Приложение 2.26.1 к ОПОП по специальности
09.02.13 Интеграция решений с применением
технологий искусственного интеллекта

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

Многопрофильный колледж

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ ЗАНЯТИЙ УЧЕБНОЙ ДИСЦИПЛИНЫ ОП.04 ЧИСЛЕННЫЕ МЕТОДЫ

для обучающихся специальности
09.02.13 Интеграция решений с применением
технологий искусственного интеллекта

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Важную часть теоретической и профессиональной практической подготовки обучающихся составляют лабораторные занятия.

Состав и содержание лабораторных занятий направлены на реализацию Федерального государственного образовательного стандарта среднего профессионального образования.

Ведущей дидактической целью лабораторных занятий является экспериментальное подтверждение и проверка существенных теоретических положений, формирование учебных практических умений, необходимых в последующей учебной деятельности.

В соответствии с рабочей программой учебной дисциплины «Численные методы» предусмотрено проведение лабораторных занятий.

В результате их выполнения обучающийся должен уметь:

– использовать основные численные методы решения математических задач;

– давать математические характеристики точности исходной информации и оценивать точность полученного численного решения;

– разрабатывать алгоритмы и программы для решения вычислительных задач, учитывая необходимую точность получаемого результата.

Содержание лабораторных занятий ориентировано на подготовку обучающихся к освоению профессиональных компетенций:

ПК 1.1 – Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

ПК 3.3 – Проводить обучение и последующую калибровку готовых моделей искусственного интеллекта.

А также формированию общих компетенций:

ОК 01 – Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

Лабораторные занятия проводятся в рамках соответствующей темы, после освоения дидактических единиц, которые обеспечивают наличие знаний, необходимых для ее выполнения.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Тема 1.1. Основные задачи численных методов

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №1
«Сравнение численных и аналитических решений для простых задач. Применение численных методов для решения инженерных задач»

Цель: получить навыки определения основных источников погрешностей и оценки точности численных решений.

Выполнив работу, Вы будете уметь:

– давать математические характеристики точности исходной информации и оценивать точность полученного численного решения; распознавать задачу в профессиональном контексте.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Численные методы дают приближённые решения математических задач. Основные источники погрешностей:

– неустранимая погрешность (погрешность исходных данных);

– погрешность метода (замена точного алгоритма приближённым);

– погрешность округления (при вычислениях на компьютере).

Абсолютная погрешность: $\Delta = |a - a\_p|$, где a – точное значение, a_p – приближённое.

Относительная погрешность: $\delta = \Delta / |a|$.

Метод половинного деления (бисекции): если $f(a)*f(b) < 0$, корень уравнения находится на отрезке [a, b]. Делим отрезок пополам $c = (a+b)/2$. Если $f(a)*f(c) < 0$, корень в [a, c], иначе в [c, b]. Повторяем до достижения точности.

Алгоритм выполнения задания:

Шаг 1. Записать уравнение $f(x) = 0$ и найти аналитическое решение.

Шаг 2. Определить отрезок [a, b], содержащий корень ($f(a)*f(b) < 0$).

Шаг 3. Реализовать метод бисекции: while $(b – a) > eps$: $c = (a + b) / 2$; выбрать половину.

Шаг 4. Вычислить абсолютную и относительную погрешности.

Шаг 5. Сравнить с аналитическим решением.

Пример выполнения:

Уравнение: $x^2 – 2 = 0$. Аналитическое решение: $x = \sqrt{2} \approx 1.41421356...$

Отрезок: [1, 2], $f(1) = –1 < 0$, $f(2) = 2 > 0$.

Итерация 1: $c = 1.5$, $f(1.5) = 0.25 > 0 \rightarrow$ отрезок [1, 1.5].

Итерация 2: $c = 1.25$, $f(1.25) = –0.4375 < 0 \rightarrow$ отрезок [1.25, 1.5].

Итерация 3: $c = 1.375$, $f(1.375) = –0.109 < 0 \rightarrow$ отрезок [1.375, 1.5].

... После 7 итераций: $x \approx 1.414$, $\Delta = |1.414 – 1.41421| \approx 0.00021$, $\delta \approx 0.015\%$.

```
>>> # Python:
>>> def bisect(f, a, b, eps=0.01):
>>>     while (b - a) > eps:
>>>         c = (a + b) / 2
>>>         if f(a) * f(c) < 0: b = c
```

```
>>>         else: a = c
>>>     return (a + b) / 2
```

Задание:

1. Изучить теоретический материал о видах погрешностей: неустранимая, погрешность метода, погрешность округления.

2. Решить уравнение $x^2 – 2 = 0$ аналитически и методом половинного деления с точностью 0,01.

3. Сравнить результаты, вычислить абсолютную и относительную погрешности.

4. Определить число верных значащих цифр приближённого решения.

5. Оформить отчёт с выводами о соотношении точности аналитического и численного решений.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 1.2. Линейные уравнения и системы уравнений

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №2

«Решение систем линейных уравнений методом Гаусса»

Цель: освоить метод Гаусса для решения СЛАУ.

Выполнив работу, Вы будете уметь:

– разрабатывать алгоритмы для решения вычислительных задач; анализировать задачу и выделять её составные части.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Метод Гаусса – прямой метод решения систем линейных алгебраических уравнений (СЛАУ) вида Ax = b.

Метод состоит из двух этапов:

Прямой ход: последовательным исключением переменных матрица системы приводится к верхнетреугольному виду. Из i-й строки вычитается k-я строка, умноженная на коэффициент m = a_ik / a_kk.

Обратный ход: из последнего уравнения находится x_n, затем подстановкой снизу вверх находятся остальные неизвестные.

Условие применимости: все ведущие элементы a_kk не должны быть равны нулю. При необходимости применяется выбор главного элемента (перестановка строк).

Алгоритм выполнения задания:

Шаг 1. Записать расширенную матрицу [A|b].

Шаг 2. Прямой ход: для k от 0 до n-1, для i от k+1 до n-1: m = A[i][k] / A[k][k]; вычесть из i-й строки k-ю, умноженную на m.

Шаг 3. Обратный ход: x[n-1] = b[n-1] / A[n-1][n-1]; для i от n-2 до 0: x[i] = (b[i] – sum(A[i][j]*x[j])) / A[i][i].

Шаг 4. Проверить решение подстановкой в исходную систему.

Пример выполнения:

Система: 2x + y – z = 8; –3x – y + 2z = –11; –2x + y + 2z = –3.

Расширенная матрица: [[2, 1, -1, 8], [-3, -1, 2, -11], [-2, 1, 2, -3]].

Прямой ход: R2 = R2 + 1.5*R1 → [0, 0.5, 0.5, 1]; R3 = R3 + R1 → [0, 2, 1, 5].

Продолжаем: R3 = R3 – 4*R2 → [0, 0, –1, 1].

Обратный ход: z = –1; y = (1 – 0.5*(–1)) / 0.5 = 3; x = (8 – 3 + (–1)) / 2 = 2.

Ответ: x = 2, y = 3, z = –1.

```
>>> import numpy as np
>>> A = np.array([[2,1,-1],[- 3,-1,2],[-2,1,2]], float)
>>> b = np.array([8,-11,-3], float)
>>> print(np.linalg.solve(A, b))  # [2. 3. -1.]
```

Задание:

1. Изучить алгоритм метода Гаусса: прямой ход (приведение к треугольному виду) и обратный ход.

2. Решить СЛАУ из 3 уравнений с 3 неизвестными вручную методом Гаусса.

3. Реализовать алгоритм метода Гаусса на языке Python.
4. Проверить решение с помощью numpy.linalg.solve.
5. Оформить отчёт с описанием алгоритма, листингом программы и результатами.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 1.2. Линейные уравнения и системы уравнений

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №3
«Решение систем линейных уравнений методом Крамера»

Цель: освоить метод Крамера для решения СЛАУ.

Выполнив работу, Вы будете уметь:

– разрабатывать алгоритмы для решения вычислительных задач; определять этапы решения задачи.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Метод Крамера – метод решения СЛАУ из n уравнений с n неизвестными при условии, что определитель матрицы системы $D \neq 0$.

Формулы Крамера: $x_i = D_i / D$, где D – определитель матрицы A, D_i – определитель матрицы, полученной заменой i-го столбца столбцом свободных членов b.

Определитель матрицы 2×2: det = a*d – b*c.

Определитель матрицы 3×3 вычисляется разложением по первой строке (правило Саррюса).

Алгоритм выполнения задания:

Шаг 1. Вычислить определитель D матрицы коэффициентов.
Шаг 2. Если D = 0, метод Крамера неприменим.
Шаг 3. Для каждого i заменить i-й столбец на столбец b и вычислить D_i.
Шаг 4. Вычислить $x_i = D_i / D$.

Пример выполнения:

Система: x + 2y = 5; 3x + 4y = 11.
D = |1 2; 3 4| = 1*4 – 2*3 = –2.
D_x = |5 2; 11 4| = 5*4 – 2*11 = –2. → x = –2 / –2 = 1.
D_y = |1 5; 3 11| = 1*11 – 5*3 = –4. → y = –4 / –2 = 2.
Проверка: 1 + 2*2 = 5 ✓; 3*1 + 4*2 = 11 ✓.

Задание:

1. Изучить формулы Крамера для решения СЛАУ.
2. Решить СЛАУ из 3 уравнений с 3 неизвестными по формулам Крамера вручную.
3. Реализовать метод Крамера на Python с вычислением определителей.
4. Сравнить результат с решением методом Гаусса.
5. Оформить отчёт.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 1.2. Линейные уравнения и системы уравнений

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №4

«Применение численных методов для систем уравнений»

Цель: освоить итерационные методы решения больших СЛАУ.

Выполнив работу, Вы будете уметь:

– разрабатывать алгоритмы и программы для решения вычислительных задач, учитывая необходимую точность.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Метод простых итераций – итерационный метод решения СЛАУ вида x = Bx + c, где B – матрица итерации, c – вектор.

Система Ax = b преобразуется к виду x_i = (b_i – sum(a_ij * x_j, j≠i)) / a_ii.

Условие сходимости: спектральный радиус матрицы B должен быть меньше 1, или достаточное условие – диагональное преобладание: |a_ii| > sum(|a_ij|, j≠i).

Итерации продолжаются до достижения условия ||x^(k+1) – x^(k)|| < eps.

Алгоритм выполнения задания:

Шаг 1. Проверить условие диагонального преобладания. При необходимости переставить строки.

Шаг 2. Привести систему к виду x_i = ... (выразить каждую переменную).

Шаг 3. Задать начальное приближение x^(0) (например, нулевой вектор).

Шаг 4. Выполнять итерации до ||x^(k+1) – x^(k)|| < eps.

Шаг 5. Записать число итераций и проверить решение.

Пример выполнения:

Система: 10x + y + z = 12; x + 10y + z = 12; x + y + 10z = 12.

Диагональное преобладание выполнено: 10 > 1+1.

Итерационные формулы: x = (12 – y – z)/10; y = (12 – x – z)/10; z = (12 – x – y)/10.

x^(0) = (0, 0, 0). Итерация 1: x = 1.2, y = 1.2, z = 1.2.

Итерация 2: x = (12 – 1.2 – 1.2)/10 = 0.96; аналогично y = z = 0.96.

... После 5–6 итераций: x ≈ y ≈ z ≈ 1.0.

Задание:

1. Изучить метод простых итераций для решения СЛАУ.
2. Реализовать на Python метод простых итераций с заданной точностью eps = 0,001.
3. Решить СЛАУ из 4 уравнений итерационным методом.
4. Исследовать сходимость метода при различных начальных приближениях.
5. Оформить отчёт с графиком сходимости.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 1.3. Нелинейные уравнения

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №5

«Реализация метода Ньютона для решения нелинейных уравнений»

Цель: освоить метод Ньютона (касательных) для решения нелинейных уравнений.

Выполнив работу, Вы будете уметь:

– использовать основные численные методы решения математических задач; оценивать точность полученного решения.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Метод Ньютона (касательных) – итерационный метод решения нелинейных уравнений $f(x) = 0$.

Формула итерации: $x_{n+1} = x_n - f(x_n) / f'(x_n)$.

Геометрический смысл: в каждой точке x_n проводится касательная к графику $f(x)$, и следующее приближение – точка пересечения касательной с осью x.

Метод имеет квадратичную скорость сходимости вблизи корня: погрешность на каждой итерации примерно возводится в квадрат.

Условия сходимости: начальное приближение должно быть достаточно близко к корню; $f'(x) \neq 0$ в окрестности корня.

Алгоритм выполнения задания:

Шаг 1. Определить функцию $f(x)$ и её производную $f'(x)$.
Шаг 2. Выбрать начальное приближение x_0.
Шаг 3. Вычислить $x_{n+1} = x_n - f(x_n) / f'(x_n)$.
Шаг 4. Проверить условие $|x_{n+1} - x_n| < eps$.
Шаг 5. Если условие не выполнено, повторить шаг 3.

Пример выполнения:

Уравнение: $x^3 - 2x - 5 = 0$. $f(x) = x^3 - 2x - 5$, $f'(x) = 3x^2 - 2$.
$x_0 = 2$. $f(2) = 8 - 4 - 5 = -1$. $f'(2) = 12 - 2 = 10$.
$x_1 = 2 - (-1)/10 = 2.1$.
$f(2.1) = 9.261 - 4.2 - 5 = 0.061$. $f'(2.1) = 13.23 - 2 = 11.23$.
$x_2 = 2.1 - 0.061/11.23 \approx 2.0946$.

```
>>> def newton(f, df, x0, eps=1e-4):
>>>     x = x0
>>>     while True:
>>>         x_new = x - f(x) / df(x)
>>>         if abs(x_new - x) < eps: return x_new
>>>         x = x_new
```

Задание:

1. Изучить алгоритм метода Ньютона: формулу итерации $x(n+1) = x(n) - f(x(n))/f'(x(n))$.
2. Решить уравнение $x^3 - 2x - 5 = 0$ методом Ньютона с точностью 0,0001, начиная с $x0 = 2$.
3. Реализовать алгоритм на Python, вывести таблицу итераций.

4. Построить график функции и показать на нём процесс приближения к корню.
5. Оформить отчёт.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 1.3. Нелинейные уравнения

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №6

«Применение численных методов для задач оптимизации в нелинейных системах»

Цель: освоить методы поиска экстремумов нелинейных функций.

Выполнив работу, Вы будете уметь:

– использовать численные методы для решения задач оптимизации; разрабатывать алгоритмы.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Метод золотого сечения – метод одномерной оптимизации для нахождения минимума унимодальной функции на отрезке [a, b].

Золотое сечение делит отрезок в пропорции $\varphi = (\sqrt{5} - 1)/2 \approx 0.618$.

На каждой итерации вычисляются две внутренние точки: $x\_1 = a + (1 - \varphi)(b - a)$ и $x\_2 = a + \varphi(b - a)$.

Если $f(x\_1) < f(x\_2)$, то минимум в [a, x_2], иначе в [x_1, b].

Преимущество: на каждой итерации нужно вычислять только одно новое значение функции.

Алгоритм выполнения задания:

Шаг 1. Задать отрезок [a, b] и точность eps.

Шаг 2. Вычислить x_1 и x_2 по формулам золотого сечения.

Шаг 3. Сравнить f(x_1) и f(x_2), сузить отрезок.

Шаг 4. Повторять, пока $|b - a| >$ eps.

Пример выполнения:

$f(x) = x^4 - 3x^2 + 2$ на [0, 2]. $\varphi = 0.618$.

$x\_1 = 0 + 0.382*2 = 0.764$; $x\_2 = 0 + 0.618*2 = 1.236$.

$f(0.764) \approx -0.406$; $f(1.236) \approx -0.406$. Почти равны → [0.764, 1.236].

... После 15 итераций: $x\_min \approx 1.225$, $f\_min \approx -0.25$.

Аналитически: $f'(x) = 4x^3 - 6x = 0 \rightarrow x = \sqrt{(3/2)} \approx 1.2247$.

Задание:

1. Изучить метод золотого сечения для поиска минимума функции одной переменной.
2. Реализовать метод на Python.
3. Найти минимум функции $f(x) = x^4 - 3x^2 + 2$ на отрезке [0, 2] с точностью 0,001.
4. Сравнить результат с аналитическим решением ($f'(x) = 0$).
5. Оформить отчёт.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 2.1. Полиномиальная интерполяция

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №7

«Интерполяция методом Лагранжа для восстановления недостающих данных»

Цель: освоить построение интерполяционного полинома Лагранжа.

Выполнив работу, Вы будете уметь:

– использовать численные методы для анализа данных; оценивать точность полученного решения.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Интерполяционный полином Лагранжа – полином степени n, проходящий через n+1 заданную точку (x_i, y_i).

Формула: L(x) = sum(y_i * l_i(x)), где l_i(x) = prod((x – x_j) / (x_i – x_j), j ≠ i) – базисные полиномы Лагранжа.

Каждый базисный полином l_i(x) принимает значение 1 в точке x_i и 0 во всех остальных узлах.

Погрешность интерполяции зависит от расположения узлов и степени полинома.

Алгоритм выполнения задания:

Шаг 1. Задать таблицу узлов (x_i, y_i).
Шаг 2. Для каждой точки вычислить базисный полином l_i(x).
Шаг 3. Вычислить L(x) = sum(y_i * l_i(x)).
Шаг 4. Проверить: L(x_i) = y_i для всех узлов.

Пример выполнения:

Точки: (1, 1), (2, 4), (3, 9).
l_0(x) = (x–2)(x–3) / ((1–2)(1–3)) = (x–2)(x–3) / 2.
l_1(x) = (x–1)(x–3) / ((2–1)(2–3)) = –(x–1)(x–3).
l_2(x) = (x–1)(x–2) / ((3–1)(3–2)) = (x–1)(x–2) / 2.
L(x) = 1*l_0 + 4*l_1 + 9*l_2 = x^2. Проверка: L(1.5) = 2.25.

Задание:

1. Изучить формулу интерполяционного полинома Лагранжа.
2. По заданным 5 точкам построить полином Лагранжа вручную.
3. Реализовать на Python функцию построения полинома Лагранжа.
4. Восстановить пропущенные значения в наборе данных.
5. Построить график полинома и исходных точек.
6. Оформить отчёт.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 2.1. Полиномиальная интерполяция

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №8

«Построение полиномиальной интерполяции для реальных данных»

Цель: применить полиномиальную интерполяцию для обработки экспериментальных данных.

Выполнив работу, Вы будете уметь:

– использовать численные методы для анализа и моделирования процессов; разрабатывать программы.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

При увеличении степени интерполяционного полинома на равномерной сетке может возникнуть явление Рунге – сильные осцилляции полинома на краях отрезка.

Для борьбы с явлением Рунге используют: узлы Чебышёва (неравномерная сетка, сгущающаяся к краям); кусочную интерполяцию (сплайны).

Узлы Чебышёва: $x_k = (a+b)/2 + (b–a)/2 * \cos((2k+1)\pi / (2n+2))$, $k = 0, ..., n$.

Алгоритм выполнения задания:

Шаг 1. Задать функцию и набор точек на равномерной сетке.

Шаг 2. Построить интерполяционный полином.

Шаг 3. Увеличить число узлов и наблюдать за поведением полинома на краях.

Шаг 4. Повторить с узлами Чебышёва и сравнить результаты.

Пример выполнения:

Функция Рунге: $f(x) = 1 / (1 + 25x^2)$ на $[–1, 1]$.

При $n = 5$ (равномерная сетка): полином хорошо приближает в центре, но осциллирует на краях.

При $n = 11$: осцилляции усиливаются – это и есть явление Рунге.

С узлами Чебышёва при $n = 11$: полином хорошо приближает функцию на всём отрезке.

```
>>> from numpy.polynomial.chebyshev import chebpts2
>>> nodes = chebpts2(12)  # 12 узлов Чебышёва на [-1, 1]
```

Задание:

1. Загрузить набор экспериментальных данных (например, зависимость температуры от времени).
2. Построить интерполяционный полином, проходящий через все точки данных.
3. Оценить погрешность интерполяции в промежуточных точках.
4. Исследовать явление Рунге при увеличении степени полинома.
5. Оформить отчёт с графиками и выводами.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 2.2. Аппроксимация функций

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №9

«Применение метода наименьших квадратов. Сплайновая аппроксимация»

Цель: освоить метод наименьших квадратов и сплайновую аппроксимацию.

Выполнив работу, Вы будете уметь:

– использовать численные методы для анализа данных; оценивать точность аппроксимации.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Метод наименьших квадратов (МНК) – метод аппроксимации, минимизирующий сумму квадратов отклонений: $S = \sum((y_i - p(x_i))^2) \rightarrow \min$.

Для линейной аппроксимации $y = ax + b$ система нормальных уравнений: $a*\sum(x_i^2) + b*\sum(x_i) = \sum(x_i*y_i)$; $a*\sum(x_i) + b*n = \sum(y_i)$.

Сплайновая интерполяция – кусочно-полиномиальная функция, гладко соединяющая соседние узлы. Кубический сплайн – на каждом отрезке полином 3-й степени.

Алгоритм выполнения задания:

Шаг 1. Задать набор данных (x_i, y_i).
Шаг 2. Для МНК: вычислить коэффициенты из системы нормальных уравнений.
Шаг 3. Для сплайна: использовать scipy.interpolate.CubicSpline.
Шаг 4. Построить графики: данные, МНК-аппроксимация, сплайн.

Пример выполнения:

Данные: (1, 2.1), (2, 3.9), (3, 6.2), (4, 7.8), (5, 10.1).
МНК (линейная): a = 1.99, b = 0.02 → $y \approx 2x + 0.02$.

```
>>> import numpy as np
>>> x = np.array([1,2,3,4,5]); y = np.array([2.1,3.9,6.2,7.8,10.1])
>>> a, b = np.polyfit(x, y, 1)  # a=1.99, b=0.02
>>> from scipy.interpolate import CubicSpline
>>> cs = CubicSpline(x, y)
>>> print(cs(2.5))  # ≈ 5.05
```

Задание:

1. Изучить метод наименьших квадратов для линейной и полиномиальной аппроксимации.
2. По заданным данным с шумом построить аппроксимирующую прямую (МНК).
3. Реализовать на Python кубическую сплайн-интерполяцию с помощью scipy.interpolate.
4. Сравнить результаты полиномиальной аппроксимации и сплайна.
5. Оформить отчёт.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 3.1. Численное дифференцирование

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №10

«Реализация методов численного дифференцирования. Анализ данных»

Цель: освоить методы конечных разностей для численного дифференцирования.

Выполнив работу, Вы будете уметь:

– использовать численные методы; давать математические характеристики точности.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Численное дифференцирование заменяет производную конечными разностями:

Правая разность: $f'(x) \approx (f(x+h) - f(x)) / h$. Погрешность $O(h)$.

Левая разность: $f'(x) \approx (f(x) - f(x-h)) / h$. Погрешность $O(h)$.

Центральная разность: $f'(x) \approx (f(x+h) - f(x-h)) / (2h)$. Погрешность $O(h^2)$ – более точная.

При уменьшении h точность растёт, но слишком малый h приводит к ошибкам округления.

Алгоритм выполнения задания:

Шаг 1. Задать функцию f(x) и точку x_0.

Шаг 2. Вычислить производную тремя формулами при h = 0.1, 0.01, 0.001.

Шаг 3. Сравнить с аналитическим значением f'(x_0).

Шаг 4. Построить график зависимости погрешности от h.

Пример выполнения:

f(x) = sin(x), x_0 = 1. Точное значение: $f'(1) = \cos(1) \approx 0.5403$.

h = 0.1: правая = $(\sin(1.1) - \sin(1))/0.1 \approx 0.4976$; центр. = $(\sin(1.1) - \sin(0.9))/0.2 \approx 0.5398$.

h = 0.01: правая ≈ 0.5361; центр. ≈ 0.54030.

Центральная разность точнее при одном и том же h.

Задание:

1. Изучить формулы конечных разностей: правая, левая, центральная.
2. Вычислить производную функции f(x) = sin(x) в точке x = 1 тремя методами с шагом h = 0,1; 0,01; 0,001.
3. Сравнить с аналитическим значением $f'(1) = \cos(1)$.
4. Построить график зависимости погрешности от шага h.
5. Оформить отчёт.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 3.2. Численное интегрирование

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №11
«Применение метода трапеций и метода Симпсона»

Цель: освоить квадратурные методы численного интегрирования.

Выполнив работу, Вы будете уметь:
– использовать численные методы; оценивать точность результата.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Метод трапеций: интеграл ≈ h/2 * (f(x_0) + 2*f(x_1) + ... + 2*f(x_{n-1}) + f(x_n)), h = (b–a)/n.

Метод Симпсона (парабол): интеграл ≈ h/3 * (f(x_0) + 4*f(x_1) + 2*f(x_2) + 4*f(x_3) + ... + f(x_n)), n – чётное.

Погрешность метода трапеций: $O(h^2)$. Погрешность метода Симпсона: $O(h^4)$ – значительно точнее.

Алгоритм выполнения задания:

Шаг 1. Задать подынтегральную функцию f(x), пределы [a, b], число разбиений n.
Шаг 2. Вычислить шаг h = (b – a) / n.
Шаг 3. Применить формулу трапеций и формулу Симпсона.
Шаг 4. Сравнить результаты при разных n.

Пример выполнения:

Интеграл: ∫ sin(x) dx от 0 до π. Точное значение: 2.

Метод трапеций, n = 4: h = π/4 ≈ 0.785. I ≈ 0.785/2 * (0 + 2*0.707 + 2*1 + 2*0.707 + 0) ≈ 1.896.

Метод Симпсона, n = 4: I ≈ 0.785/3 * (0 + 4*0.707 + 2*1 + 4*0.707 + 0) ≈ 2.005.

Симпсон при n = 4 уже даёт результат с погрешностью 0.25%, трапеции – 5.2%.

Задание:
1. Изучить формулы методов трапеций и Симпсона.
2. Вычислить интеграл функции f(x) = e^(–x²) на отрезке [0, 1] методами трапеций и Симпсона при n = 10, 100, 1000.
3. Реализовать оба метода на Python.
4. Сравнить точность методов, построить таблицу погрешностей.
5. Оформить отчёт.

Критерии оценки:
«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 4.1. Обыкновенные дифференциальные уравнения

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №12
«Решение ОДУ методом Эйлера и Рунге-Кутты»

Цель: освоить методы численного решения задачи Коши для ОДУ.

Выполнив работу, Вы будете уметь:
– разрабатывать алгоритмы для решения вычислительных задач; моделировать процессы.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Задача Коши: найти y(x) при y' = f(x, y), y(x_0) = y_0.

Метод Эйлера (1-й порядок): y_(n+1) = y_n + h * f(x_n, y_n). Прост, но неточен, погрешность O(h).

Метод Рунге-Кутты 4-го порядка (РК-4): k_1 = h*f(x_n, y_n); k_2 = h*f(x_n+h/2, y_n+k_1/2); k_3 = h*f(x_n+h/2, y_n+k_2/2); k_4 = h*f(x_n+h, y_n+k_3). y_(n+1) = y_n + (k_1 + 2*k_2 + 2*k_3 + k_4) / 6. Погрешность $O(h^4)$.

Алгоритм выполнения задания:

Шаг 1. Задать f(x, y), начальное условие y(x_0) = y_0, шаг h, отрезок [x_0, x_end].
Шаг 2. Реализовать метод Эйлера: цикл по x_n от x_0 до x_end.
Шаг 3. Реализовать РК-4: вычислить 4 коэффициента на каждом шаге.
Шаг 4. Построить графики обоих решений и аналитического (при наличии).

Пример выполнения:

y' = –2y, y(0) = 1. Аналитическое решение: y = e^(–2x).
Эйлер, h = 0.5: y_1 = 1 + 0.5*(–2*1) = 0; y(0.5) точное = 0.368. Грубая ошибка!
Эйлер, h = 0.1: y_1 = 1 + 0.1*(–2) = 0.8; y(0.1) точное ≈ 0.819. Погрешность 2.3%.
РК-4, h = 0.5: y_1 ≈ 0.3679. Погрешность < 0.03% – на порядки точнее!

Задание:
1. Изучить алгоритмы методов Эйлера и Рунге-Кутты 4-го порядка.
2. Решить задачу Коши: y' = –2y, y(0) = 1 на отрезке [0, 5] обоими методами.
3. Реализовать оба метода на Python.
4. Сравнить результаты с аналитическим решением y = e^(–2x).
5. Построить графики решений, таблицу погрешностей.
6. Оформить отчёт.

Критерии оценки:
«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 4.2. Краевые задачи

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №13

«Решение краевых задач с использованием разностных схем»

Цель: освоить метод конечных разностей для решения краевых задач.

Выполнив работу, Вы будете уметь:

– разрабатывать алгоритмы; применять численные методы для моделирования.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Краевая задача: y'' + p(x)*y' + q(x)*y = f(x) с условиями y(a) = A, y(b) = B.

Метод конечных разностей заменяет производные разностными аналогами: y'' ≈ (y_(i+1) – 2*y_i + y_(i-1)) / h².

Подстановка приводит к системе линейных уравнений, которая решается методом Гаусса или прогонки.

Алгоритм выполнения задания:

Шаг 1. Разбить отрезок [a, b] на n частей с шагом h.

Шаг 2. Заменить производные конечными разностями в каждом узле.

Шаг 3. Составить СЛАУ и решить её.

Шаг 4. Сравнить с аналитическим решением.

Пример выполнения:

y'' + y = 0, y(0) = 0, y(π) = 0. Аналитическое решение: y = sin(x).

Разностная схема: (y_(i+1) – 2*y_i + y_(i-1)) / h² + y_i = 0.

При h = π/4 (5 узлов, 3 внутренних): получаем СЛАУ 3×3.

Решение: y_1 ≈ 0.707, y_2 ≈ 1.0, y_3 ≈ 0.707 (точные: sin(π/4), sin(π/2), sin(3π/4)).

Задание:

1. Изучить метод конечных разностей для краевой задачи y'' + y = 0, y(0) = 0, y(pi) = 0.
2. Составить разностную схему и свести задачу к СЛАУ.
3. Реализовать решение на Python.
4. Сравнить с аналитическим решением y = sin(x).
5. Оформить отчёт.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.

Тема 5.1. Градиентные методы оптимизации

ЛАБОРАТОРНОЕ ЗАНЯТИЕ №14

«Реализация метода градиентного спуска. Стохастический градиентный спуск»

Цель: освоить алгоритмы градиентного спуска для оптимизации функций.

Выполнив работу, Вы будете уметь:

– разрабатывать алгоритмы; использовать численные методы для оптимизации в задачах ИИ.

Материальное обеспечение: персональный компьютер с ОС Windows/Linux, среда программирования Python (PyCharm / VS Code), библиотеки NumPy, SciPy, Matplotlib.

Краткие теоретические сведения:

Метод градиентного спуска – итерационный метод минимизации функции f(x): x_(n+1) = x_n – α * ∇ f(x_n), где α – скорость обучения (learning rate).

Градиент ∇ f указывает направление наискорейшего возрастания функции; антиградиент – направление убывания.

Стохастический градиентный спуск (SGD): градиент вычисляется не по всей выборке, а по случайному подмножеству (мини-батчу). Это ускоряет вычисления для больших данных.

Выбор α: слишком большое – расходимость; слишком малое – медленная сходимость.

Алгоритм выполнения задания:

Шаг 1. Задать функцию f(x, y) и вычислить её градиент аналитически.
Шаг 2. Задать начальную точку (x_0, y_0) и скорость обучения α.
Шаг 3. Выполнять итерации: (x, y) = (x, y) – α * (df/dx, df/dy).
Шаг 4. Остановиться при ||∇ f|| < eps или после max_iter итераций.
Шаг 5. Построить траекторию на контурном графике.

Пример выполнения:

f(x, y) = x² + y². ∇ f = (2x, 2y). Минимум в (0, 0).
(x_0, y_0) = (4, 3), α = 0.1.
Итерация 1: (x, y) = (4, 3) – 0.1*(8, 6) = (3.2, 2.4).
Итерация 2: (x, y) = (3.2, 2.4) – 0.1*(6.4, 4.8) = (2.56, 1.92).
... После 30 итераций: (x, y) ≈ (0.005, 0.004), f ≈ 0.00004.

Задание:

1. Изучить алгоритм градиентного спуска: x(n+1) = x(n) – alpha * grad(f).
2. Реализовать на Python метод градиентного спуска для функции f(x,y) = x² + y².
3. Исследовать влияние параметра скорости обучения alpha на сходимость.
4. Реализовать стохастический градиентный спуск (SGD) и сравнить с обычным GD.
5. Построить траекторию оптимизации на контурном графике.
6. Оформить отчёт.

Критерии оценки:

«5» (отлично): задание выполнено в полном объёме, программа работает корректно, отчёт оформлен в соответствии с требованиями, студент свободно владеет теоретическим материалом.

«4» (хорошо): задание выполнено, допущены незначительные ошибки в программе или оформлении.

«3» (удовлетворительно): задание выполнено частично, программа содержит существенные ошибки.

«2» (неудовлетворительно): задание не выполнено или содержит грубые ошибки.