БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
Факультет прикладной математики и информатики

**Лабораторная работа №3**  
по теме  
«Итерационные методы решения системы линейных алгебраических уравнений»

Выполнила   
Молочко Екатерина  
2 курс 7 группа

Преподаватель  
Горбачева Юлия Николаевна

Минск   
2021

## Постановка задачи:

1. Написать и отладить программу численного решения систем линейных алгебраических квадратной матрицей порядка *N* методом минимальных невязок
2. Написать и отладить программу численного решения систем линейных алгебраических квадратной матрицей порядка *N* методом релаксации
3. Провести вычислительный эксперимент решения СЛАУ с симметрическими положительно определенными матрицами с диагональным преобладанием методом релаксации, с различными параметрами релаксации

## Краткие теоретические сведения

### Метод минимальных невязок

Рассмотрим СЛАУ

А - симметрическая положительно определенная матрица с диагональным преобладанием

X – вектор неизвестных

F – столбец свободных членов

Метод минимальных невязок сходится при любом начальном приближении, если матрица положительно определенная и симметрическая

Рассмотрим переменные на k-ом шаге

- вектор невязки

- итерационный параметр

– приближенное решение

### Метод релаксации

Если матрица A симметрическая и положительно определённая  
 , то метод релаксации сходится при любом начальном приближении. При метод Гаусса Зейделя сходится.

– параметр релаксации

При = 1 - полная релаксация

В противном случае неполная релаксация

При < 1 - нижняя релаксация

При > 1 – верхняя релаксация

## Листинг программы

### Метод минимальных невязок

**import** numpy **as** np

**import** sys

EPS **=** 10 **\*\*** **-**7

K\_MAX **=** 5000

# Функция для соблюдения условия диагонального преобладания

**def** calculating\_sums**(**matrix**):**

**sum** **=** 0

**for** i **in** **range(len(**matrix**)):**

**for** j **in** **range(len(**matrix**[**i**])):**

**if** **(**j **!=** i**):**

**sum** **+=** np**.abs(**matrix**[**i**][**j**])**

A**[**i**][**i**]** **=** np**.**random**.**randint**(sum** **+** 7**,** **sum** **+** 7 **\*** 10**)**

# Функция для подсчета суммы с переменными на k+1 шаге

**def** calculating\_next **(**A**,** approx**,** i**):**

**sum** **=** 0

**for** j **in** **range(**i**):**

**sum** **+=** A**[**i**][**j**]\***approx**[**j**]**

**return** **sum**

# Функция для подсчета суммы с переменными на k шаге

**def** calculating\_present**(**A**,** approx**,** i**,** N**):**

**sum** **=** 0

**for** j **in** **range(**i **+** 1**,** N**):**

**sum** **+=** A**[**i**][**j**]\***approx**[**j**]**

**return** **sum**

# Реализация метода минимальных невязок

**def** minimal\_residual\_method**(**A**,** F**,** approx**,** X**):**

A**.**astype**(float)**

K **=** 0

**print(**"\nMinimal residual method:\n "**)**

**while** **(**K **<=** K\_MAX**):**

K **+=** 1

residual **=** np**.**matmul**(**A**,** approx**)** **-** F

scalar\_product **=** np**.**dot**(**np**.**matmul**(**A**,** residual**),** residual**)** **/** np**.**dot**(**np**.**matmul**(**A**,** residual**),**

np**.**matmul**(**A**,** residual**))**

approx **=** approx **-** scalar\_product **\*** residual

crit **=** np**.abs(**np**.**matmul**(**A**,** approx**)** **-** F**)**

index **=** np**.**argmax**(**crit**)**

**if** **(**crit**[**index**]** **<** EPS**):**

**print(**"\nNumber of iterations is:"**,** K**)**

**print(**"\nReal unknowns is: "**,** X**)**

**print(**"\n||AX\_ - F|| = "**,** crit**[**index**])**

format\_string **=** "{:.16f}"

**print(**"\nVector of computed terms is:"**)**

**for** i **in** approx**:**

**print(**format\_string**.format(**i**))**

sub **=** np**.abs(**X **-** approx**)**

index\_ **=** np**.**argmax**(**sub**)**

**print(**"\nAbsolute error is: "**,** sub**[**index\_**])**

**return**

**if** **(**K **>** K\_MAX**):**

**print(**"\nToo much iterations\n"**)**

sys**.exit()**

# Генерация симметрической положительно определённой матрицы с диагональным преобладанием

N **=** **int(input(**'Input degree of a square matrix: '**))**

A **=** np**.**full**((**N**,** N**),** 0**)**

**for** i **in** **range(len(**A**)):**

**for** j **in** **range(len(**A**[**i**])):**

**if** **(**i **>** j**):**

A**[**i**][**j**]** **=** np**.**random**.**randint**(-**100**,** 100**)**

A**[**j**][**i**]** **=** A**[**i**][**j**]**

calculating\_sums**(**A**)**

**print(**"\nSymmetric matrix with diagonal predominance:\n"**,** A**)**

# Генерация вектора неизвестных X = (1, 2, ... , N)^T

X **=** np**.**array**([**0**]** **\*** N**)**

**for** i **in** **range(len(**X**)):**

X**[**i**]** **+=** i **+** 1

**print(**'\nVector of real unknowns: \n'**,** X**)**

# Генрация столбца свободных членов F, путём умножения симметрической положительно определенной матрицы

# На вектор неизвестных X = (1, 2, ... , N)^T

F **=** np**.**matmul**(**A**,** X**)**

**print(**"\n Constant terms column:\n"**,** F**)**

# начальное приближение X0 = (0, 0, ... , 0)

init\_approx **=** np**.**array**([**0.0**]** **\*** N**).**astype**(float)**

**print(**"Initial approximation: "**,** init\_approx**)**

# Итерационный процесс

K **=** 0

Q **=** 0

approx **=** np**.**copy**(**init\_approx**)**

minimal\_residual\_method**(**A**,** F**,** approx**,** X**)**

### Метод релаксации

**import** numpy **as** np

**import** sys

EPS **=** 10 **\*\*** **-**7

K\_MAX **=** 5000

# Функция для соблюдения условия диагонального преобладания

**def** calculating\_sums**(**matrix**):**

**sum** **=** 0

**for** i **in** **range(len(**matrix**)):**

**for** j **in** **range(len(**matrix**[**i**])):**

**if** **(**j **!=** i**):**

**sum** **+=** np**.abs(**matrix**[**i**][**j**])**

A**[**i**][**i**]** **=** np**.**random**.**randint**(sum** **+** 7**,** **sum** **+** 7 **\*** 10**)**

# Функция для подсчета суммы с переменными на k+1 шаге

**def** calculating\_next **(**A**,** approx**,** i**):**

**sum** **=** 0

**for** j **in** **range(**i**):**

**sum** **+=** A**[**i**][**j**]\***approx**[**j**]**

**return** **sum**

# Функция для подсчета суммы с переменными на k шаге

**def** calculating\_present**(**A**,** approx**,** i**,** N**):**

**sum** **=** 0

**for** j **in** **range(**i **+** 1**,** N**):**

**sum** **+=** A**[**i**][**j**]\***approx**[**j**]**

**return** **sum**

# Реализация метода релаксации

**def** relaxation\_method**(**A**,** approx**,** F**,** W**,** N**):**

A**.**astype**(float)**

K **=** 0

**print(**"\nRelaxation method: "**)**

**while(**K **<=** K\_MAX**):**

K **+=** 1

**for** i **in** **range(len(**approx**)):**

approx**[**i**]** **=** **(**1 **-** W**)** **\*** approx**[**i**]** **+** W**/**A**[**i**][**i**]** **\*** **(**F**[**i**]** **-** calculating\_next**(**A**,** approx**,** i**)** **-** calculating\_present**(**A**,** approx**,** i**,** N**))**

crit **=** np**.abs(**np**.**matmul**(**A**,** approx**)** **-** F**)**

index **=** np**.**argmax**(**crit**)**

**if** **(**crit**[**index**]** **<** EPS**):**

**print(**"\nNumber of iterations is:"**,** K**)**

**print(**"\nReal unknowns is: "**,** X**)**

**print(**"\n||AX\_ - F|| = "**,** crit**[**index**])**

format\_string **=** "{:.16f}"

**print(**"\nVector of computed terms is:"**)**

**for** i **in** approx**:**

**print(**format\_string**.format(**i**))**

sub **=** np**.abs(**X **-** approx**)**

index\_ **=** np**.**argmax**(**sub**)**

**print(**"\nAbsolute error is: "**,** sub**[**index\_**])**

**return**

**if** **(**K **>** K\_MAX**):**

**print(**"\nToo much iterations\n"**)**

sys**.exit()**

# Генерация симмметрической положительно определённой матрицы с диагональным преобладанием

N **=** **int(input(**'Input degree of a square matrix: '**))**

A **=** np**.**full**((**N**,** N**),** 0**)**

**for** i **in** **range(len(**A**)):**

**for** j **in** **range(len(**A**[**i**])):**

**if** **(**i **>** j**):**

A**[**i**][**j**]** **=** np**.**random**.**randint**(-**100**,** 100**)**

A**[**j**][**i**]** **=** A**[**i**][**j**]**

calculating\_sums**(**A**)**

**print(**"\nSymmetric matrix with diagonal predominance:\n"**,** A**)**

# Генерация вектора неизвестных X = (1, 2, ... , N)^T

X **=** np**.**array**([**0**]** **\*** N**)**

**for** i **in** **range(len(**X**)):**

X**[**i**]** **+=** i **+** 1

**print(**'\nVector of real unknowns: \n'**,** X**)**

# Генрация столбца свободных членов F, путём умножения симметрической положительно определенной матрицы

# На вектор неизвестных X = (1, 2, ... , N)^T

F **=** np**.**matmul**(**A**,** X**)**

**print(**"\n Constant terms column:\n"**,** F**)**

# начальное приближение X0 = (0, 0, ... , 0)

init\_approx **=** np**.**array**([**0.0**]** **\*** N**).**astype**(float)**

**print(**"Initial approximation: "**,** init\_approx**)**

# Итерационный процесс

K **=** 0

Q **=** 0

approx **=** np**.**copy**(**init\_approx**)**

# Параметры релаксации

W **=** 0.2

relaxation\_method**(**A**,**approx**,** F**,** W**,** N**)**

## Результаты вычислительного эксперимента

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр ω | Номер итерации q, при которой достигнута требуемая точность |  |  |
| 0.2 | 145 | 9.552968549542129e-08 | 1.639071101067202e-10 |
| 0.5 | 45 | 8.680535756866448e-08 | 2.6717095202855035e-10 |
| 0.8 | 20 | 7.998642104212195e-08 | 1.608884137027644e-10 |
| 1 | 8 | 5.8234945754520595e-08 | 9.148615198739662e-11 |
| 1.2 | 18 | 2.773595042526722e-08 | 1.6637802247032596e-11 |
| 1.5 | 40 | 6.009213393554091e-08 | 1.8512635868717098e-11 |
| 1.8 | 121 | 8.224014891311526e-08 | 3.33129079876926e-11 |

## Решение одной из систем обоими методами

Input degree of a square matrix**:** 10

Symmetric matrix **with** diagonal predominance**:**

**[[** 508 75 **-**33 **-**88 **-**19 71 25 80 81 27**]**

**[** 75 1043 **-**86 **-**83 5 95 59 **-**44 9 39**]**

**[** **-**33 **-**86 1482 4 **-**5 **-**37 **-**63 69 70 **-**63**]**

**[** **-**88 **-**83 4 1947 **-**56 35 9 77 **-**83 **-**37**]**

**[** **-**19 5 **-**5 **-**56 2358 **-**48 96 **-**58 50 **-**79**]**

**[** 71 95 **-**37 35 **-**48 2904 **-**79 59 59 **-**86**]**

**[** 25 59 **-**63 9 96 **-**79 3477 **-**69 **-**64 **-**93**]**

**[** 80 **-**44 69 77 **-**58 59 **-**69 3947 30 12**]**

**[** 81 9 70 **-**83 50 59 **-**64 30 4492 70**]**

**[** 27 39 **-**63 **-**37 **-**79 **-**86 **-**93 12 70 4982**]]**

Vector of real unknowns**:**

**[** 1 2 3 4 5 6 7 8 9 10**]**

Constant terms column**:**

**[** 2352 2698 4121 7038 11122 17064 22277 32054 41501 48752**]**

Initial approximation**:** **[**0. 0. 0. 0. 0. 0. 0. 0. 0. 0.**]**

Minimal residual method**:**

Number of iterations **is:** 113

Real unknowns **is:** **[** 1 2 3 4 5 6 7 8 9 10**]**

**||**AX\_ **-** F**||** **=** 8.541519491700456e-08

Vector of computed terms **is:**

0.9999999998244169

2.0000000000210676

2.9999999999956959

3.9999999999903189

4.9999999999978364

6.0000000000040847

7.0000000000010347

8.0000000000046061

9.0000000000040252

10.0000000000057554

Absolute error **is:** 1.7558310361209806e-10

Relaxation method**:**

Number of iterations **is:** 121

Real unknowns **is:** **[** 1 2 3 4 5 6 7 8 9 10**]**

**||**AX\_ **-** F**||** **=** 8.224014891311526e-08

Vector of computed terms **is:**

0.9999999999683747

1.9999999999812401

2.9999999999789089

3.9999999999666871

5.0000000000327471

6.0000000000106040

7.0000000000016502

7.9999999999809344

9.0000000000040465

10.0000000000003144

Absolute error **is:** 3.33129079876926e-11

## Выводы

Метод минимальных невязок осуществляет меньше итераций для достижения необходимой точности, относительно метода релаксации. Однако метод релаксации имеет меньшую абсолютную погрешность.

Проведя вычислительный эксперимент по результатам работы метода релаксации с различными параметрами релаксации можно отследить следующие зависимости:   
при нижней релаксации выполняется наибольшее количество итераций, чем ближе параметр релаксации к значению 1, тем меньше итераций выполняется, уменьшается абсолютная погрешность. При верхней релаксации количество итераций увеличивается, также как и абсолютная погрешность.