**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»**

**(КНИТУ-КАИ)**

**Институт Компьютерных технологий и защиты информации**

**Кафедра Динамики процессов и управления**

**ОТЧЕТ**

**по лабораторным работам «Численные методы»**

Направление подготовки/специальность: **09.03.03 «Прикладная информатика»**

Выполнил:

гр. 4217 Старостин Э Р.

Казань, 2021 год

Оглавление

[Лабораторная работа №1 3](#_Toc74675136)

[Линейные операции над векторами. 3](#_Toc74675137)

[Лабораторная работа №2 4](#_Toc74675138)

[Разложение синуса и экспоненты в ряд Тейлора, нахождение корня формулой Герона, реализация схемы Горнера, метода половинного деления, метода Ньютона и метода последовательных приближений. 4](#_Toc74675139)

[Лабораторная работа №3 6](#_Toc74675140)

[Действия над матрицами 6](#_Toc74675141)

[Лабораторная работа №4 8](#_Toc74675142)

[Решение систем линейных алгебраических уравнений (СЛАУ) 8](#_Toc74675143)

[Лабораторная работа №5 11](#_Toc74675144)

[Решение СЛАУ методом ортогонализации и методом последовательных приближений 11](#_Toc74675145)

[Лабораторная работа №6 13](#_Toc74675149)

[Сплайн функции и метод наименьших квадратов. 13](#_Toc74675150)

[Лабораторная работа №7 15](#_Toc74675151)

[Вычисление определенного интеграла. 15](#_Toc74675152)

[Лабораторная работа №8 18](#_Toc74675153)

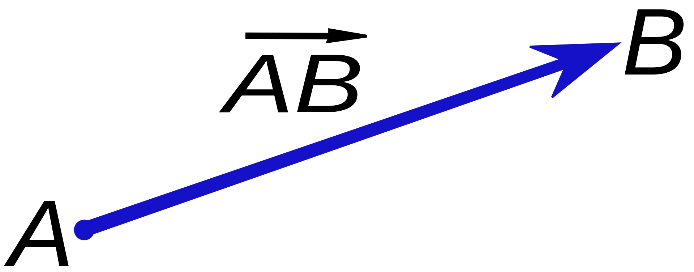
[Решение обыкновенных дифференциальных уравнений (ОДУ). 18](#_Toc74675154)

[Список литературы 22](#_Toc74675155)

# Лабораторная работа №1

## Линейные операции над векторами.

Ве́ктор (от лат. vector, «несущий») — в простейшем случае математический объект, характеризующийся величиной и направлением. Например, в геометрии и в естественных науках вектор есть направленный отрезок прямой в евклидовом пространстве (или на плоскости)



Создаем **класс Vector**.

Класс Vector **принимает** следующие значения:

* public double[] vector; (Массив векторов)
* public int size = 0; (Размерность)

Класс Vector также содержит пустой конструктор и конструктор с выше перечисленными данными.

**Методы и свойства** данного класса:

* public double this[int i] – индексатор.
* ToString() и View() – методы для просмотра векторов в консоли.
* Clear() – метод очистки векторов
* Size { get { return size; } } – свойство, определяющее размерность.
* NormaE()-нормализация вектора.
* Copy() – метод для создания копии вектора.
* Addition(Vector a) –сложение векторов
* Subtraction(Vector a) –вычитание векторов
* Multiplication(Vector a) –скалярное умножение векторов
* Multiplication\_x(double x) –умножение вектора на число.
* Len() – метод, определяющий длину вектора .
* ScalarProduct()-метод, определяющий длину вектора

Помимо стандартной реализации выше перечисленных методов, в C# существует возможность реализовать их при помощи «operator».

В **классе** Program создаётся **объект класса** Vector и соответствующие примеры. Используя выше перечисленные методы, выполняются различные вычисления.

Использованная литература: 1, 2.

# Лабораторная работа №2

## Разложение синуса и экспоненты в ряд Тейлора, нахождение корня формулой Герона, реализация схемы Горнера, метода половинного деления, метода Ньютона и метода последовательных приближений.

Для выполнения второй лабораторной работы мной был создан **класс** ChisMethod

В нем создается **делегат** double Fun, передающий параметр double x.

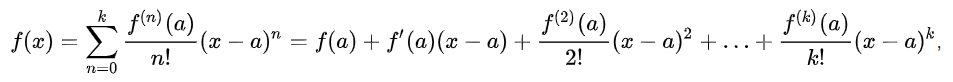
**Методы и свойства**:

* ***SinP(double x, double eps)*** - разложение синуса в ряд Тейлора.

*Входные данные*:

* x (переменная для вычислений),
* eps (эпсилон).

Ряд Тейлора — разложение функции в бесконечную сумму степенных функций



Разложение синуса в ряд Тейлора представлен:

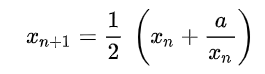


* ***SqrtP(double a, double eps)*** – метод для нахождения корня формулой Герона.

*Входные данные*:

* x (переменная для вычислений),
* eps (эпсилон).

Формула Герона представлена на:





*Условие сходимости*: |Xs – Xt| > eps.

(Xs – x следующее, Xt – x текущее)

* ***Gorner(double[] a, double x)*** – схема Горнера.

*Входные данные*:

* а (массив значений а),
* x (переменная для вычислений)

Схема Горнера – способ деления многочлена

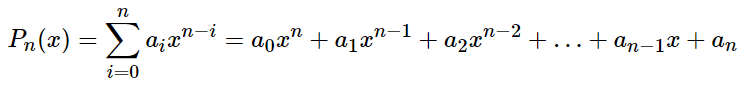


Схема Горнера- алгоритм вычисления значения многочлена, записанного в виде суммы мономов (одночленов), при заданном значении переменной. Метод Горнера позволяет найти корни многочлена[1], а также вычислить производные полинома в заданной точке.

* ***KorenPD(double a, double b, double eps, Fun f)*** – метод половинного деления.
* 

*Входные данные*:

* a (значение функции в точке а),
* b (значение функции в точке b),
* eps (эпсилон),
* f (функция для вычисления).

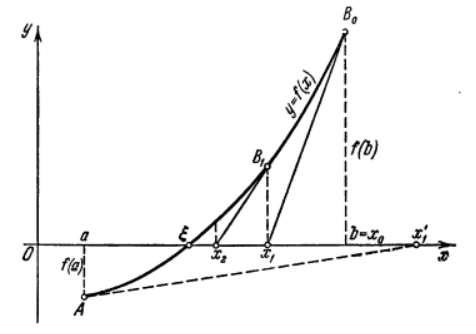
Метод деления пополам- простейший численный метод для решения нелинейных уравнений вида f(x)=0. Предполагается только непрерывность функции f(x). Поиск основывается на теореме о промежуточных значениях. *Условие сходимости*: b – a > eps.

* ***Newton(double x1, double eps, Fun f)*** – метод Ньютона.

*Входные данные*:

* x1 (переменная для вычислений),
* eps (эпсилон),
* f (функция для вычисления).

Метод Ньютона – это итерационный численный метод нахождения корня (нуля) заданной функции. Метод был впервые предложен английским физиком, математиком и астрономом Исааком Ньютоном (1643—1727). Поиск решения осуществляется путём построения последовательных приближений и основан на принципах простой итерации.



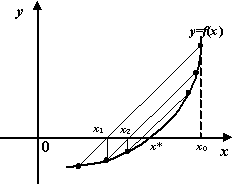
* ***PosledPribleg(double xn, double eps, Fun fi)*** – метод последовательных приближений.

*Входные данные*:

* xn (x начальное),
* eps (эпсилон),
* fi (функция для вычисления).

Метод последовательных приближений – один из простейших численных методов решения уравнений. Метод основан на принципе сжимающего отображения, который применительно к численным методам в общем виде также может называться методом простой итерации или методом последовательных приближений.

Иллюстрация последовательных приближений метода простой итерации представлена на:



*Условие сходимости*: если производная в точке x больше 1, то процесс итерации может быть расходящимся ( ). Если меньше 1, то – сходящимся.

.

В **классе** Program **создаётся объект** класса ChisMethod и соответствующие примеры. Используя выше перечисленные методы, выполняются различные вычисления.

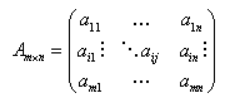
Использованная литература: 1, 3,5.

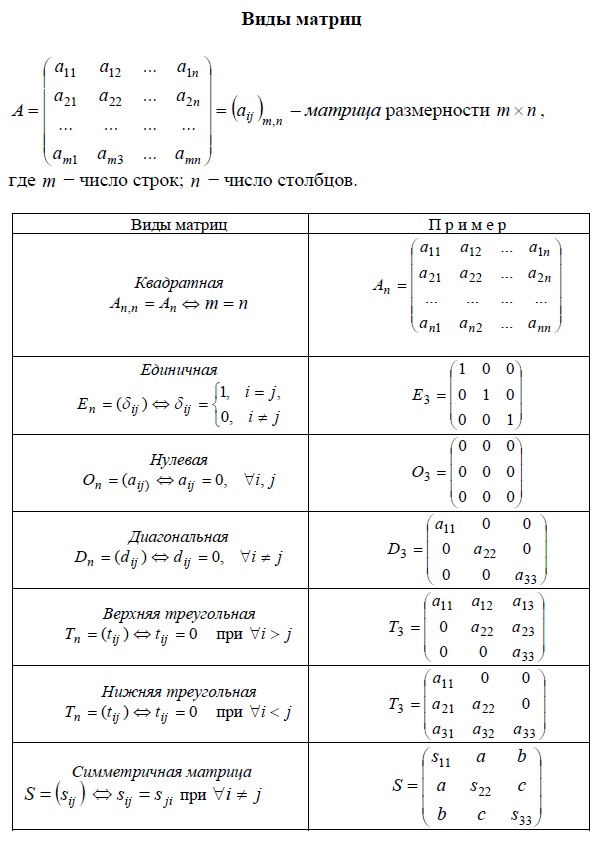
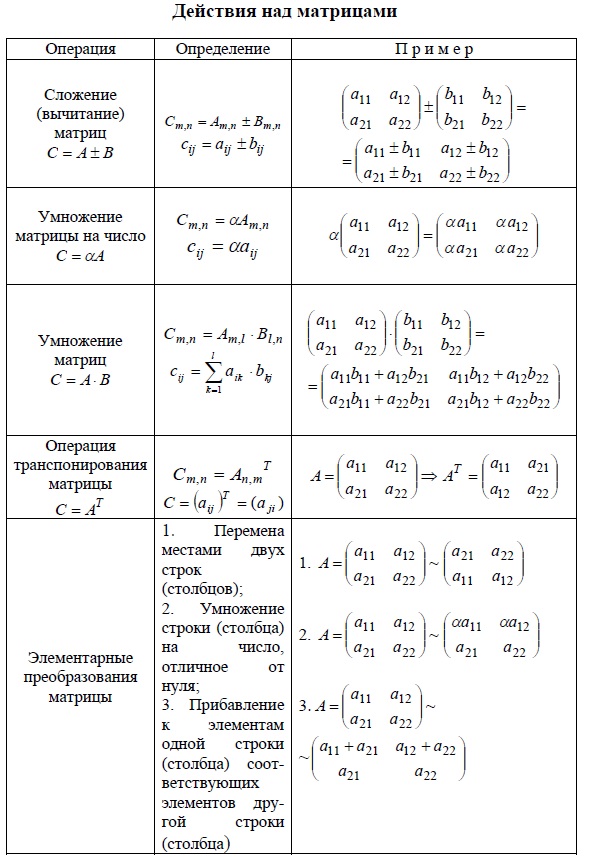
# Лабораторная работа №3

## Действия над матрицами

Ма́трица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Количество строк и столбцов задает размер матрицы. Хотя исторически рассматривались, например, треугольные матрицы[1], в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Матрицей размера m x n (читается m на n) называется прямоугольная таблица чисел, содержащая m строк и n столбцов Числа, составляющие матрицу, называются элементами матрицы.





Виды матриц Действия над матрицами

Для выполнения третий лабораторной работы мной был создан **класс** Matrix.

Класс Matrix **принимает** следующие значения:

* protected int rows, columns; (Переменные для строк и столбцов матрицы)
* protected double[,] data; (Двумерный массив, т.е. массив с 2-мя измерениями (строки и столбцы); [,,] - трехмерный массив...)

Класс Matrix также содержит пустой конструктор и конструктор с выше перечисленными данными.

**Методы и свойства** данного класса:

* GetCountRows() – получить количество строк.
* GetCountColumns() – получить количество столбцов.
* public double this[int i, int j] – индексатор.
* GetRow(int r) – получить строки.
* SetRow(int r, Vector rr) – набор строк.
* GetColumn(int c) – получить столбцы.
* SetColumn(int c, Vector cc) – набор столбцов.
* Copy() – копирование матрицы.
* PrintMatrix() – метод, который печатает матрицу.
* ToString()
* Transpr() – метод, который транспонирует матрицу.
* Umnch(Matrix a, double ch) – умножение матрицы на число.
* UmnMatrix(Matrix a, Matrix b) – умножение матрицы А на матрицу B.
* SumMatrix(Matrix a, Matrix b) – сложение матриц.
* SubtractionMatrix(Matrix a, Matrix b) – вычитание матриц.
* При помощи «operator» выполняется умножение матрицы на вектор.

В **классе** Program создаётся **объект класса** Matrix и соответствующие примеры. Используя выше перечисленные методы, выполняются различные вычисления.

Использованная литература: [1], [2], [5].

# Лабораторная работа №4

## Решение систем линейных алгебраических уравнений (СЛАУ)

* Решение СЛАУ с верхней треугольной матрицей.
* Решение СЛАУ с нижней треугольной матрицей.
* Решение СЛАУ методом Гаусса.
* Решение СЛАУ методом прогонки.

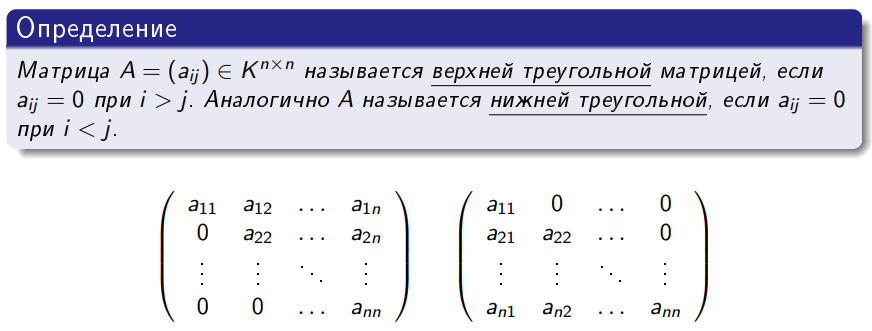
Система линейных алгебраических уравнений — система уравнений, каждое уравнение в которой является линейным — алгебраическим уравнением первой степени.

Решения всех выше перечисленных СЛАУ выполняются в **классе** **Matrix**:

* ***SLU\_DOWN(Matrix a, Vector b)*** – решение СЛАУ с верхней треугольной матрицей.

*Входные данные*:

* а (матрица а),
* b (вектор b).



Определение верхней и нижней треугольной матрицы

Пусть L – нижняя треугольная матрица, тогда система Lx=b имеет вид

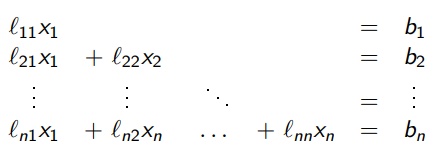
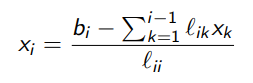


Рис. 18

Из-за особого вида системы решение легче всего получить последовательным решением уравнений в заданном порядке, что дает рекуррентные формулы



Нахождение решения по этой формуле, принято называть прямым ходом.

**Результат выполнения алгоритма**: вектор.

* ***SLU\_UP(Matrix a, Vector b)*** – решение СЛАУ с нижней треугольной матрицей.

*Входные данные*:

* а (матрица а),
* b (вектор b).

Пусть L – нижняя треугольная матрица, тогда система Lx=b имеет вид (рис. 20):

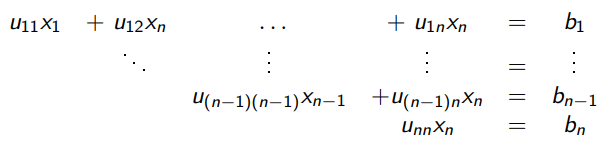


Рис. 20

Эту систему решать проще в обратном порядке, по аналогии с верхней треугольной матрицей получаем следующую формулу (рис. 21):

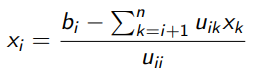


Рис. 21

Такой процесс принято называть обратным ходом.

**Результат выполнения алгоритма**: вектор.

* ***Gauss (Matrix aa, Vector bb )*** – решение СЛАУ методом Гаусса.

*Входные данные*:

* аa (матрица а),
* bb (вектор b).

Метод Гаусса **заключается** в последовательном вычитании одного уравнения из других, таким образом множество решений системы не меняется. Метод делится на n шагов, на шаге k система имеет вид (рис. 22):

То есть **целью является**: привести исходную матрицу к верхней треугольной матрице, обнуляя элементы под главной диагональю, а далее решить СЛАУ.



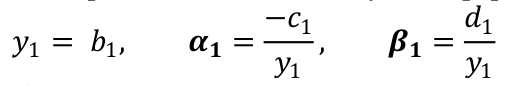
Рис. 22

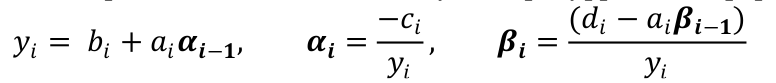
**Результат выполнения алгоритма**: вектор.

* ***MethodPoregonki (Vector v, Vector s, Vector m, Vector rav )*** – решение СЛАУ методом прогонки.

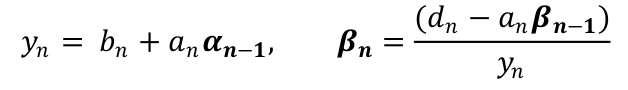
*Входные данные*:

* n (нижняя диагональ),
* s (главная диагональ),
* v (верхняя диагональ),
* rav (правая часть).

1. В первой строке матрицы (i = 1) используются формулы:  
   
2. Для строк i от 2 до n-1

 используются рекуррентные формулы:  


1. При

i = n прямая прогонка завершается вычислением:  


После этого производится обратная прогонка, в которой происходит вычисление неизвестных xi

. Этот этап выполняется при i = n...1 строго по убыванию значения i.

1. В последней строке матрицы (i = n) xn = βn

.

1. Для всех остальных строк при i от n-1 до 1 применяется формула:  
   xi

**Результат выполнения алгоритма**: вектор.

В **классе** Program создаётся объект класса Matrix и соответствующие примеры. Используя выше перечисленные методы, выполняются различные вычисления.

Использованная литература: 6, 7, 8.

# Лабораторная работа №5

## Решение СЛАУ методом ортогонализации и методом последовательных приближений

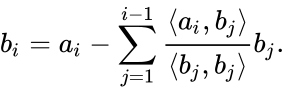
* Метод Грама - Шмидта.
* Метод последовательных приближений.

Решения всех выше перечисленных СЛАУ выполняются в **классе Matrix**:

**Процесс Грама ― Шмидта** ― наиболее известный алгоритм [ортогонализации](https://math.wikia.org/ru/wiki/%D0%9E%D1%80%D1%82%D0%BE%D0%B3%D0%BE%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), при котором по [линейно независимой](https://math.wikia.org/ru/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F_%D0%BD%D0%B5%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C) системе {\displaystyle a_{1},a_{2},...,a_{k}} строится ортогональная система {\displaystyle b_{1},b_{2},...,b_{k}} такая, что каждый вектор {\displaystyle b_{i}} линейно выражается через {\displaystyle a_{1},a_{2},...,a_{i}}, то есть матрица перехода от {\displaystyle \{a_{i}\}} к {\displaystyle \{b_{i}\}} ― верхнетреугольная матрица. При этом можно добиться того, чтобы система {\displaystyle \{b_{i}\}} была ортонормированной и чтобы диагональные элементы матрицы перехода были положительны; этими условиями система {\displaystyle \{b_{i}\}} и матрица перехода определяются однозначно.

## Алгоритм

Полагают {\displaystyle b_{1}=a_{1}}, и, если уже построены векторы {\displaystyle b_{1},b_{2},..,b_{i-1}}, то



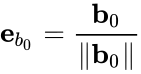
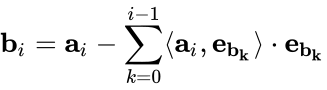
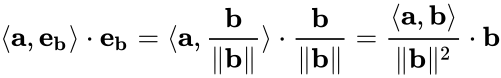
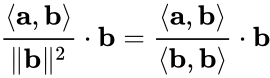
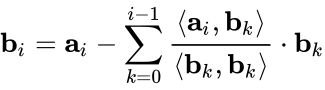
Геометрический смысл описанного процесса состоит в том, что на каждом шагу вектор {\displaystyle b_{i}} является перпендикуляром, восстановленным к [линейной оболочке](https://math.wikia.org/ru/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0) векторов {\displaystyle a_{1},...,a_{i-1}} до конца вектора {\displaystyle a_{i}}.

Нормируя полученные векторы {\displaystyle b_{i}},

## {\displaystyle c_{i}=b_{i}/|b_{i}|}

## Вывод алгоритма построения ортогонального базиса по линейно независимому набору векторов {\displaystyle {\mathbf {a} _{i}}}

Процесс построения базиса заключается в проецировании первого вектора базиса на следующие за {\displaystyle \mathbf {a} _{0}} вектора {\displaystyle \mathbf {a} _{i}} и нахождения ортогональных к этим проекциям векторов {\displaystyle \mathbf {b} _{i}}.

Первый вектор строящегося базиса выбираем так:  
1. {\displaystyle \mathbf {b} _{0}=\mathbf {a} _{0}} - так выбираем первый вектор строящегося базиса.  
2.  - нормируем вектор {\displaystyle \mathbf {b_{0}} }  
3. {\displaystyle \mathbf {b} _{1}=\mathbf {a} _{1}-\langle \mathbf {a_{1}} ,\mathbf {e_{b_{0}}} \rangle \cdot \mathbf {e_{b_{0}}} }, где {\displaystyle \langle \mathbf {a_{1}} ,\mathbf {e_{b_{0}}} \rangle \cdot \mathbf {e_{b_{0}}} } - проекция вектора {\displaystyle \mathbf {a_{1}} } на вектор {\displaystyle \mathbf {e_{b_{0}}} }, вдоль нормированного вектора {\displaystyle \mathbf {e_{b_{0}}} }.  
4. {\displaystyle \mathbf {b} _{2}=\mathbf {a} _{2}-\langle \mathbf {a} _{2},\mathbf {e_{b_{0}}} \rangle \cdot \mathbf {e_{b_{0}}} -\langle \mathbf {a} _{2},\mathbf {e_{b_{1}}} \rangle \cdot \mathbf {e_{b_{1}}} }  
5.  - в общем виде.  
Заметим что:   
  
А так как норма(длина вектора) {\displaystyle \|\mathbf {b} \|} в линейном пространстве {\displaystyle \mathbf {B} } задаётся скалярным произведением, {\displaystyle \|\mathbf {b} \|={\sqrt {\langle \mathbf {b} ,\mathbf {b} \rangle }},\quad \mathbf {b} \in \mathbf {B} .}  
получаем:   
  
*И получаем окончательный вид:*  
Первый вектор строящегося базиса определяется как:  
1. {\displaystyle \mathbf {b} _{0}=\mathbf {a} _{0}} - первый вектор с индексом {\displaystyle i=0}  
2.  - все остальные векторы с индексами: {\displaystyle i=1\ldots N-1}

**Результат выполнения алгоритма**: вектор.

* ***Vector MetodPoslPre (Matrix A, Vector B, double eps)*** – метод последовательных приближений.

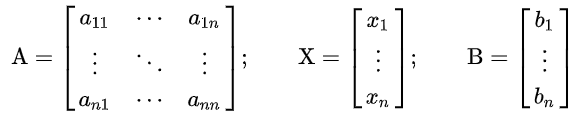
*Входные данные*:

* A (матрица),
* B (вектор),
* eps (эпсилон),

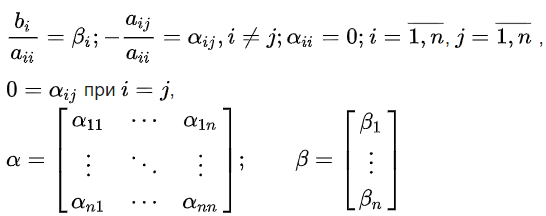
Метод последовательных приближений (метод итерации) — численный метод решения математических задач, приближённый метод решения системы линейных алгебраических уравнений вида Ax=b (рис. 26).

Суть такого метода заключается в нахождении по приближённому значению величины следующего приближения (являющегося более точным).

Метод позволяет получить значения корней системы с заданной точностью в виде предела последовательности некоторых векторов (итерационный процесс). Характер сходимости и сам факт сходимости метода зависит от выбора начального приближения корня x0.



В матричном виде получим:  , где



*Условие сходимости*: процесс итерации хорошо **сходится**, т.е. число приближений, необходимых для получения корней системы с заданной точностью, невелико, если элементы матрицы α малы по абсолютной величине. Иными словами, для успешного применения процесса итерации модуля диагональных коэффициентов системы должны быть велики по сравнению с модулями недиагональных коэффициентов этой системы.

**Результат выполнения алгоритма**: вектор.

В **классе** Program создаётся **объект класса** Matrix и соответствующие примеры. Используя выше перечисленные методы, выполняются различные вычисления.

Использованная литература: 3, 8.

# Лабораторная работа №6

## Сплайн функции и метод наименьших квадратов.

*Сплайн функции*

http://statistica.ru/upload/medialibrary/interpolyaciya-splaynami/image002.pnghttp://statistica.ru/upload/medialibrary/interpolyaciya-splaynami/image001.pngСплайн – функция, которая вместе с несколькими производными непрерывна на всем заданном отрезке [a, b], а на каждом частичном отрезке [ , ] в отдельности является некоторым алгебраическим многочленом.

Для выполнения шестой лабораторной работы мной был создан **класс Spline**.

Класс Spline **принимает** следующие значения:

* public int N; (Количество точек)
* public int countInterval; (Количество интервалов)
* public Vector xz; (x текущая)
* public Vector yz; (y текущая)
* public Vector h; (Расстояние между xi и x(i-1))

Вспомогательные векторы:

* public Vector a;
* public Vector b;
* public Vector c;
* public Vector d;

Класс Spline также содержит пустой **конструктор** и конструктор с выше перечисленными данными.

**Методы и свойства** данного класса:

* ***SolveSpline()*** – интерполирование сплайнами.

Интерполяционные формулы Лагранжа, Ньютона и Стирлинга и др. при использовании большого числа узлов интерполяции на всем отрезке [a, b] часто приводят к плохому приближению из-за накопления погрешностей в процессе вычислений. Кроме того, из-за расходимости процесса интерполяции увеличение числа узлов не обязательно приводит к повышению точности. Для снижения погрешностей весь отрезок [a, b] разбивается на частичные отрезки и на каждом из них функцию G:\Численные методы_2Курс\Интерполяция функций - Интерполирование сплайнами_files\Eqn001.png заменяют приближенно полиномом невысокой степени. Это называется кусочно-полиномиальной интерполяцией.

Один из способов интерполирования на всем отрезке [a, b] является интерполирование сплайнами.

Также для реализации данного метода используется метод прогонки [см. лаб. 4]

* ***GetValue(double x )*** – получить значение.

*Метод наименьших квадратов*

Метод наименьших квадратов – математический метод, основанный на определении аппроксимирующей функции, которая строится в ближайшей близости от точек из заданного массива экспериментальных данных. Близость исходной и аппроксимирующей функции F(x) определяется числовой мерой, а именно: сумма квадратов отклонений экспериментальных данных от аппроксимирующей кривой F(x) должна быть наименьшей.

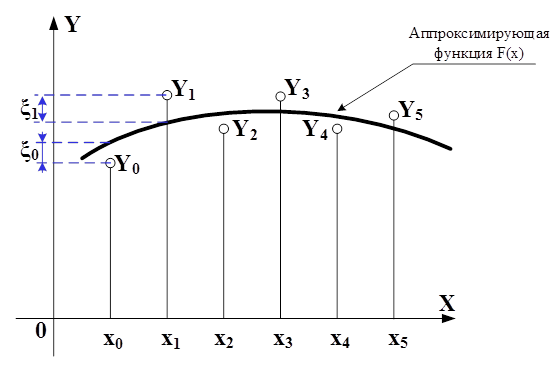


Рис. 28. Аппроксимирующая кривая, построенная по методу наименьших квадратов

Для выполнения шестой лабораторной работы мной был создан **класс MNK**.

В нем создается **делегат** Vector DelPsi, передающий параметр double x.

Класс MNK **принимает** следующие значения:

* public int N; (Количество точек)
* public int M; (Сколькими параметрами мы будем аппроксимировать (замена одних объектов другими, в каком-то смысле близкими к исходным, но более простыми)

Входные данные:

* public Vector xz; (х теоретическое)
* public Vector yz; (у теоретическое)
* public Vector p; (Вектор коэффициентов)

Класс MNK также содержит пустой **конструктор** и конструктор с выше перечисленными данными.

**Методы и свойства** данного класса:

* SolveMNK(DelPsi FPsi) – решение МНК.
* Double GetValue(double x, DelPsi FPsi) – получить значение.

В **классе** Program создаются объекты классов Spline и MNK, а также соответствующие примеры. Используя выше перечисленные методы, выполняются различные вычисления.

Использованная литература: 1, 2, 10.

# Лабораторная работа №7

## Вычисление определенного интеграла.

* Метод прямоугольников
* Метод трапеций
* Метод Симпсона
* Двумерный интеграл

*Метод* *прямоугольников*

* ***IntPR (double a, double b, double eps, Fun fun)***

*Входные данные*:

* fun (подынтегральная функция),
* a (верхняя граница),
* b (нижняя граница),
* ******eps (эпсилон).

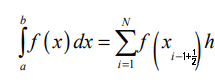
Определенный интеграл имеет вид:

Заменим интеграл (1.1) выражением f(xi-l/2)h, где xi-l/2=xi-h/2. Геометрически такая замена означает, что площадь криволинейной трапеции ABCD заменяется площадью прямоугольника ABCD (рис. 29). Тогда получаем формулу :



которая называется формулой прямоугольников на частичном отрезке [xi-l, xi].

Суммируя равенства (1.2) по i от 1 до N, получаем составную формулу прямоугольников



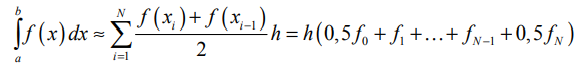
*Метод трапеций*

* ***IntTrap (double a, double b, double eps, Fun fun)***

*Входные данные*:

* fun (подынтегральная функция),
* a (верхняя граница),
* b (нижняя граница),
* eps (эпсилон).

Составная формула трапеций имеет вид:



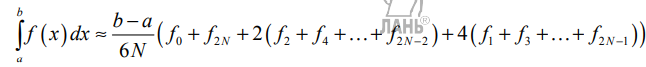
*Метод Симпсона*

***IntMeSim(double a, double b, double eps, Fun fun)***

**Входные данные**:

* fun (подынтегральная функция),
* a (верхняя граница),
* b (нижняя граница),
* eps (эпсилон).

Формула Симпсона имеет вид:



*Двумерный интеграл*

Двойной интеграл в общем виде записывается следующим образом:

Здесь D – плоская фигура, ограниченная линиями, выражения которых (равенства) даны в задании вычисления двойного интеграла. Слева и справа – равенствами, в которых слева переменная x, а сверху и снизу – равенствами, в которых слева переменная y.

*Вычисление двойного интеграла методом Монте-Карло*

* ***MonteCarlo2Integral(double xDown, double xUp, double yDown, double yUp, int N, Fun2 func)***

Для реализации метода Монте-Карло был создан **делегат** double Fun2, передающий параметр double x и double y.

*Входные данные:*

* xDown (нижняя граница по x);
* xUp (верхняя граница по y);
* yDown (нижняя граница по x);
* yUp (верхняя граница по y);
* N (общее количество генерируемых точек);
* Func (исследуемая функция для реализации программы).

Под методом Монте-Карло понимается совокупность приемов, позволяющих получать решения математических или физических задач при помощи многократных случайных величин. Оценки искомой величины выводятся статистическим путем и носят вероятностный характер. На практике случайные испытания заменяются результатами некоторых вычислений, производимых над *случайными числами*.

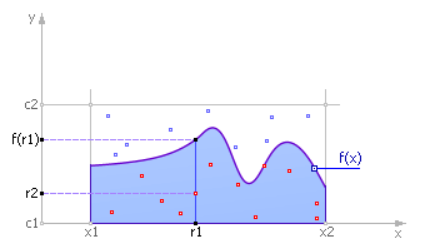


Рис. 33. Определение значения интеграла методом Монте-Карло

Решения всех выше перечисленных методов для вычисления определенного интеграла выполняются в **классе Integral**.

В **классе** Program создаётся объект класса Integral и соответствующие примеры. Используя выше перечисленные методы, выполняются различные вычисления.

Использованная литература: [1], [2], [4].

# Лабораторная работа №8

## Решение обыкновенных дифференциальных уравнений (ОДУ).

* Метод Эйлера
* Метод Рунге – Кутта 2-го порядка
* Метод Рунге – Кутта 4-го порядка
* Метод Адамса 4-го порядка

Обыкновенным дифференциальным уравнением (ОДУ) n-го порядка называется следующее уравнение, которое содержит одну или несколько производных от искомой функции y(x):



Рис. 34

здесь y(n) обозначает производную порядка n некоторой функции y(x), x – это независимая переменная.

В ряде случаев дифференциальное уравнение можно преобразовать к виду, в котором старшая производная выражена в явном виде. Такая форма записи называется уравнением, разрешенным относительно старшей производной (при этом в правой части уравнения старшая производная отсутствует):



Рис. 35

Именно такая форма записи принята в качестве стандартной при рассмотрении численных методов решения ОДУ.

Для выполнения данной лабораторной работы мной был **создан класс DiffUr**.

В нем создается **делегат** Vector PravDU, передающий параметр double t и вектор x.

**Методы и свойства** данного класса:

* ***Matrix Eiler(double tn, double tk, Vector xn, int m, PravDU prDU)***

*Входные данные*:

* tn (начальное значение отрезка),
* tk (конечное значение отрезка),
* xn (вектор),
* m (количество разбиений интервала),
* prDU (делегат правых частей).

В его основе метода Эйлера лежит аппроксимация производной отношением конечных приращений зависимой (y) и независимой (x) переменных между узлами равномерной сетки:

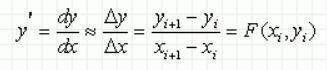
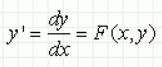
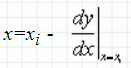


Рис. 36

где yi+1 –это искомое значение функции в точке xi+1.

Графическая интерпретация метода Эйлера также не представляет затруднений (см. рис. 36). Действительно, исходя из вида решаемого уравнения  следует, что значение F(xi, yi) есть значение производной функции y(x) в точке , и, таким образом, равно тангенсу угла наклона касательной, проведенной к графику функции y(x) в точке x=xi.

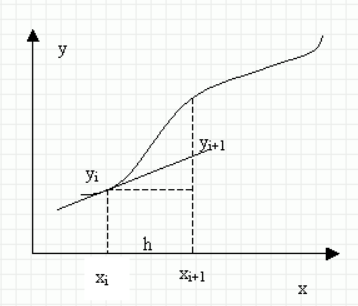


Рис. 37

Из прямоугольного треугольника на рисунке можно найти

,

откуда и получается *формула Эйлера*. Таким образом, *суть метода Эйлера* заключается в замене функции y(x) на отрезке интегрирования прямой линией, касательной к графику в точке x=xi. Если искомая функция сильно отличается от линейной на отрезке интегрирования, то погрешность вычисления будет значительной.

**Результат выполнения алгоритма**: матрица

* ***Matrix RungeKutta2(double tn, double tk, Vector xn, int m, PravDU prDu)***
* ***Matrix RungeKutta4(double tn, double tk, Vector xn, int m, PravDU prDu)***

*Входные данные*:

* tn (начальное значение отрезка),
* tk (конечное значение отрезка),
* xn (вектор),
* m (количество разбиений интервала),
* prDU (делегат правых частей).

Идея методов Рунге – Кутты 2-го и 4-го порядков основана на вычислении приближённого решения yi+1 в узле xi+1 = xi+h в виде линейной комбинации с постоянными коэффициентами:

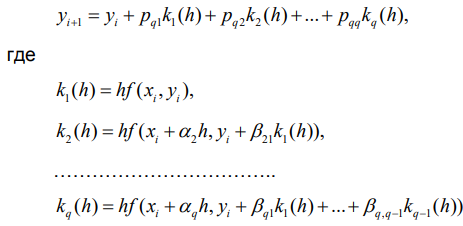


Рис. 38

При этом коэффициенты αi, βij и pqi выбираются так, чтобы разложение выражения (показанного на рис. 37) по степеням h совпадало до максимально возможной степени при произвольной правой части f(x ,y) и произвольном шаге h с разложением Тейлора искомого решения:



Рис. 39

Классическим вариантом записи метода Рунге-Кутты 2-го порядка стала следующая схема:

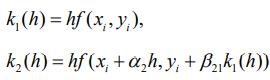


Рис. 40

Классическим вариантом записи метода Рунге-Кутты 4-го порядка стала следующая схема:

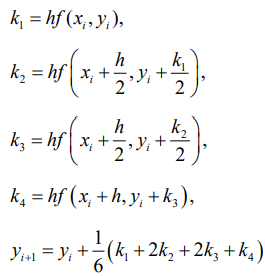


Рис. 41

**Результат выполнения алгоритмов**: матрица.

* ***Matrix Adams4(double tn, double tk, Vector xn, int m, PravDU prDu)***

*Входные данные*:

* tn (начальное значение отрезка),
* tk (конечное значение отрезка),
* xn (вектор),
* m (количество разбиений интервала),
* prDU (делегат правых частей).

Рассмотренные ранее методы (Эйлера, Рунге-Кутты) используют значение функции на одном предшествующем шаге, поэтому они относятся к так называемым *одношаговым методам*. Точность вычислений можно увеличить, если использовать при нахождении решения в некотором узле xi информацию о значениях функции, полученных в нескольких (k) *предыдущих узлах сетки интегрирования* (xi-1, xi-2 … xi-k).

Значения квадратурных коэффициентов для k от 2 до 4 приведены в таблице.

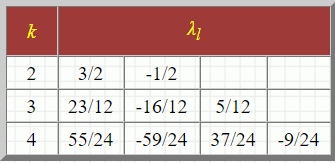


Табл. 1

Полученное таким образом семейство формул называется явной k-шаговой схемой Адамса (методы Адамса-Башфорта).

Например, четырехшаговая явная формула Адамса может быть записана так:

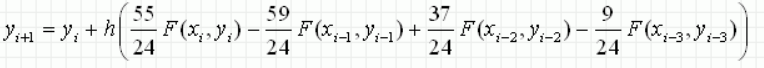


Рис. 42

**Результат выполнения алгоритма:** матрица.

В **классе** Program создаётся объект класса DiffUr и соответствующие примеры. Используя выше перечисленные методы, выполняются различные вычисления

# Список литературы

1. Б. П. Демидович и И. А. Марон, Основы вычислительной математики, М., 1966 г., 664 стр.
2. Н. В. Копченова, И. А. Марон, Вычислительная математика в примерах и задачах. Главная редакция физико-математической литературы изд-ва «Наука», М., 1972.
3. Н. Я. Виленкин, Метод последовательных приближений (Серия «Популярные лекции по математике»), М., 1968 г., 108 стр.
4. Н. А. Тюкачев, C#. Алгоритмы и структуры данных: учебное пособие для СПО [Глава 6. Некоторые численные методы. 6.3. Приближенное вычисление интегралов] / Н. А. Тюкачев, В. Г. Хлебостроев. — Санкт Петербург: Лань, 2021. — 232 с.: ил.
5. М. Дж. Сальвадори, Численные методы в технике, М., 1955 г., 243 стр.
6. Н. Н. Калиткин, Численные методы, Главная редакция физико-математической литературы изд-ва «Наука», М., 1978.
7. М. А. Черкасов, Численные методы. Решение задач. М., 2007, 88 стр.
8. Л. И. Турчак, Основы численных методов: Учеб. пособие. – М.: Наука. Гл. ред. физ.-мат. лит., 1987. – 320 с.
9. [Электронный ресурс]. Кафедра физхимии ЮФУ (РГУ) ЧИСЛЕННЫЕ МЕТОДЫ И ПРОГРАММИРОВАНИЕ. Материалы к лекционному курсу Лектор – ст. преп. Щербаков И.Н.
10. [Электронный ресурс]. http://www.machinelearning.ru/wiki/index.php?title. Интерполяция кубическими сплайнами.