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**

**по**

**численным методам**

Направление подготовки/специальность: 09.03.03 «Прикладная информатика»

Выполнил:

Шарипов Р.Р. 4216

Проверил:

Семенов П.К.

**Казань 2023**

**1.Класс поиска корней уравнения** class RootFindingAlgorithms

Класс "RootFindingAlgorithms" предназначен для реализации алгоритмов нахождения корней (корневых значений) уравнений. Корень уравнения - это значение переменной, при подстановке которого уравнение обращается в ноль.

Класс **RootFindingAlgorithms** содержит три статических метода для численного нахождения корня уравнения: **BisectionMethod**, **NewtonsMethod** и **FixedPointIteration**.

1. Метод **BisectionMethod** использует метод половинного деления (или метод бисекции) для нахождения корня уравнения на заданном интервале. Он принимает левую и правую границы интервала (**left\_board** и **right\_board** соответственно), точность (**eps**) и функцию (**func**) в качестве параметров. Метод итеративно делит интервал пополам до достижения заданной точности **eps** или нахождения корня. Если функция **func** не меняет знак на интервале, метод возвращает **double.NaN** для обозначения отсутствия корня. В противном случае метод возвращает приближенное значение корня.
2. Метод **NewtonsMethod** использует метод Ньютона для нахождения корня уравнения. Он принимает начальную точку (**currentPoint**), точность (**eps**) и функцию (**func**) в качестве параметров. Метод итеративно обновляет значение **currentPoint** на основе функции и её производной, пока разность между текущей и предыдущей точкой больше заданной точности **eps**. Если метод расходится (то есть разность между текущей и предыдущей точкой начинает увеличиваться), он возвращает **double.NaN**. В противном случае, метод возвращает приближенное значение корня.
3. Метод **FixedPointIteration** использует метод простой итерации (метод фиксированной точки) для нахождения корня уравнения. Он принимает начальную точку (**point**), точность (**eps**) и функцию (**f**) в качестве параметров. Метод итеративно обновляет значение **point**, используя функцию **f**, пока разность между текущей и предыдущей точкой больше заданной точности **eps**. Если метод расходится, он возвращает **double.NaN**. В противном случае, метод возвращает приближенное значение корня.

Все три метода используют приближенные итеративные подходы для нахождения корней уравнений и требуют задания начальной точки или интервала, а также задания требуемой точности **eps**.

**2. Класс матриц** class Matrix

Класс "Matrix" представляет собой структуру данных, используемую для работы с матрицами. Матрица — это двумерный массив чисел, организованных в виде строк и столбцов.

Класс "Matrix" обычно содержит методы и операции, которые позволяют выполнять различные операции над матрицами, такие как сложение, вычитание, умножение, нахождение определителя, решение систем линейных уравнений и другие.

Класс Matrix определяет базовые операции и методы для работы с матрицами. Вот объяснение каждого метода на русском языке:

1. Конструктор public Matrix(int r, int c): Создает матрицу с указанным числом строк (r) и столбцов (c). Все элементы матрицы инициализируются нулями.
2. Конструктор public Matrix(double[,] mm): Создает матрицу на основе двумерного массива (mm). Число строк и столбцов матрицы определяется размерностью массива mm.
3. Свойства Rows и Columns: возвращают число строк и столбцов матрицы соответственно.
4. Метод GetRow(int r): возвращает указанную строку матрицы в виде вектора.
5. Метод GetColumn(int c): возвращает указанный столбец матрицы в виде вектора.
6. Метод SetRow(int index, Vector r): заменяет указанную строку матрицы на заданный вектор. Возвращает true, если замена выполнена успешно, иначе false.
7. Метод SetColumn(int index, Vector c): Заменяет указанный столбец матрицы на заданный вектор. Возвращает true, если замена выполнена успешно, иначе false.
8. Метод SwapRows(int r1, int r2): Меняет местами указанные строки матрицы.
9. Метод Copy(): Создает и возвращает копию текущей матрицы.
10. Метод Trans(): Возвращает транспонированную матрицу.
11. Оператор +: Выполняет сложение двух матриц одинаковой размерности и возвращает новую матрицу-сумму.
12. Метод Dot(Matrix secondMatrix): Выполняет умножение текущей матрицы на другую матрицу и возвращает результат.
13. Метод Dot(Vector vector): Выполняет умножение текущей матрицы на вектор и возвращает результат.
14. Метод Dot(double value): Умножает каждый элемент матрицы на заданное значение и возвращает новую матрицу-результат.
15. Оператор \*: выполняет умножение двух матриц или матрицы на число и возвращает результат.
16. Метод SolveLUDownTriangle(Matrix A, Vector B): Решает систему линейных уравнений с нижнетреугольной матрицей A и вектором B. Возвращает вектор неизвестных значений.
17. Метод SolveLUUpTriangle(Matrix A, Vector B): Решает систему линейных уравнений с верхнетреугольной матрицей A и вектором B. Возвращает вектор неизвестных значений.
18. Метод Inverse(Matrix A): Вычисляет обратную матрицу методом LU-разложения. Возвращает обратную матрицу.

Метод прямых ходов (GaussianElimination):

Этот метод решает СЛАУ путем приведения матрицы системы к верхнетреугольному виду с помощью прямого хода метода Гаусса. Процесс прямого хода включает в себя преобразования строк, чтобы привести матрицу к треугольному виду, с нулями под главной диагональю. Затем происходит обратный ход, в котором значения неизвестных находятся путем обратной подстановки. Если матрица системы вырожденная (то есть имеет нулевую главную диагональ), или система несовместная, будет выброшено исключение. В итоге метод возвращает вектор неизвестных, который является решением СЛАУ.

Метод LU-разложения с прямым ходом (SolveLUDownTriangle):

Этот метод решает СЛАУ, используя LU-разложение матрицы системы. Сначала происходит проверка матрицы системы на корректность (невырожденность и ненулевые элементы на главной диагонали). Затем выполняется прямой ход, чтобы преобразовать матрицу к верхнетреугольному виду (матрица L - нижнетреугольная, матрица U - верхнетреугольная). После этого система разбивается на две подсистемы: Ly = b и Ux = y. Решение для первой подсистемы (нахождение вектора y) выполняется путем прямой подстановки, а затем решение для второй подсистемы (нахождение вектора x) - путем обратной подстановки. Метод возвращает вектор неизвестных, который является решением СЛАУ.

Метод LU-разложения с обратным ходом (SolveLUUpTriangle):

Этот метод также использует LU-разложение матрицы системы. После проверки матрицы системы на корректность выполняется обратный ход, чтобы привести матрицу к нижнетреугольному виду. Затем система разбивается на две подсистемы: Ux = y и Ly = b. Решение для первой подсистемы (нахождение вектора y) выполняется путем обратной подстановки, а затем решение для второй подсистемы (нахождение вектора x) - путем прямой подстановки. Метод также возвращает вектор неизвестных, который является решением СЛАУ.

Метод прогонки (RunThroughMethod): Этот метод решает СЛАУ с трехдиагональной матрицей с использованием метода прогонки. Он принимает векторы c, d, e, b, которые представляют трехдиагональную матрицу и вектор правой части, и возвращает вектор неизвестных.

Метод Гивенса (SolveUsingGivens): Этот метод решает СЛАУ с использованием преобразований Гивенса. Он приводит матрицу системы к верхнетреугольному виду с помощью преобразований Гивенса и затем использует метод LU-разложения с обратным ходом для нахождения неизвестных значений. Метод возвращает вектор неизвестных.

**3. Класс кубический сплайн интерполяции** class CubicSpline

Класс "CubicSpline" представляет собой реализацию кубического сплайна интерполяции. Кубический сплайн - это метод интерполяции, который используется для аппроксимации функции на заданном интервале с использованием кубических полиномов.

Кубический сплайн интерполяции разбивает заданный интервал на несколько подинтервалов и аппроксимирует функцию отдельно на каждом подинтервале с использованием кубического полинома. Кубический полином определяется четырьмя коэффициентами, что позволяет более гибко аппроксимировать функцию и сохранять свойства гладкости на границах подинтервалов.

1) на каждом отрезке  ,  функция  является кубическим многочленом;

2) функция , а также ее первая и вторая производные непрерывны на отрезке [*a*, *b*] ;

3) 

Третье условие называется *условием интерполирования*. Сплайн, определяемый условиями 1) – 3), называется *интерполяционным кубическим сплайном*.

Класс **CubicSpline** содержит следующие приватные поля:

* **x** - вектор узловых точек **x**
* **y** - вектор узловых точек **y**
* **h** - вектор расстояний между соседними узловыми точками
* **a** - коэффициенты **a** для сплайнов
* **b** - коэффициенты **b** для сплайнов
* **c** - коэффициенты **c** для сплайнов
* **d** - коэффициенты **d** для сплайнов

Конструктор класса принимает два вектора **x** и **y**, представляющих узловые точки. В конструкторе происходит проверка на совпадение размеров векторов **x** и **y**. Затем создаются копии входных векторов **x** и **y**, и вычисляются расстояния между соседними узловыми точками в векторе **h**. Затем вызывается метод **CalculateC()**, который вычисляет коэффициенты **c** для сплайнов. После этого вычисляются коэффициенты **a**, **b** и **d** для сплайнов.



Изображение выглядит как Шрифт, линия, типография

Автоматически созданное описание

Метод **CalculateC()** вычисляет вектор коэффициентов **c** для сплайнов. В методе создается квадратная матрица **A** размера **n x n**, где **n** - размер вектора **x**, и вектор **B** размера **n**. Задаются граничные условия, заполняется матрица **A** и вектор **B**, а затем применяется метод Гаусса (метод исключения Гаусса) для решения системы линейных уравнений и получения вектора **c**. Вектор **c** возвращается из метода.

Метод **Interpolate()** выполняет интерполяцию для заданного значения **xi**. В методе сначала находится индекс **index**, соответствующий **xi**, с помощью метода **FindIndex()**. Затем вычисляются разности **dx = xi - x[index]** и интерполированное значение **result** с использованием кубического сплайна и его коэффициентов **a**, **b**, **c** и **d**. Результат интерполяции возвращается из метода.

Метод **FindIndex()** находит индекс элемента вектора **x**, ближайшего к заданному значению **xi**. В методе проверяются граничные условия, а затем применяется бинарный поиск для нахождения индекса элемента вектора **x**.

**4. Метод наименьших квадратов** class LeastSquares

Класс "LeastSquares" представляет собой реализацию метода наименьших квадратов (МНК) для решения задачи аппроксимации данных. Метод наименьших квадратов используется для поиска оптимальных параметров модели, которая наилучшим образом соответствует набору данных.

Класс **LeastSquares** содержит следующие приватные поля:

* **x** - вектор аргументов функции
* **y** - вектор значений функции
* **p** - вектор параметров, которые будут найдены методом наименьших квадратов
* **func** - массив функций базисных функций, заданных через делегата **FuncPsi**
* **n** - количество точек (аргументов) функции
* **m** - количество базисных функций

Конструктор класса **LeastSquares** принимает векторы **x** и **y**, представляющие аргументы и значения функции соответственно, а также массив функций **func**, представляющий базисные функции. В конструкторе происходит проверка на совпадение размеров векторов **x** и **y**. Затем создаются копии входных векторов **x** и **y**, и инициализируются поля **n** и **m**. Затем вызывается метод **CalculateParameters()**, который вычисляет параметры модели методом наименьших квадратов.

Метод **CalculateParameters()** вычисляет параметры модели методом наименьших квадратов. В методе создается матрица **H** размера **n x m**, в которой каждая строка представляет значения базисных функций для соответствующей точки **x**. Затем вычисляются транспонированная матрица **HTransposed**, обратная матрица **HTHInverse** (произведение транспонированной матрицы **H** на **H**), и произведение **HTHInverse** на транспонированную матрицу **H** (**HTHInverseHT**). Вектор параметров **p** вычисляется как произведение **HTHInverseHT** на вектор **y**.

Общая формула МНК выглядит следующим образом: ***p = (H^T \* H)^(-1) \* H^T \* y,*** где:

* p - вектор параметров модели;
* H - матрица пси-функций, каждая строка которой соответствует одному наблюдению, а каждый столбец - одной пси-функции;
* H^T - транспонированная матрица H;
* y - вектор значений, которые требуется аппроксимировать;
* ^(-1) - оператор обратной матрицы.

residuals[i] = y[i] - p \* GetFunc(x[i]);

Метод GetFunc() используется для вычисления вектора пси-функций для заданного значения аргумента x.

Метод **GetFunc()** возвращает вектор значений базисных функций для заданного значения аргумента **xValue**. В методе происходит вычисление значений каждой базисной функции с использованием делегата **func** и сохранение результатов в вектор **result**, который затем возвращается из метода.

Метод **GetCriteria()** вычисляет критерий остатков, который представляет собой норму L1 (сумма модулей) вектора остатков между истинными значениями **y** и приближенными значениями, полученными с помощью найденных параметров **p**. В методе создается вектор **residuals**, в котором для каждой точки **i** вычисляется остаток как разность между истинным значением **y[i]** и значением функции, вычисленным через линейную комбинацию базисных функций и найденных параметров **p**. Затем вычисляется норма L1 вектора остатков и возвращается в качестве результата.

Класс **LeastSquares** предоставляет удобный интерфейс для аппроксимации функции с использованием метода наименьших квадратов. После создания объекта класса можно получить вектор параметров **p** и вычислить критерий остатков, что позволяет оценить точность аппроксимации.

**5. Интегралы** (class IntegralCalculator)

Класс "Integrals" представляет собой инструмент для вычисления интегралов функций. Интеграл — это математическая операция, обратная операции дифференцирования, и она используется для вычисления площади под кривой, определенного значения функции, суммы изменений или других величин, связанных с функцией.

Метод **RectangleMethod** вычисляет приближенное значение определенного интеграла на заданном интервале **[a, b]** с использованием метода прямоугольников. Метод принимает аргументы:

* **a** - левый конец интервала
* **b** - правый конец интервала
* **eps** - максимальная допустимая погрешность в численном приближении
* **function** - делегат, представляющий подынтегральную функцию

Метод использует итеративный процесс для уточнения результата. Начиная с одного интервала, он вычисляет сумму значений функции в средней точке каждого интервала и умножает ее на длину интервала. Затем количество интервалов удваивается, и процесс повторяется до тех пор, пока разница между результатами текущей и предыдущей итераций не станет меньше заданной погрешности **eps**. В конце метод возвращает приближенное значение интеграла.

Метод **TrapezoidalMethod** работает аналогично методу прямоугольников, но вычисляет интеграл с использованием метода трапеций. Он также использует итеративный процесс с удваиванием количества интервалов и вычислением суммы значений функции на каждом интервале. Результаты каждой итерации сравниваются с предыдущей итерацией до достижения заданной погрешности.

Метод **SimpsonMethod** вычисляет интеграл на заданном интервале **[a, b]** с использованием метода Симпсона. Он также принимает аргументы:

* **a** - левый конец интервала
* **b** - правый конец интервала
* **eps** - максимальная допустимая погрешность в численном приближении
* **function** - делегат, представляющий подынтегральную функцию

Метод использует итеративный процесс для приближенного вычисления интеграла. Он начинает с небольшого числа отрезков разбиения и вычисляет суммы значений функции на концах интервала и внутри интервалов. Затем он удваивает количество отрезков разбиения, уменьшает шаг разбиения и пересчитывает суммы. Процесс продолжается до тех пор, пока разница между текущим и предыдущим результатами не станет меньше заданной погрешности **eps**. В конце метод возвращает приближенное значение интеграла.

Метод **SimpsonMethod** также содержит перегруженную версию, которая вычисляет двойной интеграл функции на прямоугольнике **[ax, bx] x [ay, by]** с использованием метода Симпсона. Он принимает дополнительные аргументы **ax**, **bx**, **ay** и **by**, определяющие границы прямоугольника интегрирования. Метод использует сетку для вычисления интеграла с заданной точностью, аналогично одномерной версии метода.

Оба метода **SimpsonMethod** реализуют итерационный процесс с увеличением количества отрезков разбиения и пересчетом значений функции на каждом отрезке. Они уточняют результаты до достижения заданной погрешности **eps** и возвращают приближенное значение интеграла.

В обоих методах используется делегат **Integral** для представления подынтегральной функции, а метод **SimpsonMethod** с двумя аргументами использует делегат **DoubleIntegral** для представления двумерной подынтегральной функции.

Класс **IntegralCalculator** предоставляет удобный интерфейс для численного вычисления интегралов с помощью различных методов. Он позволяет пользователям получать приближенные значения интегралов с заданной точностью, что может быть полезно в различных математических и научных приложениях.

**6. Решение систем дифференциальных уравнений**

class DifferentialEquations

Класс **DifferentialEquations** представляет собой инструмент для численного решения дифференциальных уравнений с использованием различных методов. Класс содержит статические методы для решения дифференциальных уравнений методом Эйлера, методом Рунге-Кутты второго порядка, методом Рунге-Кутты четвертого порядка и методом Адамса.

Метод **MethodEuler** решает дифференциальное уравнение на заданном интервале **[t0, t1]** с использованием метода Эйлера. Он принимает аргументы:

* **t0** - начальное значение **t**
* **t1** - конечное значение **t**
* **h** - шаг интегрирования
* **x0** - вектор начальных значений **x**
* **f** - делегат, представляющий правую часть дифференциального уравнения

Метод использует итеративный процесс для численного решения уравнения. Он вычисляет значение функции **f** на текущем шаге и обновляет вектор **x** с использованием шага **h**. Процесс повторяется до достижения конечного значения **t1**. В конце метод возвращает вектор **x**, содержащий численное решение дифференциального уравнения.

Метод **MethodRungeKutta2** решает дифференциальное уравнение с использованием метода Рунге-Кутты второго порядка. Он работает аналогично методу Эйлера, но использует два вычисления функции **f** на каждом шаге и соответствующие коэффициенты **k1** и **k2** для обновления вектора **x**. Это позволяет получить более точное приближенное решение.

Метод **MethodRungeKutta4** решает дифференциальное уравнение с использованием метода Рунге-Кутты четвертого порядка. Он работает аналогично методу Рунге-Кутты второго порядка, но использует дополнительные коэффициенты **k3** и **k4**, чтобы еще точнее обновить вектор **x**. Метод Рунге-Кутты четвертого порядка обеспечивает еще более точное численное решение.

Метод **MethodAdams** решает дифференциальное уравнение с использованием метода Адамса (Адамса-Бэшфорта-Моултона). Он комбинирует предсказание и коррекцию для получения численного решения. Сначала метод вычисляет первые несколько шагов с использованием метода Рунге-Кутты четвертого порядка, а затем применяет формулы Адамса-Бэшфорта и Адамса-Моултона для последующих шагов. Это позволяет достичь еще более высокой точности приближенного решения.

Этап предиктор

Изображение выглядит как Шрифт, текст, рукописный текст, каллиграфия

Автоматически созданное описание

Этап корректор



Каждый из методов принимает параметры **t0** и **t1**, определяющие интервал интегрирования, а также шаг **h** и начальные значения **x0**. Делегат **f** представляет правую часть дифференциального уравнения. Методы используют итерационные процессы для приближенного численного решения уравнений и возвращают вектор **x**, содержащий результаты.

**Список литературу**

1. Б.П. Демидович, И.А. Марон - "Численные методы анализа"
2. Numerical Recipes - [www.nr.com](http://www.nr.com/): Официальный сайт книги "Numerical Recipes", которая предоставляет множество алгоритмов и методов численного анализа.
3. GitHub - github.com