Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«Введение в численные методы Задание 1»

ОТЧЕТ

о выполненном задании

студента 206 учебной группы факультета ВМК МГУ
Николайчука Артёма Константиновича

гор. Москва
2020 г.

Содержание

Цели

Целью данной работы является реализация численных методов нахождения решения заданных систем линейных алгебраических уравнений методом Гаусса, в том числе методом Гаусса с выбором главного элемента, и методом верхней релаксации.

- Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента
- Вычислить определителю матрицы $det(A)$
- Вычислить обратную матрицу A^{-1}
- Исследовать вопрос вычислительной устойчивости метода Гаусса
- Решить заданную СЛАУ итерационным методом верхней релаксации
- Разработать критерий остановки итерационного процесса для гарантированного получения приближенного решения исходной СЛАУ с заданной точностью
- Изучить скорость сходимости итераций к точному решению задачи при различных итреационных параметрах ω
- Проверить правильность решения СЛАУ на различных тестах, используя wolframalpha.com

-

Постановка задачи

Дана система линейных уравнений $A\overline{x} = \overline{f}$ порядка $n * n$ с невырожденной матрицей A.
Написать программу, решающую СЛАУ заданного пользователем размера методом Гаусса и методом Гаусса с выбором главного элемента.
Написать программу численного решения СЛАУ заданного пользователем размера, использующую итерационный метод верхней релаксации. Итерационный процесс имеет вид:

$$(D + \omega A^{(-)})\frac{x^{k+1} - x^k}{\omega} + Ax^k = f),$$

где ω -итерационный параметр.
Предусмотреть возможность задания элементов матрицы системы и ее правой части как на входной файле данных, так и задания специальных формул.

Описание алгоритмов

- Метод Гаусса
  Цель - найти решение системы.
  Решения системы линейных алгебраических уравнений методом Гаусса производится в два этапа:"прямой"и "обратный"ход. В результате "прямого хода"матрица коэффициентов, входящая в расширенную матрицу, путем линейных пребразований строк и их перестановок последовательно приводится к верхнему треугольному виду. На этапе "обратного хода"выполняется восстановление решения путем прохода по строкам матрицы в обратном направлении.
  В методе Гаусса с выбором главного элемента на каждой итерации выбирается максимальный из всех элементов матрицы, что позволяет уменьшить погрешность вычислений.

  После приведения матрицы к диагональному виду с помощью классического метода Гаусса, определитель вычисляется, как произведение элементов, стоящих на диагонали.

  Задача нахождения обратной матрицы решается с помощью метода Гаусса-Жордана. Над расширенной матрицей, составленной из столбцов исходной матрицы и единичной того же порядка, производятся преобразования метода Гаусса, в результате которых исходная матрица принимает вид единичной матрицы, а на месте единичной образуется матрица, обратная исходной.

- Метод верхней релаксации

  Метод вехний релакцации это стационарным итерационным методом, в котором каждый следующий вектор приближения точного решения вычисляется по формуле:
  $(D+\omega T)\frac{y_{k+1}-y_k}{\omega}+Ay_k=f$

  где

  ω - итерационный параметр,

  y_k - k-й вектор приближения,

  y_{k+1} - $(k+1)$-й вектор приближения,

  A - матрица коэффициентов СЛАУ,

  f - правая часть СЛАУ,

  T - нижняя диагональная матрица матрицы A,

  D - матрица диагональных элементов матрицы A.

  Метод применим только к положительноопределённым матрицам. По теореме Самарского для положительно определенных матриц A метод верхней релаксации сходится, если $0<\omega<2$. Значением по умолчанию является значение $\omega=\frac{4}{3}$. За вектор начального приближения берется нулевой вектор.

Описание и код программы

Весь код написан на языке программирования python.
Программа размещена на гитхабе. https://github.com/Strannik2001/Computing_Methods

Код основной программы:

```
import sys
import random
from math import sin


class Test:
  def __init__(self, mode, n, a, b=1):
    self.mode = mode
    self.n = n
    self.accuracy = 1e-9
    self.max_iterations = 1000
    self.omega = 4 / 3
    self.need_optimization = b
    self.matrix = []
    self.b = []
    self.q = 0
    self.determinant = 1
    self.invert_matrix = []
    for i in range(n):
      self.invert_matrix.append([])
      self.invert_matrix[i] = [0] * self.n
      self.invert_matrix[i][i] = 1
    self.prepare_matrix(a)
    self.matrix_copy = []
    for i in range(n):
      self.matrix_copy.append([])
      for j in range(n):
        self.matrix_copy[i].append(self.matrix[i][j])

  def prepare_matrix(self, a):
    if self.mode == 'static':
      with open(a, "r") as f:
        for i in range(self.n):
          self.matrix.append(list(map(float, f.readline().split())))
        self.b = list(map(float, f.readline().split()))
    else:
      for i in range(self.n):
        self.matrix.append([])
        self.matrix[i] = [0] * self.n
      self.b = [0] * self.n
      self.q = 1.001 - 2 * a / 1000
      x = random.random()
      for i in range(self.n):
        self.b[i] = abs(x - self.n / 10) * (i + 1) * sin(x)
      for i in range(1, self.n + 1):
        for j in range(1, self.n + 1):
          if i == j:
            self.matrix[i - 1][j - 1] = pow(self.q - 1, i + j)
          else:
            self.matrix[i - 1][j - 1] = pow(self.q, i + j) + 0.1 * (j - i)

  def out_matrix(self, a='matrix'):
    if a == 'invert matrix':
      print("Invert matrix")
      m = self.invert_matrix
    else:
      print("Matrix")
      m = self.matrix
```

```
    out_matrix(m)

  def get_invert_matrix(self):
    if self.determinant != 0:
      return self.invert_matrix
    else:
      print("No such matrix")
      return None

  def gaus(self):
    sign = 1
    matrix_copy = []
    b_copy = []
    for i in range(self.n):
      b_copy.append(self.b[i])
      matrix_copy.append([])
      for j in self.matrix[i]:
        matrix_copy[i].append(j)
    for i in range(self.n):
      if self.need_optimization:
        maxi = i
        for j in range(i, self.n):
          if abs(self.matrix[maxi][i]) < abs(self.matrix[j][i]):
            maxi = j
        if maxi != i:
          sign *= -1
        self.b[i], self.b[maxi] = self.b[maxi], self.b[i]
        self.matrix[i], self.matrix[maxi] = self.matrix[maxi], self.matrix[i]
        self.invert_matrix[i], self.invert_matrix[maxi] = self.invert_matrix[maxi], self.invert_matrix[i]
      cur = self.matrix[i][i]
      if cur == 0:
        self.determinant = 0
        continue
      self.determinant *= cur
      self.matrix[i] = [z / cur for z in self.matrix[i]]
      self.b[i] /= cur
      self.invert_matrix[i] = [z / cur for z in self.invert_matrix[i]]
      for j in range(i + 1, self.n):
        cur = self.matrix[j][i]
        self.b[j] -= cur * self.b[i]
        self.matrix[j] = [self.matrix[j][z] - cur * self.matrix[i][z] for z in range(self.n)]
        self.invert_matrix[j] = [self.invert_matrix[j][z] - cur * self.invert_matrix[i][z] for z in range(self.n)]
    self.determinant *= sign
    print("Determinant = {}".format(self.determinant))
    for i in range(self.n)[::-1]:
      for j in range(0, i):
        self.invert_matrix[j] = [self.invert_matrix[j][z] - self.matrix[j][i] * self.invert_matrix[i][z] for z in range(self.n)]
        self.b[j] -= self.b[i] * self.matrix[j][i]
        self.matrix[j] = [self.matrix[j][z] - self.matrix[j][i] * self.matrix[i][z] for z in range(self.n)]
    self.matrix = matrix_copy
    if self.determinant == 0:
      self.b = b_copy
      print("No invert_matrix(((")
      return
    print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
    print("Answer by gaus method")
    for j in range(self.n - 1):
      print("%015.10f" % self.b[j], end=' |^| ')
    print("%015.10f" % self.b[self.n - 1])
    print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
```

```
    self.b = b_copy

  def relax_method(self):
    res = [0] * self.n
    t = [0] * self.n
    for k in range(self.max_iterations):
      for i in range(self.n):
        s1 = 0
        s2 = 0
        for j in range(i):
          s1 += self.matrix[i][j] * t[j]
        for j in range(i, self.n):
          s2 += self.matrix[i][j] * res[j]
        t[i] = res[i] + (self.omega / self.matrix[i][i]) * (self.b[i] - s1 - s2)
      d = 0
      for i in range(self.n):
        d += (res[i] - t[i]) * (res[i] - t[i])
      if d < self.accuracy:
        print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
        print("Answer by relax method")
        for j in range(self.n - 1):
          print("%015.10f" % t[j], end=' |^| ')
        print("%015.10f" % t[self.n - 1])
        print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
        return t
      for i in range(self.n):
        res[i] = t[i]
    print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
    print("Answer by relax method")
    for j in range(self.n - 1):
      print("%015.10f" % res[j], end=' |^| ')
    print("%015.10f" % res[self.n - 1])
    print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
    return res


def out_matrix(a):
  if a is None:
    return
  n = len(a)
  if n > 20:
    print('too much size')
    return
  print("##################################################")
  for i in range(n):
    for j in range(n - 1):
      print("%015.10f" % a[i][j], end=' |^| ')
    print("%015.10f" % a[i][n - 1])
  print("##################################################")


def mul_matrix(a, b):
  n = len(a)
  res = []
  for i in range(n):
    res.append([])
    for j in range(n):
      res[i].append(0)
      for k in range(n):
        res[i][j] += a[i][k] * b[k][j]
  return res


def out_usage():
  print("##Usage##")
```

```
    print("First argument - 'static' or 'dynamic'")
    print("Second argument - matrix order")
    print("Third argument - path_to_file if static, starting number if dynamic")


def main():
    try:
        if sys.argv[1] == 'static':
            matrix = Test('static', int(sys.argv[2]), sys.argv[3])
        elif sys.argv[1] == 'dynamic':
            matrix = Test('dynamic', int(sys.argv[2]), float(sys.argv[3]))
        else:
            out_usage()
            exit(0)
    except:
        out_usage()
        print("Type parameters")
        a = input().split()
        if a[0] == 'static':
            matrix = Test('static', int(a[1]), a[2])
        elif a[0] == 'dynamic':
            matrix = Test('dynamic', int(a[1]), float(a[2]))
        else:
            out_usage()
            exit(0)
    matrix.gaus()
    matrix.relax_method()
    out_matrix(matrix.get_invert_matrix())


if __name__ == "__main__":
    main()
```

Тестирование

Тестирование проводится на наборах СЛАУ из приложения 1-11 и приложения 2-5 и тестах, которые я придумал сам.
Результат работы на невырожденных матрицах сравниваем с точным ответом, полученном на сайте wolframalpha.com.

Например для системы: $$\begin{array}{r} 24x_1 + 6x_2 + -12x_3 = 16 \\ 6x_1 + 33x_2 + 6x_3 = 8 \\ -12x_1 + 6x_2 + 24x_3 = 4 \end{array}$$

Матрица является невырожденной и положительноопределённой, значит можно применять и метод Гаусса, и метод верхней релаксации.

```
##Usage##
First argument - 'static' or 'dynamic'
Second argument - matrix order
Third argument - path_to_file if static, starting number if dynamic
Type parameters
static 3 StaticTests/test7.txt
Determinant = 11664.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Answer by gaus method
0001.0370370370  |^|  -000.0740740741  |^|  0000.7037037037
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Answer by relax method
0001.0370347614  |^|  -000.0740778166  |^|  0000.7037016466
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Invert Matrix:
##########################################
0000.0648148148  |^|  -000.0185185185  |^|  0000.0370370370
-000.0185185185  |^|  0000.0370370370  |^|  -000.0185185185
0000.0370370370  |^|  -000.0185185185  |^|  0000.0648148148
##########################################
```

Решение системы методом Гаусса с выбором главного элемента полностью совпадает с решением, которое выдаёт wolframalfa. Решение методом верхней релаксации за 20 итераций достигает точности 10^{-9}

Случай, когда матрица вырождена, программа также обрабатывает корректно:

$2x_1 - x_2 + 3x_3 + 4x_4 = 5$

$4x_1 - 2x_2 + 5x_3 + 6x_4 = 7$

$6x_1 - 3x_2 + 7x_3 + 8x_4 = 9$

$8x_1 - 4x_2 + 9x_3 + 10x_4 = 11$

```
##Usage##
First argument - 'static' or 'dynamic'
Second argument - matrix order
Third argument - path_to_file if static, starting number if dynamic
Type parameters
static 4 StaticTests/test3.txt
Determinant = 0
No invert_matrix(((
No such matrix
```

Ещё один тест: $\begin{matrix} x_1 + 2x_2 + 4x_3 = 31 \\ 5x_1 + x_2 + 2x_3 = 29 \\ 3x_1 - 1x_2 + x_3 = 10 \end{matrix}$

```
static 3 StaticTests/test4.txt
Determinant = -27.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Answer by gaus method
0003.0000000000  |^|  0004.0000000000  |^|  0005.0000000000
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Answer by relax method
000000000000nan  |^|  000000000000nan  |^|  000000000000nan
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Invert Matrix:
##########################################
-000.1111111111  |^|  0000.2222222222  |^|  0000.0000000000
-000.0370370370  |^|  0000.4074074074  |^|  -000.6666666667
0000.2962962963  |^|  -000.2592592593  |^|  0000.3333333333
##########################################
```

Решение методом Гаусса и определитель совпадают с решением wolframalpha. Метод верхней релаксации неприменим, так как матрица не является положительноопределённой.

Выводы

Метода Гаусса находит решение с высокой точностью, особенно, если использовать языки программирования с точной арифметикой.

Метод верхней релаксации даёт решение с высокой точностью уже всего через $10 - 20$ итераций, что будет даёт значительную выгоду по сравнению с методами Гаусса. Однако множество применимости этого метода сильно уже, чем у методов Гаусса.