Выполнил: Чакирян А.А.

Группа: М8О-408Б-17

Лабораторная работа № 1

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.

```
A = np.array([
    [7, -5, 6, -7],
    [-2,  0,  8, -4],
    [-7, -2,  2, -2],
    [2,  -4, -4,  4]
])

b = np.array([18, -12, 6, -12]).astype(np.float32)
```

Результат работы:

```
P:
[0 1 2 3]
L:
[[ 1.          0.          0.          0.        ]
 [-0.28571429  1.          0.          0.        ]
 [-1.          4.9         1.          0.        ]
 [ 0.28571429  1.8         0.58585859  1.        ]]
U:
[[  7.          -5.           6.          -7.        ]
 [  0.          -1.42857143   9.71428571  -6.        ]
 [  0.           0.         -39.6         20.4       ]
 [  0.           0.           0.           4.84848485]]
x:
[-0.3   0.45 -5.55 -7.95]
B:
[ 18. -12.   6. -12.]
my B:
[ 18. -12.   6. -12.]
```

```
det(A): 1919.9999999999993
A^(-1):
[[ 0.03333333  0.00833333 -0.10833333  0.0125    ]
 [-0.05       -0.0125     -0.0875     -0.14375   ]
 [-0.05        0.2375     -0.0875      0.10625   ]
 [-0.11666667  0.22083333 -0.12083333  0.20625   ]]
A * A^(-1):
[[ 1.00000000e+00  1.11022302e-16  0.00000000e+00  0.00000000e+00]
 [-5.55111512e-17  1.00000000e+00  0.00000000e+00  2.22044605e-16]
 [ 1.94289029e-16  9.43689571e-16  1.00000000e+00  7.77156117e-16]
 [ 5.55111512e-17  2.22044605e-16  2.77555756e-16  1.00000000e+00]]
```

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.

```
c = [-9, -4, -8, 5]
b = [18, -9, 21, -10, 12]
a = [2, -9, -4, 7]
f = [-81, 71, -39, 64, 3]
```

Результат работы:

```
answer: [-8 -7 -6 -3  2]
f:
[-81, 71, -39, 64, 3]
my f:
[-81  71 -39  64   3]
```

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.

Главная идея метода Зейделя в сравнении с методом простых итераций – переиспользование коэффициентов, посчитанных на текущей итерации.

```
A = np.array([
    [21, -6, -9, -4],
    [-6, 20, -4, 2],
    [-2, -7, -20, 3],
    [4,  9, 6, 24]
])

b = np.array([127, -144, 236, -5]).astype(np.float32)
```

Результат работы:

```
condition number: 4.553464977796038
norm: 46.9148164229596
iterations
iterations: 13
x: [ 0.99969256 -9.00010663 -8.0001808   5.00006692]
Ax: [ 126.99554312 -143.99943092  236.00517809   -5.00166828]
Zeidel
iterations: 7
x: [ 1.00003705 -8.99997205 -8.00001895  4.99998808]
Ax: [ 127.00082862 -143.99961145  236.00007342   -5.        ]
```

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.

Алгоритм состоит в том, чтобы на каждом шаге обнулять максимальный по модулю элемент. Данная задача сводится к поиску ортогональной матрицы вращения, на пересечении строк и столбцов, соответствующих максимальному элементу, находятся синусы и косинусы угла вращения.

```
A = np.array([
    [8, -3, 9],
    [-3, 8, -2],
    [9, -2, -8],
])
```

Результат работы:

```
default criterion:  9.695359714832659
it: 1, norm: 3.6055512754639896
it: 2, norm: 0.5955119569103899
it: 3, norm: 0.29951665483432294
it: 4, norm: 0.008567321831287715
after iterations:
[[ 1.41143308e+01 -8.56676126e-03 -2.96902931e-18]
 [-8.56676126e-03  5.94541747e+00  9.80043440e-05]
 [ 6.78448118e-16  9.80043440e-05 -1.20597483e+01]]
eigenvalues:
[14.114330836196043, 5.945417474082699, -12.05974831027874]
Values:
[array([ 0.78387401, -0.50264086,  0.3645459 ]), array([0.47034713, 0.86395803, 0.17986134]), array([-0.40535802,  0.03047448,  0.91364992])]
orotogonality:
(x0, x1) =  1.3877787807814457e-17
(x0, x2) =  0.0
(x1, x2) =  0.0
Ax = lambda*x
[11.0598277  -7.10184067  5.14378058] = [11.06385705 -7.09443935  5.14532141]
[2.78965505 5.14090015 1.06631735] = [2.79641004 5.13659116 1.06935078]
[  4.88856174  -0.36742993 -11.01837051] = [  4.88851565  -0.3675146  -11.01838813]
```

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.

В качестве примера матрицы был взят образец из методички, так как на моем варианте отсутствовали комплексно-сопряженные корни. С помощью алгоритма мы пытаемся найти такие $A=QR$, где Q — ортогональная, a R — верхняя треугольная.

```
A = np.array([
    [1, 3, 1],
    [1, 1, 4],
    [4, 3, 1],
])
```

Результат работы:

```
cr: 5.09901951359278 45
Q:
[[-0.23570226  0.96582428  0.10783277]
 [-0.23570226  0.05083286 -0.97049496]
 [-0.94280904 -0.25416428  0.21566555]]
R:
[[-4.24264069e+00 -3.77123617e+00 -2.12132034e+00]
 [ 1.48094774e-17  2.18581284e+00  9.14991422e-01]
 [ 1.10030137e-16  1.49547129e-16 -3.55848152e+00]]
QR:
[[1. 3. 1.]
 [1. 1. 4.]
 [4. 3. 1.]]
Q-1:
[[-0.23570226 -0.23570226 -0.94280904]
 [ 0.96582428  0.05083286 -0.25416428]
 [ 0.10783277 -0.97049496  0.21566555]]
```

```
Num iterations:  5
eigen values:
[(6.313174860729379+0j), (-1.6761220268750678+1.5529441689226442j), (-1.6761220268750678-1.5529441689226442j)]
eigen values from numpy:
[ 6.34280359+0.j         -1.6714018 +1.55214776j -1.6714018 -1.55214776j]
```

Лабораторная работа № 2

2.1. Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

```
def f(x):
    return x*np.exp(x) + x**2 - 1


def df(x):
    return np.exp(x)*(1 + x) + 2*x


def phi(x, lmbd=0.1):
    return x - lmbd*f(x)


def dphi(x, lmbd=0.1):
    return 1 - lmbd*df(x)


def getQ(l, r):
    return max(abs(dphi(l)), abs(dphi(r)))
```

Результат работы:

```
simple iterations
q: 0.9
it: 1, dx: 4.500000000000001
it: 2, dx: 0.0669245718150578
it: 3, dx: 0.04383309222432547
it: 4, dx: 0.028873130772942686
it: 5, dx: 0.019089521905553634
it: 6, dx: 0.012651770063255909
it: 7, dx: 0.008398522530342458
it: 8, dx: 0.005581030447877612
it: 9, dx: 0.003711340165028277
it: 10, dx: 0.002469161742569726
it: 11, dx: 0.0016432469489274795
it: 12, dx: 0.0010938193158787082
final it: 13, dx: 0.00072819528311735
root = 0.47833362186501366
checking: f(root) = 0.0005387003240253652
Newton method
```

```
it: 1, dx: 0.6
it: 2, dx: 0.110140538649859 34
it: 3, dx: 0.011566112889711821
final it: 4, dx: 0.0001209380902851831
root = 0.47817241037014363
checking: f(root) = 4.38638101396549e-08
```

2.2. Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

Метод простых итераций идейно аналогичен алгоритму из предыдущей задачи, а метод Ньютона преобразуется к следующему виду:

$$\begin{cases} x_1^{(k+1)} = x_1^{(k)} - \dfrac{\det \mathbf{A}_1^{(k)}}{\det \mathbf{J}^{(k)}} \\ x_2^{(k+1)} = x_2^{(k)} - \dfrac{\det \mathbf{A}_2^{(k)}}{\det \mathbf{J}^{(k)}} \end{cases} \qquad k = 0,1,2,...$$

Где J — матрица Якоби, а матрицы A имеют вид:

$$\mathbf{A}_1^{(k)} = \begin{bmatrix} f_1(x_1^{(k)}, x_2^{(k)}) & \dfrac{\partial f_1(x_1^{(k)}, x_2^{(k)})}{\partial x_2} \\ f_2(x_1^{(k)}, x_2^{(k)}) & \dfrac{\partial f_2(x_1^{(k)}, x_2^{(k)})}{\partial x_2} \end{bmatrix}, \qquad \mathbf{A}_2^{(k)} = \begin{bmatrix} \dfrac{\partial f_1(x_1^{(k)}, x_2^{(k)})}{\partial x_1} & f_1(x_1^{(k)}, x_2^{(k)}) \\ \dfrac{\partial f_2(x_1^{(k)}, x_2^{(k)})}{\partial x_1} & f_2(x_1^{(k)}, x_2^{(k)}) \end{bmatrix}.$$

Результат работы:

```
Newton method
iterations: 3
Newton answer: [0.37372831 0.72657115]
f1(x) = 5.555556015224283e-13, f2(x) = -5.792255564074367e-12
simple iterations
iterations: 9
simple iterations answer: [0.37370172 0.72629289]
f1(x) = -0.00023799730317619616, f2(x) = -0.0005178677490138561
```

Лабораторная работа № 3

3.1. Используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках X_i, $i = 0,...,3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, Y_i\}$. Вычислить значение погрешности интерполяции в точке X^*.

Интерполяционный многочлен Лагранжа принимает вид:

$$L_n(x) = \sum_{i=0}^{n} f_i \prod_{j=0,\ j\neq i}^{n} \frac{(x - x_i)}{(x_i - x_j)}$$

Интерполяционный многочлен Ньютона принимает вид:

$$P_n(x) = f(x_0) + (x - x_0) f(x_1, x_0) + (x - x_0)(x - x_1) f(x_0, x_1, x_2) + ... +$$
$$+ (x - x_0)(x - x_1)...(x - x_n) f(x_0, x_1, ..., x_n).$$

Результат работы:

```
points: [0.1 0.5 0.9 1.3], x = 0.8
Lagrange
real: 0.5768564486857903
Lagrange: 0.5996357042970935
error: 0.022779255611303117
Newton
real: 0.5768564486857903
Newton: 0.5996357042970932
error: 0.022779255611302895
points: [0.1 0.5 1.1 1.3], x = 0.8
Lagrange
real: 0.5768564486857903
Lagrange: 0.6341106211498607
error: 0.057254172464070385
Newton
real: 0.5768564486857903
Newton: 0.6341106211498606
error: 0.057254172464070274
```

3.2. Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

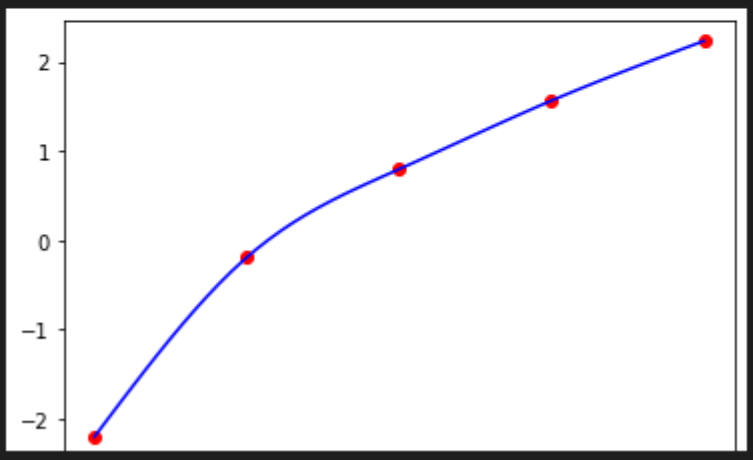
Для построения кубического сплайна необходимо построить n многочленов третьей степени, т.е. определить $4n$ неизвестных a_i, b_i, c_i, d_i. Эти коэффициенты ищутся из условий в узлах сетки.

$$
\begin{aligned}
&S(x_{i-1}) = a_i = a_{i-1} + b_{i-1}(x_{i-1} - x_{i-2}) + c_{i-1}(x_{i-1} - x_{i-2})^2 + d_{i-1}(x_{i-1} - x_{i-2})^3 = f_{i-1} \\
&S'(x_{i-1}) = b_i = b_{i-1} + 2c_{i-1}(x_{i-1} - x_{i-2}) + 3d_{i-1}(x_{i-1} - x_{i-2})^2, \\
&S''(x_{i-1}) = 2c_i = 2c_{i-1} + 6d_{i-1}(x_{i-1} - x_{i-2}), \qquad i = 2,3,...,n \\
&S(x_0) = a_1 = f_0 \qquad , \qquad (3.12) \\
&S''(x_0) = c_1 = 0 \\
&S(x_n) = a_n + b_n(x_n - x_{n-1}) + c_n(x_n - x_{n-1})^2 + d_n(x_n - x_{n-1})^3 = f_n \\
&S''(x_n) = c_n + 3d_n(x_n - x_{n-1}) = 0
\end{aligned}
$$

Результат работы:

```
splines coefs(abcd):
1. [-2.2026, 5.6729, 0.0, -4.0581]
2. [-0.1931, 3.725, -4.8698, 4.3272]
3. [0.7946, 1.9063, 0.3229, -0.7253]
4. [1.5624, 1.8165, -0.5474, 0.4562]

f(0.8) = 0.6029
```

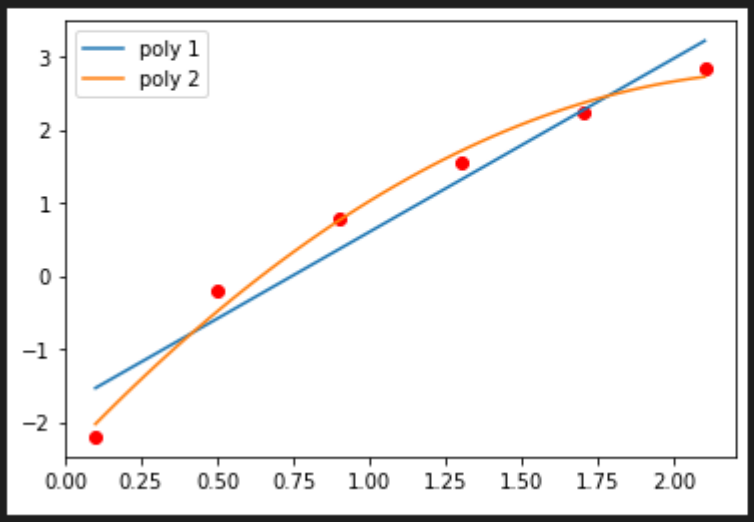


3.3. Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

```
x = np.array([0.1, 0.5, 0.9, 1.3, 1.7, 2.1])
y = np.array([-2.2026, -0.19315, 0.79464, 1.5624, 2.2306, 2.8419])
```

Результат работы:

```
pol. of power 1: [ 2.37582214 -1.77443936]
error: 0.9854121221771428
pol. of power 2: [-0.92289174  4.40618397 -2.46045555]
error: 0.171386167509286
```



3.4. Вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_i)$, $i = 0,1,2,3,4$ в точке $x = X^*$.

```
x = np.array([0, 1, 2, 3, 4])
y = np.array([0, 2, 3.4142, 4.7321, 6])
x_test = 2
```

Первая производная считается как приращение функции к значению приращения аргумента, усредненная для концов отрезка.

Результат работы:

```
f'(2) = 1.36605
f''(2) = -0.09630000000000027
```

3.5. Вычислить определенный интеграл $F = \int_{X_0}^{X_1} y\,dx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

Результат работы:

```
rectangles      h1:0.01001      h2:0.01011
error: 0.00014  Runge-Romberg err: 0.00018
trapezium       h1:0.01042      h2:0.01022
error: 0.00027  Runge-Romberg err: 0.00036
Simpson h1:0.01014      h2:0.01015
error: 0.0000045        Runge-Romberg err: 0.0000048
```

Лабораторная работа № 4

4.1. Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Результат работы:

Euler

x	y	y_true	abs error	Runge eps
2.0	7.0	7.0	0.0	0.0
2.2	8.08	8.04	0.03999999999999915	0.5600000000000005
2.4	9.239	9.16	0.07916666666666572	1.1792253176930618
2.6	10.477	10.36	0.1168347338935547	1.8570626175690226
2.8	11.792	11.64	0.15245905306566776	2.5930131132934857
3.0	13.186	13.0	0.1855837396687008	3.386662607444789

Runge-Kutta

x	y	y_true	abs error	Runge eps
2.0	7.0	7.0	0.0	0.0
2.2	8.04	8.04	4.8159200556341375e-06	0.035333643290066344
2.4	9.16	9.16	9.622676051534995e-06	0.07466728619759676
2.6	10.36	10.36	1.4590014544069163e-05	0.11800093986977452
2.8	11.64	11.64	1.9820201957898576e-05	0.1653346110001338
3.0	13.0	13.0	2.537802078528273e-05	0.21666830381439262

Adams

x	y	y_true	abs error	Runge eps
2.0	7.0	7.0	0.0	0.0
2.2	8.04	8.04	4.8159200556341375e-06	0.0
2.4	9.16	9.16	9.622676051534995e-06	0.0
2.6	10.36	10.36	1.4590014544069163e-05	0.0
2.8	11.64	11.64	1.8207704721717732e-05	0.04733356961116056
3.0	13.0	13.0	2.1780372886581745e-05	0.09958842787724174

4.2. Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Результат работы:

Shooting method

x	y	y_true	abs error	Runge eps
1.0	2.993	3.0	0.006958631126713577	0.0012071979738029388
1.2	3.026	3.64	0.6136566655860771	0.006865945914784326
1.4	3.107	4.36	1.2528769641746997	0.025720372728492407
1.6	3.218	5.16	1.9424134189418742	0.050588865588562015
1.8	3.348	6.04	2.6921575573814986	0.07900576212751102
2.0	3.492	7.0	3.5080459830275488	0.10960198411664675
2.2	3.646	8.04	4.393857090667423	0.14157541544140942
2.4	3.808	9.16	5.35211001807844	0.17443608433857882
2.6	3.975	10.36	6.3845488695810495	0.20787464528201557
2.8	4.148	11.64	7.49241947241613	0.24169057830438026
3.0	4.323	13.0	8.676635451631274	0.27575114700517184

Finite difference method

x	y	y_true	abs error	Runge eps
1.0	2.979	3.0	0.021215917751532753	0.015853912696333516
1.2	3.019	3.64	0.6214987966548864	0.013881081750496627
1.4	3.104	4.36	1.2560894123497088	0.07432176856431116
1.6	3.218	5.16	1.9420041938929704	0.15190211523072827
1.8	3.351	6.04	2.6887489426608786	0.23958332302403074
2.0	3.498	7.0	3.5020486458171143	0.33345097681827207
2.2	3.654	8.04	4.3855562968863575	0.4312108929359004
2.4	3.818	9.16	5.3417120877624935	0.5314625272493991
2.6	3.988	10.36	6.372207953394071	0.6333229232845952
2.8	4.162	11.64	7.478253854047148	0.7362210309104631
3.0	4.339	13.0	8.660737932420137	0.8397800015068944