Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу «Численные методы»**

Студент: П. А. Гамов
Преподаватель: Д. Л. Ревизников
Группа: М8О-407Б
Дата:
Оценка:
Подпись:

**Москва, 2021**

# 1 LU алгоритм разложения матрицы

Начальные данные: матрица и правая часть.

```
A = [[-1,-7,-3,-2],
     [-8,1,-9,0],
     [8,2,-5,-3],
     [-5,3,5,-9]]
b = [-12,-60,-91,-43]
```

Функция разложения матрицы на L и U матрицы.

```
def LUP_decomposition(A):
    A_ = deepcopy(A)
    size = len(A_)
    P = [i for i in range(size)]
    k_ = 0
    for k in range(size):
        flag = 0
        for i in range(k, size):
            if abs(A_[i][k]) > flag:
                flag = abs(A_[i][k])
                k_ = i
        if flag == 0:
            return -1
        swap(P, k ,k_)
        swap(A_, k, k_)
        for i in range(k + 1, size):
            A_[i][k] = A_[i][k] / A_[k][k]
            for j in range(k + 1, size):
                A_[i][j] = A_[i][j] - A_[i][k] * A_[k][j]
    return A_, P
```

Функция решения системы уравнений из матрицы на основе LU разложения.

```
def LUP_solve(A, P, B):
    size = len(A)
    X = [0 for i in range(size)]
    Y = [0 for i in range(size)]
    for i in range(size):
        if i == 0:
            Y[i] = B[P[i]]
        else:
            suma_y = sum(map(lambda u, y: u * y, A[i][:i], Y[:i]))
            Y[i] = B[P[i]] - suma_y
    for i in range(size - 1, -1, -1):
        if i == size - 1:
            X[i] = Y[i] / A[i][i]
        else:
            suma_x = sum(map(lambda l, x: l * x, A[i][i + 1 :], X[i + 1 :]))
            X[i] = (Y[i] - suma_x) / A[i][i]
```

```
    return X
```

# 2 Метод прогонки

```
A = [[-14,-6],[-9,15,-1],[1,-11,1],[-7,12,3],[6,-7]]
b = [-78,-73,-38,77,91]
```

Функция решения диагональной матрицы методом прогонки.

```
def solve(m, b):
    y = [None] * len(m)
    alpha = [None] * len(m)
    beta = [None] * len(m)
    for i in range(len(m)):
        if i == 0:
            y[i] = m[i][0]
            alpha[i] = -1 * m[i][1] / y[i]
            beta[i] = b[i] / y[i]
        elif i == len(m) - 1:
            y[i] = m[i][1] + m[i][0] * alpha[i-1]
            beta[i] = (b[i] - m[i][0] * beta[i-1]) / y[i]
        else:
            y[i] = m[i][1] + m[i][0] * alpha[i-1]
            alpha[i] = -1 * m[i][2] / y[i]
            beta[i] = (b[i] - m[i][0] * beta[i-1]) / y[i]
    x = [0] * len(m)
    for i in range(len(m)):
        if i == 0:
            x[len(m)-i-1] = beta[len(m)-i-1]
        else:
            x[len(m)-i-1] = alpha[len(m)-i-1] * x[len(m)-i] + beta[len(m)-i-1]
    return x
```

# 3 Итерационные методы решения СЛАУ

## 1 Метод простых итераций

```
def error(x,x_, err):
    if x_[0] == None:
        return False
    res = [0] * len(x)
    for i in range(len(x)):
        res[i] = pow(x[i] - x_[i],2)
    if math.sqrt(sum(res)) > err:
        return False
```

```
    else:
        return True

def solve(A, b, err):
    x_ = [None] * len(A)
    x = [0] * len(A)
    num_of_it = 0
    while True:
        for i in range(len(A)):
            s = 0
            for j in range(len(A)):
                if i != j:
                    s += A[i][j] * x[j]
            x_[i] = (b[i] - s) / A[i][i]
        num_of_it += 1
        if error(x,x_,err):
            break
        x = copy.copy(x_)
    return x_, num_of_it
```

## 2 Метод Зейделя

```
def solveZeidel(A, b, err):
    x_ = [None] * len(A)
    x = [0] * len(A)
    num_of_it = 0
    while True:
        for i in range(len(A)):
            s = 0
            for j in range(len(A)):
                if j < i:
                    s += A[i][j] * x_[j]
                elif i != j:
                    s += A[i][j] * x[j]
            x_[i] = (b[i] - s) / A[i][i]
        num_of_it += 1
        if error(x,x_,err):
            break
        x = copy.copy(x_)
    return x_, num_of_it
```

# 4 Численные методы решения задач на собственные значения и собственные векторы матриц

## 1 Метод вращений Якоби

```
def jacobi(A, err):
    A_ = copy.deepcopy(A)
    num_of_it = 0
    while True:
        i, j = find_max(A)
        P = math.pi / 4
        if A[i][i] - A[j][j] != 0:
            P = 2 * A[i][j] / (A[i][i] - A[j][j])
        c = math.cos(math.atan(P) / 2)
        s = math.sin(math.atan(P) / 2)
        rotate = rotate_matrix(A,s,c,i,j)
        A_ = prois(transpose(rotate),prois(A,rotate))
        num_of_it += 1
        A = copy.deepcopy(A_)
        if error(A_, err):
            break
    return A, num_of_it

def rotate_matrix(A,s,c,i,j):
    res = copy.deepcopy(A)
    for k in range(len(A)):
        for l in range(len(A)):
            if k == l:
                res[k][l] = 1
            else:
                res[k][l] = 0
    res[i][i] = c
    res[i][j] = -s
    res[j][i] = s
    res[j][j] = c
    return res

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

def error(A_, err):
    s = sum([sum([math.pow(A_[i][j], 2) for j in range(i+1,len(A_))]) for i in range(
        len(A_))])
    return False if math.sqrt(s) > err else True

def find_max(A):
    m, ib, jb = None, None, None
    for i in range(len(A)):
```

```
        for j in range(len(A)):
            if i < j:
                if m == None or abs(A[i][j]) > m:
                    m = A[i][j]
                    ib, jb = i, j
    return ib, jb
```

# 5 QR алгоритм нахождения собственных значений матриц

```
def solve_QR(A):
    iter = 0
    Q,R = find_QR(A)
    A_ = prois(Q, R)
    while error(A, A_, err) != True:
        Q,R = find_QR(A)
        A = A_
        A_ = prois(Q, R)
        iter += 1
    return A_

def find_QR(A):
    R_ = copy.deepcopy(A)
    Q_ = None
    for i in range(len(R_) - 1):
        H = find_housholder(R_, i)
        if Q_ == None:
            Q_ = H
        else:
            Q_ = prois(H, Q_)
        R_ = prois(R_, H)
    return Q_, R_

def find_housholder(A, it):
    v = [0] * len(A)
    for i in range(it, len(A)):
        v[i] = A[i][it]
    s = 0
    for i in range(len(A)):
        s += math.pow(v[i],2)
    if A[it][it] < 0:
        v[it] -= math.sqrt(s)
    elif A[it][it] > 0:
        v[it] += math.sqrt(s)
    else:
        print('eq 0 house')
```

```
    H = copy.deepcopy(A)
    dim = 0
    for i in range(len(v)):
        dim += math.pow(v[i],2)
    for i in range(len(A)):
        for j in range(len(A)):
            if i == j:
                H[i][j] = 1 - 2 * v[i] * v[j] / dim
            else:
                H[i][j] = 0 - 2 * v[i] * v[j] / dim
    return H
```

# 6 Выводы

В данной лабораторной работе я научился применять алгоритмы решения линейных уравнений, находить собственные значения и векторы матриц.