МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Дисциплина: «Численные методы»

Лабораторное задание № 1

Отчет по лабораторной работе № 1

Тема: «Численные методы решения систем линейных алгебраических уравнений с матрицами специального вида»

Выполнил:

студент 3 курса 8 группы

Казанин Андрей Алексеевич

(*фио*)

Проверил:

преподаватель

Махинова О.А.

1. **Постановка задачи**

Решить систему линейных уравнений Ax=f.

Матрица A, представленная на Рис. 1, определяется пятью векторами a, b, c, q, p.

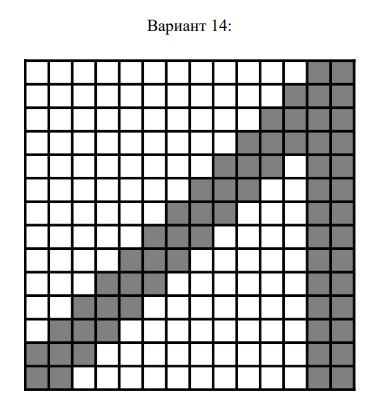


Рис. 1

**2. Анализ задачи**

Систему уравнений задают 6 векторов a, b, c, f, p, q ∈ и матрица А ∈ .

a, b, c – векторы для элементов матрицы A, расположенных на побочной диагонали по направлению сверху вниз.

p, q – векторы для элементов n-1 и n столбцов матрицы.

f – вектор правой части системы уравнений.

Исходная матрица имеет вид:

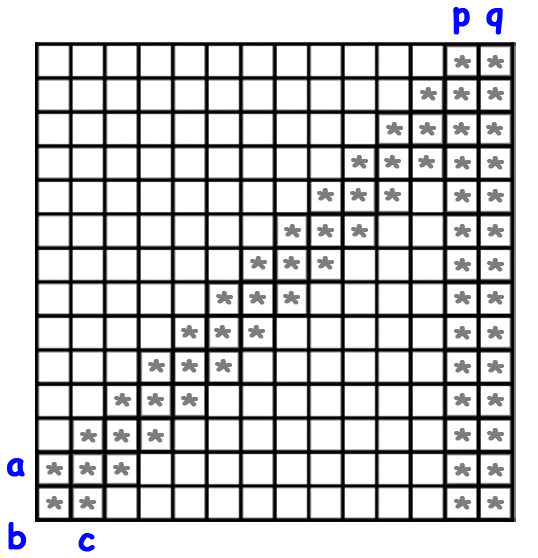
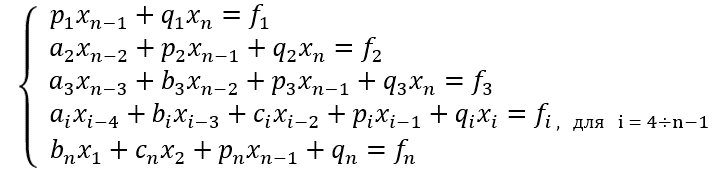


Рис. 2

Во введенных обозначениях система уравнений Ax = f записывается в виде:



При этом положим крайние элементы векторов n-1 размера как = 0 = 0

А для повторяющихся элементов по столбцам условимся в равенствах:

для n-1 столбца: = = =

для n столбца: = =

Рассмотрим алгоритм приведения исходной матрицы (рис. 2) к матрице с единицами на побочной диагонали (рис. 3)

Для упрощения разобьем алгоритм на 3 шага.

**1 шаг**

Обращение в нули элементов верхней побочной кодиагонали (вектор *a*) и обращение в единицы элементов побочной диагонали (вектор *b*).

Суть шага заключается в последовательном построчном подъеме и преобразовании коэффициентов, отвечающих за соответствующие строки. С точки зрения системы это выглядит как преобразование всех уравнений таким образом, что исчезает первое слагаемое в каждом из них. Рассмотрим подробнее:

Для начала, n уравнение делится на ведущий член , таким образом получая = 1. Далее из n-1-го уравнения вычитается n, умноженное на коэффициент и получаем новое уравнение, в котором ведущим членом будет слагаемое с , т.к. коэффициент обратиться в ноль. Повторяя эту процедуру n-1 раз мы гарантируем обнуление всех координат вектора *a* и создание единиц вместо координат вектора *b* (за исключением , т.к. последнюю строку и по совместительству первое уравнение нужно обработать отдельно, просто поделив его на – его ведущий член). Демонстрация результата представлена на рис. 3.

Опасность шага заключается в 2 ключевых моментах:

1. Программно – коэффициенты, хранящиеся в векторах «дублируются» и на 1, 2 и 3 строках (уравнениях системы) потребуется замена дубликатов по измененным оригиналам элементов побочных диагоналей.
2. В случае, когда = и = для i = 2n, коэффициент обращается в ноль и впоследствии алгоритм неприменим либо в шаге 1 – деление на 0, либо во шаге 2 – невозможность его осуществления (в случае когда i = 2). К матрицам с таким дефектом алгоритм не может быть применен (в случае необусловленных матриц такая ситуация возникает редко, а в случае хорошо обусловленных и вовсе не возникает).

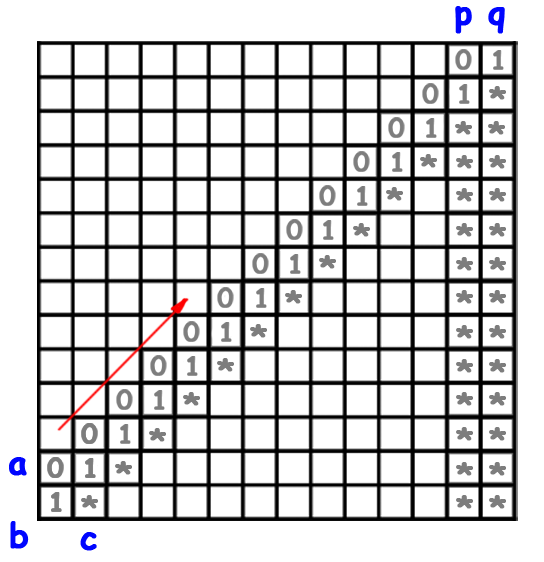


Рис. 3

**Псевдокод шага 1**

для

нц

*// Деление строки и создание ведущей единицы*

*// Вычитание уравнений и обращение в 0 элемента верхней кодиагонали*

*// Проверки на случай повторяющихся элементов*

еслито

еслито

кц

**2 шаг**

Обращение в нули оставшихся нетронутыми элементов n и n-1 столбцов

(столбцы *p* и *q* соответственно)

Суть шага заключается в том, чтобы, используя 1 уравнение системы (уже сведенное к единичному коэффициенту), путем его умножения на коэффициент , где i = 2n и вычитания получившегося уравнения из i уравнения системы, свести все n-1 уравнения к виду без члена (все коэффициенты обращаются в ноль).

После обнуления n столбца *q*, 2 уравнение системы таким же способом можно использовать для преобразования оставшихся n-2 уравнений к двучленному виду (все коэффициенты обращаются в ноль). Демонстрация результата представлена на рис. 4.

Как и в предыдущем шаге, опасность заключается в дублировании элементов при хранении на компьютере, и решение этой проблемы будет аналогичным, с тем лишь исключением что теперь главными будут считаться измененные элементы в столбцах *p*, *q* и по ним изменятся соответствующие элементы в векторах побочных диагоналей.

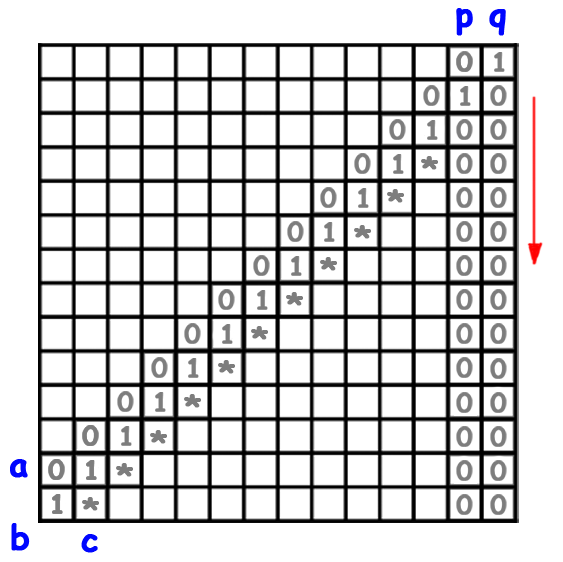


Рис. 4

**Псевдокод шага** **2**

*// Преобразование в 0 столбца q*

нц

то

кц

// Преобразование в 0 столбца p

нц

то

кц

**3 шаг**

Обращение в нули элементов нижней побочной кодиагонали (вектор *c*)

Суть шага (аналогично шагу 1) заключается в том, чтобы, используя 3 уравнение системы, и оставшиеся n-4 двучленных уравнений – свети их к одночленному виду, заполнив тем самым нижнюю побочную кодиагональ нулями. С помощью

и уравнения =, путем его предварительного умножения на и вычитание из 4 уравнения системы, сводим 4 уравнение к виду =1, . Повторяя итеративно этот шаг n-2 раз, спускаясь постоянно построчно вниз мы преобразуем матрицу к единичному виду (рис. 5).

В отличии от предыдущих шагов, никаких опасностей 3 шаг не утаивает и дойдя до него, он всегда окажется выполненным.

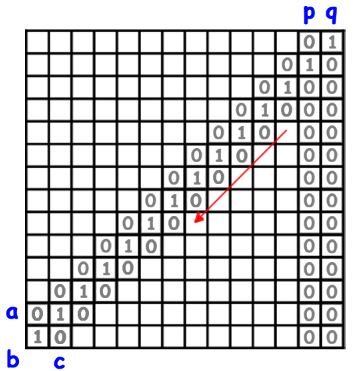


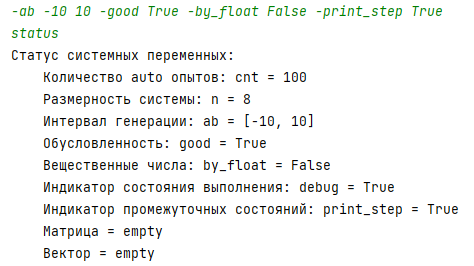
Рис. 5

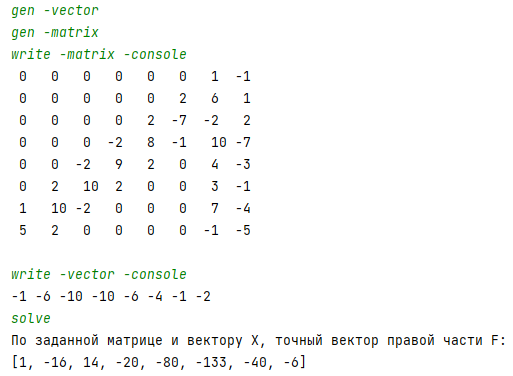
**Псевдокод шага 3**

нц

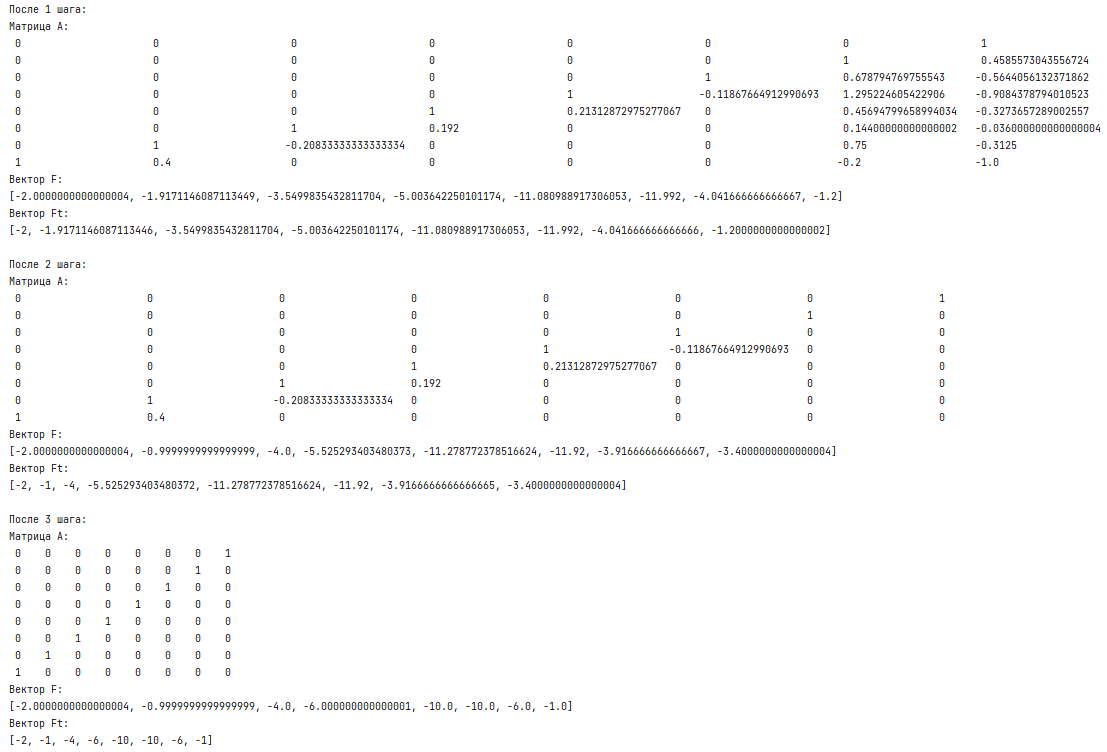
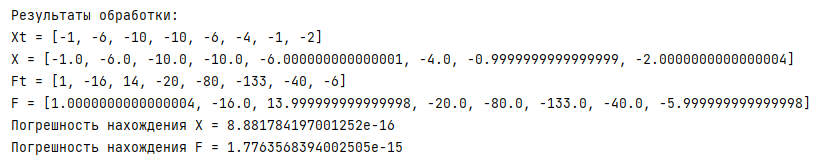
кц

**3. Тестовая задача**

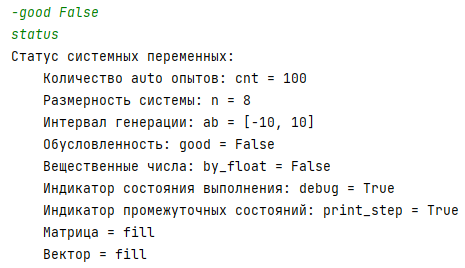
Программный интерфейс предусматривает взаимодействие с пользователем в реальном времени путем ввода необходимых команд со специальными флагами, статус которых можно узнать. Для начала настроим программу нужными флагами. Для наглядности опыта будем использовать только целые числа из маленького интервала.

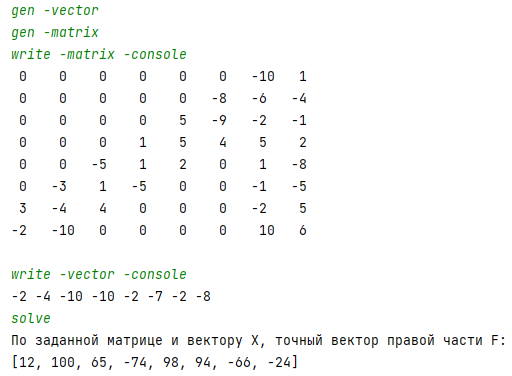
Сгенерируем и рассмотрим обусловленную матрицу и вектор ‘неизвестных’ переменных *Х*.

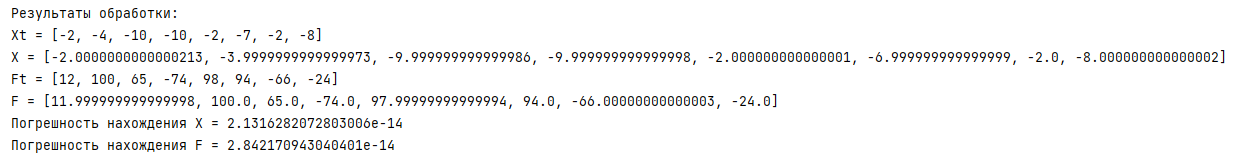
Посмотрим на пошаговые преобразования матрицы и вектора правой части во время решения:



Как нетрудно заметить, получившийся на 3 шаге вектор *F* совпадает с перевернутым вычисленным вектором *X*, что очевидно, учитывая равенство *Ax=F*, где *A* – единичная диагональная матрица с *побочной* диагональю.

Проведем аналогичные действия, но на этот раз будем генерировать плохо обусловленную матрицу.



Посмотрим на пошаговые преобразования матрицы и вектора правой части во время решения:



В результате тестов мы можем наблюдать вычислительную погрешность, вызванную модифицированием матрицы в процессе метода прогонки.

**4. Вычислительный эксперименты**

Исследуем зависимость погрешности от размерности систем, обусловленности и интервалов генерации.

Во всех случаях будем генерировать 10 тыс. вещественных систем и находить среднее арифметическое их решений путем выбора минимума из классической и итеративной формулы.

Для начала рассмотрим случай хорошо обусловленной матрицы

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность системы | Интервал генерации | Ср. относительная погрешность Х | Ср. относительная погрешность F |
| 8 | (-10; 10) | 4.9573560534366004e-15 | 1.0977100658506799e-14 |
| (-100; 100) | 5.013582327073603e-14 | 1.0476149794828664e-12 |
| (-1000; 1000) | 5.050639906301058e-13 | 1.0278737410238063e-10 |
| 16 | (-10; 10) | 7.13191936063673e-15 | 1.6096772672584536e-14 |
| (-100; 100) | 7.348503749948776e-14 | 1.57244993725775e-12 |
| (-1000; 1000) | 7.120450229436415e-13 | 1.5484731059558744e-10 |
| 32 | (-10; 10) | 8.910183701971163e-15 | 2.051901981658942e-14 |
| (-100; 100) | 9.415747070917746e-14 | 2.0074782725032434e-12 |
| (-1000; 1000) | 8.862575562318398e-13 | 1.9826670722977707e-10 |
| 64 | (-10; 10) | 1.1548552392159894e-14 | 2.4635599116251683e-14 |
| (-100; 100) | 1.2297010631989736e-13 | 2.3980959440450534e-12 |
| (-1000; 1000) | 1.2533682536286506e-12 | 2.3806528588465883e-10 |
| 128 | (-10; 10) | 1.5801682531901673e-14 | 2.862560188887631e-14 |
| (-100; 100) | 1.4250929325498873e-13 | 2.8267375995483173e-12 |
| (-1000; 1000) | 1.439681929227276e-12 | 2.750975909293622e-10 |
| 256 | (-10; 10) | 1.860763418348732e-14 | 3.2168201435922585e-14 |
| (-100; 100) | 1.8931765044527538e-13 | 3.259487968421126e-12 |
| (-1000; 1000) | 1.8378774058191994e-12 | 3.1309587029682005e-10 |
| 512 | (-10; 10) | 4.7766945954208494e-14 | 3.6298131167455855e-14 |
| (-100; 100) | 2.4374557927586695e-13 | 3.590843888900963e-12 |
| (-1000; 1000) | 2.37927419988182e-12 | 3.550954716047272e-10 |
| 1024 | (-10; 10) | 4.5788708935567967e-14 | 4.062139513649754e-14 |
| (-100; 100) | 2.936634407868499e-13 | 3.913460666638002e-12 |
| (-1000; 1000) | 3.116433155980758e-12 | 3.9769283830537445e-10 |
| 2048 | (-10; 10) | 3.96143236269797e-14 | 4.591960700572879e-14 |
| (-100; 100) | 5.780736223037761e-13 | 4.236558481807148e-12 |
| (-1000; 1000) | 3.6997930052962146e-12 | 4.3152830357939785e-10 |

Перейдем к случаю плохо обусловленной матрицы

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность системы | Интервал генерации | Ср. относительная погрешность Х | Ср. относительная погрешность F |
| 8 | (-10; 10) | 2.5866574143024085e-13 | 1.7178857570102737e-13 |
| (-100; 100) | 4.018165555841335e-12 | 1.59190676625659e-11 |
| (-1000; 1000) | 9.202622415216411e-11 | 1.5027848313753845e-09 |
| 16 | (-10; 10) | 2.14429956707507e-12 | 7.906070629256975e-13 |
| (-100; 100) | 2.010255082396378e-11 | 4.6940400514086734e-11 |
| (-1000; 1000) | 1.3337938860047413e-10 | 4.087802344828295e-09 |
| 32 | (-10; 10) | 6.016557230203998e-12 | 2.2432924970996916e-12 |
| (-100; 100) | 8.587830174211497e-11 | 3.3744669872302457e-10 |
| (-1000; 1000) | 7.906316405126279e-10 | 1.2893736402475042e-08 |
| 64 | (-10; 10) | 2.2537507310405046e-11 | 6.4621148529235824e-12 |
| (-100; 100) | 5.143363142279249e-10 | 6.1829688488209325e-09 |
| (-1000; 1000) | 2.416994798082322e-09 | 1.160322536634336e-07 |
| 128 | (-10; 10) | 1.1304959473545928e-10 | 3.306380733992036e-11 |
| (-100; 100) | 5.821966109285925e-09 | 3.464403179569573e-09 |
| (-1000; 1000) | 4.530868292107465e-09 | 1.5416288642882069e-07 |
| 256 | (-10; 10) | 5.709992634272938e-10 | 5.707740012218061e-11 |
| (-100; 100) | 2.6165187515969607e-09 | 4.118708592397251e-09 |
| (-1000; 1000) | 2.663589175480037e-08 | 9.801450204577122e-07 |
| 512 | (-10; 10) | 8.92945333468493e-09 | 1.6723240829595017e-10 |
| (-100; 100) | 1.7737079530100175e-08 | 9.538729588056665e-09 |
| (-1000; 1000) | 9.776518259334561e-08 | 1.6911397901890896e-06 |
| 1024 | (-10; 10) | 1.1964422797564045e-09 | 3.186251405544258e-10 |
| (-100; 100) | 1.006262244227063e-07 | 1.2185013181052954e-08 |
| (-1000; 1000) | 1.0935796198285886e-07 | 2.437208551964413e-06 |
| 2048 | (-10; 10) | 3.1403385373440786e-09 | 1.1301832182523897e-09 |
| (-100; 100) | 1.1261792819770004e-07 | 4.9017432372370097e-08 |
| (-1000; 1000) | 1.3602688059936672e-06 | 6.519250564451795e-06 |

**5. Выводы**

Из опытов видно, что погрешность зависит как от размерностей системы и интервала генерации, так и от обусловленности матрицы особого вида.

С ростом размерности системы неизбежно растет и погрешность.

Независимо от роста размерности, при росте размаха интервала, погрешность так же увеличивается, в среднем на 1 знак при увеличении интервала на порядок.

Обусловленность матрицы особого вида в свою очередь влияет на то, насколько большим будет рост от вышеописанных факторов:

* В случае хорошо обусловленной матрицы – рост от размерности почти незаметен (погрешность увеличилась на порядок при увеличении матрицы в 210 раза).
* В случае плохо обусловленной матрицы – рост размерности всего в 4 раза увеличивает порядок погрешности на единицу.

**6. Приложение**

Для реализации поставленной задачи были созданы вспомогательные классы *Vector* и *Matrix*. Далее представлены выкройки самого основного кода (необходимого для проведения опытов) в обоих классах.

**Выкройка из класса *Vector* в пакете *MyPackage*:**

**import** io  
**import** random  
  
*# Класс для работы с векторами основанный на python-list***class** Vector:  
 *# Инициализатор (заполняет default значением = 0)* **def** \_\_init\_\_(self, size=0):  
 self.\_\_buf = [0 **for** \_ **in** range(size)]

*# Переопределение вызова функции len()* **def** \_\_len\_\_(self):  
 **return** len(self.\_\_buf)  
  
 *# Переопределение вызова функции abs()* **def** \_\_abs\_\_(self):  
 v = Vector()  
 **for** x **in** self.\_\_buf:  
 v.\_\_buf.append(abs(x))  
 **return** v  
  
 *# Переопределение оператора [] для получения значения* **def** \_\_getitem\_\_(self, item):  
 **return** self.\_\_buf[item - (item > 0)]  
  
 *# Переопределение оператора [] для присваивания значения* **def** \_\_setitem\_\_(self, key, value):  
 self.\_\_buf[key - (key > 0)] = value

*# Переопределение пользовательского строкового представления* **def** \_\_str\_\_(self):  
 **return** self.\_\_buf.\_\_str\_\_()  
  
 *# Подсчет нормы как максимума по модулю* **def** norm(self):  
 **return** max(abs(self))

*# Заполнение случайными числами* **def** fill\_rand(self, a, b, by\_float=**False**, rand\_sign=**False**):  
 **if** a >= b **or** (b - a) <= 1:  
 **raise** ValueError(**"Неверно заданный интервал (расширьте и проверьте границы)"**)  
 **if** by\_float:  
 **for** i **in** range(len(self.\_\_buf)):  
 self.\_\_buf[i] = random.uniform(float(a), float(b))  
 *# В случае нуля оставляем что-то маленькое, если это возможно* **if** self.\_\_buf[i] == 0:  
 **if** b != 0:  
 self.\_\_buf[i] = b  
 **else**:  
 self.\_\_buf[i] = a  
 **if** rand\_sign:  
 self.\_\_buf[i] \*= random.choice([1, -1])  
 **else**:

**for** i **in** range(len(self.\_\_buf)):  
 self.\_\_buf[i] = random.randint(a, b)  
 **if** self.\_\_buf[i] == 0:  
 **if** b != 0:  
 self.\_\_buf[i] = b  
 **else**:  
 self.\_\_buf[i] = a  
 **if** rand\_sign:  
 self.\_\_buf[i] \*= random.choice([1, -1])  
 **return** self  
  
 *# Глубокое копирование* **def** copy(self):  
 v = Vector()  
 v.\_\_buf = self.\_\_buf[:]  
 **return** v  
  
 *# Переворот списка* **def** reverse(self):  
 v = self.copy()  
 v.\_\_buf = v.\_\_buf[::-1]  
 **return** v  
  
 *# Поиск минимума* **def** min(self):  
 **return** min(self.\_\_buf)  
  
 *# Поиск максимума* **def** max(self):  
 **return** max(self.\_\_buf)

**Выкройка из класса *Matrix* в пакете *MyPackage*:**

**from** MyPackage.Vector **import** Vector  
**import** random  
  
**class** Matrix:  
 *# Инициализатор матрицы (создает квадратную матрицу NxN где N-натуральное)* **def** \_\_init\_\_(self, size=1):  
 *# В случае неподходящего типа или размера default размер = 1* **if** type(size) != int **or** size <= 0:  
 size = 1  
 self.\_\_a, self.\_\_b, self.\_\_c, self.\_\_p, self.\_\_q = \

(Vector(size) **for** \_ **in** range(5))  
  
 *# Переопределение оператора \* (только для умножения матрицы на вектор)* **def** \_\_mul\_\_(self, other):  
 *# Ставим защиту от неподходящих типов* **if** type(other) != int **and** type(other) != float **and** type(other) != Vector:  
 **raise** TypeError(**"Операция \* возможна только в случае типов Matrix \* Vector"**)  
 size = len(other)  
 *# Защита от разных размерностей* **if** size != len(self.\_\_a):  
 **raise** ValueError(**"Несоразмерные вектор и матрица"**)  
 v = Vector(size)  
 *# Сначала рассмотрим частные случаи* **if** size >= 1:  
 v[1] = self.\_\_b[1] \* other[-1]  
 **if** size >= 2:  
 v[1] += self.\_\_a[1] \* other[-2]  
 v[2] = self.\_\_b[2] \* other[-2] + self.\_\_c[2] \* other[-1]  
 **if** size >= 3:  
 v[2] += self.\_\_a[2] \* other[-3]  
 v[3] = self.\_\_b[3] \* other[-3] + self.\_\_c[3] \* other[-2] + \

self.\_\_q[3] \* other[-1]  
 *# Затем рассмотрим общий случай* **if** size >= 4:  
 v[3] += self.\_\_a[3] \* other[-4]  
 **for** i **in** range(4, size + 1):  
 v[i] = self.\_\_b[i] \* other[-i] + self.\_\_c[i] \* other[1 - i] + \  
 self.\_\_p[i] \* other[-2] + self.\_\_q[i] \* other[-1]  
 **if** i != size:  
 v[i] += self.\_\_a[i] \* other[-i - 1]  
 **return** v

*# Метод заполнения побочной диагонали для хорошей обусловленности матрицы* **def** \_\_fill\_diag\_rand\_good(self, a, b, by\_float=**True**):  
 **if** a >= 0:  
 *# В случае [a>0, b>0] меньшие абсолютные значения лежат слева* begin, end = a, a + (b - a - 1) // 4  
 **elif** b <= 0:  
 *# В случае [a<0, b<0] меньшие абсолютные значения лежат права* begin, end = b - (b - a - 1) // 4, b  
 **else**:  
 *# В случае [a<0, b>0] - меньшие значения лежат в области 0* d = min((abs(b), abs(a))) // 5  
 begin, end = 0, 0  
 **if** a < -d:  
 begin = -d  
 **if** b > d:  
 end = d  
 self.\_\_a.fill\_rand(begin, end, by\_float=by\_float)  
 self.\_\_c.fill\_rand(begin, end, by\_float=by\_float)  
 n = len(self.\_\_a)  
 **for** i **in** range(1, n + 1):  
 sum = 0  
 **if** i != 1:  
 sum += abs(self.\_\_a[i-1])  
 **if** i != n:  
 sum += abs(self.\_\_c[i+1])  
 **if** a >= 0:  
 *# В случае [a>0, b>0] устанавливаем нижнюю грань генерации* self.\_\_b[i] = [random.randint(int(sum), b),  
 random.uniform(float(sum), float(b))][by\_float]  
 **else**:  
 *# a < 0* gen\_a, gen\_b = **False**, **False** sgn\_b = (1, -1)[b < 0]  
 **if** -sum >= a **and** sum \* sgn\_b <= b:  
 *# Если сумма модулей лежит в области обеих границ, то рандомно*

*# выбираем из какой генерировать* gen\_a = random.choice([**True**, **False**])  
 gen\_b = **not** gen\_a  
 **if** gen\_a **or** -sum >= a **and** sum \* sgn\_b >= b:  
 *# Если сумма модулей лежит в области только левой границы* self.\_\_b[i] = [random.randint(a, int(-sum)),  
 random.uniform(float(a), float(-sum))][by\_float]  
 **elif** gen\_b **or** -sum <= a **and** sum \* sgn\_b <= b:  
 *# Если сумма модулей лежит в области только правой границы*self.\_\_b[i] = [random.randint(sgn\_b \* (int(sum) + 1), b),  
 random.uniform(float(sum \* sgn\_b),

float(b))][by\_float]  
 **else**:  
 **raise** ValueError(**"Невозможно сгенерировать нужную матрицу по заданному диапазону"**)  
 **if** self.\_\_b[i] == 0:  
 self.\_\_b[i] = (a, b)[max(abs(a), abs(b)) == abs(b)]  
 **if** by\_float:  
 self.\_\_b[i] += random.random()  
 **if** self.\_\_b[i] > b:  
 self.\_\_b[i] = b - 0.001  
  
 *# Метод заполнения побочной диагонали для плохой обусловленности матрицы* **def** \_\_fill\_diag\_rand\_bad(self, a, b, by\_float=**True**):  
 *# Для плохой обусловленности* self.\_\_a.fill\_rand(a, b, by\_float=by\_float)  
 self.\_\_c.fill\_rand(a, b, by\_float=by\_float)  
 n = len(self.\_\_a)  
 **for** i **in** range(1, n + 1):  
 s = 0  
 **if** i != 1:  
 s += abs(self.\_\_a[i - 1])  
 **if** i != n:  
 s += abs(self.\_\_c[i + 1])  
 s -= 1  
 sgn\_a, sgn\_b = (-1, 1)[a >= 0], (-1, 1)[b >= 0]  
 **if** a >= 0:  
 *# [a >= 0, b > 0]* **try**:  
 self.\_\_b[i] = [random.randint(a, int(min(b, s))),  
 random.uniform(float(a),

float(min(b, s)))][by\_float]  
 **except** ValueError:  
 self.\_\_b[i] = a  
 **else**:  
 **if** b <= 0:  
 *# [a < 0, b <= 0]* **try**:  
 self.\_\_b[i] = [random.randint(int(max(a, -s)), b),  
 random.uniform(float(max(a, -s)),

float(b))][by\_float]  
 **except** ValueError:  
 self.\_\_b[i] = b  
 **else**:  
 *# [a < 0, b > 0]* self.\_\_b[i] = [random.randint(int(max(a, -s)), int(min(b, s))),  
 random.uniform(float(max(a, -s)),

float(min(b, s)))][by\_float]  
 **if** self.\_\_b[i] == 0:  
 self.\_\_b[i] = (sgn\_a, sgn\_b)[max(abs(a), abs(b)) == abs(b)]  
 **if** by\_float:  
 self.\_\_b[i] += random.random()  
 **if** self.\_\_b[i] > b:  
 self.\_\_b[i] = b - 0.001  
  
 *# Заполнение случайными числами  
 # good - показатель обусловленности* **def** fill\_rand(self, a, b, good=**True**, by\_float=**True**):  
 **if** a >= b:  
 **raise** ValueError(**"Неверно заданный интервал"**)  
 **if** good:  
 self.\_\_fill\_diag\_rand\_good(a, b, by\_float)  
 **else**:  
 self.\_\_fill\_diag\_rand\_bad(a, b, by\_float)  
 *# Заполняем столбцы по изначальным интервалам т.к. они ни на что не влияют* self.\_\_p.fill\_rand(a, b, by\_float=by\_float)  
 self.\_\_q.fill\_rand(a, b, by\_float=by\_float)  
 size = len(self.\_\_a)  
 *# Перезапись дублирующихся элементов* **if** size >= 1:  
 self.\_\_q[1] = self.\_\_b[1]  
 **if** size >= 2:  
 self.\_\_p[1] = self.\_\_a[1]  
 self.\_\_p[2], self.\_\_q[2] = self.\_\_b[2], self.\_\_c[2]  
 **if** size >= 3:  
 self.\_\_p[3] = self.\_\_c[3]  
  
 *# Нахождение решение системы уравнений* **def** solution(self, v, print\_step=**False**):  
 size = len(v)  
 f = v.copy()  
 m = Matrix()  
 *# Заводим копии чтобы не испортить исходную матрицу и работаем в последствии с ними* m.\_\_a, m.\_\_b, m.\_\_c, m.\_\_p, m.\_\_q = \  
 self.\_\_a.copy(), self.\_\_b.copy(), self.\_\_c.copy(), self.\_\_p.copy(),

self.\_\_q.copy()  
 *# 1 шаг - создание нулей на верхней кодиагонали и единиц на диагонали* **for** i **in** range(size, 1, -1):  
 tmp = m.\_\_b[i]  
 **if** tmp != 1:  
 *# Если единица есть изначально, то делить вектор на главный элемент не нужно* **if** tmp == 0:  
 mes = **"Встречено деление на 0 в операции на "** + str(i) + **" строке"  
 raise** ZeroDivisionError(mes)  
 m.\_\_b[i] = 1  
 m.\_\_c[i] /= tmp  
 m.\_\_p[i] /= tmp  
 m.\_\_q[i] /= tmp  
 f[i] /= tmp  
 tmp = m.\_\_a[i - 1]  
 **if** tmp != 0:  
 *# Если ноль уже на верхней кодиагонали то ничего делать не надо* m.\_\_a[i - 1] = 0  
 m.\_\_b[i - 1] -= tmp \* m.\_\_c[i]  
 m.\_\_p[i - 1] -= tmp \* m.\_\_p[i]  
 m.\_\_q[i - 1] -= tmp \* m.\_\_q[i]  
 *# Защита от дублирующихся элементов* **if** i == 4:  
 m.\_\_c[i - 1] = m.\_\_p[i - 1]  
 **if** i == 3:  
 m.\_\_c[i - 1] = m.\_\_q[i - 1]  
 f[i - 1] -= tmp \* f[i]  
 *# Устанавливаем финальные единицы в последней правой верхней клетке матрицы* f[1] /= m.\_\_b[1]  
 m.\_\_b[1], m.\_\_q[1] = 1, 1  
 **if** print\_step:  
 print(**f"После 1 шага:"**,  
 **f"Матрица A:"**,  
 m,  
 **"Вектор F:"**,  
 f,  
 sep=**'\n'**, end=**'\n\n'**)  
 *# 2 шаг - преобразование в 0 столбца q* **for** i **in** range(2, size + 1):  
 f[i] -= f[1] \* m.\_\_q[i]  
 m.\_\_q[i] = 0  
 **if** i == 2:  
 m.\_\_c[i] = 0  
 *# и преобразование в 0 столбца p* **for** i **in** range(3, size + 1):  
 f[i] -= f[2] \* m.\_\_p[i]  
 m.\_\_p[i] = 0  
 **if** i == 3:  
 m.\_\_c[i] = 0  
 **if** print\_step:  
 print(**f"После 2 шага:"**,  
 **f"Матрица A:"**,  
 m,  
 **"Вектор F:"**,  
 f,  
 sep=**'\n'**, end=**'\n\n'**)  
 *# 3 шаг - создание 0 в нижней кодиагонали* **for** i **in** range(4, size + 1):  
 f[i] -= m.\_\_c[i] \* f[i - 1]  
 m.\_\_c[i] = 0  
 *# поскольку матрица единичная с побочной диагональю, вектор решения*

*# будет перевернут* **if** print\_step:  
 print(**f"После 3 шага:"**,  
 **f"Матрица A:"**,  
 m,  
 **"Вектор F:"**,  
 f,  
 sep=**'\n'**, end=**'\n\n'**)  
 **return** f.reverse()  
  
 *# Метод подсчета ошибки для тестов  
 # x - полученный численный вектор  
 # xt - точный исходный вектор  
 # q - неотрицательное число подбираемое с учетом особенностей решаемой*

*# системы уравнений* @staticmethod  
 **def** error\_rate(x, xt, q):  
 size = len(x)  
 d = Vector()  
 **for** i **in** range(1, size + 1):  
 **if** x[i] > q:  
 d.push\_back(abs((x[i] - xt[i]) / abs(xt[i])))  
 **else**:  
 d.push\_back(abs(x[i] - xt[i]))  
 **return** d.norm()

**Выкройка с процедурой тестирования:**

**from** MyPackage.Matrix **import** Matrix  
**from** MyPackage.Vector **import** Vector  
**import** time  
  
  
*# Метод подсчета ошибок по эксперименту  
# n - размерность матрицы для генерации  
# [a, b] - интервал для заполенения случайными значениями  
# cnt - количество проводимых испытаний  
# good - флаг обусловленности матрицы  
# by\_float - флаг заполенения целыми или же вещественными числами***def** errors\_by\_test(n=1, a=10, b=50, cnt=10, good=**True**, by\_float=**True**, debug=**False**):  
 q = b + 1  
 x\_errors, f\_errors = [], []  
 p = 1  
 **for** i **in** range(cnt):  
 *# Генерируем данные* m = Matrix(n)  
 m.fill\_rand(a, b, good=good, by\_float=by\_float)  
 xt = Vector(n)  
 xt.fill\_rand(a, a + (b - a) // 2, by\_float=by\_float)  
 *# Решаем систему и находим ошибки* ft = m \* xt  
 **try**:  
 x = m.solution(ft)  
 f = m \* x  
 x\_err = Matrix.error\_rate(x, xt, q)  
 f\_err = Matrix.error\_rate(f, ft, q)  
 *# Записываем их в список* x\_errors.append(x\_err)  
 f\_errors.append(f\_err)  
 **except** ZeroDivisionError:  
 **pass  
 if** debug **and** i >= p \* (cnt / 100):  
 print(**"\r"** \* (p % 10 + 1), end=**''**)  
 print(**f"{**p**}%"**, end=**''**)  
 p += 1  
 **if** debug:  
 print(**"100%"**, end=**''**)  
 print(**"\r\r\r\r"**, end=**''**)  
 **return** x\_errors, f\_errors  
  
  
*# Метод подсчета среднего по ошибкам для теста  
# n - размерность матрицы для генерации  
# [a, b] - интервал для заполенения случайными значениями  
# cnt - количество проводимых испытаний  
# good - флаг обусловленности матрицы  
# by\_float - флаг заполенения целыми или же вещественными числами***def** average\_errors\_by\_test(n=1, a=10, b=50, cnt=10, good=**True**, by\_float=**True**, debug=**False**):  
 x, f = errors\_by\_test(n, a, b, cnt, good=good, by\_float=by\_float, debug=debug)  
 *# Учитывая машинную точность, считаем среднее по минимуму из 2 возможных формул* x\_rate, f\_rate = 0., 0.  
 **for** i **in** range(len(x)):  
 x\_rate = (x\_rate \* i + x[i]) / (i + 1)  
 f\_rate = (f\_rate \* i + f[i]) / (i + 1)  
 **return** min((x\_rate, sum(x) / len(x))), min((f\_rate, sum(f) / len(f)))  
  
*# Функция с интерфейсом для проведения тестов***def** test():

. . .

*# Определяет пользовательский интерфейс и вызывает вышеописанные процедуры*

*. . .*

*# Основное тело программы***def** main():  
 print(**"Программа запущена"**)  
 start\_work\_time = time.time()  
 test()  
 work\_time = time.time() - start\_work\_time  
 print(**"Программа завершила работу"**,  
 **f"Время работы программы: {**time\_in\_min\_sec(work\_time)**}"**,  
 sep=**'\n'**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()