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 № 2

Отчет по лабораторной работе № 2

Тема: «Применение точных методов решения систем линейных алгебраических уравнений»

Выполнил:

студент 3 курса 8 группы

Казанин Андрей Алексеевич

(*фио*)

Проверил:

преподаватель

Махинова О.А.

1. **Постановка задачи**

Решить систему линейных уравнений методом Халецкого для случая с симметричной ленточной матрицей.

Матрица , представленная на рис. 1, определяется массивом строк размера вида, представленного на рис.1.

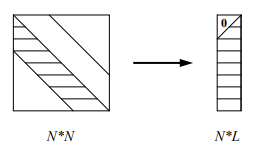


рис. 1

1. **Метод решения**

Для начала опишем алгоритм решения классическим методом Халецкого

Если все главные миноры матрицы отличны от нуля, то матрица ,

согласно известной -теореме линейной алгебры, представима в виде

произведения двух матриц

(1)

где – нижняя треугольная матрица, – верхняя треугольная матрица с

единицами на главной диагонали. Соотношение (1) символически обозначено на рис. 2

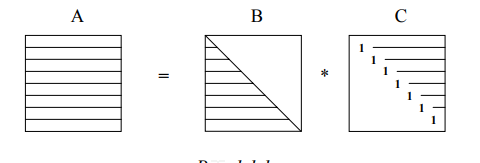


рис. 2

Нетрудно видеть, что справедливы формулы:

Которые выведены из правила перемножения матриц:

Для того чтобы по этим формулам вычислять матрицы и , необходимо указать правильный порядок вычисления элементов.

Для вычисления необходимо знать (рис. 3):

– элементы нижней треугольной матрицы *B*, стоящие в *i-*ой строке с

первого по (*j-1*)*-*й столбец;

– элементы верхней треугольной матрицы *C*, стоящие в *j-*ом столбце,

начиная с первого по (*j-1*)-й включительно.

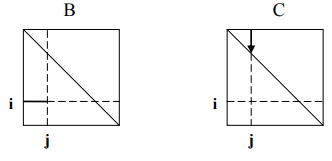


рис. 3

Для вычисления элемента нужно знать (рис. 4):

– все элементы нижней треугольной матрицы *B*, стоящие в *i-*ой строке;

– элементы верхней треугольной матрицы *С*, стоящие в *j-*ом столбце с

первой по (*i-1*)-ую строки.

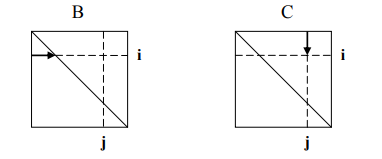


рис. 4

Решение системы линейных алгебраических уравнений

сводится к последовательному решению двух систем уравнений с треугольными матрицами:

которые в символическом виде изображены на рис. 3

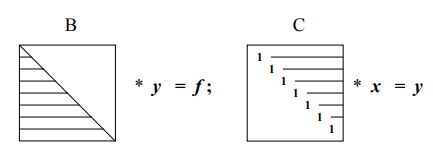


рис. 5

Компоненты векторов и , определяются по формулам:

Вычисление матриц и – *прямой ход**классического метода Халецкого*, а восстановление векторов , – *обратный ход метода Халецкого*.

Теперь модифицируем этот метод в соответствии с задачей.

Поскольку исходная матрица – симметрична, то вычислять коэффициенты матрицы С нет никакой необходимости, ведь их можно задать формулой:

В соответствии с этим, вычислять матрицу нет необходимости.

Перепишем формулы прямого и обратного хода основываясь на этом.

Теперь из условия « – ленточная матрица» перепишем матрицы и (рис. 6):

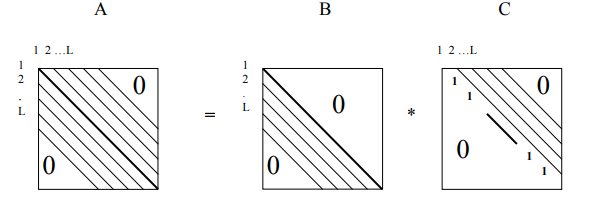


рис. 6

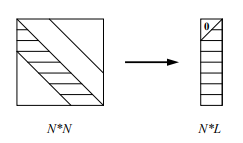
При этом определим способ хранения матрицы в виде ведущей ленты в массиве размера (рис. 7):

рис. 7

Для более удобного индексирования при последующем преобразовании формул условимся, что проход будет осуществляться по «ведущим элементам матрицы» (элементам получившегося массива с сохранением матричных индексов элемента.

Обозначим:

Свяжем эти индексы между собой (с учетом симметричности – *min, max*):

Введем отображение

Эта функция связывает элементы матрицы, с соответствующими элементами ленты-массива (либо 0, в случае если элемент лежит за пределами ленты).

Из условия, можно легко определить ее следующим образом:

С учетом этого, можно переписать формулы Халецкого под поставленную задачу:

Подсчитаем количество операций выполняемых алгоритмом.

Обозначим:

Подсчитаем верхний предел операций алгоритма:

т.к.

Очевидным образом, точный предел операций будет выглядеть иметь вид:

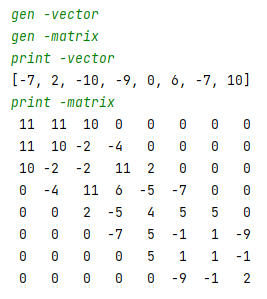
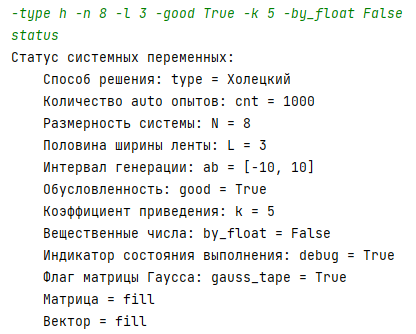
В случае же, когда , нижний предел операций будет иметь вид:

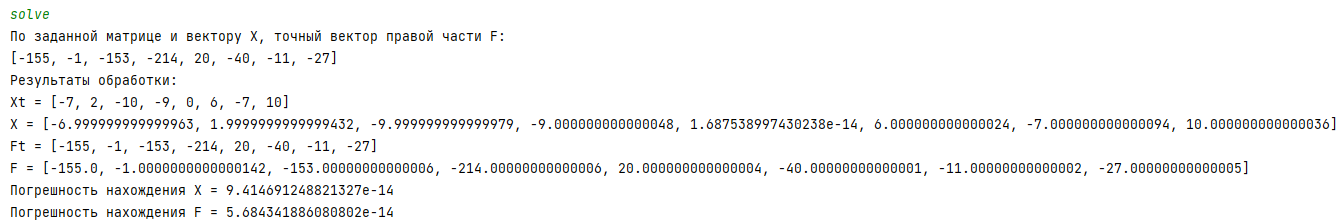
Основные процедуры приведены в виде кода, в приложении:

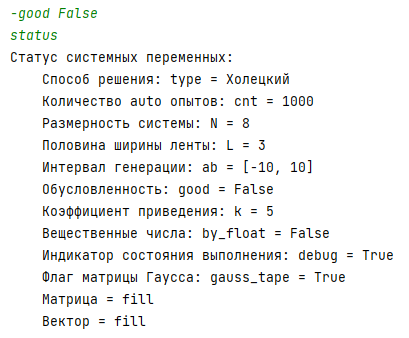
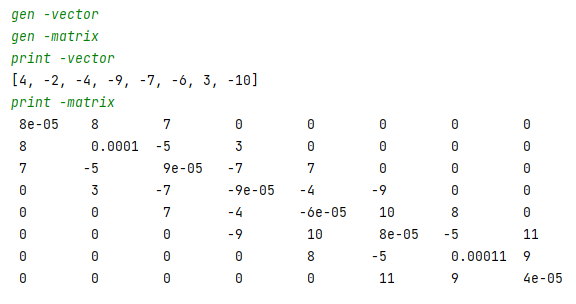
* sbh (решение системы уравнений, метод класса TapeMatrix, стр. 10)
* rate (подсчет ошибки, функция модуля BaseFunctions, стр. 11)

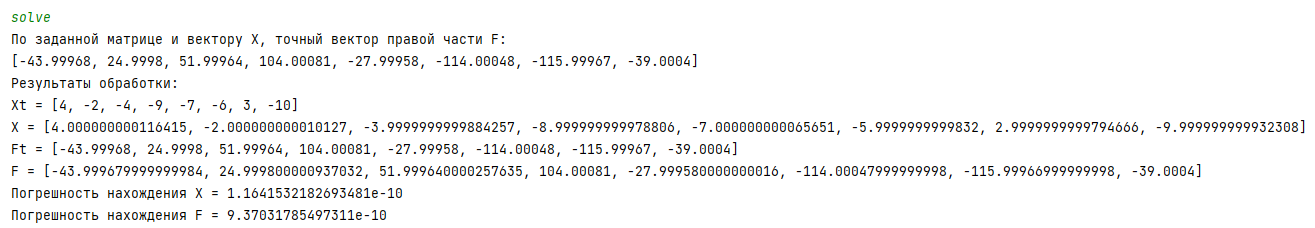
1. **Тестовая задача**

Для начала рассмотрим случай хорошо обусловленной матрицы.

Сгенерируем ее последующим параметрам:

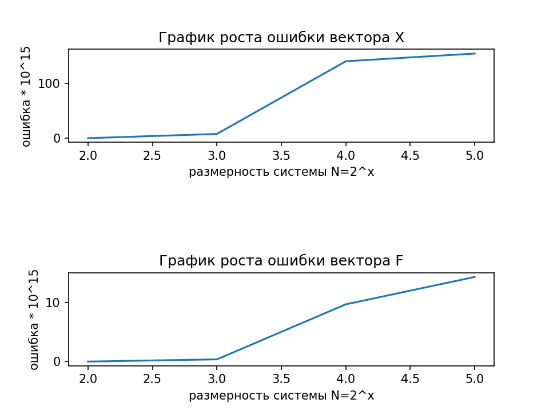
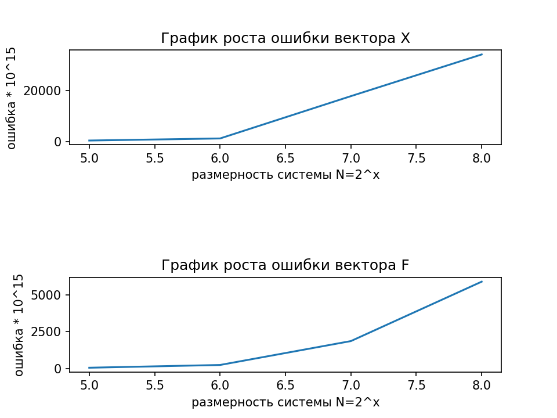
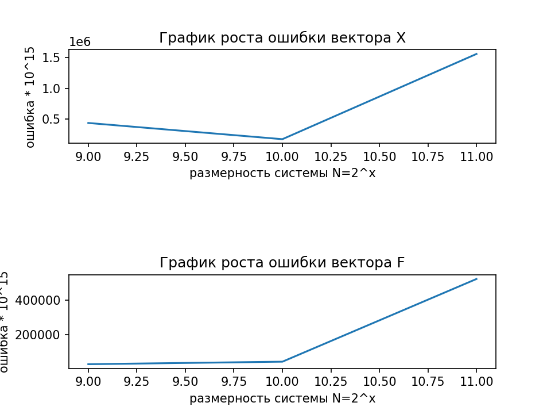
И посмотрим на вывод процедуры решения:

Проделаем то же, но уже для плохо обусловленной матрицы:

Видим заметный рост погрешности:

1. **Вычислительные эксперименты**

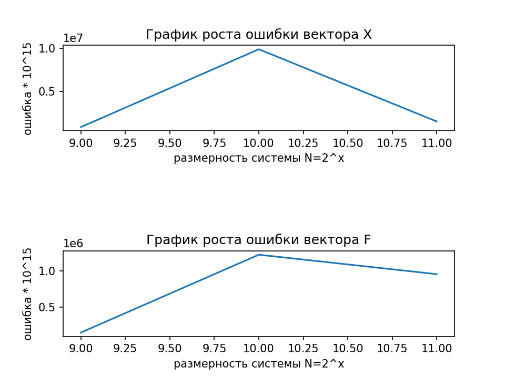
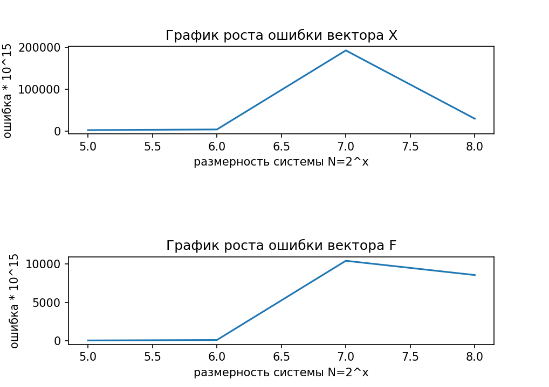
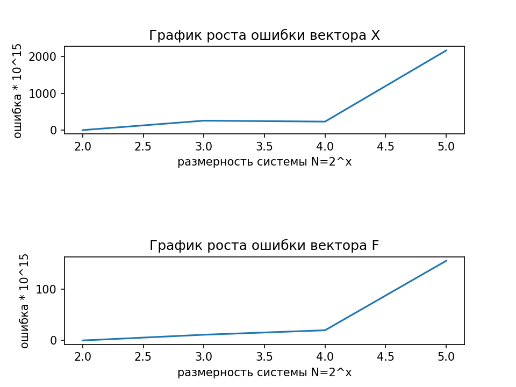
**4.1 Эксперименты с хорошо обусловленными матрицами:**

Для начала проанализируем зависимость точности решения от роста размерности системы при условии что ширина ленты = const. Отрисуем график ошибок с помощью процедуры analyze (процедура модуля MenuOptions, стр. 13). Процедура генерирует обусловленные матрицы в интервале , с шириной ленты удовлетворяющей условию для размерностей и отображает ошибку нахождения решения (для наглядности умноженную на 1015)

Теперь рассмотрим зависимость погрешности хорошо обусловленной матрицы в интервале от порядка системы и отношения (погрешность считаем как среднее от погрешностей матриц):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | Размерность системы | Отношение | Средняя относительная погрешность Х | Средняя относительная погрешность F |
| 1 | 10 | 1/10 | 4.1216162427e-16 | 0.0 |
| 2 | 10 | 2/10 | 1.1599105009e-13 | 3.97270438767e-13 |
| 3 | 30 | 1/10 | 7.3793082755e-13 | 4.8191672874e-12 |
| 4 | 30 | 1/6 ~ 15/100 | 3.9210279467e-12 | 4.376516926e-12 |
| 5 | 30 | 2/10 | 3.37235794845e-12 | 1.47082346302e-12 |
| 6 | 50 | 1/10 | 9.593692507e-13 | 4.18615364594e-12 |
| 7 | 50 | 7/50 ~ 15/100 | 1.35731093031e-12 | 5.4718629627e-12 |
| 8 | 50 | 2/10 | 3.43614914299e-12 | 3.4066860443e-11 |
| 9 | 50 | 12/50 ~ 25/100 | 2.49238851779e-12 | 1.08522613118e-11 |
| 14 | 100 | 1/10 | 1.75346848152e-12 | 9.6165120311e-12 |
| 15 | 100 | 15/100 | 3.8660419310e-11 | 1.1379967190e-10 |
| 16 | 100 | 2/10 | 1.01288577525e-11 | 4.82422279901e-11 |
| 17 | 100 | 25/100 | 9.44179845419e-12 | 9.51649248293e-11 |
| 18 | 300 | 1/10 | 1.29175405927e-10 | 3.22103499428e-10 |
| 19 | 300 | 15/100 | 6.76344857897e-11 | 7.20578681168e-10 |
| 20 | 300 | 2/10 | 8.96357499335e-11 | 5.81191947901e-10 |
| 21 | 300 | 25/100 | 9.829051350607e-11 | 1.416961126210e-09 |
| 22 | 500 | 1/10 | 5.155109450427e-10 | 1.572294863194e-09 |
| 23 | 500 | 15/100 | 2.243419983116e-10 | 2.253133217777e-09 |
| 24 | 500 | 2/10 | 2.913559660200e-10 | 3.478020751401e-09 |
| 25 | 500 | 25/100 | 4.619878612999e-10 | 3.907952059734e-09 |

**4.2 Эксперименты с плохо обусловленными матрицами:**

Проанализируем зависимость точности решения от роста размерности системы при условии что ширина ленты = const. Отрисуем график ошибок с помощью процедуры analyze (процедура модуля MenuOptions, стр. 13). Процедура генерирует плохо обусловленные матрицы в интервале , с шириной ленты удовлетворяющей условию для размерностей и отображает ошибку нахождения решения (для наглядности умноженную на 1015)

Теперь рассмотрим зависимость погрешности плохо обусловленной матрицы в интервале с коэффициентом приведения диагональных элементов от порядка системы и отношения (погрешность считаем как среднее от погрешностей матриц):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | Размерность системы | Отношение | Средняя относительная погрешность Х | Средняя относительная погрешность F |
| 1 | 10 | 1/10 | 4.3465231414e-16 | 0.0 |
| 2 | 10 | 2/10 | 3.84909881745e-13 | 1.904234547822e-12 |
| 3 | 30 | 1/10 | 1.02576502847e-12 | 2.431308487871e-12 |
| 4 | 30 | 1/6 ~ 15/100 | 1.75621406306e-11 | 6.313336520236e-12 |
| 5 | 30 | 2/10 | 1.76945347263e-12 | 6.405967978295e-12 |
| 6 | 50 | 1/10 | 1.91559212936e-12 | 7.8149153814e-12 |
| 7 | 50 | 7/50 ~ 15/100 | 8.23312973352e-12 | 4.6144705834e-11 |
| 8 | 50 | 2/10 | 1.19927656694e-11 | 5.476754882938e-11 |
| 9 | 50 | 12/50 ~ 25/100 | 8.2274320689e-12 | 3.730965314474e-11 |
| 14 | 100 | 1/10 | 7.32245665491e-11 | 3.93816090848e-10 |
| 15 | 100 | 15/100 | 6.26941432102e-11 | 1.00226387189e-10 |
| 16 | 100 | 2/10 | 7.75407071884e-11 | 1.05599147204e-10 |
| 17 | 100 | 25/100 | 1.51896362154e-11 | 8.7104194967e-11 |
| 18 | 300 | 1/10 | 7.14946080115e-11 | 2.07291126486e-10 |
| 19 | 300 | 15/100 | 1.91581823738e-10 | 9.02207926145e-10 |
| 20 | 300 | 2/10 | 1.33421260706e-10 | 1.20801252112e-09 |
| 21 | 300 | 25/100 | 2.52787322185e-10 | 8.573614729e-10 |
| 22 | 500 | 1/10 | 3.46075652579e-10 | 1.330012402434e-09 |
| 23 | 500 | 15/100 | 5.0280151597e-10 | 3.724821580064e-09 |
| 24 | 500 | 2/10 | 7.66277358232e-10 | 2.74871633010e-09 |
| 25 | 500 | 25/100 | 3.30852065744e-10 | 3.59443935726e-09 |

1. **Выводы**

Из вычислительных экспериментов видно, что несмотря на единичные исключения (очевидно возникающие по вине случайной генерации), в целом, можно определить следующие зависимости:

* При увеличении размерности системы ошибка увеличивается
* При увеличении длины ленты ошибка увеличивается
* При переходе от обусловленной матрице к необусловленной ошибка увеличивается (на опытах это не сильно бросается в глаза т.к. коэффициент приведения k брался сравнительно малым, и при его увеличении ошибка вырастет сильнее)

1. **Приложение**

Ниже приведены основные методы и функции, используемые приложением для проведения вычислительных экспериментов:

1. **Модуль TapeMatrix:**

*"""Определение класса симметричной ленточной матрицы"""***import** random  
  
  
**class** TapeMatrix:  
 *# Инициализатор:  
 # N - размерность матрицы  
 # L - половина ширины ленты (с учетом главной диагонали)  
 # value - значение по умолчанию* **def** \_\_init\_\_(self, N, L, value=0):  
 self.N = N  
 self.L = L  
 *# A - матрица ленты* self.A = [[value **for** j **in** range(L)] **for** i **in** range(N)]  
  
 *# Конвертация строки в матрицу* **def** from\_str(self, str):  
 elems = str.strip().split()  
 **if** len(elems) != self.N \*\* 2:  
 **raise** ValueError(**f"Для записи матрицы было необходимо {**self.N\*\*2**} значений"**)  
 *# Обход по столбцам матрицы self.A* **for** i **in** range(self.L):  
 **for** j **in** range(self.N - i):  
 ind = self.N \* j + i + j  
 self.A[j + i][-1 - i] = float(elems[ind])  
  
 *# Получение элемента по индексам с внутреннего буфера* **def** get\_elem(self, i, j):  
 **if not**(0 <= i < self.N **and** 0 <= j < self.N):  
 **raise** IndexError(**"Выход за границы матрицы"**)  
 **if** abs(i - j) >= self.L:  
 **return** 0  
 **else**:  
 **if** i < j:  
 i, j = j, i  
 **return** self.A[i][-1 - i + j]  
  
 *# Аналог получения элемента по индексам с внешнего буфера* @staticmethod  
 **def** gget\_elem(arr, i, j, N, L):  
 **if not**(0 <= i < N **and** 0 <= j < N):  
 **raise** IndexError(**"Выход за границы матрицы"**)  
 **if** abs(i - j) >= L:  
 **return** 0  
 **else**:  
 **if** i < j:  
 i, j = j, i  
 **return** arr[i][-1 - i + j]  
  
 *# Переопределение пользовательского строчного представления (вывода)* **def** \_\_str\_\_(self):  
 res = **""** *# Находим максимальную длину строкового представления числа (для форматирования вывода)* max\_length = max([len(str(self.A[i][j])) **for** i **in** range(self.N) **for** j **in** range(self.L)]) + 1  
 **for** i **in** range(self.N):  
 **for** j **in** range(self.N):  
 elem = self.get\_elem(i, j)  
 s\_elem = str(elem)  
 **if** elem >= 0:  
 s\_elem = **' '** + s\_elem  
 res += s\_elem.ljust(max\_length) + **' '** res += **'\n'  
 return** res  
  
 *# Переопределение умножения* **def** \_\_mul\_\_(self, x):  
 **if** type(x) != list **and** type(x) != tuple():  
 **raise** TypeError(**"Умножение возможно только в случае Matrix \* tuple/list"**)  
 **if** len(x) != self.N:  
 **raise** ValueError(**f"Размерность вектора должна быть = {**self.N**}"**)  
 f = [0 **for** \_ **in** range(self.N)]  
 x = [0 **for** \_ **in** range(self.L - 1)] + x  
 **for** i **in** range(self.N):  
 sum\_horizontal = 0  
 sum\_diagonal = 0  
 **for** j **in** range(self.L):  
 sum\_horizontal += self.A[i][j] \* x[i+j]  
 **if** i + j < self.N:  
 sum\_diagonal += self.A[i+j][-1 - j] \* x[i + j + self.L - 1]  
 f[i] = sum\_diagonal + sum\_horizontal - self.A[i][self.L - 1] \* x[i + self.L - 1]  
 **return** f  
  
 *# Решение системы методом Холецкого* **def** sbh(self, f):  
 f = f[:]  
 *# Заполнение матрицы B (обозначим ее в коде как H)* H = [[0 **for** \_ **in** range(self.L)] **for** \_ **in** range(self.N)]  
 **for** ii **in** range(self.N):  
 **for** jj **in** range(self.L - 1, -1, -1):  
 **if** ii < jj:  
 **continue** i = ii  
 j = i - jj  
 a = self.get\_elem(i, j)  
 sm = 0  
 **for** k **in** range(j):  
 bik = self.gget\_elem(H, i, k, self.N, self.L)  
 bjk = self.gget\_elem(H, j, k, self.N, self.L)  
 bkk = self.gget\_elem(H, k, k, self.N, self.L)  
 sm += bik \* bjk / bkk  
 H[ii][-1 - jj] = a - sm  
 *# Восстановление ответа* y = [0 **for** \_ **in** range(self.N)]  
 x = [0 **for** \_ **in** range(self.N)]  
 **for** i **in** range(self.N):  
 sm = 0  
 **for** k **in** range(i):  
 bik = self.gget\_elem(H, i, k, self.N, self.L)  
 sm += bik \* y[k]  
 bii = self.gget\_elem(H, i, i, self.N, self.L)  
 y[i] = (f[i] - sm) / bii  
  
 **for** i **in** range(self.N - 1, -1, -1):  
 sm = 0  
 **for** k **in** range(i + 1, self.N):  
 bki = self.gget\_elem(H, k, i, self.N, self.L)  
 bii = self.gget\_elem(H, i, i, self.N, self.L)  
 sm += bki \* x[k] / bii  
 x[i] = y[i] - sm  
 **return** x  
  
 *# Заполнение матрицы случайными числами по заданным флагам* **def** fill\_rand(self, a, b, k=**None**, by\_float=**False**, good=**True**):  
 **if** k **is None**:  
 k = 2  
 **if** by\_float:  
 **for** i **in** range(self.N):  
 **for** j **in** range(self.L):  
 **if** i < j:  
 **continue  
 while** self.A[i][-1 - j] == 0:  
 self.A[i][-1 - j] = random.uniform(float(a), float(b))  
 **else**:  
 **for** i **in** range(self.N):  
 **for** j **in** range(self.L):  
 **if** i < j:  
 **continue  
 while** self.A[i][-1 - j] == 0:  
 self.A[i][-1 - j] = random.randint(int(a), int(b) + 1)  
 **if not** good:  
 k = 10\*\*k  
 **for** i **in** range(self.N):  
 self.A[i][-1] /= k

1. **Модуль BaseParams**

*"""Набор глобальных пременных - флагов"""  
  
# Набор констант для типа решения*UNDEF = 0  
GAUSS = 1  
HOL = 2  
  
*# Параметры задающие решение*cnt = 1000 *# кол-во тестов*N = 6 *# размерность матрицы*L = 2 *# половина ширины ленты для матрицы Холецкого (ленточная симметричная)*a, b = -10, 10 *# границы генерации элементов матрицы*by\_float = **False** *# флаг генерации вещественных элементов*good = **True** *# флаг генерации обусловленной матрицы*debug = **True** *# флаг для вывода прогресса отладки*k = 2 *# коэффициент приведения для генерации необусловленной матрицы Холецкого*vector = **None** *# вектор для решения конкретной задачи пользователя*matrix = **None** *# матрица для решения конкретной задачи пользователя*type = GAUSS *# тип алгоритма применяемого для решения задачи*gauss\_tape = **True** *# генерация ленточной матрицы для гаусса*

1. **Модуль BaseFunctions**

*"""Набор вспомогательных функций, используемых в решении"""*  
  
*# функция подсчета накопленной вычислительной погрешности***def** rate(x, xt):  
 size = len(x)  
 d = []  
 **for** i **in** range(size):  
 d.append(abs(x[i] - xt[i]))  
 *# if x[i] > q:  
 # d.append(abs((x[i] - xt[i]) / abs(xt[i])))  
 # else:  
 # d.append(abs(x[i] - xt[i]))* **return** max(map(**lambda** x: abs(x), d))  
  
  
*# Подсчет средней ошибки для заданного метода***def** average\_by\_test(func):  
 x, f = func()  
 *# 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)))

1. **Модуль HolAutoTesting**

*"""Множественное тестирование методом Холецкого"""***from** BaseFunctions **import** \*  
**import** BaseParams **as** BP  
**from** TapeMatrix **import** \*  
  
  
**def** HAT():  
 x\_errors, f\_errors = [], []  
 p = 1  
 **for** i **in** range(BP.cnt):  
 *# Генерируем данные* m = TapeMatrix(BP.N, BP.L)  
 m.fill\_rand(BP.a, BP.b, k=BP.k, good=BP.good, by\_float=BP.by\_float)  
 xt = []  
 **if** BP.by\_float:  
 xt = [random.uniform(float(BP.a), float(BP.b)) **for** \_ **in** range(BP.N)]  
 **else**:  
 xt = [random.randint(int(BP.a), int(BP.b)) **for** \_ **in** range(BP.N)]  
 *# Решаем систему и находим ошибки* ft = m \* xt  
 **try**:  
 x = m.sbh(ft)  
 f = m \* x  
 x\_err = rate(x, xt)  
 f\_err = rate(f, ft)  
 *# Записываем их в список* x\_errors.append(x\_err)  
 f\_errors.append(f\_err)  
 **except** ZeroDivisionError:  
 **pass  
 if** BP.debug **and** i >= p \* (BP.cnt / 100):  
 print(**"\r"** \* (p % 10 + 1), end=**''**)  
 print(**f"{**p**}%"**, end=**''**)  
 p += 1  
 **if** BP.debug:  
 print(**"100%"**, end=**''**)  
 print(**"\r\r\r\r"**, end=**''**)  
 **return** x\_errors, f\_errors

1. **Модуль MenuOptions**

**def** analyze(cmd\_list):  
 ind = cmd\_list.index(**"analyze"**) + 1  
 **if** cmd\_list[ind] == **"n"**:  
 start, end = int(cmd\_list[ind + 1]), int(cmd\_list[ind + 2])  
 x\_err, f\_err = [], []  
 old\_N, old\_cnt, old\_L = BP.N, BP.cnt, BP.L  
 BP.cnt = int(cmd\_list[ind + 3])  
 start, end = math.ceil(math.log2(start)), math.floor(math.log2(end))  
 n\_cur = 2\*\*(start-1)  
 **for** n **in** range(start, end + 1):  
 n\_cur \*= 2  
 BP.N = n\_cur  
 BP.L = int(BP.N \* old\_L/old\_N)  
 xe, fe = average\_by\_test(HAT)  
 x\_err.append(xe\*10\*\*15), f\_err.append(fe\*10\*\*13)  
 print(**f"N = {**BP.N**}, L = {**BP.L**}:"**,  
 **f"x\_err = {**xe**}"**,  
 **f"f\_err = {**fe**}\n"**,  
 sep=**'\n'**)  
 fig = plt.figure(**"Анализ роста ошибки при увеличении N"**)  
 xx = fig.add\_subplot(3, 1, 1)  
 fx = fig.add\_subplot(3, 1, 3)  
 xx.set\_xlabel(**"размерность системы N=2^x"**)  
 xx.set\_ylabel(**"ошибка \* 10^15"**)  
 xx.set\_title(label=**"График роста ошибки вектора X"**)  
 fx.set\_xlabel(**"размерность системы N=2^x"**)  
 fx.set\_ylabel(**"ошибка \* 10^15"**)  
 fx.set\_title(label=**"График роста ошибки вектора F"**)  
 xx.plot(range(start, end+1), x\_err)  
 fx.plot(range(start, end+1), f\_err)  
 plt.show()  
 BP.N = old\_N  
 BP.cnt = old\_cnt  
 BP.L = old\_L  
 **elif** cmd\_list[ind] == **"-l"**:  
 **pass**