Национальный исследовательский институт
«Московский Авиационный Институт»

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ**

**Кафедра вычислительной математики и программирования**

# Отчет по лабораторным работам

## по курсу «Численные методы»

Работу выполнил
студент **3** курса
**очного** отделения
Куликов А. В.
группы М80-308Б

преподаватель:
Сластушенский Ю.В.

Оценка:

Подпись преподавателя
_______
Подпись студента
_______

Москва-2020

# Лабораторная работа №1

***Задача 1.1.*** Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ).

$$\begin{cases} -4 \cdot x_1 - 9 \cdot x_2 + 4 \cdot x_3 + 3 \cdot x_4 = -51 \\ 2 \cdot x_1 + 7 \cdot x_2 + 9 \cdot x_3 + 8 \cdot x_4 = 76 \\ 4 \cdot x_1 - 4 \cdot x_2 - 2 \cdot x_4 = 26 \\ -8 \cdot x_1 + 5 \cdot x_2 + 2 \cdot x_3 + 9 \cdot x_4 = -73 \end{cases}$$

Для матрицы СЛАУ вычислить определитель и обратную матрицу.

**Метод решения**

Для решения задачи реализован стандартный алгоритм LU разложения с выбором главного элемента (LUP разложение) согласно методичке.
Хранение результата осуществляется компактным образом в одной матрице.

Пример работы:

```
$ ./prog < input2
--- INPUT ---
A:
4
-4 -9 4 3
2 7 9 8
4 -4 0 -2
-7 -9 -8 -5
b: -51 76 26 -73

--- SOLUTION ---
x: 28.5806 -7.22581 -44.3871 58.6129
det: -558
inverse matrix:
4
-0.225806 0.415771 0.437276 0.354839
0.0322581 -0.18638 -0.213262 -0.193548
0.483871 -0.90681 -0.643369 -0.903226
-0.516129 1.2043 0.801075 1.09677

--- CHECKOUT ---
A*x:
-51 76 26 -73
A*A^(-1):
4
1 4.44089e-16 0 8.88178e-16
0 1 0 0
0 0 1 0
0 0 0 1
```

***Задача 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.

$$\begin{cases} -11 \cdot x_1 - 8 \cdot x_2 = 99 \\ 9 \cdot x_1 - 17 \cdot x_2 + x_3 = -75 \\ -4 \cdot x_2 + 20 \cdot x_3 + 9 \cdot x_4 = 66 \\ -4 \cdot x_3 - 14 \cdot x_4 + 3 \cdot x_5 = 54 \\ -6 \cdot x_4 + 14 \cdot x_5 = 8 \end{cases}$$

**Метод решения**

Для решения задачи реализован стандартный метод прогонки.

По сути своей метод прогонки – частный случаей метода Гаусса для трехдиагональной матрицы. Решение осуществялется в два прохода. При проходе в прямом направлении осуществляется поиск прогоночных коэффициентов. На обратном ходе метода на основе подсчитанных прогоночных коэффициентов вычисляется решение.

Пример работы:

```
$ ./prog < input1
--- INPUT ---
a: 9 -4 -4 -6
b: -11 -17 20 -14 14
c: -8 1 9 3
d: 99 -75 66 54 8

--- SOLUTION ---
x: -9 0 6 -6 -2
```

***Задача 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.

$$\begin{cases} -7 \cdot x_1 - x_2 + 2 \cdot x_3 + 2 \cdot x_4 = -24 \\ 3 \cdot x_1 - 20 \cdot x_2 - 8 \cdot x_4 = -47 \\ -9 \cdot x_1 + x_2 + 18 \cdot x_3 - 6 \cdot x_4 = 28 \\ -x_1 - x_3 - 6 \cdot x_4 = -50 \end{cases}$$

**Метод решения**

Для решения задачи реализованы итеративные методы решения СЛАУ: метод простых итераций и метод Зейделя.

Суть этих методов довольно проста. Берется начальное приближение, далее вычисляется очередное решение с использованием предыдущего. После каждого вычисления проверяется одно из возможных условий окончания вычислений.

Различаются эти два метода только тем, что в методе Зейделя уже подсчитанные компоненты решения на текущей итерации используются для вычисления последующих компонент.

Пример работы:

```
$ ./prog iterative < input1
--- INPUT ---
A: 4
-7 -1 2 2
3 -20 0 -8
-9 1 18 -6
-1 -1 0 -6
b: -24 -47 28 -50

--- SOLUTION ---
iterative method
with precision: 0.001
iterations required: 12
x: 7.48979 0.685099 7.58597 6.97087

--- CHECKOUT ---
A*x:
-23.9999 -46.9996 27.9993 -50.0001


$ ./prog seidel < input1
--- INPUT ---
A: 4
-7 -1 2 2
3 -20 0 -8
-9 1 18 -6
-1 -1 0 -6
b: -24 -47 28 -50

--- SOLUTION ---
seidel method
with precision: 0.001
iterations required: 8
x: 7.48982 0.685133 7.58602 6.97084

--- CHECKOUT ---
A*x:
-24.0001 -46.9999 28 -50
```

***Задача 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.

$$\begin{pmatrix} 9 & -2 & 3 \\ -2 & 6 & 8 \\ 3 & 8 & -6 \end{pmatrix}$$

**Метод решения**

Для решения задачи реализован метод вращений Якоби.

В данном методе на каждой итерации выбирается наибольший по модулю внедиагональный элемент, и при помощи специального ортогонального преобразования этот элемент обнуляется. При достижении состояния, в котором все внедиагональные элементы матрицы малы на диагонали останутся собственные значения. А соответствующимим им собственными векторами будут столбцы матрицы, полученной перемножением матриц указанных выше ортогональных преобразований.

Пример работы:

```
$ ./prog < input1
-- INPUT --
M:
3
9 -2 3
-2 6 8
3 8 -6
-- SOLUTION --
jacobi method
with precision: 1e-06
iterations required: 6
 e-val                e-vec
9.35252: ( 0.813453 0.435283 0.385775 )
10.2991: ( -0.553337 0.78351 0.282718 )
-10.6517: ( -0.179196 -0.443442 0.878207 )
-- CHECKOUT --
   M * e-vec: 7.60784 4.07099 3.60797
e-val * e-vec: 7.60784 4.07099 3.60797
   M * e-vec: -5.6989 8.06948 2.91176
e-val * e-vec: -5.6989 8.06948 2.91176
   M * e-vec: 1.90874 4.7234 -9.35437
e-val * e-vec: 1.90874 4.7234 -9.35437
```

***Задача 1.5.*** 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.

$$\begin{pmatrix} -9 & 2 & 2 \\ -2 & 0 & 7 \\ 8 & 2 & 0 \end{pmatrix}$$

**Метод решения**

Для решения задачи реализован алгоритм QR-разложения квадратной матрицы.
В данном алгоритме ищутся такие матрицы Q, R, что A=QR, и Q — ортогональная, а R — верхняя треугольная, а A здесь – исходная матрица. Для этого используется преобразование Хаусхолдера, являющееся ортогональным и позволяющее обратить в нуль группу поддиагональных элементов столбца матрицы.

Реализованный алгоритм QR-разложения многократно используется при поиске собственных значений. На каждой итерации ищется QR-разложение. Из него получается матрица T=RQ, для которой проверяются два отдельных критерия окончания: для действительных комплексных собственных значений. Для комплексного СЗ  проверяется блок 2x2 на диагонали матрицы T. Из него вычисляется пара комплексно-сопряженных СЗ и сравнивается с соответствующими СЗ на предыдущей итерации. Для действительного же собственного значения критерием окончания будет малость элементов, находящихся под ним в матрице.

Пример работы:

```
$ ./prog < input1
-- INPUT --
M:
3
-9 2 2
-2 0 7
8 2 0
-- SOLUTION --
QR algorithm method
with precision: 0.001
iterations required: 35
e-val: (-9.07107,0) (5.05923,0) (-4.98817,0)
```

# Лабораторная работа №2

***Задача 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.

$\ln(x+1)-2x^2+1=0$

**Метод решения**

Для решения этой задачи реализованы методы простой итерации и Ньютона.
Метод простых итераций основан на принципе сжимающих отображений. По сути весь метод сводится к многократному вычислению решений по формуле

$$x^{(k+1)}=\varphi(x^{(k)})$$

и проверки критерия окончания. При удачно подобранной функции метод всегда сходится к решению.

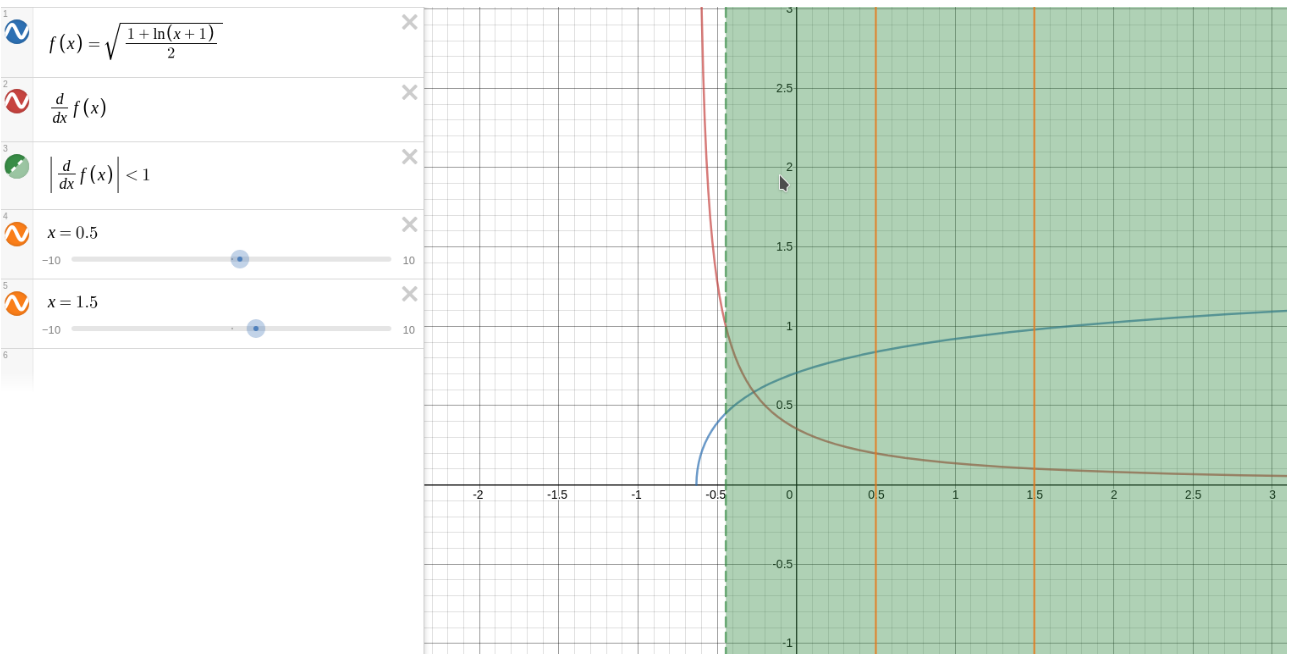
Для решения конкретно этой задачи подошла функция $\varphi(x)=\sqrt{(1+\log(x+1)/2)}$ .

Метод Ньютона так же сводится к последовательному вычислению решений по формуле

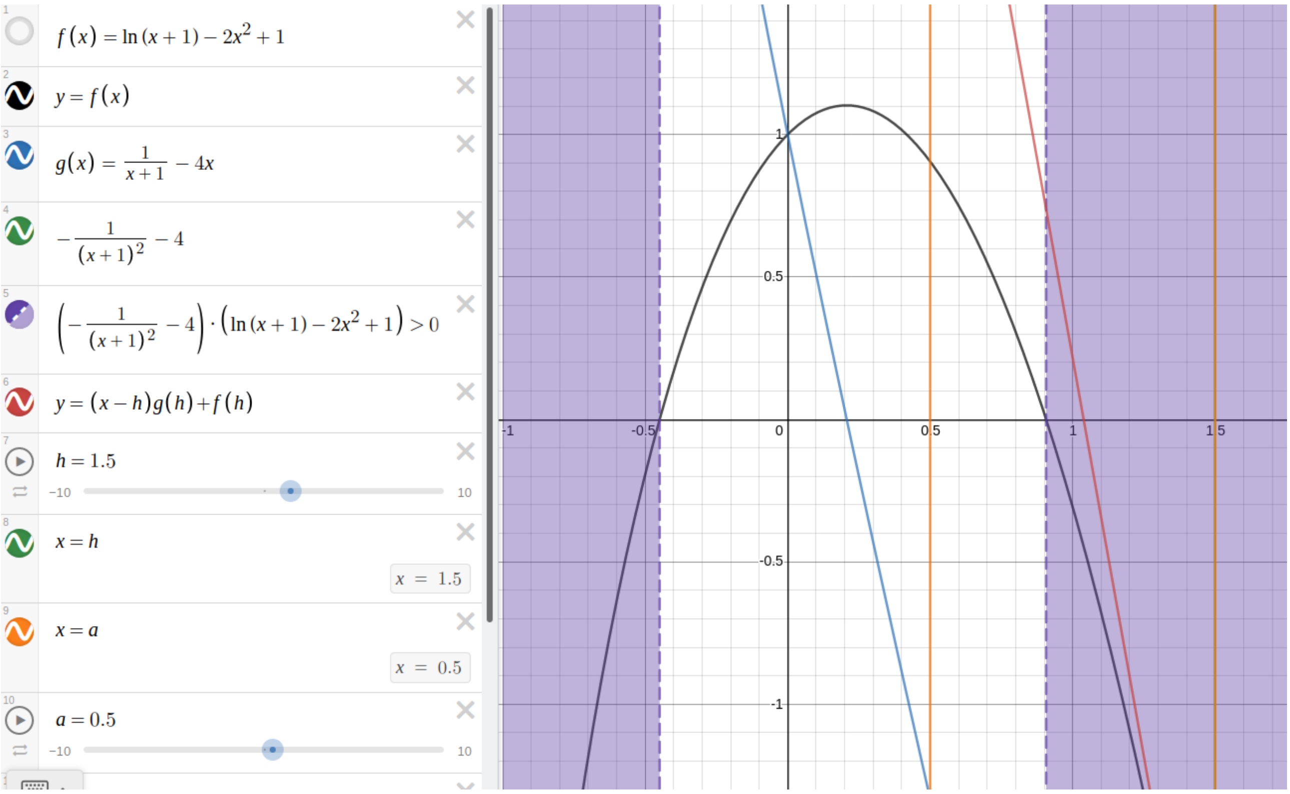
$$x^{(k+1)}=x^{(k)}-\frac{f(x^{(k)})}{f'(x^{(k)})}$$

И при правильно выбранной начальной точке данный метод сходится к решению.

Выбор функции для метода простых итераций.




Выбор начальной точки для метода Ньютона.




Пример работы:

```
$ ./prog fixed-point 0.0001 0.5 1.5
-- SOLUTION --
fixed-point iteration method
q: 0.198817
initial approximation: 1
with precision: 0.0001
iterations required: 4
x_0: 0.907114
-- CHECKOUT --
f(x_0): -0.000121353

$ ./prog newton 0.0001 0.5 1.5
-- SOLUTION --
newton method
initial approximation: 1.5
with precision: 0.0001
iterations required: 4
x_0: 0.907075
-- CHECKOUT --
f(x_0): -2.50443e-08

$ ./prog dichotomy 0.0001 0.5 1.5
-- SOLUTION --
dichotomy method
with precision: 0.0001
iterations required: 13
x_0: 0.907043
-- CHECKOUT --
f(x_0): 9.84486e-05
```

***Задача 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.

$$\begin{cases} x_1^2 + x_2^2 - a^2 = 0, \\ x_1 - e^{x_2} + a = 0. \end{cases}$$

при $a = 3$

**Метод решения**

Для решения этой задачи реализованы методы простой итерации и Ньютона решения систем нелинейных уравнений.

Метод простых итераций по сути своей аналогичен одномерному. Система так же приводится к специальному виду, а за тем решение последовательно вычисляется по формуле

$$\mathbf{x}^{(k+1)} = \boldsymbol{\varphi}(\mathbf{x}^{(k)})$$

только здесь x – вектор, $\varphi$ – вектор функция.

Метод Ньютона для систем так же схож с методом Ньютона для уравнений.

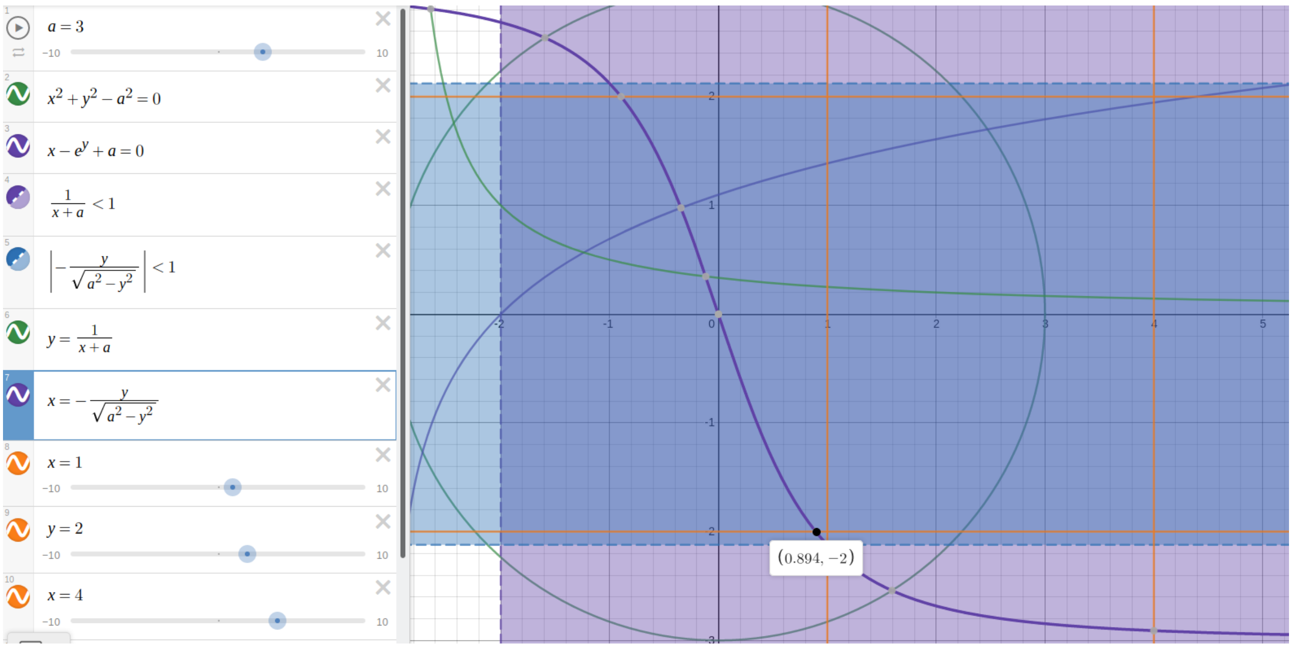
Решения последовательно вычисляются по формуле

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{J}^{-1}(\mathbf{x}^{(k)})\mathbf{f}(\mathbf{x}^{(k)})$$

до выполнения критерия окончания

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| \le \varepsilon$$

Выбор начального приближения



Пример работы:

```
$ ./prog < input1
newton method
initial approximation: 2.5 0
x_0: 2.47208 1.69966
iterations required: 9
-- CHECKOUT --
func1(x_0): 1.50113e-07
func2(x_0): -4.20108e-13


$ ./prog < input2
fixed-point method
q: 0.928709
initial approximation: 2.5 0
x_0: 2.47208 1.69966
iterations required: 15
-- CHECKOUT --
func1(x_0): -1.29177e-06
func2(x_0): -2.45739e-07
```

# Лабораторная работа №3

***Задача 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 $X^*$.

$y = \arcsin(x)$, а) $X_i = -0.4, -0.1, 0.2, 0.5$ ; б) $X_i = -0.4, 0, 0.2, 0.5$;

**Метод решения**

Для решения этой задачи реализованы интерполяционные многочлены Лагранжа и Ньютона.
Интерполяционный многочлен Лагранжа рассчитывается следующим образом:

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

Интерполяционный многочлен Лагранжа рассчитывается следующим образом:

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

Пример работы:

```
$ ./prog < input1
x : f(x)
-0.4 : -0.411517
-0.1 : -0.100167
0.2 : 0.201358
0.5 : 0.523599

x*: 0.1
f(x): 0.100167
f(ip): 0.100056
error:0.000111543
-- CHECKOUT --
f(-0.4): -0.411517
ip(-0.4): -0.411517
f(-0.1): -0.100167
ip(-0.1): -0.100167
f(0.2): 0.201358
ip(0.2): 0.201358
f(0.5): 0.523599
ip(0.5): 0.523599


$ ./prog < input2
x : f(x)
-0.4 : -0.411517
0 : 0
0.2 : 0.201358
0.5 : 0.523599

x*: 0.1
f(x): 0.100167
f(ip): 0.100094
error:7.37745e-05
-- CHECKOUT --
```

```
f(-0.4): -0.411517
ip(-0.4): -0.411517
f(0): 0
ip(0): 0
f(0.2): 0.201358
ip(0.2): 0.201358
f(0.5): 0.523599
ip(0.5): 0.523599
```

***Задача 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 $x = X^*$.

$X^* =$0.1

| $i$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $x_i$ | -0.4 | -0.1 | 0.2 | 0.5 | 0.8 |
| $f_i$ | -0.41152 | -0.10017 | 0.20136 | 0.52360 | 0.92730 |

**Метод решения**

Для решения этой задачи реализован алгоритм построения кубического сплайна согласно методичке.

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.

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

Если ввести обозначение $h_i = x_i - x_{i-1}$, и исключить из системы (3.12) $a_i, b_i, d_i$, то можно получить систему из $n-1$ линейных алгебраических уравнений относительно $c_i,\ i = 2,...,n$ с трехдиагональной матрицей:

$$
\begin{aligned}
&2(h_1 + h_2)c_2 + h_2 c_3 = 3[(f_2 - f_1)/h_2 - (f_1 - f_0)/h_1] \\
&h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = 3\left[(f_i - f_{i-1})/h_i - (f_{i-1} - f_{i-2})/h_{i-1}\right],\ i = 3,...,n-1 \\
&h_{n-1}c_{n-1} + 2(h_{n-1} + h_n)c_n = 3[(f_n - f_{n-1})/h_n - (f_{n-1} - f_{n-2})/h_{n-1}]
\end{aligned}
\tag{3.13}
$$

Остальные коэффициенты сплайнов могут быть восстановлены по формулам:

$$
\begin{aligned}
&a_i = f_{i-1},\ i = 1,..n; \quad b_i = (f_i - f_{i-1})/h_i - \frac{1}{3}h_i(c_{i+1} + 2c_i), \quad d_i = \frac{c_{i+1} - c_i}{3h_i}, i = 1,...,n-1 \\
&c_1 = 0, \quad b_n = (f_n - f_{n-1})/h_n - \frac{2}{3}h_n c_n, \quad d_n = -\frac{c_n}{3h_n}
\end{aligned}
\tag{3.14}
$$

Пример работы:

```
$ ./prog < input1
x: -0.4 -0.1 0.2 0.5 0.8
f: -0.41152 -0.10017 0.20136 0.5236 0.9273
x*: 0.1
CS(x*): 0.10134
```

***Задача 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.

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $x_i$ | -0.7 | -0.4 | -0.1 | 0.2 | 0.5 | 0.8 |
| $y_i$ | -0.7754 | -0.41152 | -0.10017 | 0.20136 | 0.5236 | 0.9273 |

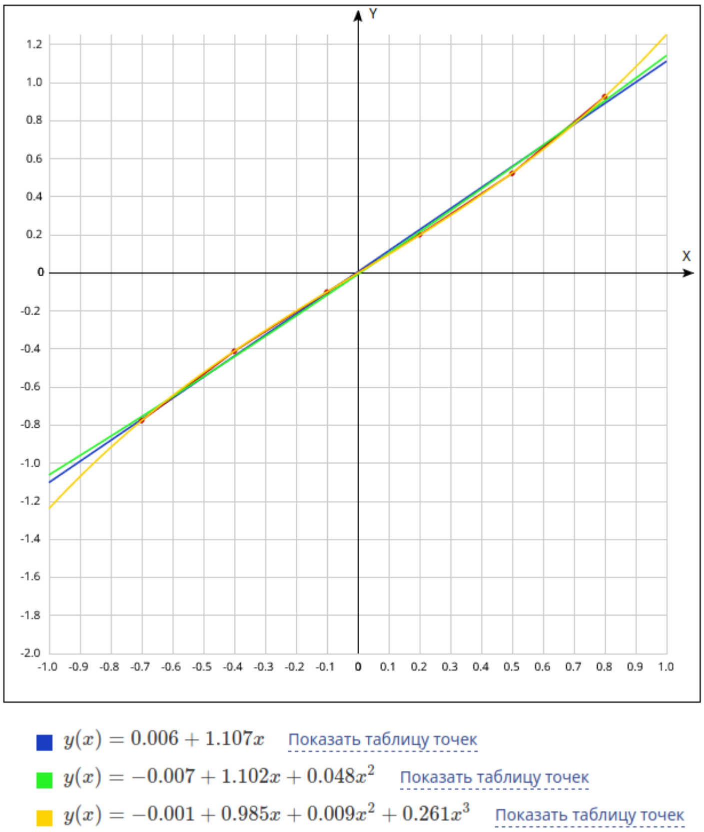
**Метод решения**

Для решения этой задачи реализован алгоритм метода наименьших квадратов.
В нем из соображений минимизации суммы квадратичных отклонений по всем точкам строится система линейных уравнений. Ее решаем при помощи LU-разложения.

Пример работы:

```
$ ./prog < input1
POINTS:
(-0.7;-0.7754)(-0.4;-0.41152)(-0.1;-0.10017)(0.2;0.20136)(0.5;0.5236)
(0.8;0.9273)
COEFS:
0.00552648 1.1067
err: 0.00394166


$ ./prog < input2
POINTS:
(-0.7;-0.7754)(-0.4;-0.41152)(-0.1;-0.10017)(0.2;0.20136)(0.5;0.5236)
(0.8;0.9273)
COEFS:
-0.0069917 1.10189 0.0481468
err: 0.00324066
```




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

$X^* =$ `1.0`

| $i$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $x_i$ | -1.0 | 0.0 | 1.0 | 2.0 | 3.0 |
| $y_i$ | -0.7854 | 0.0 | 0.78540 | 1.1071 | 1.249 |

**Метод решения**

Первая производная для табличной функции в области ее определения рассчитывается с первым порядком точности по формуле

$$y'(x) \approx \varphi'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Вторая производная рассчитывается со вторым порядком точности по формуле

$$y''(x) \approx \varphi''(x) = 2\frac{\dfrac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \dfrac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}$$

Пример работы:

```
$ ./prog < input1
POINTS:
(-1;-0.7854)(0;0)(1;0.7854)(2;1.1071)(3;1.249)
x*: 1
```

```
f'(x*): 0.7854
f''(x*): -0.4637
```

***Задача 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.

$$y = \frac{1}{x^2 + 4}, \quad X_0 = -2, \quad X_k = 2, \quad h_1 = 1.0, \quad h_2 = 0.5$$

**Метод решения**
Для решения задачи реализованы методы численного интегрирования: метод прямоугольников, трапеций, Симпсона.

Численное интегрирование методом прямоугольников производится по формуле

$$\int_a^b f(x)\, dx \approx \sum_{i=1}^{N} h_i\, f\left(\frac{x_{i-1} + x_i}{2}\right)$$

Методом трапеций по формуле

$$F = \int_a^b f(x)\, dx \approx \frac{1}{2} \sum_{i=1}^{N} (f_i + f_{i-1})\, h_i$$

Методом Симпсона по формуле

$$F = \int_a^b f(x)\, dx \approx \frac{1}{3} \sum_{i=1}^{N} (f_{i-1} + 4 f_{i-\frac{1}{2}} + f_i)\, h_i$$

Пример работы:

```
$ ./prog < input1
rectangle method
with h: 1: 0.790588
with h: 0.5: 0.7867
error estimation: 0.00129604

$ ./prog < input2
trapezoid method
with h: 1: 0.775
with h: 0.5: 0.782794
error estimation: 0.00259804

$ ./prog < input3
Simpson method
with h: 1: 0.785392
with h: 0.5: 0.785398
error estimation: 3.97917e-07
```

# Лабораторная работа №4

***Задача 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.

| Задача Коши | Точное решение |
|---|---|
| $y'' - 4xy' + (4x^2 - 3)y - e^{x^2} = 0$,<br>$y(0) = 1$,<br>$y'(0) = 0$,<br>$x \in [0,1], h = 0.1$ | $y = (e^x + e^{-x} - 1)e^{x^2}$ |

**Метод решения**

Для решения задачи реализованы методы Эйлера, Рунге-Кутты и Адамса 4-го порядка.

Методы Эйлера и Рунге-Кутты являются одношаговыми методами. Метод же Адамся является многошаговым.

При поставленной задаче

$$y' = f(x, y)$$
$$y(x_0) = y_0$$

решение методом Эйлера производится в точках сетки $y_i$ с шагом *h* по формуле

$$y_{k+1} = y_k + hf(x_k, y_k)$$

Реализованный метод Рунге-Кутты 4-го порядка точности расчитывается по формулам

$$y_{k+1} = y_k + \Delta y_k$$
$$\Delta y_k = \frac{1}{6}(K_1^k + 2K_2^k + 2K_3^k + K_4^k)$$

$$K_1^k = hf(x_k, y_k)$$
$$K_2^k = hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1^k)$$
$$K_3^k = hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2^k)$$
$$K_4^k = hf(x_k + h, y_k + K_3^k)$$

Решение методом Адамса осуществляется по формуле

$$y_{k+1} = y_k + \frac{h}{24}(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}),$$

где $f_k$ значение подынтегральной функции в узле $x_k$ .

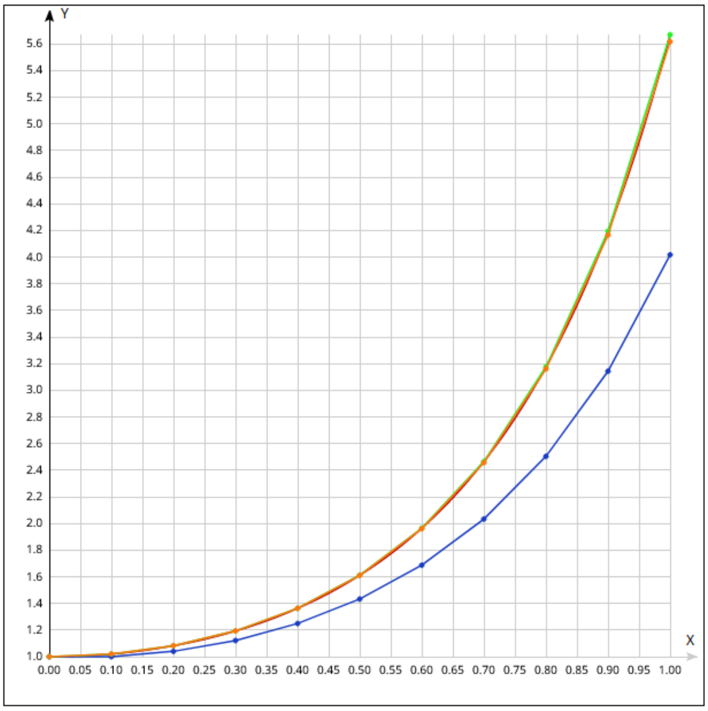
Таким образом решаются задачи Коши первого порядка. Системы ОДУ первого порядка решаются аналогично и теми же методами, что и ОДУ первого порядка. Задачи Коши большего порядка или такие же системы решаются путем к сведению к системе первого порядка и применению одного из указанных методов.

Пример работы:

```
$ ./prog < input1
euler method
(0;1)(0.1;1)(0.2;1.04)(0.3;1.1213)(0.4;1.24905)(0.5;1.43267)(0.6;1.68689)
(0.7;2.03344)(0.8;2.50381)(0.9;3.14336)(1;4.01754)
error estimation: 0.224686
absolute error: 1.65323

$ ./prog < input2
runge-kutta method
(0;1)(0.1;1.02016)(0.2;1.08258)(0.3;1.19338)(0.4;1.36377)(0.5;1.61174)
(0.6;1.96493)(0.7;2.46522)(0.8;3.17612)(0.9;4.19455)(1;5.66996)
error estimation: 5.01152e-05
absolute error: 0.000809489

$ ./prog < input3
adams method
(0;1)(0.1;1.02016)(0.2;1.08258)(0.3;1.19338)(0.4;1.36336)(0.5;1.60998)
(0.6;1.96086)(0.7;2.45693)(0.8;3.16035)(0.9;4.16557)(1;5.61769)
error estimation: 0.00318218
absolute error: 0.0530816
```




$y(x) = (e^{x} + e^{-x} - 1)e^{x^2}$ Показать таблицу точек

Здесь красным цветом показано аналитическое решение, синим цветом показано решение методом Эйлера, рыжим – методом Рунге-Кутты, зеленым – методом Адамса.

***Задача 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.

| Краевая задача | Точное решение |
|---|---|
| (2x+1) y″+4xy′-4y=0,<br>y′ (-2)+2y(-2)= -9,<br>y′(0)=1 | $y(x) = 3x + e^{-2x}$ |

**Метод решения**
Для решения задачи реализованы метод стрельбы и конечно-разностный метод.

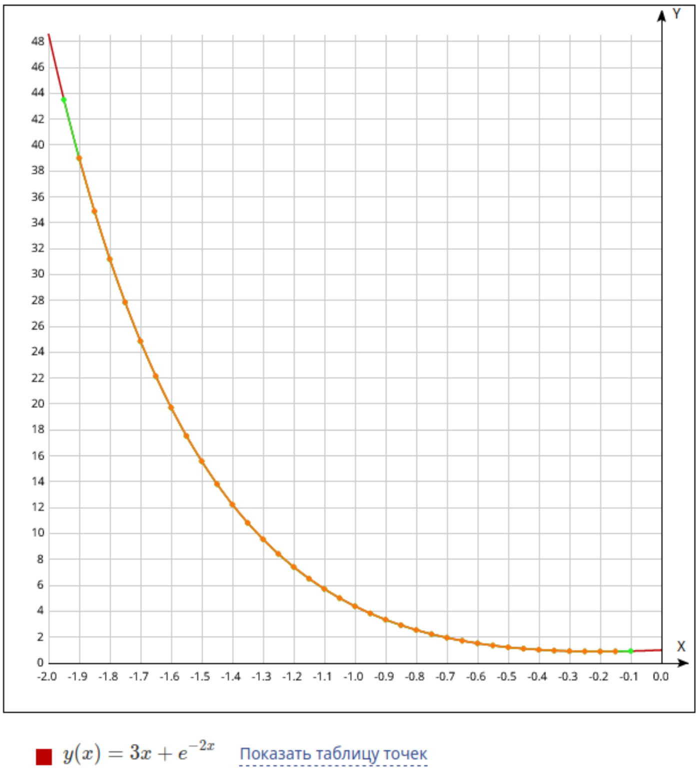
Метод стрельбы заключается в варьировании одного из параметров (значение функции или значение производной) на правом конце интересующего отрезка с тем, чтобы после двух «пристрелочных» решений задачи скорректировать этот параметр и постепенно приблизиться к заданной точности. Решения при это ищутся приближенно в данном случае методом Рунге-Кутты.

Конечно-разностный метод по сути своей состоит из замены производных и функций их конечно-разностными аналогами в точках введенной сетки и последующем решении получившейся СЛАУ, причем матрица которой имеет трехдиагональный вид. Такая система с легкостью решается методом прогонки.

Пример работы:

```
$ ./prog < input1
INPUT:
a0:2 b0:1 c0:-9
a1:0 b1:1 c1:1
a:-2 b:0
shooting method
POINTS:
(-2;48.5441)(-1.95;43.5035)(-1.9;38.9569)(-1.85;34.8572)(-1.8;31.162)(-
1.75;27.8326)(-1.7;24.8344)(-1.65;22.1358)(-1.6;19.7082)(-1.55;17.526)(-
1.5;15.5656)(-1.45;13.
8061)(-1.4;12.2284)(-1.35;10.815)(-1.3;9.5504)(-1.25;8.42042)(-1.2;7.41225)(-
1.15;6.5143)(-1.1;5.71607)(-1.05;5.00808)(-1;4.38173)(-0.95;3.82927)(-
0.9;3.34365)(-0.85
;2.91852)(-0.8;2.54812)(-0.75;2.22725)(-0.7;1.95118)(-0.65;1.71566)(-
0.6;1.51683)(-0.55;1.35119)(-0.5;1.21559)(-0.45;1.10716)(-0.4;1.02331)(-
0.35;0.961706)(-0.3;0.92
0225)(-0.25;0.896954)(-0.2;0.89016)(-0.15;0.898274)(-0.1;0.919879)(-
0.05;0.953691)
CHECKOUT:
-0.000250006 -0.000214297 -0.000183233 -0.000156011 -0.000132392 -0.000111772 -
9.39315e-05 -7.84117e-05 -6.50582e-05 -5.35036e-05 -4.35796e-05 -3.50521e-05 -
2.77536e
-05 -2.15254e-05 -1.62155e-05 -1.17436e-05 -7.96467e-06 -4.80005e-06 -2.17108e-
06 -6.5559e-07 1.78183e-06 3.22462e-06 4.37391e-06 5.27926e-06 5.96872e-06
6.48182e-06
6.84285e-06 7.0757e-06 7.20293e-06 9.75077e-06 7.20449e-06 7.11188e-06
6.97105e-06 6.79271e-06 6.58737e-06 6.35525e-06 6.11079e-06 5.854e-06
```

```
$ ./prog < input2
INPUT:
a0:2 b0:1 c0:-9
a1:0 b1:1 c1:1
a:-2 b:0
finite-difference method
POINTS:
(-2;48.5981)(-1.95;43.5524)(-1.9;39.0012)(-1.85;34.8973)(-1.8;31.1982)(-
1.75;27.8654)(-1.7;24.8641)(-1.65;22.1626)(-1.6;19.7325)(-1.55;17.5479)(-
1.5;15.5855)(-1.45;1
3.8241)(-1.4;12.2446)(-1.35;10.8297)(-1.3;9.56373)(-1.25;8.43249)(-1.2;7.42317)
(-1.15;6.52418)(-1.1;5.72501)(-1.05;5.01617)(-1;4.38905)(-0.95;3.83589)(-
0.9;3.34965)(
-0.85;2.92395)(-0.8;2.55303)(-0.75;2.23169)(-0.7;1.9552)(-0.65;1.7193)(-
0.6;1.52012)(-0.55;1.35417)(-0.5;1.21828)(-0.45;1.1096)(-0.4;1.02554)(-
0.35;0.963752)(-0.3;0.
922118)(-0.25;0.898721)(-0.2;0.891824)(-0.15;0.899859)(-0.1;0.921402)(-
0.05;0.955171)
CHECKOUT:
-0.000253422 -0.000217499 -0.000185945 -0.000157988 -0.000134331 -0.000113897 -
9.57915e-05 -8.00548e-05 -6.64031e-05 -5.46266e-05 -4.45145e-05 -3.58426e-05 -
2.84464e
-05 -2.21582e-05 -1.68049e-05 -1.22544e-05 -8.44518e-06 -5.23725e-06 -2.48281e-
06 -3.08626e-07 1.53087e-06 3.01364e-06 4.18764e-06 5.11837e-06 5.82536e-06
6.3599e-06
6.73796e-06 6.98613e-06 7.12577e-06 7.17628e-06 7.15273e-06 7.06888e-06
6.93606e-06 6.76506e-06 6.56162e-06 6.33544e-06 6.09637e-06 5.84186e-06
```




Здесь красным цветом показано аналитическое решение, зеленым – решение методом стрельбы, рыжим – конечно-разностным методом.