Московский Авиационный Институт

(Государственный Технический Университет)

Факультет прикладной математики и физики.

Кафедра вычислительной математики и программирования.

**Лабораторная работа №1**

**по курсу «Численные методы»**

VI семестр.

Студент Баскаков О.А.

Группа 08-306

Вариант 2

Москва, 2011.

# Постановка задачи

1.1. Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.



1.2. Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.



1.3. Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.



1.4. Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

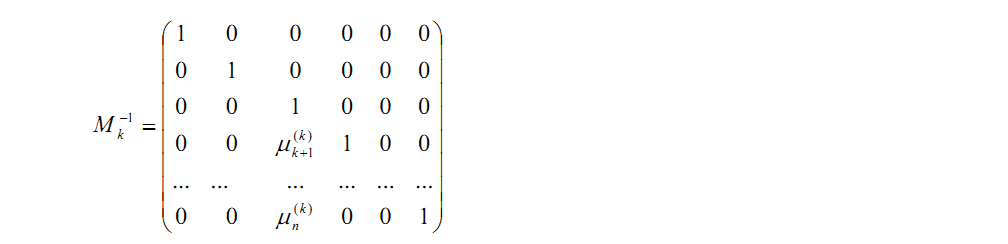
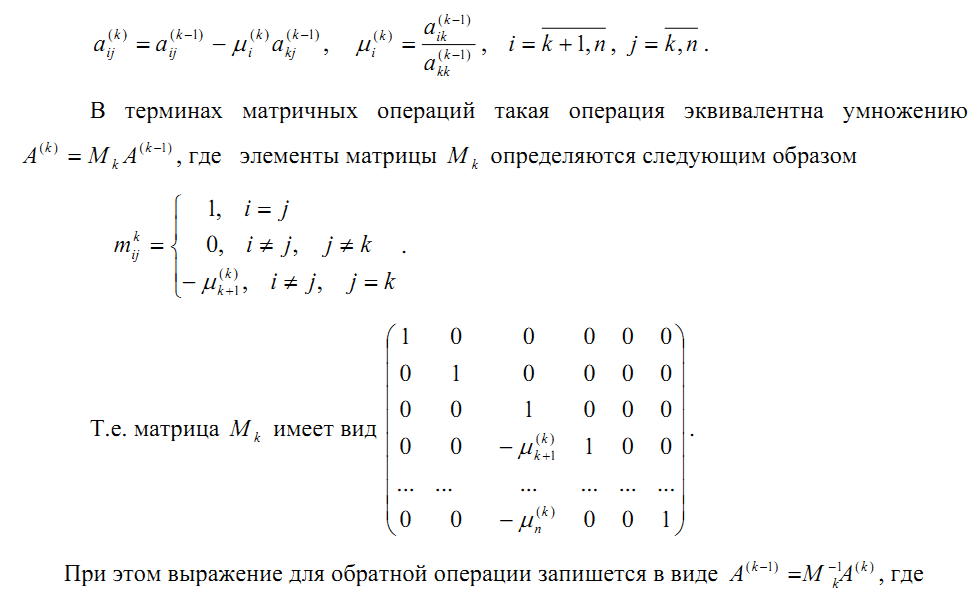
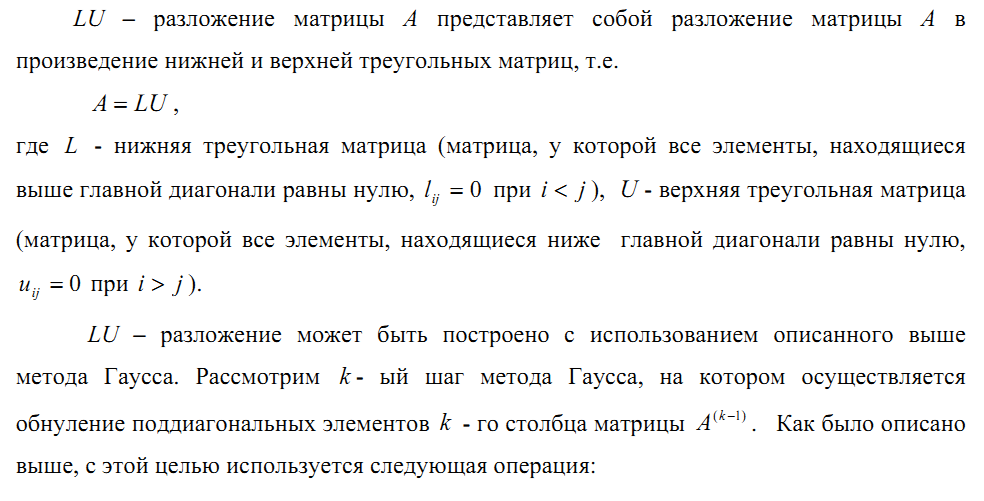


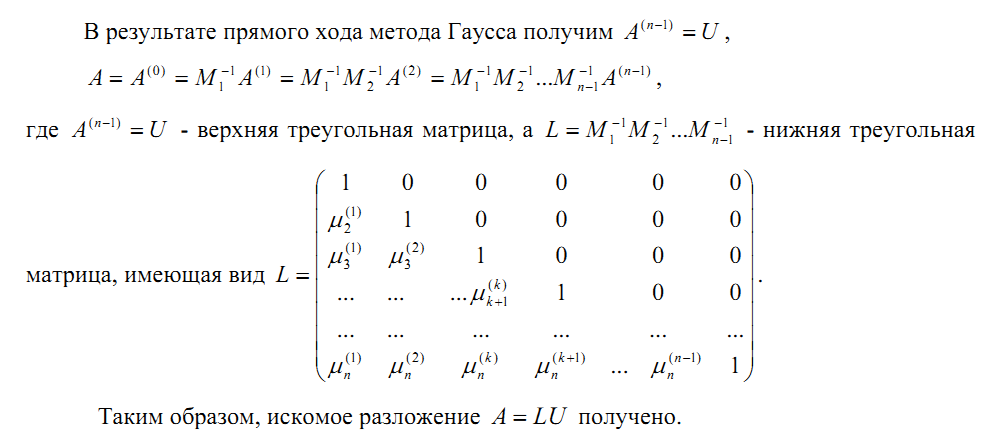
1.5. Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.



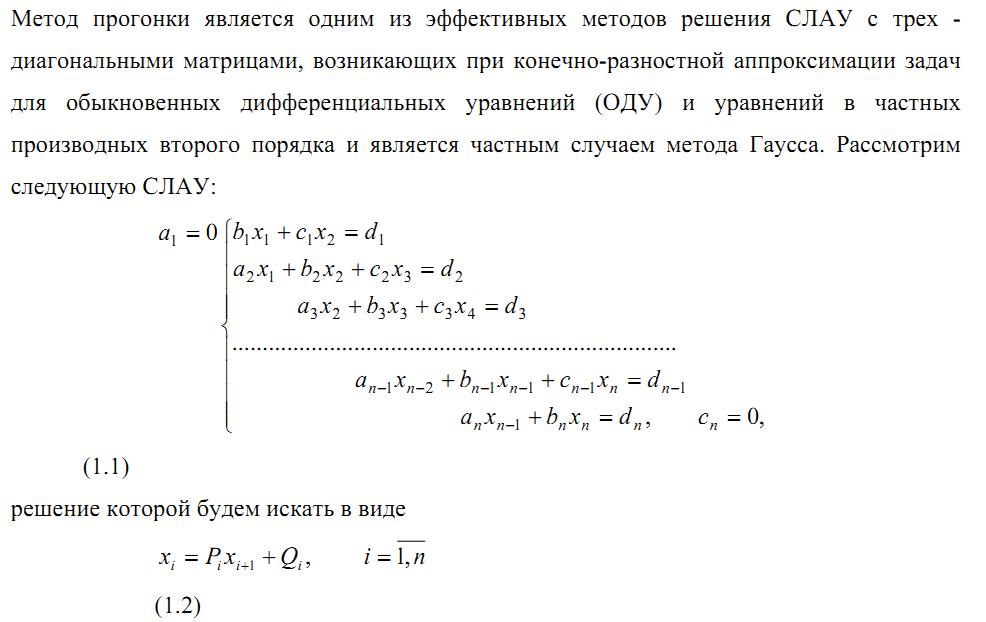
# Теория

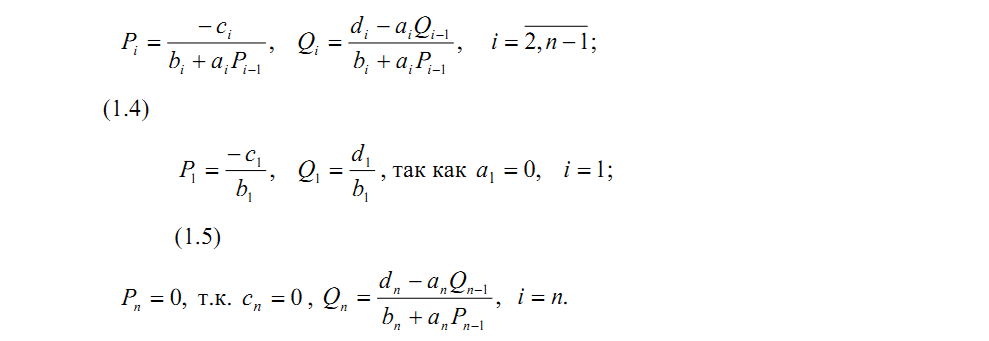
## 1. LU-разложение

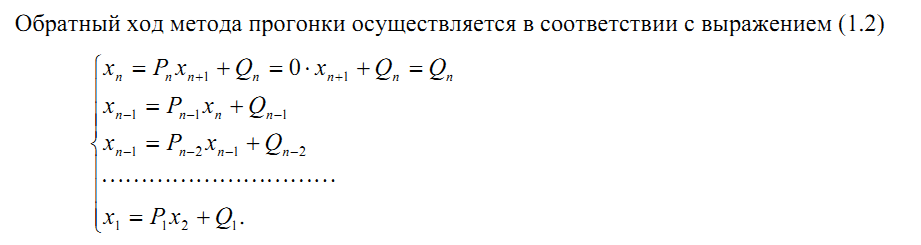




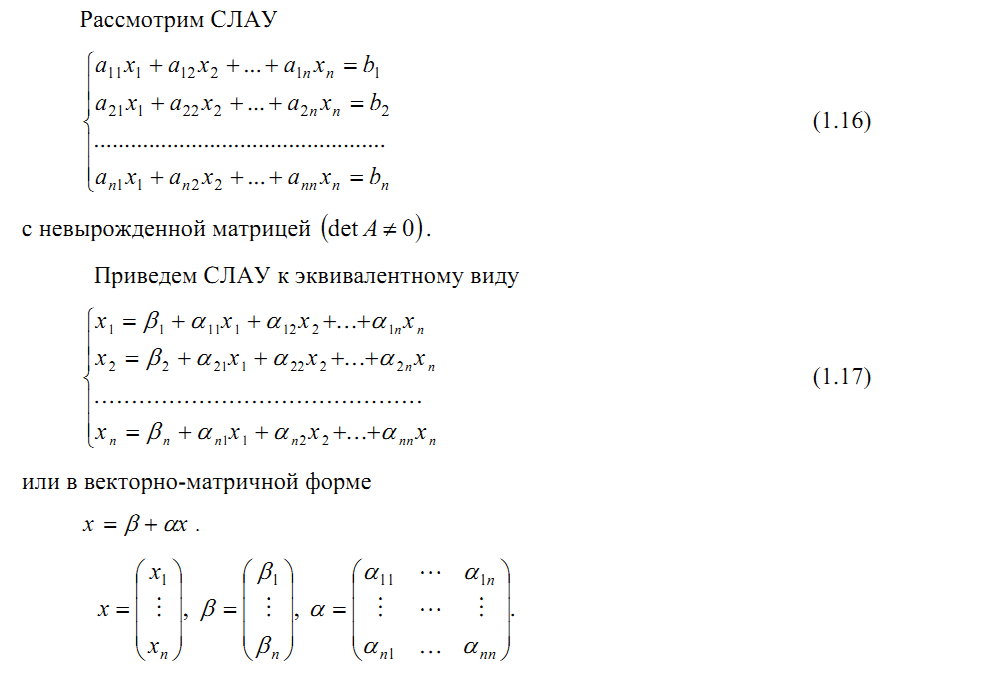
## 2. Метод прогонки

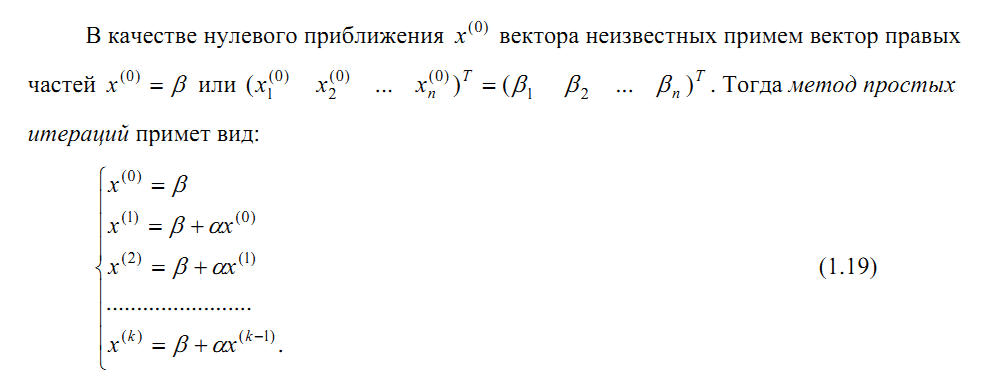


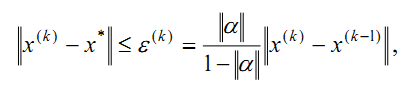


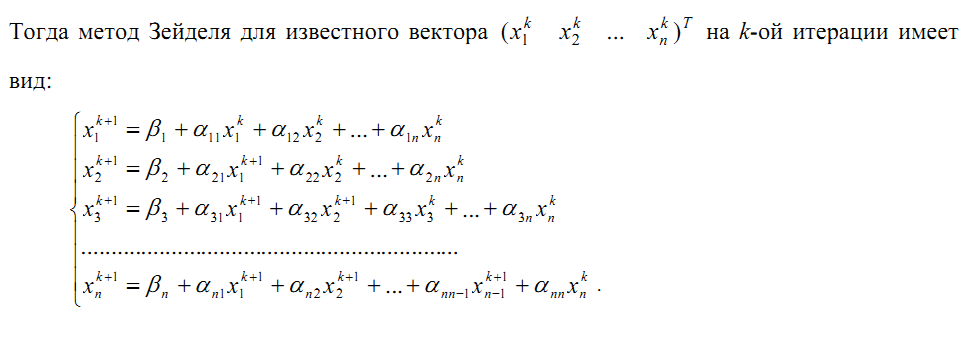


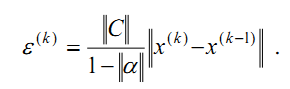
## 3. Метод простых итераций и метод Зейделя



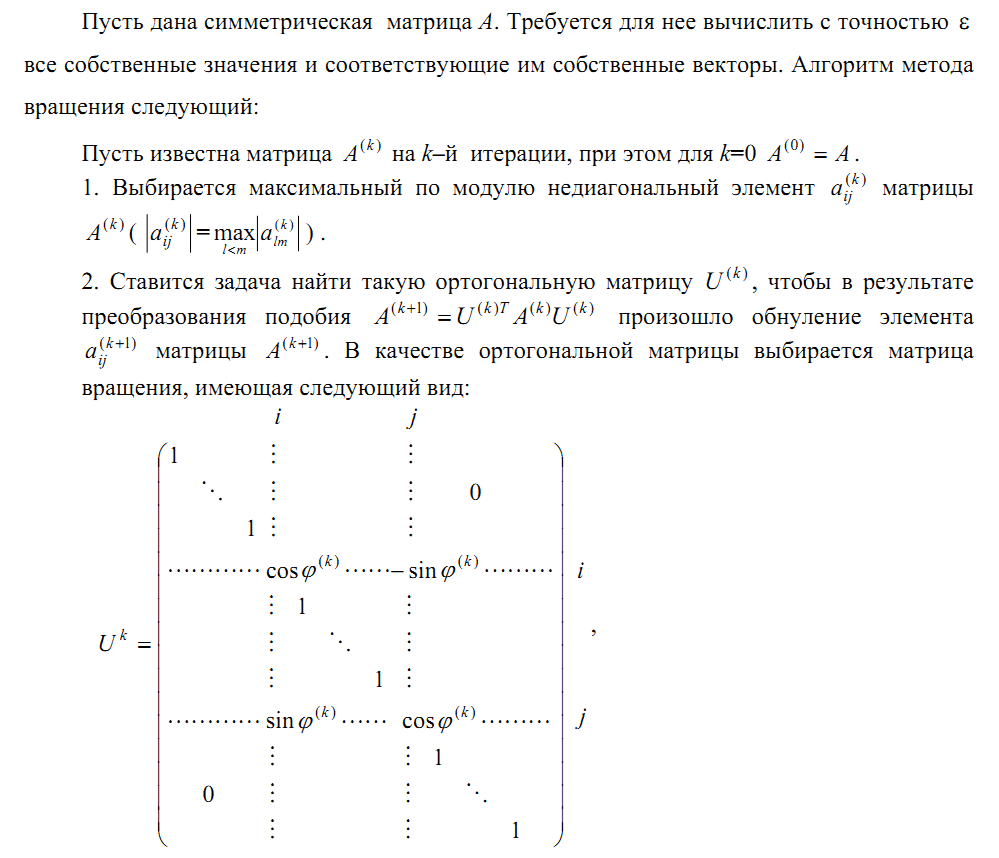


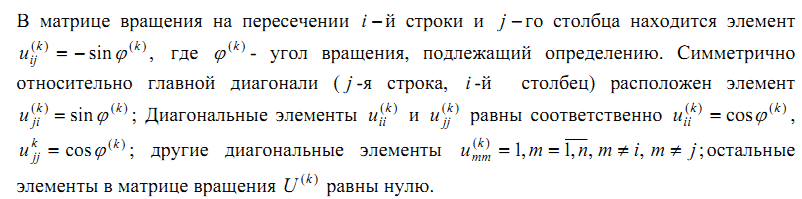


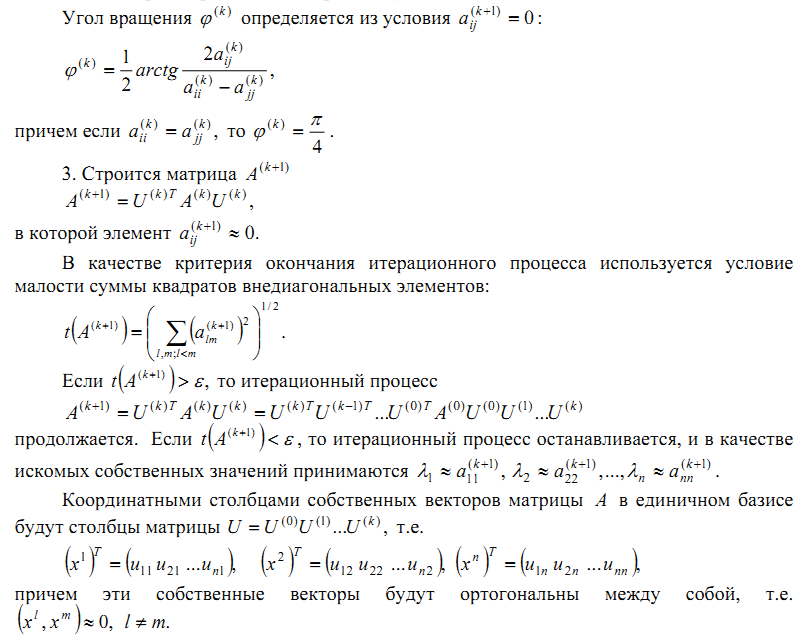




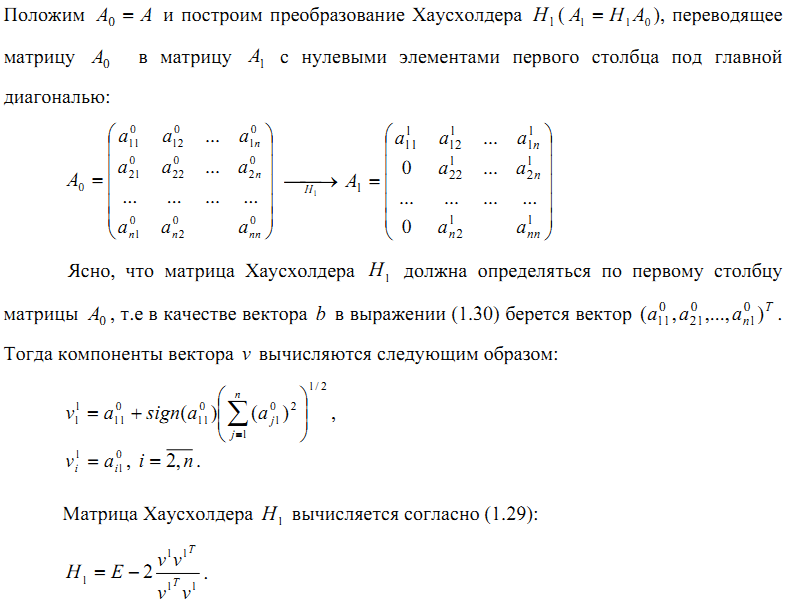
## 4. Метод вращений

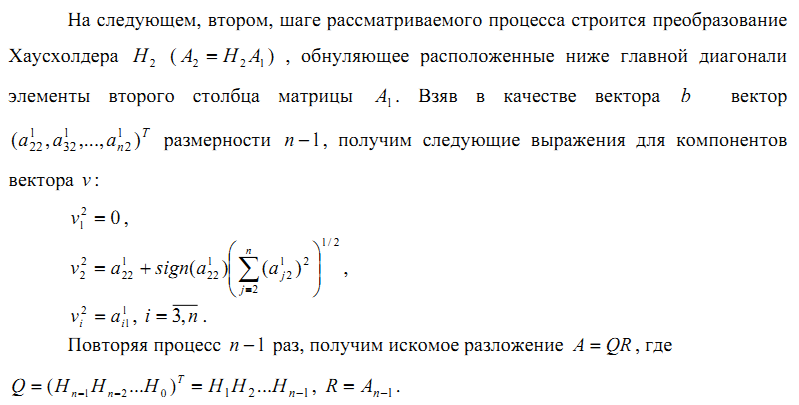


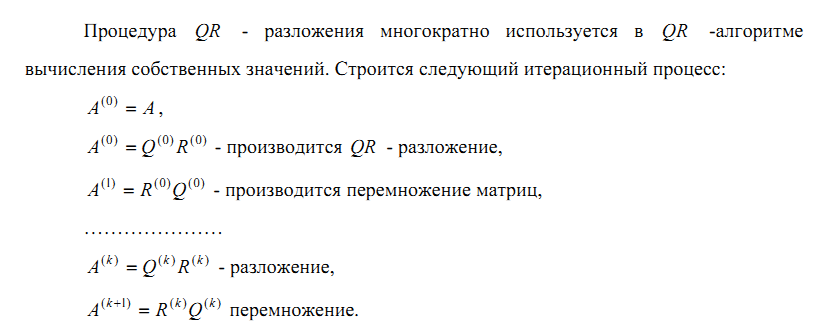




## 5. QR-разложение







# Исходный код на языке Python

## 1. LU-разложение

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from copy import copy, deepcopy

class Matrix:

def \_\_init\_\_(self,f = "",m = 0,n = 0):

if f:

input1 = open(f, 'r')

self.M = []

self.b = []

self.p = 0

self.p\_count = 0

for str1 in input1.readlines():

try:

z = [float(x) for x in str1.split(' ')]

self.b.append(z.pop()) #deleting b

self.M.append(z)

except:

continue

self.m = len(self.M)

self.n = len(self.M[0])

elif m and n:

l = [0] \* n

self.M = [copy(l) for i in range(m)]

self.m = m

self.n = n

for i in range(min(n, m)):

self.M[i][i] = 1

else:

self.M = []

self.m = 0

self.n = 0

def \_\_getitem\_\_(self,i):

return self.M[i]

def \_\_setitem\_\_(self,i,y):

self.M[i] = y

def \_\_mul\_\_(self, M2):

M1 = self

result = Matrix(m = M1.m, n = M2.n)

for i in range(result.m):

for j in range(result.n):

result[i][j] = 0

for k in range(M1.n):

result[i][j] += M1[i][k] \* M2[k][j]

return result

def transponate(self):

res = Matrix("", m = self.n, n = self.m)

for i in range(self.n):

for j in range(self.m):

res[i][j] = self[j][i]

return res

def pr(self):

print('Matrix ' + str(self.m)+ 'x' + str(self.n) + " :")

for xx in self.M :

for x in xx:

print("\t%.2f"%x, end = ' ')

print()

print("-----")

return

def det(self):

"""Calculating det"""

det = 1

for i in range(self.n):

det\*=self.U[i][i]

if self.p\_count %2:

det = -det

return det

def swap\_rows(self, k, s):

"""Swap rows k and s"""

z = self[k]

self[k] = self[s]

self[s] = z

def swap\_cols(self, k, s):

"""Swap cols k and s"""

for j in range(self.n):

z = self[j][k]

self[j][k] = self[j][s]

self[j][s] = z

def build\_LU(self):

n = self.n

A = Matrix()

A.M = deepcopy(self.M)

A.n = self.n

A.m = self.m

self.p = [x for x in range(n)]

self.p\_count = 0

U = Matrix("", n, n)

L = Matrix("", n, n)

for k in range(n-1):

# Find max

s = k

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

if (abs(A[s][k])< abs(A[j][k])) : s = j

A.swap\_rows(k, s)

L.swap\_rows(k, s)

L.swap\_cols(k, s)

# construct vector p

z = self.p[k] #equal k

self.p[k] = self.p[s]

self.p[s] = z

if (k!=s): self.p\_count += 1

for i in range(k+1, n): #col number

koef = A[i][k]/A[k][k]

A.M[i] =[(xx-xo\*koef)for(xx,xo) in zip(A.M[i],A.M[k])]

L[i][k] = koef

self.L = L

self.U = A

def solve(self,b):

n = len(b)

z = [0]\*n

z[0] = b[0]

for i in range(1,n):

z[i] = b[i]

for j in range(i):

z[i] -= self.L[i][j]\*z[j]

x = [0]\*n

x[n-1] = z[n-1]

i = n-1

while i > -1 :

x[i] = z[i]

for j in range(i+1,n):

x[i] -= self.U[i][j]\*x[j]

x[i] /= self.U[i][i]

i-=1

return x

def inverse(self):

E = Matrix("",self.n,self.n)

A = Matrix("",self.n, self.n)

for i in range(self.n):

e = [0]\*self.n

e[i] = 1.0

A[self.p[i]] = self.solve(e)

return A.transponate()

def shift\_b(self, b):

v = [0]\*self.n

for i in range(self.n):

v[i] = b[self.p[i]]

return v

def main():

m = Matrix("lab1\_1.in")

print ("Input matrix:")

m.pr()

print ("U:")

m.build\_LU()

m.U.pr()

print ("L:")

m.L.pr()

print ("L\*U:")

(m.L\*m.U).pr()

m1 = m.inverse()

print ("Solve:")

x = m.solve(m.shift\_b(m.b))

x = [round(xx, 2) for xx in x]

print("x = ", x)

print ("Determinant:")

print (m.det())

print ("A^-1:")

m1.pr()

print ("A\*A^-1:")

(m\*m1).pr()

if (\_\_name\_\_ == "\_\_main\_\_"):

main()

## 2. Метод прогонки

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

class Tridiagonal\_Matrix:

def \_\_init\_\_(self, f = ""):

if f:

inp = open(f, 'r')

self.a = []

self.b = []

self.c = []

self.d = []

for line in inp.readlines():

(a, b, c, d) = ( float(x) for x in line.split() )

self.a.append(a)

self.b.append(b)

self.c.append(c)

self.d.append(d)

self.n = len(self.a)

else:

print ("File wasn't specified")

exit(1)

return None

def solve(self):

"""Method progonki"""

a = self.a

b = self.b

c = self.c

d = self.d

n = self.n

P = []

Q = []

P.append(-c[0]/b[0])

Q.append( d[0]/b[0])

for i in range(1, n):

P.append( -c[i] / (b[i]+a[i]\*P[i-1]) )

Q.append( (d[i] - a[i]\*Q[i-1]) / (b[i] + a[i]\*P[i-1]) )

x = [0]\*n

x[n-1] = Q[n-1]

for i in range(n-2, -1, -1):

x[i] = P[i]\*x[i+1] + Q[i]

return x

def main():

e = Tridiagonal\_Matrix("lab1\_2.in")

print("a =",e.a)

print("b =",e.b)

print("c =",e.c)

print("d =",e.d)

print()

x = e.solve()

x = [round(z,2) for z in x]

print("x = ", x)

return

if (\_\_name\_\_ == "\_\_main\_\_"):

main()

## 3. Метод простых итераций и метод Зейделя

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from copy import copy, deepcopy

class Matrix:

def \_\_init\_\_(self,f = "",m = 0,n = 0):

self.nrm = 0

if f:

input1 = open(f, 'r')

self.M = []

self.b = []

self.p = 0

for str1 in input1.readlines():

try:

z = [float(x) for x in str1.split(' ')]

self.b.append(z.pop()) #deleting b

self.M.append(z)

except:

continue

self.m = len(self.M)

self.n = len(self.M[0])

elif m and n:

l = [0] \* n

self.M = [copy(l) for i in range(m)]

self.m = m

self.n = n

for i in range(min(n, m)):

self.M[i][i] = 1

else:

self.M = []

self.m = 0

self.n = 0

def \_\_getitem\_\_(self,i):

return self.M[i]

def \_\_setitem\_\_(self,i,y):

self.M[i] = y

def \_\_neg\_\_(self):

res = Matrix("",self.m,self.n)

for i in xrange(self.m):

for j in xrange(self.n):

res[i][j] = -self.M[i][j]

def \_\_len\_\_(self):

return len(self.M)

def \_\_add\_\_(self,y):

res = Matrix("",self.m,self.n)

for i in range(self.m):

for j in range(self.n):

res[i][j] = self.M[i][j] + y[i][j]

return res

def \_\_sub\_\_(self,y):

res = Matrix("",self.m,self.n)

for i in range(self.m):

for j in range(self.n):

res[i][j] = self.M[i][j] - y.M[i][j]

return res

def \_\_mul\_\_(self, M2):

M1 = self

result = Matrix(m = M1.m, n = M2.n)

for i in range(result.m):

for j in range(result.n):

result[i][j] = 0

for k in range(M1.n):

result[i][j] += M1[i][k] \* M2[k][j]

return result

def norm(self):

tp = 0.

mx = [0.,0.]

for i in range(self.m):

s = 0.

for j in range(self.n):

s+=self.M[i][j]

#~ print self.M[i][j],

if mx[0] < s:

mx = [s,i]

#~ print s

self.nrm = mx[0]

return self.nrm

def transponate(self):

res = Matrix("", m = self.n, n = self.m)

for i in range(self.n):

for j in range(self.m):

res[i][j] = self[j][i]

self.M = res.M

self.n = res.n

self.m = res.m

return self

def pr(self):

print('Matrix ' + str(self.m)+ 'x' + str(self.n) + " :")

for xx in self.M :

for x in xx:

print("\t %.2f"%x, end = ' ')

print()

print("-----")

return

class Iteration(Matrix):

def \_\_init\_\_(self, f="", m=0, n=0):

inp = open(f, 'r')

line = inp.readline().split()

self.eps = float(line[-1])

Matrix.\_\_init\_\_(self, f, m, n)

def solve(self):

b = self.b

A = self.M

alpha = Matrix("",self.n,self.n)

beta = Matrix("",self.n,1)

for i in range(0, beta.m):

beta[i][0] = b[i]/A[i][i]

for i in range(0, alpha.n):

for j in range(0, alpha.n):

if i==j:

alpha[i][j] = 0

else:

alpha[i][j] = - A[i][j] / A[i][i]

xk = beta

eps\_k = 1

norma = alpha.norm()

counter = 0

while (eps\_k > self.eps):

xk1 = beta + alpha\*xk

if norma < 1:

eps\_k = norma/(1-norma) \* (xk1-xk).norm()

else:

eps\_k = (xk1-xk).norm()

counter+=1

if counter > 100 : break

xk = xk1

print ("Simple calculate %d iterations; " %counter)

return xk

def solve\_zeidel(self):

b = self.b

a = self.M

alpha = Matrix("",self.m,self.n)

beta = Matrix("",self.n,1)

for i in range(self.m):

beta.M[i] = [b[i]/a[i][i]]

for j in range(self.n):

if i != j:

alpha.M[i][j] = - a[i][j] / a[i][i]

else:

alpha.M[i][j] = 0

C = Matrix("",self.m,self.n)

for i in range(self.m):

for j in range(self.n):

if i >= j:

C.M[i][j] = alpha.M[i][j]

E = Matrix("",self.m,self.n)

for i in range(E.n):

E.M[i][i] = 1

ek = 1.

count = 0

xk = Matrix("",self.n,1)

xkm = deepcopy(beta)

while abs(ek) >= self.eps :

tp = Matrix("",self.n,1)

for i in range(self.m):

tp[i][0] = 0

for j in range(i):

tp[i][0]+=alpha[i][j]\*xk[j][0]

for j in range(i,self.n):

tp[i][0]+=alpha[i][j]\*xkm[j][0]

xk[i][0] = beta[i][0] + tp[i][0]

if alpha.norm() < 1:

ek = C.norm()/(1-alpha.norm()) \* (xk-xkm).norm()

else:

ek = (xk-xkm).norm()

count+=1

xkm = deepcopy(xk)

print ("Zeidel finish in %d iteratrions;" %count)

return xk

def main():

m = Iteration("lab1\_3.in")

m.solve().pr()

m.solve\_zeidel().pr()

return

main()

## 4. Метод вращений

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from math import sqrt,sin,cos,atan,pi

from copy import copy, deepcopy

class Matrix:

def \_\_init\_\_(self,f = "",m = 0,n = 0):

self.nrm = 0

if m and n:

l = [0] \* n

self.M = [copy(l) for i in range(m)]

self.m = m

self.n = n

for i in range(min(n, m)):

self.M[i][i] = 1

else:

self.M = []

self.m = 0

self.n = 0

def \_\_getitem\_\_(self,i):

return self.M[i]

def \_\_setitem\_\_(self,i,y):

self.M[i] = y

def \_\_neg\_\_(self):

res = Matrix("",self.m,self.n)

for i in xrange(self.m):

for j in xrange(self.n):

res[i][j] = -self.M[i][j]

def \_\_len\_\_(self):

return len(self.M)

def \_\_add\_\_(self,y):

res = Matrix("",self.m,self.n)

for i in range(self.m):

for j in range(self.n):

res[i][j] = self.M[i][j] + y[i][j]

return res

def \_\_sub\_\_(self,y):

res = Matrix("",self.m,self.n)

for i in range(self.m):

for j in range(self.n):

res[i][j] = self.M[i][j] - y.M[i][j]

return res

def \_\_mul\_\_(self, M2):

M1 = self

result = Matrix(m = M1.m, n = M2.n)

for i in range(result.m):

for j in range(result.n):

result[i][j] = 0

for k in range(M1.n):

result[i][j] += M1[i][k] \* M2[k][j]

return result

def norm(self):

tp = 0.

mx = [0.,0.]

for i in range(self.m):

s = 0.

for j in range(self.n):

s+=self.M[i][j]

if mx[0] < s:

mx = [s,i]

self.nrm = mx[0]

return self.nrm

def transponate(self):

res = Matrix("", m = self.n, n = self.m)

for i in range(self.n):

for j in range(self.m):

res[i][j] = self[j][i]

return res

def pr(self):

print('Matrix ' + str(self.m)+ 'x' + str(self.n) + " :")

for xx in self.M :

for x in xx:

print("\t%.2f"%x, end = ' ')

print()

print("-----")

return

class Rotations(Matrix):

def \_\_init\_\_(self,f="",m=0,n=0):

if f:

input1 = open(f, 'r')

self.M = []

self.b = []

self.p = 0

self.eps = float( input1.readline().split(' ').pop() )

for str1 in input1.readlines():

try:

z = [float(x) for x in str1.split(' ')]

self.M.append(z)

except:

continue

self.m = len(self.M)

self.n = len(self.M[0])

elif m and n:

l = [0] \* n

self.M = [copy(l) for i in range(m)]

self.m = m

self.n = n

for i in range(min(n, m)):

self.M[i][i] = 1

else:

self.M = []

self.m = 0

self.n = 0

def max\_nd(self):

a\_m = 0

i\_m = 0

j\_m = 0

for i in range(self.n):

for j in range(i):

if i!=j:

if (abs(self[i][j])>abs(a\_m)):

a\_m = abs(self[i][j])

i\_m = i

j\_m = j

return (a\_m, j\_m, i\_m)

def solve(self):

E = Matrix("",self.m,self.n)

Uk = Matrix("",self.m,self.n)

U\_es = []

A = Matrix("",self.m,self.n)

A.M = deepcopy(self.M)

Akp = A

eps\_k = 1

counter = 0

while abs(eps\_k) > self.eps:

counter+=1

if counter > 9000: break

Ak = Akp

(a, i, j) = Rotations.max\_nd(Ak)

if (Ak[i][i] - Ak[j][j]):

phi = 0.5 \* atan(2\*Ak[i][j]/(Ak[i][i] - Ak[j][j]))

else:

phi = pi/4.

Uk = deepcopy(E)

Uk[i][i] = cos(phi)

Uk[j][j] = cos(phi)

Uk[i][j] = -sin(phi)

Uk[j][i] = sin(phi)

U\_es.append(Uk)

Akp = Uk.transponate()\*Ak\*Uk

eps\_k = 0.

for m in range(self.n):

for l in range(m):

eps\_k += Akp[l][m]\*Akp[l][m]

eps\_k = sqrt(eps\_k)

#reduce manual

for i in range(1, len(U\_es)):

U\_es[0] = U\_es[0]\*U\_es[i]

tp = U\_es[0].transponate()

print("finish in %d iters."%counter)

lambdas = []

vectors = []

for i in range(Akp.n):

lambdas.append(Akp[i][i])

vectors.append(tp[i])

return [lambdas,vectors]

def main():

r = Rotations("lab1\_4.in")

l, v = r.solve()

for i in range(len(v)):

v[i] = [round(xx,3) for xx in v[i] ]

for (l\_i, v\_i) in zip(l, v): print(" l = \t%.2f\t v = "%l\_i, v\_i)

if (\_\_name\_\_ == "\_\_main\_\_"):

main()

## 5. QR-разложение

#! /usr/bin/python3.1

# -\*- coding: utf-8 -\*-

from math import sqrt,sin,cos,atan,pi

from copy import copy, deepcopy

from functools import reduce

def product(L):

mul = lambda x, y: x\*y

return reduce(mul, L)

def sign(x):

if (x<0): return -1

elif(x>0): return 1

else: return 0

class Matrix:

def \_\_init\_\_(self,f = "",m = 0,n = 0):

if f:

self.M = None

elif m and n:

l = [0] \* n

self.M = [copy(l) for i in range(m)]

self.m = m

self.n = n

else:

self.M = []

self.m = 0

self.n = 0

def \_\_getitem\_\_(self,i):

return self.M[i]

def \_\_setitem\_\_(self,i,z):

self.M[i] = z

def \_\_len\_\_(self):

return len(self.M)

def \_\_neg\_\_(self):

M1 = self

res = Matrix("",self.m,self.n)

res.M = [ [(-M1\_ij) for M1\_ij in M1\_i]

for M1\_i in M1.M]

return res

def \_\_add\_\_(self, M2):

M1 = self

res = Matrix("",self.m,self.n)

res.M = [ [(M1\_ij + M2\_ij) for (M1\_ij, M2\_ij) in zip(M1\_i,M2\_i)]

for (M1\_i, M2\_i) in zip(M1.M,M2.M)]

return res

def \_\_sub\_\_(self, M2):

M1 = self

res = Matrix("",self.m,self.n)

res.M = [ [(M1\_ij - M2\_ij) for (M1\_ij, M2\_ij) in zip(M1\_i,M2\_i)]

for (M1\_i, M2\_i) in zip(M1.M,M2.M)]

return res

def \_\_mul\_\_(self, M2):

M1 = self

if type(M2) in [int,float]:

res = Matrix("",M1.m, M1.n)

res.M = [ [(M2\*xx) for xx in M1\_i] for M1\_i in M1]

return res

res = Matrix(m = M1.m, n = M2.n)

res.M = [[ sum([ (M1[i][k] \* M2[k][j]) for k in range(M1.n)])

for j in range(res.n)]

for i in range(res.m)]

return res

def transponate(self):

res = Matrix("", m = self.n, n = self.m)

res.M = [ [(self[j][i]) for j in range(res.n)]

for i in range(res.m)]

return res

def pr(self):

print('Matrix ' + str(self.m)+ 'x' + str(self.n) + " :")

for xx in self.M :

for x in xx:

print("\t %.2f"%x, end = ' ')

print()

print("-----")

return

def QRdecompositon(self):

n = self.n

E = Matrix("",self.m,self.n)

for i in range(E.n):

E[i][i] = 1

Ak = deepcopy(self)

Hes = []

for k in range(n):

vec = Matrix("",n,1)

for j in range(k,n):

vec[k][0] += Ak[j][k]\*\*2

vec[k][0] = sqrt(vec[k][0]) \* sign(Ak[k][k]) + Ak[k][k]

for i in range(k+1,n):

vec[i][0] = Ak[i][k]

vec\_t = vec.transponate()

tp = (vec\_t \* vec)[0][0]

tp2 = vec \* vec\_t

H = E - tp2\*(2.0/tp)

Hes.append(H)

Ak = H\*Ak

return [product(Hes),Ak]

class QR(Matrix):

def \_\_init\_\_(self,f="",m=0,n=0):

if f:

inp = open(f,"r")

s = inp.readline().split()

self.eps = float(s[-1])

self.M = []

self.b = []

self.p = 0

for i in inp.readlines():

s = [float(x) for x in i.split()]

self.M.append(s)

self.m = len(self.M)

self.n = len(self.M[0])

elif m and n:

l = [0] \* n

self.M = [copy(l) for i in range(m)]

self.m = m

self.n = n

else:

self.M = []

self.m = 0

self.n = 0

self.f = f

def solve(self):

eps = self.eps

n = self.n

m = self.m

Ak = deepcopy(self)

lambdas = []

diag = [False]\*n

l1 = [0 + 0j]\*n

l2 = [0 + 0j]\*n

l1p = [0 + 0j]\*n

l2p = [0 + 0j]\*n

count = 0

while 42 and (count < 9000) and (len(lambdas) < n):

Q,R = Ak.QRdecompositon()

Ak = R\*Q

for j in range(n):

s = sum( [Ak[i][j]\*\*2 for i in range(j+1,m)])

if sqrt(s) < eps:

if j < n-1 and not diag[j]:

lambdas.append(Ak[j][j] + 0j)

diag[j] = True

elif j == n-1 and len(lambdas) == n-1 and not diag[j]:

lambdas.append(Ak[j][j] + 0j)

diag[j] = True

else:

try: # if j+1 != n

s-= Ak[j+1][j]\*\*2

except:

continue

if sqrt(s) < eps:

det = (Ak[j][j]-Ak[j+1][j+1])\*\*2+4\*Ak[j+1][j]\*Ak[j][j+1]

b = Ak[j][j] + Ak[j+1][j+1]

if (det < 0):

det = sqrt(abs(det))

l1[j] = b/2.0 + 0.5j\*det

l2[j] = b/2.0 - 0.5j\*det

else:

det = sqrt(det)

l1[j] = (b + det)/2.0 + 0j

l2[j] = (b - det)/2.0 + 0j

if abs(l1[j]-l1p[j]) < eps and abs(l2[j]-l2p[j]) < eps and not diag[j] and not diag[j+1]:

lambdas.append(l1[j])

lambdas.append(l2[j])

diag[j] = True

diag[j+1] = True

else:

l1p = l1

l2p = l2

count += 1

return lambdas

def round\_c(c, n = 2):

im = c.imag

re = c.real

return round(re, n) + 1j\*round(im, n)

def main():

m = QR("lab1\_5.in")

print ("eps = ", m.eps)

lam = m.solve()

for xx in lam:

print(round\_c(xx))

if (\_\_name\_\_ == "\_\_main\_\_"):

main()

# Протокол

## 1. LU-разложение

oleg@debian:~/lab1\_release$ python3.1 1.1\_LUP.py

Input matrix:

Matrix 4x4 :

2.00 7.00 -8.00 6.00

4.00 4.00 0.00 -7.00

-1.00 -3.00 6.00 3.00

9.00 -7.00 -2.00 -8.00

-----

U:

Matrix 4x4 :

9.00 -7.00 -2.00 -8.00

0.00 8.56 -7.56 7.78

0.00 0.00 7.17 -9.91

0.00 0.00 0.00 8.92

-----

L:

Matrix 4x4 :

1.00 0.00 0.00 0.00

0.22 1.00 0.00 0.00

0.44 0.83 1.00 0.00

-0.11 -0.44 0.34 1.00

-----

L\*U:

Matrix 4x4 :

9.00 -7.00 -2.00 -8.00

2.00 7.00 -8.00 6.00

4.00 4.00 0.00 -7.00

-1.00 -3.00 6.00 3.00

-----

Solve:

x = [8.0, -3.0, 2.0, -3.0]

Check A\*x = b:

Matrix 4x1 :

-39.00

41.00

4.00

113.00

-----

Vector b:

[-39.0, 41.0, 4.0, 113.0]

Determinant:

-4924.0

A^-1:

Matrix 4x4 :

0.10 0.07 0.16 0.07

0.04 0.11 0.03 -0.05

-0.00 0.09 0.15 -0.02

0.08 -0.04 0.11 0.01

-----

A\*A^-1:

Matrix 4x4 :

1.00 -0.00 0.00 0.00

0.00 1.00 0.00 0.00

0.00 0.00 1.00 0.00

-0.00 0.00 0.00 1.00

-----

## 2. Метод прогонки

oleg@debian:~/lab1\_release$ cat lab1\_2.in

0 -11 -9 -122

5 -15 -2 -48

-8 11 -3 -14

6 -15 4 -50

3 6 0 42

oleg@debian:~/lab1\_release$ python3.1 1.2\_Progonka.py

x = [7.0, 5.0, 4.0, 6.0, 4.0]

oleg@debian:~/lab1\_release$ cat lab1\_2.in

0 10 5 -120

3 10 -2 -91

2 -9 -5 5

5 16 -4 -74

-8 16 0 -56

oleg@debian:~/lab1\_release$ python3.1 1.2\_Progonka.py

x = [-9.0, -6.0, 2.0, -7.0, -7.0]

oleg@debian:~/lab1\_release$ cat lab1\_2.in

0.0 -12.0 -10.0 11

23.0 -45.0 -21.0 12

34.0 -100.0 -32.0 13

45.0 -177.0 -43.0 14

56.0 -276.0 -54.0 15

67.0 -397.0 -65.0 16

78.0 -540.0 -76.0 17

89.0 -705.0 -87.0 18

100.0 -892.0 -98.0 19

111.0 -1101.0 0.0 20

oleg@debian:~/lab1\_release$ python3.1 1.2\_Progonka.py

x = [-0.55, -0.44, -0.24, -0.12, -0.07, -0.05, -0.03, -0.03, -0.02, -0.02]

## 3. Метод простых итераций и метод Зейделя

oleg@debian:~/lab1\_release$ cat lab1\_3.in

eps = 0.001

19 -4 -9 -1 100

-2 20 -2 -7 -5

6 -5 -25 9 34

0 -3 -9 13 69

oleg@debian:~/lab1\_release$ python3.1 1.3\_Zeidel.py

eps = 0.001

Simple calculate 31 iterations;

Matrix 1x4 :

7.68 3.57 2.63 7.95

-----

Zeidel finish in 14 iteratrions;

Matrix 1x4 :

7.68 3.57 2.63 7.95

-----

test A\*x = b

Matrix 1x4 :

100.00 -5.00 34.00 69.00

-----

b = [100.0, -5.0, 34.0, 69.0]

oleg@debian:~/lab1\_release$ cat lab1\_3.in

eps = 0.01

24 2 4 -9 -9

-6 -27 -8 -6 -76

-4 8 19 6 -79

4 5 -3 -13 -70

oleg@debian:~/lab1\_release$ python3.1 1.3\_Zeidel.py

eps = 0.01

Simple calculate 8 iterations;

Matrix 1x4 :

4.00 1.99 -6.99 9.00

-----

Zeidel finish in 5 iteratrions;

Matrix 1x4 :

4.00 2.00 -7.00 9.00

-----

test A\*x = b

Matrix 1x4 :

-9.04 -76.00 -78.98 -70.00

-----

b = [-9.0, -76.0, -79.0, -70.0]

## 4. Метод вращений

oleg@debian:~/lab1\_release$ cat lab1\_4.in

eps = 0.01

3 4 -4

4 -7 -4

-4 -4 3

oleg@debian:~/lab1\_release$ python3.1 1.4\_Rotations.py

finish in 6 iters.

l = 9.00 v = [0.667, 0.333, -0.667]

l = -9.00 v = [-0.236, 0.943, 0.236]

l = -1.00 v = [0.707, -0.0, 0.707]

oleg@debian:~/lab1\_release$ cat lab1\_4.in

eps = 0.001

-9 7 5

7 8 9

5 9 8

oleg@debian:~/lab1\_release$ python3.1 1.4\_Rotations.py

finish in 5 iters.

l = -11.70 v = [0.951, -0.288, -0.11]

l = 19.53 v = [0.286, 0.691, 0.664]

l = -0.84 v = [-0.115, -0.663, 0.74]

test A^t\*R\*A:

Matrix 3x3 :

-11.70 0.00 -0.00

0.00 19.53 0.00

-0.00 0.00 -0.84

-----

## 5. QR-разложение

oleg@debian:~/lab1\_release$ cat lab1\_5.in

eps = 0.001

3 -7 -1

-9 -8 7

5 2 2

oleg@debian:~/lab1\_release$ python3.1 1.5\_QR.py

eps = 0.001

(3.82+2.6j)

(3.82-2.6j)

(-13.5+0j)

oleg@debian:~/lab1\_release$ cat lab1\_5.in

eps = 0.001

6 -4 0

-7 6 -7

-2 -6 -7

oleg@debian:~/lab1\_release$ python3.1 1.5\_QR.py

(2.85+0j)

(-9.2+0j)

(12.22+0j)

# Вывод

## 1. LU-разложение

Алгоритм LU разложения позволяет эффективно находить решения СЛАУ, а также вычислять обратную матрицу и детерминант. Алгоритм особенно полезен в случае, когда необходимо найти решение нескольких СЛАУ с одинаковыми коэффициентами, и различными свободными членами. LUP-разложение используется для вычисления обратной матрицы по компактной схеме, решая СЛАУ. По сравнению с алгоритмом LU-разложения алгоритм LUP-разложения может обрабатывать любые невырожденные матрицы и при этом обладает более высокой численной устойчивостью.

## 2. Метод прогонки

Метод прогонки позволяет чрезвычайно быстро решать СЛАУ, матрица коэффициентов которых имеет трёхдиагональный вид. Значимость метода объясняется тем, что трехдиагональные СЛАУ приходится решать при решении более сложных задач, примером может служить сплайн интерполяция или дифференциальные уравнения. Для применимости формул метода прогонки достаточно свойства строгого диагонального преобладания.

## 3. Метод простых итераций и метод Зейделя

Методы простых итераций и Зейделя позволяют численно решать СЛАУ с заданной точностью. Главным достоинством методов является простота реализации, а недостатком - требование диагонального преобладания матрицы коэффициентов.

## 4. Метод вращений

Метод вращений – итерационный метод для вычисления собственных значений и собственных векторов вещественной симметричной матрицы. Примечателен тем, что он решает полную проблему собственных значений и собственных векторов, но применим только для симметрических матриц. Он назван в честь Карла Густава Якоба Якоби, предложившего этот метод в 1846 году, хотя использоваться метод начал только в 1950ых с появлением компьютеров.

## 5. QR-разложение

QR алгоритм использует QR-разложение при решении полной проблемы собственных значений матриц. QR-разложение матрицы — представление в виде произведения ортогональной и верхнетреугольной матрицы. Для этого используется преобразование Хаусхолдера — линейное преобразование векторного пространства, которое описывает его отображение относительно гиперплоскости, проходящей через начало координат.

Замечания:

QR алгоритм позволяет находить комплексные собственные значения.

Проще всего QR может быть вычислено, как побочный продукт ортогонализации Грама—Шмидта.

Для нахождения максимального собственного значения существуют более эффективные методы.