**Лабораторная 1**

метод прямоугольников

Метод прямоугольников — это один из способов численного интегрирования, который используется для приближенного вычисления определенного интеграла функции. Суть метода заключается в следующем:

1. \*\*Разбиение области интегрирования\*\*: Интервал \([a, b]\), на котором нужно вычислить интеграл, разбивается на \(n\) маленьких подинтервалов равной ширины \(\Delta x\).

2. \*\*Выбор высоты прямоугольников\*\*: Для каждого подинтервала выбирается значение функции \(f(x)\), которое будет высотой соответствующего прямоугольника. Это значение может быть взято в начале интервала (левые прямоугольники), в конце (правые прямоугольники) или в середине (средние прямоугольники).

3. \*\*Вычисление площадей прямоугольников\*\*: Площадь каждого прямоугольника вычисляется как произведение его высоты на ширину \(\Delta x\).

4. \*\*Суммирование площадей\*\*: Сумма площадей всех прямоугольников дает приближенное значение интеграла.

Этот метод является базовым и простым, но он может быть не очень точным, если функция сильно меняется на интервале интегрирования или если интервалы слишком велики. Для повышения точности обычно уменьшают ширину интервалов \(\Delta x\).

метод трапеций

Метод трапеций — это еще один способ численного интегрирования, который используется для приближенного вычисления определенного интеграла функции. Вот как он работает:

1. \*\*Разбиение интервала\*\*: Интервал \([a, b]\), на котором нужно вычислить интеграл, разбивается на \(n\) равных подинтервалов.

2. \*\*Создание трапеций\*\*: На каждом подинтервале рассматривается функция \(f(x)\), и на основе ее значений на концах подинтервала строится трапеция.

3. \*\*Вычисление площади трапеций\*\*: Площадь каждой трапеции вычисляется как среднее значение высот (значений функции на концах подинтервала) умноженное на ширину подинтервала.

4. \*\*Суммирование площадей\*\*: Сумма площадей всех трапеций дает приближенное значение интеграла.

Метод трапеций обычно более точен, чем метод прямоугольников, потому что он лучше аппроксимирует область под кривой, особенно если функция линейна или близка к линейной на подинтервалах. Однако, как и в случае с методом прямоугольников, точность метода трапеций увеличивается с уменьшением ширины подинтервалов \(\Delta x\).

метод симпсона

Метод Симпсона — это метод численного интегрирования, который позволяет вычислять определенные интегралы с высокой точностью. Вот как он работает:

1. \*\*Разбиение интервала\*\*: Интервал \([a, b]\), на котором нужно вычислить интеграл, разбивается на четное число \(n\) равных подинтервалов.

2. \*\*Применение парабол\*\*: Вместо того чтобы аппроксимировать функцию прямыми линиями (как в методе трапеций), метод Симпсона использует параболы для приближения формы подграфика функции на каждом подинтервале.

3. \*\*Вычисление площади\*\*: Площадь под каждой параболой вычисляется и суммируется, что дает приближенное значение интеграла.

Метод Симпсона обеспечивает высокую точность, особенно если функция хорошо аппроксимируется параболами на подинтервалах. Он особенно эффективен, когда функция гладкая или когда нужна высокая точность вычислений.

метод Гаусса

Метод Гаусса для численного интегрирования — это высокоточный метод, который использует взвешенные суммы значений функции в определенных точках для приближенного вычисления интеграла. Вот его основные шаги:

1. \*\*Выбор точек и весов\*\*: Вместо равномерного разбиения интервала, как в методах прямоугольников или трапеций, метод Гаусса использует специально подобранные точки (узлы Гаусса) и соответствующие им веса. Эти точки и веса выбираются таким образом, чтобы максимизировать точность аппроксимации интеграла.

2. \*\*Вычисление суммы\*\*: Суммируются значения функции в узлах Гаусса, умноженные на их веса. Это дает приближенное значение интеграла.

Метод Гаусса считается одним из самых точных методов численного интегрирования, особенно если функция гладкая. Он эффективен даже при малом количестве узлов, что делает его популярным выбором для многих инженерных и научных приложений.

метод монте-карло

Метод Монте-Карло — это статистический метод численного интегрирования, который использует случайные выборки для приближенного вычисления интеграла. Вот как он работает:

1. \*\*Генерация случайных точек\*\*: В области, где определен интеграл, генерируется большое количество случайных точек.

2. \*\*Вычисление значений функции\*\*: Для каждой случайной точки вычисляется значение функции \(f(x)\).

3. \*\*Оценка интеграла\*\*: Среднее значение функции по всем случайным точкам умножается на объем области интегрирования, что дает приближенное значение интеграла.

Метод Монте-Карло особенно полезен, когда интегрирование проводится по многомерным областям, где традиционные методы могут быть неэффективными. Он также хорошо подходит для задач, где функция имеет сложную форму или когда точные границы интегрирования трудно определить. Точность метода увеличивается с увеличением числа случайных точек \(N\).

метод правило рунге

Правило Рунге — это метод оценки погрешности численного интегрирования, который помогает улучшить точность вычислений без значительного увеличения вычислительных затрат. Вот как он работает:

1. \*\*Вычисление с двумя разными шагами\*\*: Сначала вычисляется приближенное значение интеграла с определенным шагом \(h\). Затем вычисление повторяется с шагом, который в два раза меньше, то есть \(h/2\).

2. \*\*Оценка погрешности\*\*: Сравниваются два полученных значения интеграла. Разница между ними дает оценку погрешности.

3. \*\*Корректировка результата\*\*: Если разница между двумя значениями больше заданной погрешности, шаг уменьшается и вычисления повторяются. Этот процесс продолжается до тех пор, пока разница не станет меньше или равна заданной погрешности.

Правило Рунге позволяет контролировать точность численного интегрирования, адаптируя шаг интегрирования в зависимости от требуемой точности. Это делает метод очень полезным для практических вычислений, где необходимо достичь баланса между точностью и вычислительными ресурсами.

метод адаптивное программирование

Адаптивное численное интегрирование — это метод, который автоматически корректирует процесс вычисления интеграла для достижения высокой точности. Вот его основные принципы:

1. \*\*Начальное разбиение интервала\*\*: Интервал \([a, b]\), на котором нужно вычислить интеграл, разбивается на подинтервалы.

2. \*\*Оценка погрешности\*\*: На каждом подинтервале вычисляется приближенное значение интеграла и оценивается погрешность.

3. \*\*Адаптация шага\*\*: Если погрешность на подинтервале превышает заданный порог, интервал делится на более мелкие части. На этих частях снова вычисляется интеграл и оценивается погрешность.

4. \*\*Итеративный процесс\*\*: Процесс продолжается до тех пор, пока погрешность на всех подинтервалах не станет меньше заданного порога.

Этот метод позволяет более точно вычислять интегралы для функций с переменной сложностью, автоматически увеличивая количество вычислений там, где функция изменяется более резко, и уменьшая их там, где функция более плавная. Это делает адаптивное интегрирование очень эффективным и точным для широкого спектра функций.

**Лабораторная 2**

метод Гаусса

Метод Гаусса — это классический способ решения систем линейных алгебраических уравнений (СЛАУ). Суть метода заключается в приведении системы уравнений к такому виду, чтобы можно было последовательно найти все неизвестные. Давайте разберем основные шаги метода Гаусса:

1. \*\*Прямой ход\*\* — система уравнений преобразуется таким образом, чтобы под главной диагональю матрицы коэффициентов стояли нули. Это достигается путем элементарных преобразований строк: сложение, вычитание, умножение строки на число и замена строк местами.

2. \*\*Обратный ход\*\* — начиная с последнего уравнения системы, постепенно находим все неизвестные. В результате прямого хода в последнем уравнении остается только одна неизвестная, которую мы и находим. Затем подставляем найденное значение в предыдущее уравнение, чтобы найти следующую неизвестную, и так далее, поднимаясь вверх по системе.

В результате этих двух шагов система уравнений приводится к треугольному виду, из которого легко найти все неизвестные, последовательно подставляя их значения из нижних уравнений в верхние.

Пример простой системы из двух уравнений с двумя неизвестными:

a\_1x + b\_1y = c\_1 \\

a\_2x + b\_2y = c\_2

Применим метод Гаусса:

1. Из второго уравнения вычитаем первое, умноженное на коэффициент так, чтобы коэффициент при \( x \) во втором уравнении обратился в ноль.

2. Решаем получившееся уравнение относительно \( y \).

3. Подставляем найденное значение \( y \) в первое уравнение и находим \( x \).

Таким образом, метод Гаусса позволяет последовательно исключать неизвестные и находить их значения². Это один из самых эффективных методов для решения СЛАУ, особенно когда количество уравнений и неизвестных велико.

метод гаусса-жордана

Метод Гаусса-Жордана — это усовершенствованная версия метода Гаусса для решения систем линейных алгебраических уравнений (СЛАУ). Он заключается в приведении матрицы системы к диагональному или единичному виду, что позволяет найти решение системы напрямую. Вот основные шаги метода Гаусса-Жордана:

1. \*\*Приведение к ступенчатому виду\*\* — так же, как и в методе Гаусса, сначала систему приводят к верхнетреугольному виду, обнуляя элементы под главной диагональю.

2. \*\*Приведение к диагональному виду\*\* — затем, используя элементарные преобразования строк, обнуляют элементы над главной диагональю, чтобы в каждом уравнении осталась только одна неизвестная с коэффициентом 1.

3. \*\*Приведение к единичному виду\*\* — если это возможно, добиваются того, чтобы на главной диагонали стояли единицы, а все остальные элементы были равны нулю. Таким образом, каждое уравнение системы будет содержать только одну неизвестную с коэффициентом 1, что сразу дает решение.

Пример системы из двух уравнений с двумя неизвестными:

a\_1x + b\_1y = c\_1 \\

a\_2x + b\_2y = c\_2

Применение метода Гаусса-Жордана:

1. Сначала делаем коэффициент при \( x \) в первом уравнении равным 1, если он уже не равен 1.

2. Затем вычитаем первое уравнение из второго так, чтобы коэффициент при \( x \) во втором уравнении стал равен 0.

3. Далее делаем коэффициент при \( y \) во втором уравнении равным 1.

4. После этого вычитаем второе уравнение из первого так, чтобы коэффициент при \( y \) в первом уравнении стал равен 0.

В итоге получаем систему:

x = d\_1 \\

y = d\_2

где \( d\_1 \) и \( d\_2 \) — числа, полученные в результате преобразований. Это и будут решениями системы.

Метод Гаусса-Жордана особенно удобен, когда нужно найти обратную матрицу или решить систему с несколькими правыми частями, так как после приведения матрицы к единичному виду решения находятся непосредственно. Он требует больше вычислительных операций по сравнению с классическим методом Гаусса, но зато дает более прямой путь к решению.

метод зейделя

Метод Зейделя — это итерационный метод решения систем линейных алгебраических уравнений, который является улучшением метода простых итераций. Он используется для нахождения приближенного решения СЛАУ, особенно когда система велика и прямые методы (такие как метод Гаусса) становятся неэффективными. Вот как работает метод Зейделя:

1. \*\*Выбор начального приближения\*\* — сначала выбирается начальное приближение для каждой неизвестной переменной. Это могут быть любые числа, но чем ближе они к истинному решению, тем быстрее сойдется метод.

2. \*\*Итерационный процесс\*\* — на каждом шаге итерации значения неизвестных обновляются одно за другим. Ключевое отличие от метода простых итераций заключается в том, что при вычислении нового значения для переменной используются уже обновленные значения предыдущих переменных в текущей итерации.

3. \*\*Критерий остановки\*\* — итерационный процесс продолжается до тех пор, пока разница между последовательными приближениями не станет меньше заданного порога (эпсилон), что свидетельствует о достижении приемлемой точности.

Пример работы метода Зейделя:

Допустим, у нас есть система из двух уравнений:

a\_{11}x + a\_{12}y = b\_1 \\

a\_{21}x + a\_{22}y = b\_2

Начальное приближение: \( x^{(0)} \) и \( y^{(0)} \).

Итерационный процесс:

1. Вычисляем \( x^{(1)} \) используя \( y^{(0)} \).

2. Сразу же вычисляем \( y^{(1)} \) используя новое значение \( x^{(1)} \).

3. Повторяем процесс, используя \( x^{(1)} \) и \( y^{(1)} \) для вычисления \( x^{(2)} \) и \( y^{(2)} \), и так далее.

Метод Зейделя обычно сходится быстрее, чем метод простых итераций, благодаря использованию последних доступных значений переменных. Он хорошо подходит для систем с диагональным преобладанием, то есть когда элементы на главной диагонали матрицы коэффициентов значительно больше остальных элементов в строке. Это обеспечивает более быструю и стабильную сходимость метода.

метод наискорейшего спуска

Метод наискорейшего спуска — это итерационный метод оптимизации, который используется для нахождения минимума функции. В контексте решения систем линейных алгебраических уравнений (СЛАУ), он применяется для минимизации квадратичной формы ошибки. Суть метода заключается в следующем:

1. \*\*Выбор начального приближения\*\* — сначала выбирается начальное приближение решения системы.

2. \*\*Вычисление градиента\*\* — на каждом шаге вычисляется градиент функции ошибки в текущей точке. Градиент показывает направление наискорейшего роста функции, а его противоположное направление указывает на направление наискорейшего спуска.

3. \*\*Определение шага\*\* — затем определяется величина шага в направлении противоположном градиенту. Шаг должен быть таким, чтобы достигнуть максимального уменьшения функции ошибки.

4. \*\*Обновление приближения\*\* — после этого текущее приближение корректируется на найденный шаг в направлении наискорейшего спуска.

5. \*\*Критерий остановки\*\* — процесс повторяется до тех пор, пока не будет достигнута необходимая точность решения, то есть пока изменения между последовательными приближениями не станут достаточно малыми.

Пример работы метода наискорейшего спуска:

Допустим, у нас есть функция ошибки \( f(x) \), которую мы хотим минимизировать. Начальное приближение — \( x^{(0)} \).

Итерационный процесс:

1. Вычисляем градиент \( \nabla f(x^{(0)}) \).

2. Находим шаг \( \alpha \), при котором функция \( f(x^{(0)} - \alpha \nabla f(x^{(0)})) \) минимальна.

3. Обновляем приближение: \( x^{(1)} = x^{(0)} - \alpha \nabla f(x^{(0)}) \).

4. Повторяем процесс с новым приближением \( x^{(1)} \).

Метод наискорейшего спуска хорошо подходит для решения задач оптимизации, где функция ошибки имеет квадратичный вид, и особенно эффективен, когда размерность задачи велика. Однако, выбор оптимального шага на каждой итерации может быть нетривиальной задачей.

метод градиентного спуска

Метод градиентного спуска — это итерационный метод оптимизации, используемый для нахождения минимума функции. В контексте решения систем линейных алгебраических уравнений (СЛАУ), он применяется для минимизации функции ошибки, которая измеряет расстояние от текущего решения до истинного. Вот как работает метод градиентного спуска:

1. \*\*Выбор начального приближения\*\* — сначала выбирается начальное приближение для решения системы. Это может быть любой вектор, который мы считаем близким к решению.

2. \*\*Вычисление градиента\*\* — на каждом шаге вычисляется градиент функции ошибки в текущем приближении. Градиент указывает направление наискорейшего увеличения функции.

3. \*\*Шаг в противоположном направлении\*\* — чтобы минимизировать функцию, нужно двигаться в направлении, противоположном градиенту. Для этого из текущего приближения вычитается градиент, умноженный на небольшой коэффициент, который называется скоростью обучения или размером шага.

4. \*\*Обновление приближения\*\* — после вычитания градиента получается новое приближение, которое должно быть ближе к минимуму функции.

5. \*\*Повторение процесса\*\* — процесс повторяется до тех пор, пока изменения в приближении не станут незначительными, что свидетельствует о приближении к минимуму функции.

Пример работы метода градиентного спуска:

Допустим, у нас есть функция ошибки $$ f(x, y) = (x-3)^2 + (y+5)^2 $$, которую мы хотим минимизировать. Начальное приближение — точка \( (x^{(0)}, y^{(0)}) \).

Итерационный процесс:

1. Вычисляем градиент $$ \nabla f(x^{(0)}, y^{(0)}) = (2(x^{(0)}-3), 2(y^{(0)}+5)) $$.

2. Выбираем размер шага \( \alpha \), например, 0.1.

3. Обновляем приближение: $$ (x^{(1)}, y^{(1)}) = (x^{(0)}, y^{(0)}) - \alpha \nabla f(x^{(0)}, y^{(0)}) $$.

4. Повторяем процесс с новым приближением \( (x^{(1)}, y^{(1)}) \).

Метод градиентного спуска эффективен для решения многих задач оптимизации, но его производительность сильно зависит от выбора размера шага и начального приближения. Если шаг слишком большой, метод может "перепрыгнуть" минимум, а если слишком маленький — сходимость будет медленной. Поэтому важно тщательно подбирать параметры метода.

метод овражный

Овражный метод — это один из итерационных методов оптимизации, который используется для нахождения минимума функций многих переменных. Он особенно эффективен, когда функция имеет "овраги", то есть узкие и глубокие области в пространстве решений, где изменение функции в одном направлении значительно больше, чем в других.

Суть овражного метода заключается в следующем:

1. \*\*Выбор начальной точки\*\* — сначала выбирается начальная точка в пространстве переменных, откуда начнется поиск минимума.

2. \*\*Вычисление направления\*\* — на каждом шаге вычисляется направление движения, которое должно вести к минимуму. В овражных методах это направление определяется не только градиентом функции, но и информацией о предыдущих шагах, чтобы эффективнее "спускаться" по оврагу.

3. \*\*Выбор шага\*\* — определяется размер шага в выбранном направлении. Шаг должен быть достаточным, чтобы приблизиться к минимуму, но не настолько большим, чтобы "перепрыгнуть" его.

4. \*\*Итерационный процесс\*\* — процесс повторяется с новой точкой, полученной после предыдущего шага, до тех пор, пока не будет достигнута необходимая точность или не будет выполнен критерий остановки.

Пример работы овражного метода:

Допустим, у нас есть функция $$ f(x, y) = x^2 + 100y^2 $$, которая имеет овражную форму из-за большого коэффициента при \( y^2 \).

Итерационный процесс:

1. Начинаем с точки \( (x^{(0)}, y^{(0)}) \).

2. Вычисляем направление спуска, учитывая градиент и возможно предыдущие направления.

3. Определяем размер шага, например, с помощью метода золотого сечения.

4. Делаем шаг и получаем новую точку \( (x^{(1)}, y^{(1)}) \).

5. Повторяем процесс с новой точкой.

Овражный метод хорошо подходит для функций с овражной структурой, так как он позволяет быстрее найти минимум, "следуя" за оврагом. Однако, для его эффективного применения необходимо правильно выбирать начальную точку и учитывать особенности функции.

метод сопряженных градиентов

Метод сопряженных градиентов — это итерационный метод для решения систем линейных уравнений, особенно тех, которые имеют большое количество неизвестных и являются разреженными. Он часто используется для решения уравнений вида \(Ax = b\), где \(A\) — симметричная и положительно определенная матрица. Вот основные шаги метода:

1. \*\*Начальное приближение\*\* — выбирается начальное приближение решения \(x\_0\) и вычисляется начальный градиент \(g\_0 = b - Ax\_0\), который также является направлением спуска \(d\_0\).

2. \*\*Итерационный процесс\*\* — на каждом шаге \(k\) вычисляется значение \(x\_{k+1}\) как приближение решения, а также новое направление спуска \(d\_{k+1}\). Это делается с помощью следующих формул:

- \(x\_{k+1} = x\_k + \alpha\_k d\_k\), где \(\alpha\_k\) — размер шага, который выбирается так, чтобы минимизировать функцию \(f(x) = \frac{1}{2}x^TAx - x^Tb\).

- \(g\_{k+1} = g\_k + \alpha\_k Ad\_k\) — новый градиент.

- \(d\_{k+1} = -g\_{k+1} + \beta\_k d\_k\), где \(\beta\_k\) — коэффициент, вычисляемый по одной из формул сопряженности, например, по формуле Флетчера-Ривса: \(\beta\_k = \frac{g\_{k+1}^Tg\_{k+1}}{g\_k^Tg\_k}\).

3. \*\*Критерий остановки\*\* — процесс продолжается до тех пор, пока норма градиента не станет меньше заданного порога, что указывает на достижение приемлемой точности решения.

Суть метода сопряженных градиентов заключается в том, что каждое новое направление спуска \(d\_k\) выбирается таким образом, чтобы оно было сопряжено относительно матрицы \(A\) с предыдущими направлениями. Это означает, что все направления спуска взаимно ортогональны (перпендикулярны) в пространстве, предопределенном матрицей \(A\). Благодаря этому свойству, метод сопряженных градиентов обеспечивает более быструю сходимость к решению по сравнению с другими итерационными методами, такими как метод градиентного спуска. Он особенно эффективен для больших систем, где прямые методы, такие как метод Гаусса, становятся слишком ресурсоемкими.

метод регуляризации

Метод регуляризации — это подход к решению систем линейных алгебраических уравнений, который особенно полезен, когда система плохо обусловлена или имеет неточные данные. Суть метода заключается в добавлении дополнительной информации к системе для стабилизации решения. Вот как он работает:

1. \*\*Добавление регуляризации\*\* — к исходной системе уравнений \(Ax = b\) добавляется дополнительное условие, которое называется регуляризатором. Часто это условие имеет вид \( \lambda R(x) \), где \( \lambda \) — параметр регуляризации, а \( R(x) \) — регуляризирующий член, например, норма решения.

2. \*\*Формирование новой задачи\*\* — вместо исходной задачи решается новая задача минимизации функции \( F(x) = \|Ax - b\|^2 + \lambda \|R(x)\|^2 \), где первый член отвечает за соответствие решения исходным данным, а второй — за "гладкость" или "простоту" решения.

3. \*\*Выбор параметра регуляризации\*\* — параметр \( \lambda \) выбирается таким образом, чтобы достичь компромисса между точностью решения и его стабильностью. Если \( \lambda \) слишком мал, решение может быть неустойчивым; если слишком велик — решение может быть слишком "грубым".

4. \*\*Решение регуляризованной задачи\*\* — решается новая задача, которая обычно более устойчива к погрешностям исходных данных и позволяет получить "сглаженное" решение.

Пример применения метода регуляризации:

Допустим, у нас есть система \(Ax = b\), где матрица \(A\) плохо обусловлена. Мы можем добавить регуляризатор в виде нормы решения \( \|x\|^2 \) и решить задачу минимизации \( \|Ax - b\|^2 + \lambda \|x\|^2 \).

Метод регуляризации широко используется в численных методах и машинном обучении, где часто встречаются плохо обусловленные задачи или задачи с большим количеством параметров. Он помогает предотвратить переобучение и улучшить обобщающую способность моделей. Подбор оптимального значения параметра регуляризации является ключевым моментом при использовании этого метода.

**Лабораторная 3**

метод дихотомии

Метод дихотомии, также известный как метод бисекции или половинного деления, — это численный метод для нахождения корней нелинейных уравнений. Суть метода заключается в следующем:

1. Выбирается отрезок \([a, b]\), на концах которого функция \(f(x)\) принимает значения разных знаков, то есть \(f(a) \cdot f(b) < 0\). Это гарантирует, что между \(a\) и \(b\) есть корень уравнения \(f(x) = 0\).

2. Отрезок делится пополам, и находится середина \(c = \frac{a + b}{2}\).

3. Проверяется значение функции в точке \(c\). Если \(f(c) = 0\), то корень найден. Если нет, то определяется, в какой половине отрезка находится корень — в \([a, c]\) или \([c, b]\). Это делается путём проверки знака \(f(c)\). Если \(f(a) \cdot f(c) < 0\), корень находится в левой половине; если \(f(c) \cdot f(b) < 0\), то в правой.

4. Выбирается новый отрезок, содержащий корень, и процесс повторяется до тех пор, пока длина отрезка не станет меньше заданной точности \(\varepsilon\).

Таким образом, метод дихотомии последовательно сужает область поиска корня, делая отрезок всё меньше и меньше, пока не будет достигнута необходимая точность. Это простой, но эффективный метод для решения уравнений, когда другие методы не применимы или слишком сложны².

метод золотого сечения

Метод золотого сечения — это численный метод оптимизации, который используется для нахождения минимума или максимума функции на заданном отрезке. В контексте решения нелинейных уравнений и систем он помогает найти точку, в которой функция достигает экстремума (минимального или максимального значения). Вот как он работает:

1. Выбирается отрезок \([a, b]\), на котором предполагается наличие экстремума функции \(f(x)\).

2. Рассчитываются две внутренние точки \(x\_1\) и \(x\_2\) внутри отрезка, используя пропорции золотого сечения: \(x\_1 = b - \frac{b - a}{\varphi}\) и \(x\_2 = a + \frac{b - a}{\varphi}\), где \(\varphi\) (фи) — золотое число, приблизительно равное 1.618.

3. Сравниваются значения функции в точках \(x\_1\) и \(x\_2\). Если \(f(x\_1) > f(x\_2)\), то новый отрезок для поиска будет \([x\_1, b]\), иначе — \([a, x\_2]\).

4. Процесс повторяется для нового отрезка, и на каждом шаге отрезок уменьшается в соотношении золотого сечения, пока разница между \(a\) и \(b\) не станет меньше заданной точности \(\varepsilon\).

Таким образом, метод золотого сечения позволяет эффективно находить экстремумы функции без необходимости вычисления её производных, что делает его полезным для оптимизации сложных нелинейных функций.

метод ньютона

Метод Ньютона, также известный как метод касательных, — это мощный численный метод для нахождения корней нелинейных уравнений. Вот как он работает:

1. Выбирается начальное приближение \( x\_0 \), которое предположительно близко к истинному корню уравнения \( f(x) = 0 \).

2. На каждом шаге метода строится касательная к графику функции в точке \( x\_n \). Уравнение касательной можно выразить как \( y = f'(x\_n)(x - x\_n) + f(x\_n) \), где \( f'(x\_n) \) — производная функции в точке \( x\_n \).

3. Точка пересечения касательной с осью абсцисс (где \( y = 0 \)) берется как следующее приближение корня. Это значение рассчитывается по формуле \( x\_{n+1} = x\_n - \frac{f(x\_n)}{f'(x\_n)} \).

4. Процесс повторяется с новым приближением \( x\_{n+1} \), пока разница между двумя последовательными приближениями не станет меньше заданной точности \( \varepsilon \), или пока \( f(x\_{n+1}) \) не станет достаточно близко к нулю.

Суть метода заключается в использовании свойств касательной для быстрого нахождения корня. Метод эффективен, когда функция хорошо дифференцируема и начальное приближение выбрано достаточно близко к истинному корню. Однако метод может не сходиться, если начальное приближение выбрано неправильно или функция имеет сложное поведение.

метод секущих

Метод секущих — это численный метод для нахождения корней нелинейных уравнений, который является альтернативой методу Ньютона. Основное отличие состоит в том, что метод секущих не требует вычисления производной функции. Вот как он работает:

1. Выбираются две начальные точки \( x\_0 \) и \( x\_1 \), которые предполагаются близкими к корню уравнения \( f(x) = 0 \).

2. Строится секущая — прямая линия, соединяющая точки на графике функции с координатами \( (x\_0, f(x\_0)) \) и \( (x\_1, f(x\_1)) \).

3. Точка пересечения секущей с осью абсцисс (где \( y = 0 \)) принимается за следующее приближение корня. Это значение рассчитывается по формуле:

$$ x\_{n+1} = x\_n - f(x\_n) \frac{x\_n - x\_{n-1}}{f(x\_n) - f(x\_{n-1})} $$

4. Процесс повторяется с новыми точками \( x\_n \) и \( x\_{n+1} \), пока разница между двумя последовательными приближениями не станет меньше заданной точности \( \varepsilon \), или пока значение \( f(x\_{n+1}) \) не станет достаточно близко к нулю.

Суть метода заключается в использовании линейной аппроксимации функции для быстрого нахождения корня. Метод секущих обычно сходится быстрее, чем метод бисекции, но может быть менее надежным, если начальные приближения выбраны далеко от истинного корня.

метод ньютона-рафсона

Метод Ньютона-Рафсона — это усовершенствованная версия метода Ньютона для нахождения корней нелинейных уравнений. Он используется, когда необходимо решить систему нелинейных уравнений. Вот как он работает:

1. Выбирается начальное приближение для вектора переменных \( \mathbf{x}\_0 \), который предположительно близок к истинному решению системы.

2. На каждом шаге метода вычисляется значение функции \( \mathbf{f}(\mathbf{x}) \) и её Якобиан \( J(\mathbf{x}) \) — матрица всех первых производных системы уравнений.

3. Решается система линейных уравнений \( J(\mathbf{x}\_n)(\mathbf{x}\_{n+1} - \mathbf{x}\_n) = -\mathbf{f}(\mathbf{x}\_n) \) для нахождения приращения \( \Delta\mathbf{x}\_n \).

4. Находится новое приближение: \( \mathbf{x}\_{n+1} = \mathbf{x}\_n + \Delta\mathbf{x}\_n \).

5. Процесс повторяется до тех пор, пока норма вектора \( \mathbf{f}(\mathbf{x}\_{n+1}) \) не станет меньше заданной точности \( \varepsilon \), или пока изменения между последовательными приближениями не станут незначительными.

Суть метода Ньютона-Рафсона заключается в итеративном уточнении приближения к решению, используя информацию о градиенте и кривизне функции. Этот метод часто сходится быстрее, чем обычный метод Ньютона, особенно для систем уравнений.

**Лабораторная 4**

метод полином Лагранжа

Метод полинома Лагранжа — это способ нахождения неизвестных значений функции на основе её известных значений. Допустим, у нас есть набор точек, и мы знаем значения функции в этих точках. Метод Лагранжа позволяет нам "проложить" через эти точки гладкую кривую, так чтобы она проходила через все известные точки.

Суть метода заключается в создании \*\*интерполяционного многочлена\*\* — это многочлен, который точно соответствует заданным значениям в определённых точках (узлах интерполяции). Полином Лагранжа — это один из способов построения такого многочлена.

Вот как это работает:

1. Для каждой известной точки создаётся \*\*базисный полином\*\*. Этот полином равен 1 в этой точке и 0 во всех остальных известных точках.

2. Затем, каждый базисный полином умножается на значение функции в соответствующей точке.

3. Все полученные произведения суммируются, и результатом является интерполяционный многочлен Лагранжа.

Формула для полинома Лагранжа выглядит так:

$$ L(x) = \sum\_{i=0}^{n} y\_i \cdot l\_i(x) $$

где \( l\_i(x) \) — базисные полиномы, а \( y\_i \) — значения функции в известных точках.

Базисные полиномы \( l\_i(x) \) вычисляются по формуле:

$$ l\_i(x) = \prod\_{j=0, j \neq i}^{n} \frac{x - x\_j}{x\_i - x\_j} $$

где \( x\_i \) и \( x\_j \) — координаты известных точек.

Этот метод полезен, когда у нас есть дискретный набор данных и мы хотим найти значение функции в точке, которая не включена в этот набор. Он широко используется в численном анализе и компьютерных науках для аппроксимации функций.

метод кубический сплайн

Метод кубических сплайнов — это способ интерполяции, который позволяет создать гладкую кривую, проходящую через ряд точек данных. В отличие от полинома Лагранжа, который создаёт один многочлен для всех точек, кубический сплайн создаёт набор кубических многочленов, по одному для каждого интервала между точками.

Вот основные моменты метода кубических сплайнов:

1. \*\*Разбиение на интервалы\*\*: Данные разбиваются на интервалы, и для каждого интервала создаётся свой кубический многочлен.

2. \*\*Гладкость кривой\*\*: Кривая должна быть гладкой, то есть первая и вторая производные многочленов должны совпадать в точках соединения интервалов.

3. \*\*Краевые условия\*\*: Для определения уникального сплайна необходимо задать краевые условия, например, значения производных на концах интервалов.

Формула кубического сплайна для интервала между точками \( x\_i \) и \( x\_{i+1} \) выглядит так:

$$ S\_i(x) = a\_i + b\_i(x - x\_i) + c\_i(x - x\_i)^2 + d\_i(x - x\_i)^3 $$

где коэффициенты \( a\_i \), \( b\_i \), \( c\_i \), и \( d\_i \) вычисляются так, чтобы сплайн был непрерывным, имел непрерывные первую и вторую производные, и удовлетворял краевым условиям.

Кубические сплайны широко используются в компьютерной графике для создания плавных кривых и в инженерных расчётах, где требуется высокая точность аппроксимации данных. Они обеспечивают хорошее приближение функции, сохраняя при этом гладкость и изгибы кривой.

метод кривые Безье

Кривые Безье — это математический инструмент, который используется для создания гладких кривых в компьютерной графике и анимации. Они позволяют легко моделировать сложные формы и изгибы, используя набор контрольных точек.

Вот как работают кривые Безье:

1. \*\*Контрольные точки\*\*: Вы выбираете несколько точек, которые определяют форму кривой. Эти точки не обязательно лежат на самой кривой.

2. \*\*Построение кривой\*\*: Кривая Безье строится так, что она начинается в первой контрольной точке и заканчивается в последней, при этом "тянется" к промежуточным контрольным точкам, формируя изгибы.

3. \*\*Параметрическое уравнение\*\*: Кривая описывается параметрическим уравнением, которое использует параметр \( t \), принимающий значения от 0 до 1. При изменении \( t \) от 0 до 1, точка на кривой перемещается от начальной до конечной контрольной точки.

Формула кривой Безье для кривой с четырьмя контрольными точками (кубическая кривая Безье) выглядит так:

$$ B(t) = (1-t)^3P\_0 + 3(1-t)^2tP\_1 + 3(1-t)t^2P\_2 + t^3P\_3 $$

где \( P\_0, P\_1, P\_2, P\_3 \) — это контрольные точки, а \( t \) — параметр.

Суть метода в том, что он позволяет легко и интуитивно управлять формой кривой, перемещая контрольные точки. Это делает кривые Безье очень популярными в дизайне и анимации, так как они предоставляют большую гибкость и контроль над формой кривой. Кривые Безье также используются для интерполяции и аппроксимации данных, когда нужно создать плавный переход между различными точками.

метод численное дифференцирование

Метод численного дифференцирования — это способ вычисления производной функции, когда у нас нет её аналитической формулы, а есть только набор её значений в определённых точках. Это может быть полезно, например, когда функция задана таблично или её аналитическое выражение слишком сложно для прямого дифференцирования.

Суть метода заключается в приближении производной функции с помощью разностей её значений. Вот основные шаги:

1. \*\*Выбор точек\*\*: Выбираются точки, в которых известны значения функции.

2. \*\*Разностное приближение\*\*: Используются разности значений функции в этих точках для приближения производной.

3. \*\*Формулы разностей\*\*: Производная в точке может быть приближена с помощью формул, использующих значения функции в этой и соседних точках.

Например, простейшая формула для приближения первой производной в точке \( x \) использует значения функции в точках \( x \) и \( x + h \), где \( h \) — это маленький шаг:

$$ f'(x) \approx \frac{f(x + h) - f(x)}{h} $$

Это называется \*\*формулой конечных разностей\*\*. Есть и более сложные формулы, которые используют больше точек и дают более точные результаты.

Однако, важно помнить, что численное дифференцирование может быть неточным и зависит от выбора шага \( h \). Если \( h \) слишком велико, результат может быть неточным. Если \( h \) слишком мало, результаты могут быть искажены из-за ошибок округления. Поэтому выбор оптимального значения \( h \) является ключевым моментом в численном дифференцировании¹.

метод наименьших квадратов

Метод наименьших квадратов — это статистический метод, который используется для нахождения лучшей линии (или кривой) аппроксимации для набора точек данных. Он широко применяется для решения задач регрессии, когда нужно определить зависимость одной переменной от другой.

Суть метода заключается в минимизации суммы квадратов разностей между наблюдаемыми значениями и значениями, предсказанными моделью. Вот как это работает:

1. \*\*Выбор модели\*\*: Сначала выбирается модель аппроксимации, например, линейная функция вида \( y = ax + b \).

2. \*\*Расчёт ошибок\*\*: Для каждой точки данных вычисляется ошибка, то есть разность между наблюдаемым значением и значением, предсказанным моделью.

3. \*\*Минимизация суммы квадратов\*\*: Сумма квадратов этих ошибок минимизируется, чтобы найти наилучшие параметры модели (например, коэффициенты \( a \) и \( b \) для линейной модели).

Формула для минимизации суммы квадратов выглядит так:

$$ S = \sum\_{i=1}^{n} (y\_i - (ax\_i + b))^2 $$

где \( y\_i \) — наблюдаемые значения, \( x\_i \) — соответствующие им значения независимой переменной, а \( a \) и \( b \) — параметры модели, которые нужно найти.

Метод наименьших квадратов находит такие значения \( a \) и \( b \), при которых сумма \( S \) будет наименьшей. Это и будет лучшая линия аппроксимации для данных точек.

Этот метод особенно полезен, когда данные содержат шум или неточности, и мы хотим найти общую тенденцию или закономерность в данных. Метод наименьших квадратов помогает "сгладить" эти неточности и выявить основную зависимость между переменными.

**Лабораторная 5**

метод полином Лагранжа

Метод рядов Фурье позволяет представить любую периодическую функцию в виде суммы простых тригонометрических функций, таких как синусы и косинусы. Это очень полезно, потому что тригонометрические функции имеют хорошо изученные свойства и легко интегрируются и дифференцируются, что делает их удобными для анализа и приближения сложных функций.

Суть метода заключается в следующем:

1. **Периодичность**: Рассматриваемая функция должна быть периодической, то есть повторяться через определённые интервалы.
2. **Разложение**: Функция разлагается на бесконечную сумму синусов и косинусов различных частот, которые называются гармониками.
3. **Коэффициенты Фурье**: Для каждой гармоники вычисляются коэффициенты, которые показывают, насколько сильно данная гармоника присутствует в исходной функции.
4. **Сумма ряда**: Сумма всех этих тригонометрических функций с их коэффициентами даёт приближение исходной функции.

**Лабораторная 6**

метод решения задач линейного программирования

Методы решения задач линейного программирования в контексте многомерной оптимизации непрерывных функций основаны на поиске оптимального решения, которое максимизирует или минимизирует целевую функцию при заданных ограничениях. Вот основные принципы:

1. **Целевая функция**: Это функция, которую нужно оптимизировать, например,

f(x)=c1​x1​+c2​x2​+…+cn​xn​

, где ( c\_i ) - коэффициенты, а ( x\_i ) - переменные.

1. **Ограничения**: Это уравнения или неравенства, которые ограничивают значения переменных, например,

a1​x1​+a2​x2​≤b

.

1. **Допустимое множество**: Множество всех возможных решений, которые удовлетворяют ограничениям.
2. **Вершины допустимого множества**: Точки, в которых пересекаются ограничения. В линейном программировании оптимальное решение часто находится в одной из этих вершин.
3. **Симплекс-метод**: Популярный алгоритм, который перемещается от одной вершины к другой в поисках оптимального решения.
4. **Метод ветвей и границ**: Используется для решения задач целочисленного программирования, где переменные должны быть целыми числами.

Суть этих методов заключается в систематическом поиске решения в пределах допустимого множества, используя математические алгоритмы для нахождения точки, в которой целевая функция достигает своего оптимального значения. Это требует тщательного анализа ограничений и стратегического перемещения по допустимому множеству для нахождения лучшего решения.

метод минимизации нелинейных функционалов без ограничений

Методы минимизации нелинейных функционалов без ограничений в многомерной оптимизации непрерывных функций направлены на нахождение точки, в которой функция достигает своего минимального значения. Вот основные идеи:

1. **Градиентный спуск**: Это итеративный метод, который использует градиент функции (вектор, указывающий направление наискорейшего роста функции) для поиска минимума. На каждом шаге текущее приближение улучшается в направлении, противоположном градиенту.
2. **Шаг алгоритма**: На каждом шаге выбирается размер шага, который определяет, насколько далеко следует переместиться в направлении, противоположном градиенту.
3. **Критерии остановки**: Алгоритм продолжает итерации до тех пор, пока не будет достигнут критерий остановки, например, когда изменение функции становится незначительным или градиент приближается к нулю.
4. **Метод Ньютона**: Этот метод использует вторые производные (матрицу Гессе) для более быстрого нахождения минимума. Он может быть более эффективным, чем градиентный спуск, но требует вычисления вторых производных.
5. **Квазиньютоновские методы**: Эти методы приближают матрицу Гессе, чтобы избежать её прямого вычисления, что делает их более практичными для больших задач.
6. **Метод сопряженных градиентов**: Подходит для больших задач, где прямое вычисление градиента и матрицы Гессе затруднительно. Он использует свойства сопряженности для эффективного поиска минимума.

Суть этих методов заключается в последовательном улучшении текущего решения, используя информацию о градиенте и, возможно, вторых производных функции, чтобы найти точку, где функция достигает минимального значения. Это требует математического анализа и вычислительных алгоритмов для эффективного поиска минимума функции.

**Лабораторная 7**

метод многомерной дискретной оптимизации.

Методы многомерной дискретной оптимизации - это математические подходы, используемые для нахождения наилучшего возможного решения или набора решений для задач с несколькими переменными, где каждая переменная может принимать только дискретные значения. Суть этих методов заключается в поиске оптимального значения функции, которая оценивает качество каждого возможного решения.

Вот несколько ключевых понятий, которые помогут понять суть этих методов:

* **Функция цели**: Это функция, которую мы хотим минимизировать или максимизировать. Например, если мы хотим минимизировать затраты или максимизировать прибыль, функция цели будет отражать эти затраты или прибыль.
* **Дискретные переменные**: В отличие от непрерывных переменных, которые могут принимать любое значение в заданном диапазоне, дискретные переменные ограничены определенным набором значений, например, целыми числами.
* **Пространство поиска**: Это набор всех возможных решений. В многомерной дискретной оптимизации пространство поиска состоит из всех возможных комбинаций значений дискретных переменных.
* **Методы поиска**: Это алгоритмы, которые используются для исследования пространства поиска и нахождения оптимального решения. Они могут включать методы перебора, градиентного спуска, генетические алгоритмы и другие.

[Примером метода многомерной дискретной оптимизации может служить **метод покоординатного спуска**, который исследует пространство поиска, изменяя одну переменную за раз, пока не будет найдено оптимальное решение1](https://pnu.edu.ru/media/filer_public/2013/02/26/popova_methods-mo.pdf).

[Другой пример - **метод градиентного спуска**, который использует информацию о производной функции цели для определения направления, в котором функция уменьшается быстрее всего, и движется в этом направлении для поиска минимума1](https://pnu.edu.ru/media/filer_public/2013/02/26/popova_methods-mo.pdf).

Эти методы могут быть очень сложными и включать продвинутые математические концепции, но в основе их работы лежит идея систематического поиска через пространство возможных решений для нахождения того, которое лучше всего соответствует нашим целям.

метод ветвей и границ

Метод ветвей и границ - это алгоритмический метод для решения задач многомерной дискретной оптимизации, особенно полезный, когда задача имеет большое количество переменных и решений. Он помогает найти оптимальное решение, систематически разделяя пространство всех возможных решений на меньшие подмножества (ветвление) и оценивая их с помощью ограничений (границы), чтобы отсечь те, которые не могут содержать оптимальное решение.

Вот как это работает:

1. **Ветвление**: Пространство решений делится на подпространства, или “ветви”. Это делается путем выбора переменной и создания двух новых задач, каждая из которых имеет разные ограничения для этой переменной.
2. **Оценка границ**: Для каждой ветви вычисляется нижняя или верхняя граница функции цели. Это значение показывает, какое лучшее решение можно получить в данной ветви.
3. **Отсечение**: Если граница хуже, чем уже найденное решение, ветвь отсекается, так как она не может содержать лучшее решение.
4. **Выбор**: Алгоритм выбирает одну из ветвей с наилучшей границей для дальнейшего исследования и повторяет процесс.
5. **Остановка**: Процесс продолжается до тех пор, пока не будут исследованы все ветви или пока не будет найдено удовлетворительное решение.

Примером использования метода ветвей и границ может быть задача о рюкзаке, где нужно выбрать предметы с определенным весом и стоимостью так, чтобы максимизировать общую стоимость, не превышая весового ограничения рюкзака.

Этот метод эффективен, потому что он позволяет сократить количество решений, которые нужно рассмотреть, используя математические оценки, и таким образом сэкономить время и вычислительные ресурсы. Он широко используется в различных областях, таких как логистика, финансовое планирование и производственное планирование.

метод муравьиного алгоритма

Метод муравьиного алгоритма в многомерной дискретной оптимизации вдохновлен поведением муравьев при поиске пищи и основан на использовании простых правил для решения сложных задач. Вот как он работает:

1. **Поиск пути**: Муравьи случайным образом исследуют пространство в поисках пищи (в нашем случае, оптимального решения).
2. **Выделение феромона**: По мере движения муравьи оставляют за собой след феромона, который может быть обнаружен другими муравьями.
3. **Следование за феромоном**: Другие муравьи чувствуют феромон и с большей вероятностью следуют по пути с более сильным феромонным следом, что увеличивает вероятность нахождения пищи.
4. **Испарение феромона**: Со временем феромон испаряется, что уменьшает его привлекательность. Это предотвращает застревание муравьев на неоптимальных путях.
5. **Усиление феромона**: Когда муравей находит путь к пище, он возвращается к колонии, усиливая феромонный след. Если путь был коротким (эффективным), след феромона становится сильнее.
6. **Поиск оптимального пути**: Со временем, муравьи начинают выбирать более короткие пути, так как они имеют более сильный феромонный след. Это приводит к тому, что большинство муравьев следует за оптимальным путем.

В контексте оптимизации, “пища” - это оптимальное решение задачи, а “пути” - это различные возможные решения. Метод муравьиного алгоритма эффективен для нахождения оптимальных решений в задачах, таких как задача коммивояжера, где необходимо найти самый короткий путь через ряд городов, посещая каждый город только один раз.

Этот метод хорош тем, что он адаптивен и может находить решения в сложных динамических условиях, так как муравьи постоянно обновляют информацию о путях на основе свежих данных. Таким образом, метод муравьиного алгоритма использует коллективное “знание” муравьев для решения задач, которые были бы слишком сложны для одного индивидуума.

метод жадного алгоритма

Жадный алгоритм — это метод решения задач оптимизации, который делает выбор на каждом шаге, исходя из локального оптимума, надеясь, что эти локальные оптимумы приведут к глобальному оптимальному решению. Вот основные принципы работы жадного алгоритма:

1. **Локальный выбор**: На каждом шаге алгоритм выбирает локально оптимальное решение, не рассматривая более широкие последствия. Это может быть, например, выбор самого дешевого пути или самого тяжелого предмета.
2. **Однонаправленность**: Жадный алгоритм не возвращается назад, чтобы изменить свой выбор. Он продолжает двигаться вперед, делая новые выборы на основе текущей ситуации.
3. **Оптимизация**: Цель алгоритма — максимизировать или минимизировать функцию цели, например, максимизировать общую стоимость или минимизировать общее время.
4. **Простота и скорость**: Жадные алгоритмы часто проще и быстрее, чем другие методы оптимизации, потому что они не требуют обратного просмотра или рассмотрения большого количества комбинаций.

Примером использования жадного алгоритма может быть задача о выборе монет для сдачи: чтобы дать сдачу, используя как можно меньше монет, алгоритм на каждом шаге выбирает монету с наибольшим номиналом, который не превышает требуемую сумму.

Однако жадные алгоритмы не всегда приводят к оптимальному решению для всех задач. Иногда они могут давать только приближенное решение, которое достаточно хорошо справляется с задачей, но не является абсолютно оптимальным. Это происходит из-за того, что локально оптимальные решения не всегда совпадают с глобальным оптимумом. Тем не менее, жадные алгоритмы полезны благодаря своей эффективности и простоте во многих практических ситуациях.

метод перебора

Метод перебора, также известный как полный перебор или “brute force”, — это простой, но мощный метод решения задач многомерной дискретной оптимизации. Суть этого метода заключается в проверке всех возможных вариантов решения задачи для нахождения наилучшего результата. Вот основные шаги:

1. **Генерация вариантов**: Алгоритм создает список всех возможных комбинаций переменных в задаче. Если у нас есть две переменные, каждая из которых может принимать значения от 1 до 3, то алгоритм переберет все пары: (1,1), (1,2), (1,3), (2,1), и так далее.
2. **Оценка каждого варианта**: Для каждой комбинации переменных вычисляется значение функции цели. Например, если мы хотим минимизировать расходы, алгоритм вычислит общие расходы для каждой комбинации.
3. **Сравнение результатов**: Алгоритм сравнивает результаты всех комбинаций и выбирает ту, которая дает наилучшее значение функции цели.
4. **Выбор оптимального решения**: После того как все варианты были оценены и сравнены, алгоритм выбирает комбинацию с наилучшим результатом как окончательное решение задачи.

Преимущество метода перебора в том, что он гарантированно найдет оптимальное решение, если оно существует, поскольку проверяет каждый возможный вариант. Однако основной недостаток в том, что он может быть очень медленным и неэффективным, особенно если пространство решений велико, так как время выполнения алгоритма растет экспоненциально с увеличением количества переменных.

Метод перебора часто используется как последний вариант, когда другие более эффективные методы оптимизации не подходят или когда точность решения критически важна. Это может быть полезно для решения задач с небольшим количеством переменных или когда необходимо проверить все возможные решения, например, в криптографии для взлома шифров.