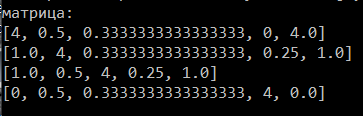
1. **Раздел А**

Исходная матрица с приписанным справа столбцом b

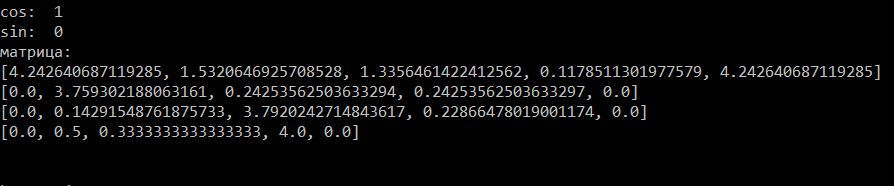


Значения синусов и косинусов на каждом шаге алгоритма(для матрицы порядка N=4):

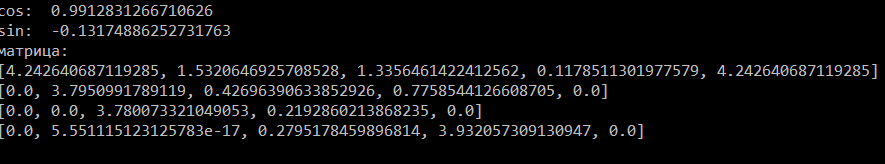
|  |  |  |
| --- | --- | --- |
| номера вращаемых строк | cos | sin |
| 1,2 | 0.9701425001453319 | -0.24253562503633297 |
| 1,3 | 0.9718253158075499 | -0.2357022603955158 |
| 1,4 | 1 | 0 |
| 2,3 | 0.9992781554489787 | -0.03798905161341708 |
| 2,4 | 0.9912831266710626 | -0.13174886252731763 |
| 3,4 | 0.9972772229253439 | -0.07374374979829819 |

Промежуточные матрицы с очередным занулённым столбцом:

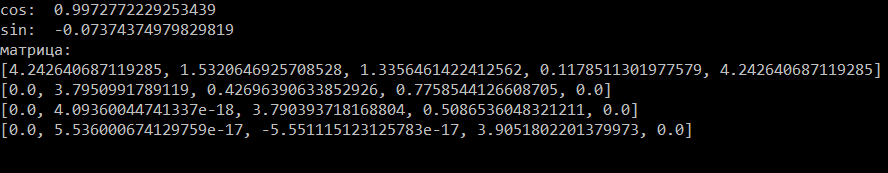
зануление 1-го столбца



зануление 2-го столбца



зануление 3-го столбца



Решение:



1. **Раздел Б**

Время расчёта и погрешность при различных N

|  |  |  |
| --- | --- | --- |
| N | Время расчёта | Абсолютная погрешность |
| 4 | 0.000999212 | 0.0 |
| 10 | 0.00199913 | 0.0 |
| 50 | 0.0779290 | 4.44089e-16 |
| 100 | 0.609439 | 0.0 |
| 200 | 4.56880 | 1.77635e-15 |
| 500 | 71.0466 | 2.66453e-15 |

1. **Исходный код**

import time

import math

# Вывод матрицы

def vivod(A):

print('матрица:')

for k in A:

print(k)

print('\n')

# генерация первоначальной квадратной матрицы A размера NxN

def generate(N,k=9):

A = []

K = N // 2

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

res = []

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

if i == j:

res.append(min(N,10))

elif abs(i-j) > K:

res.append(0)

else:

res.append(1/j)

A.append(res)

return A

# формирование вектора правых частей b

def vector\_b(A,N,k=9):

x = []

for v in range(1,N+1): # здесь индексы начинаем с 1 для удобства(в задании они указаны с 1)

if v == 1:

x.append(1)

elif v == (1+k):

x.append(1)

elif v == (1+5\*k):

x.append(1)

else:

x.append(0)

print('x: ',x)

b = []

for i in range(len(A)):

res = 0 # значение одного элемента вектора b

v = 0

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

#print(A[i][j])

res += A[i][j] \* x[v]

v += 1

#print('res: ',res)

b.append(res)

return [b,x]

# Вращение Гивенса

def givens(a,b):

if b == 0:

c = 1

s = 0

else:

if abs(b)>abs(a):

t = -a/b

s = 1/(math.sqrt(1+t\*t))

c = s\*t

else:

t = -b/a

c = 1/(math.sqrt(1+t\*t))

s = c\*t

return [c,s]

# Умножение на матрицу Гивенса

def row\_rot(A,k,j,c,s): # i,k - индексы строк матрицы, которые будут изменяться

q = len(A[0]) # кол-во столбцов матрицы

for i in range(q):

t1 = A[j][i]

t2 = A[k][i]

A[j][i] = t1\*c + t2\*s

A[k][i] = t2\*c - t1\*s

return A

# приписываем к матрице A справа столбец b

def Add\_b(A,b,N):

v = 0

A2 = []

for st in A:

res = st

res.append(b[v])

A2.append(res)

v += 1

return A2

# QR разложение с Гауссовским порядком обхода элементов(мой вариант)

def gauss(A,N):

for k in range(0,N-1):

for j in range(1+k,N):

c,s = givens(A[k][k],A[j][k])

print('k: ',k,' j: ',j)

print('cos: ',c)

print('sin: ',s)

A = row\_rot(A,k,j,c,s)

# решение системы линейных уравнений после приведения матрицы к верхнетреугольному виду

def syst(A,N):

b = []

# отделяем вектор b от матрицы A

for i in range(len(A)):

b.append(A[i][-1])

A[i] = A[i][:-1]

X = [] # в список будем добавлять составляющие вектора X

x = 0

k = -1

for st in reversed(A):

x = b[k]

for i in range(len(st)):

# умножаем на bool(k != (-i-1)) чтобы не вычитался диагональный элемент

if k < -1: # на первом шаге ничего не вычитаем

x = x - st[-i-1]\*bool(k != (-i-1))\*X[k+1]

x = x / st[k]

X.append(x)

k -= 1

X = X[::-1] # так как мы добавляли в конец то вектор x вышел перевернутым

return X

# считаем абсолютную погрешность на основе второй нормы

def pogr(X\_true,x\_calcul):

A\_true = 0

A\_calcul = 0

for v in X\_true:

A\_true += abs(v)

for k in x\_calcul:

A\_calcul += abs(k)

return abs(A\_true-A\_calcul)

# тест

def test(N):

t1 = time.time()

A = generate(N) # генерируем матрицу размера N

vivod(A)

b,x = vector\_b(A,N) # вычисляем вектор b

print('b: ',b)

print('Вектор x который должен получиться: ',x)

A = Add\_b(A,b,N) # столбец b припписываем к матрице A

vivod(A)

gauss(A,N) # QR разложение

X = syst(A,N) # решаем верхнетреугольную систему

print('Посчитанное решение X: ',X)

delta = pogr(x,X) # абсолютная погрешность

print('абсолютная погрешность: ',delta)

t2 = time.time()

print('время вычисления при N=',N,': ',t2-t1)

test(4)

#test(10)

#test(50)

#test(100)