision by infinity norm: {eps_1:.6f}")
print(f"x = {np.ones(n)}")
print("Seidel method solution:")
print(seidel_solution_2)
print(f"Solution precision by infinity norm: {eps_2:.6f}")
```

2.3 Результат работы программы

Программа находит верные решения системы, совпадающие с корнями, найденными встроенным пакетом numpy.linalg, а различия обусловлены машинной точностью при вычислениях и заданным числом итераций. Так как норма матрицы B меньше единицы, то отображение является сжимающим и итерационный метод сходится вне зависимости от начального приближения, но с разной точностью за 10 итераций.

```
Numpy solution: [-5.16161616  3.5        -2.5         0.21212121  1.        ]

System convergence with Seidel iterative method: True
x = [0. 0. 0. 0. 0.]
Seidel method solution:
[-5.16167521  3.49996837 -2.50000698  0.21216197  1.00003386]
Solution precision by infinity norm: 0.000059
x = [1. 1. 1. 1. 1.]
Seidel method solution:
[-5.16168826  3.49994571 -2.50002082  0.21217691  1.00004949]
Solution precision by infinity norm: 0.000072
```

3 Метод простой итерации (5.3)

3.1 Формулировка задачи

Для системы уравнений $Ax = b$ из задачи № 1 (5.1) выполнить 10 итераций по методу простой итерации. Оценить абсолютную погрешность полученного решения теоретически. Найти реальную величину абсолютной погрешности, приняв за точное решение - решение, полученное с помощью встроенной функции пакетов языка. Объяснить результаты.

3.2 Теоретическая оценка

Так как это сжимающее отображение, сведенное к виду $x = Bx+c$, то можно оценить погрешность как:

$$||x - x_k|| \leq ||B||^k ||x_0|| + \frac{||B||^k}{1 - ||B||}||c||$$

$$||B||_\infty = 0.9949, ||c||_\infty = 5.91, ||x_0||_\infty = 0$$

$$||x - x_{10}|| \leq 0.9949^{10} \times 0 + \frac{0.9949^{10}}{1 - 0.9949} \times 5.91$$

$$||x - x_{10}|| \leq 1101.06$$

Это достаточно высокая оценка погрешности, так как при таком коэффициенте сжимающего отображения за 10 итераций и таких условиях верхняя граница из-за большой нормы вектора велика.

3.3 Код на Python

```
import numpy as np
from solve import simple_iterative

A = np.array([
    [79.2, 0, 35, 19.8, 24],
    [39.6, 85, 0, 19.8, 25],
    [19.8, -15, 45, 0, 10],
    [49.5, 18, 20, 89.1, 0],
    [9.9, 15, 20, -49.5, 95],
])
b = np.array([-468.1, 122.3, -257.2, -223.6, 35.9])
n = 5


np_solution = np.linalg.solve(A, b)
print("Numpy solution:")
print(np_solution)

si_solution, iter_cnt = simple_iterative(A, b, max_iter=10)
eps = np.linalg.norm(np_solution - si_solution, ord=np.inf)
print("Simple Iterative solution:")
print(si_solution)
print(f"Precision: {eps:.6f}")
```

3.4 Результат работы программы

Решение сходится за 10 итераций с заданной точностью к решению встроенными методами пакета numpy.linalg, как и должно.

```
Numpy solution:
[-5.16161616  3.5        -2.5         0.21212121  1.        ]
Simple Iterative solution:
[-5.16167521  3.49996837 -2.50000698  0.21216197  1.00003386]
Precision: 0.000059
```

4 Метод Зейделя для разреженной матрицы (5.6.4)

4.1 Формулировка задачи

Дана система уравнений $Ax = b$, где A – симметричная положительно определенная разреженная матрица размерности $N * N$. Методом Зейделя найти решение системы с точностью $\epsilon = 10^{-9}$. Определить число итераций, потребовавшихся для достижения заданной точности. Матрицы, в которых большинство элементов равно нулю, называются разреженными. Компактное хранение элементов матрицы A в памяти ЭВМ организовать с использованием одномерных массивов.

4.2 Структура данных разреженной матрицы

Реализована на основе словаря с ключами по индексам элементов структура данных для хранения разреженной матрицы, многократно экономящая память ЭВМ.

```python
import math
import numpy as np
from typing import Dict, Tuple


class SparseMatrix:
    """Sparse matrix class"""

    def __init__(self,
                 data: Dict[Tuple[int, int], float] = None,
                 n: int = 2):
        """
        :param data: A dictionary of (i, j) -> value
        """
        self.data = data or {}
        self.n = n

    def from_dense_matrix(self, matrix) -> "SparseMatrix":
        self.n = len(matrix)
        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                if matrix[i][j] != 0:
                    self[(i, j)] = matrix[i][j]
        return self

    def dense(self) -> "SparseMatrix":
        res = np.zeros((self.n, self.n))
        for (i, j), v in self.data.items():
            res[i, j] = v
        return res

    def __getitem__(self, key: Tuple[int, int]) -> float:
        return self.data.get(key, 0)

    def __setitem__(self, key: Tuple[int, int], value: float):
        self.data[key] = value

    def __delitem__(self, key: Tuple[int, int]):
        del self.data[key]

    def __iter__(self):
        return iter(self.data.items())

    def __len__(self):
        return self.n
```

```python
def __str__(self):
    return str(self.data)

def __repr__(self):
    return repr(self.data)

def __add__(self, other: "SparseMatrix") -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] += self[key]
    for key in other:
        result[key] += other[key]
    return result

def __sub__(self, other: "SparseMatrix") -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = self[key]
    for key in other:
        result[key] -= other[key]
    return result

def __mul__(self, other: "SparseMatrix") -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        for key2 in other:
            result[(key[0], key2[1])] += self[key] * other[key2]
    return result

def __rmul__(self, other: float) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = self[key] * other
    return result

def __truediv__(self, other: float) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = self[key] / other
    return result

def __neg__(self) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = -self[key]
    return result

def __pow__(self, other: float) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
```

```
        result[key] = self[key] ** other
    return result

def __rpow__(self, other: float) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = other ** self[key]
    return result

def __eq__(self, other: "SparseMatrix") -> bool:
    return self.data == other.data

def __ne__(self, other: "SparseMatrix") -> bool:
    return self.data != other.data

def __lt__(self, other: "SparseMatrix") -> bool:
    return self.data < other.data

def __le__(self, other: "SparseMatrix") -> bool:
    return self.data <= other.data

def __gt__(self, other: "SparseMatrix") -> bool:
    return self.data > other.data

def __ge__(self, other: "SparseMatrix") -> bool:
    return self.data >= other.data

def __abs__(self) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = abs(self[key])
    return result

def __round__(self) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = round(self[key])
    return result

def __floor__(self) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = math.floor(self[key])
    return result

def __ceil__(self) -> "SparseMatrix":
    result = SparseMatrix()
    for key in self:
        result[key] = math.ceil(self[key])
    return result
```

```python
    def __trunc__(self) -> "SparseMatrix":
        result = SparseMatrix()
        for key in self:
            result[key] = math.trunc(self[key])
        return result
```

4.3 Код на Python

```python
import numpy as np

from sparse_matrix import SparseMatrix
from solve import seidel


A = SparseMatrix(n=50)
b = np.zeros(50)
for i in range(50):
    A[i, i] = 100
    j = i + 1
    b[i] = j * np.exp(10 / j) * np.cos(9 / j)
for i in range(50 - 1):
    A[i, i + 1] = 27
for i in range(50 - 3):
    A[i, i + 3] = 15
for i in range(50 - 7):
    A[i, i + 7] = 1

np_solution = np.linalg.solve(A.dense(), b)
seidel_solution, iter_cnt = seidel(A, b, eps=1e-9)
eps = np.linalg.norm(np_solution - seidel_solution, ord=np.inf)

print("Numpy solution:")
print(np_solution)

print("\nSeidel method solution")
print(seidel_solution)
print(f"Iterations count = {iter_cnt}")
print(f"Solution precision by infinity norm: {eps:.10f}")
```

4.4 Результат работы программы

Решение сходится за 25 итераций с заданной точностью к решению встроенными методами пакета numpy.linalg, как и должно.

```
Numpy solution:
[-2.00536996e+02 -4.08531015e-01 -7.55047636e-01 -2.88261628e-01
 -9.50545518e-02 -5.13636347e-03  4.38024409e-02  7.42724568e-02
  9.54649068e-02  1.11578106e-01  1.24713624e-01  1.36005027e-01
  1.46101302e-01  1.55394210e-01  1.64128831e-01  1.72466428e-01
```

```
  1.80512895e-01  1.88344822e-01  1.96011063e-01  2.03554488e-01
  2.10995297e-01  2.18365049e-01  2.25663461e-01  2.32925789e-01
  2.40127453e-01  2.47325045e-01  2.54452103e-01  2.61618723e-01
  2.68675892e-01  2.75850133e-01  2.82815038e-01  2.90047512e-01
  2.96861089e-01  3.04280907e-01  3.10817917e-01  3.18589313e-01
  3.24448525e-01  3.33107448e-01  3.38013621e-01  3.48599638e-01
  3.49940344e-01  3.63253985e-01  3.60197144e-01  3.86828188e-01
  3.72846537e-01  4.00625586e-01  3.55106165e-01  4.65095477e-01
  4.28601688e-01  6.00834700e-01]

Seidel method solution
[-2.00536996e+02 -4.08531015e-01 -7.55047636e-01 -2.88261628e-01
 -9.50545517e-02 -5.13636336e-03  4.38024411e-02  7.42724569e-02
  9.54649069e-02  1.11578106e-01  1.24713624e-01  1.36005027e-01
  1.46101302e-01  1.55394210e-01  1.64128831e-01  1.72466428e-01
  1.80512895e-01  1.88344822e-01  1.96011063e-01  2.03554488e-01
  2.10995297e-01  2.18365049e-01  2.25663461e-01  2.32925789e-01
  2.40127453e-01  2.47325045e-01  2.54452103e-01  2.61618723e-01
  2.68675892e-01  2.75850133e-01  2.82815038e-01  2.90047512e-01
  2.96861089e-01  3.04280907e-01  3.10817917e-01  3.18589313e-01
  3.24448525e-01  3.33107448e-01  3.38013621e-01  3.48599638e-01
  3.49940344e-01  3.63253985e-01  3.60197144e-01  3.86828188e-01
  3.72846537e-01  4.00625586e-01  3.55106165e-01  4.65095477e-01
  4.28601688e-01  6.00834700e-01]
Iterations count = 25
Solution precision by infinity norm: 0.0000000002
```