ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

tpu.ru

Углубленный курс информатики

Лекция 12

Численные методы решения систем линейных уравнений. Минимизация функций нескольких переменных

23 мая 2022 г.

Вячеслав Алексеевич Чузлов,
к.т.н., доцент ОХИ ИШПР

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Содержание

Системы линейных уравнений

Рассмотрим систему из n линейных алгебраических уравнений с n неизвестными:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases} \tag{1}$$

Совокупность коэффициентов данной системы можно записать в виде матрицы:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \tag{2}$$

Системы линейных уравнений

Таким образом, систему (1) можно представить в матричном виде:

$$AX = B \tag{3}$$

где X и B – вектор-столбец неизвестных переменных и вектор-столбец правых частей соответственно:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \qquad B = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix} \tag{4}$$

Детерминант или определитель матрицы A ($\det A$) – это число D, равное

$$D = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \sum (-1)^k \, a_{1\alpha} a_{2\beta} \dots a_{n\overline{\omega}} \tag{5}$$

где индексы α, β, …, $\overline{\omega}$ пробегают все возможные $n!$ перестановки индексов $1, 2, \dots, n$; k – число инверсий в данной перестановке.

Системы линейных уравнений

- Необходимое и достаточное условие существования единственного решения системы линейных уравнений – определитель матрицы не должен быть равен нулю: $D \neq 0$.
- Если определитель системы равен нулю, матрица называется *вырожденной*, при этом сама система линейных уравнений (1) либо не имеет решений, либо имеет бесконечное множество решений.

Методы решения систем линейных уравнений

Прямые методы

- В *прямых методах* используются соотношения (формулы) для вычисления неизвестных. Такие методы дают решение после выполнение заранее известного числа операций.
- Данные методы обладают достаточной простотой и пригодны для широкого класса систем линейных уравнений.
- Среди наиболее известных прямых методов можно выделить метод Гаусса, метод прогонки и схему Жордана.

Итерационные методы

- Подразумевают задание *начального приближения*. Затем производится цикл вычислений на основе некоторого алгоритма, который называется *итерацией*.
- В результате каждой итерации определяется новое приближение. Повторение итераций происходит до тех пор, пока не будет получена требуемая точность.
- Наиболее распространенным является метод Гаусса-Зейделя.

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Метод Гаусса

Метод Гаусса

Метод Гаусса является прямым методом решения систем линейных уравнений, заключающийся в том, что последовательным исключением неизвестных систему (1) приводят к эквивалентной системе с треугольной матрицей:

$$\begin{cases} x_1 + c_{12}x_2 + \ldots + c_{1n}x_n = d_1 \\ x_2 + \ldots + c_{2n}x_n = d_2 \\ \ldots \\ x_n = d_n \end{cases} \tag{6}$$

решение которой можно найти по рекуррентным формулам:

$$\begin{gathered} x_n = d_n \\ x_i = d_i - \sum_{k=i+1}^{n} c_{ik}x_k; \qquad i = n-1, n-2, \ldots, 1 \end{gathered} \tag{7}$$

Метод Гаусса

Рассмотрим схему метода Гаусса с выбором главного элемента. Сделаем предположение о том, что $a_{11} \neq 0$, и разделим обе части первого уравнения системы на a_{11}, после чего получим уравнение:

$$x_1 + b_{12}x_2 + \ldots + b_{1n}x_n = b_1 \tag{8}$$

При помощи уравнения (8) исключим во всех уравнениях системы, кроме первого, слагаемые, содержащие x_1. С этой целью, умножим последовательно обе части уравнения (8) на a_{21}, a_{31}, …, a_{n1} и вычтем из соответствующих уравнений. В итоге будет получена система, имеющая порядок на единицу меньше порядка исходной системы. Аналогичным образом преобразуем полученную систему. После n-кратного повторения этих преобразований получим систему с треугольной матрицей. Условие применимости данной схемы состоит в неравенстве нулю элементов главной диагонали матрицы коэффициентов:

$$a_{ii} \neq 0; \qquad i = 1, 2, \ldots, n \tag{9}$$

иначе будет необходимо сделать перестановку уравнений системы с тем, чтобы добиться выполнения данного условия.

Пример

Решим методом Гаусса следующую систему уравнений:

$$\begin{cases} 4x_1 - x_2 + x_3 = 4 \\ x_1 + 6x_2 + 2x_3 = 9 \\ -x_1 - 2x_2 + 5x_3 = 2 \end{cases}$$

В данном случае элементы главной диагонали матрицы коэффициентов системы линейных уравнений не равны нулю. Следовательно, можно исключить x_1 из 2-го и 3-го уравнений системы, для этого разделим первое уравнение на 4:

$$\begin{cases} x_1 - \frac{1}{4}x_2 + \frac{1}{4}x_3 = 1 \\ \left(6 + \frac{1}{4}\right)x_2 + \left(2 - \frac{1}{4}\right)x_3 = 9 - 1 \\ \left(-2 - \frac{1}{4}\right)x_2 + \left(5 + \frac{1}{4}\right)x_3 = 2 + 1 \end{cases} \Rightarrow \begin{cases} x_1 - \frac{1}{4}x_2 + \frac{1}{4}x_3 = 1 \\ \frac{25}{4}x_2 + \frac{7}{4}x_3 = 8 \\ -\frac{9}{4}x_2 + \frac{21}{4}x_3 = 3 \end{cases}$$

Пример

По такому же принципу исключим x_2 из уравнения 3:

$$\begin{cases} x_1 - \frac{1}{4}x_2 + \frac{1}{4}x_3 = 1 \\ x_2 + \frac{7}{25}x_3 = \frac{8\cdot 4}{4}\cdot\frac{4}{25} \\ \left(\frac{21}{4} + \frac{7}{25}\cdot\frac{9}{4}\right)x_3 = 3 + \frac{32}{25}\cdot\frac{9}{4} \end{cases} \Rightarrow \begin{cases} x_1 - \frac{1}{4}x_2 + \frac{1}{4}x_3 = 1 \\ x_2 + \frac{7}{25}x_3 = \frac{32}{25} \\ \frac{3}{4}\cdot\left(7 + \frac{7\cdot 3}{25}\right)x_3 = \frac{3}{4}\cdot\left(4 + \frac{32\cdot 3}{25}\right) \end{cases}$$

Выпишем уравнение 3, сократим его на $\frac{3}{4}$ и выразим x_3:

$$\left(\frac{7\cdot 25}{25} + \frac{21}{25}\right)x_3 = \left(\frac{4\cdot 25}{25} + \frac{96}{25}\right) \Rightarrow \left(\frac{175+21}{25}\right)x_3 = \left(\frac{100+96}{25}\right) \Rightarrow$$

$$\Rightarrow \frac{196}{25}x_3 = \frac{196}{25} \Rightarrow x_3 = 1$$

Пример

Подставим значение x_3 во второе уравнение системы и найдем значение x_2:

$$x_2 + \frac{7}{25} \cdot 1 = \frac{32}{25} \Rightarrow x_2 = \frac{32 - 7}{25} = \frac{25}{25} = 1$$

Теперь осталось подставить значения x_2 и x_3 в первое уравнение системы и вычислить значение x_1:

$$x_1 - \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 1 = 1 \Rightarrow x_1 = 1$$

Таким образом, решение системы уравнений $x_1 = x_2 = x_3 = 1$ или :

$$X = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Программная реализация


```python
def gauss(a, b):
    """
    :param a: матрица коэффициентов системы линейных уравнений
    :param b: вектор-столбец правых частей линейных уравнений
    :return: вектор решений
    """
    n = len(a)
    x = [0 for _ in range(n)]

    arr = [a[i] + [b[i]] for i in range(n)]

    for k in range(n):
        for j in range(k + 1, n + 1):
            arr[k][j] /= arr[k][k]

            for i in range(k + 1, n):
                arr[i][j] -= arr[i][k] * arr[k][j]

    for i in range(n - 1, -1, -1):
        s = 0

        for k in range(i + 1, n):
            s += arr[i][k] * x[k]

        x[i] = arr[i][n] - s

    return x

```

Программная реализация


```
29
30  sol = gauss([[4, -1, 1],
31               [1, 6, 2],
32               [-1, -2, 5]], [4, 9, 2])
33
34  print(sol)
35
```

```
[1.0, 1.0, 1.0]
```

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Метод Гаусса-Жордана

Метод Гаусса-Жордана

Пусть дана система линейных алгебраических уравнений 3×3 (3 уравнения с 3-мя неизвестными):

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{10}$$

или в матричном виде:

$$Ax = b \tag{11}$$

где

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}; \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}; \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{12}$$

где A – матрица коэффициентов системы, b – правая часть ограничений, x – вектор искомых переменных.

Метод Гаусса-Жордана

Составим расширенную матрицу системы:

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right] \tag{13}$$

После в результате применения метода Гаусса-Жордана матрица (13) будет приведена к следующему виду:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & c_1 \\ 0 & 1 & 0 & c_2 \\ 0 & 0 & 1 & c_3 \end{array}\right] \tag{14}$$

следовательно, искомый вектор переменных будет равен:

$$x = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \tag{15}$$

Метод Гаусса-Жордана

Алгоритм метода Гаусса-Жордана состоит в следующих шагах:

1. В качестве опорной выбирают первую строку матрицы.
2. Если элемент опорной строки, индекс которой равен номеру опорной строки, равен нулю, то меняем всю опорную строку на строку снизу, в столбце которой нет нуля.
3. Каждый элемент опорной строки делится на элемент, индекс которого равен индексу этой строки.
4. Из оставшихся строк вычитают опорную строку, умноженную на элемент, индекс которого равен индексу опорной строки.
5. В качестве опорной выбирается следующая строка.
6. Повторяются действия 2-5 пока номер опорной строки не превысит число строк.

Пример

Пусть требуется решить следующую систему линейных алгебраических уравнений:

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ 4x_1 + 2x_2 + x_3 = 1 \\ 9x_1 + 3x_2 + x_3 = 3 \end{cases}$$

методом Гаусса-Жордана
Запишем расширенную матрицу системы уравнений:

$$a = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 4 & 2 & 1 & 1 \\ 9 & 3 & 1 & 3 \end{array}\right]$$

Пример

1. Выбираем первую строку матрицы в качестве опорной. Разделим элементы этой строки на элемент с индексом, равным индексу этой строки, т.е. $a[0][0] = 1$. Получим:

$$a = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 4 & 2 & 1 & 1 \\ 9 & 3 & 1 & 3 \end{array}\right]$$

Далее выбираем для второй строки разрешающий элемент $a[1][0] = 4$, а для третьей строки элемент $a[2][0] = 9$. Вычтем из второй и третьей строки первую строку, домноженную на соответствующий разрешающий элемент, т. е. выполним следующие преобразования:

- из строки 2 вычтем $4 \times$ строку 1
- из строки 3 вычтем $9 \times$ строку 1.

В результате получим:

$$a = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & -2 & -3 & 1 \\ 0 & -6 & -8 & 3 \end{array}\right]$$

Пример

2. Теперь выбираем вторую строку матрицы в качестве опорной. Разделим элементы этой строки на элемент с индексом, равным индексу этой строки, т.е. $a[1][1] = -2$. В итоге получим:

$$a = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & 1 & -3/2 & -1/2 \\ 0 & -6 & -8 & 3 \end{array}\right]$$

Выбираем для первой строки разрешающий элемент $a[0][1] = 1$, а для третьей строки элемент $a[2][1] = -6$. Вычтем из первой и третьей строк вторую строку, домноженную на соответствующий разрешающий элемент, т.е. выполним следующие преобразования:

- из строки 1 вычтем $1 \times$ строку 2
- из строки 3 вычтем $-6 \times$ строку 2.

В результате получим:

$$a = \left[\begin{array}{ccc|c} 1 & 0 & -1/2 & 1/2 \\ 0 & 1 & -3/2 & -1/2 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

Пример

- Выбираем третью строку матрицы в качестве опорной. Разделим элементы этой строки на элемент с индексом, равным индексу этой строки $a[2][2] = 1$. Получим:

$$a = \left[\begin{array}{ccc|c} 1 & 0 & -1/2 & 1/2 \\ 0 & 1 & -3/2 & -1/2 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

Теперь для первой строки выберем разрешающий элемент $a[0][2] = -1/2$, а для второй строки $a[1][2] = -3/2$. Вычтем из первой и второй строки третью строку, домноженную на соответствующий разрешающий элемент, т.е. выполним следующие действия:

- из строки 1 вычтем $-1/2 \times$ строку 3
- из строки 2 вычтем $-3/2 \times$ строку 3.

В результате получим:

$$a = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1/2 \\ 0 & 1 & 0 & -1/2 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

Пример

Таким образом, искомый вектор решений:

$$x = \begin{bmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ 0 \end{bmatrix}$$

иначе говоря, $x_1 = \frac{1}{2}$; $x_2 = -\frac{1}{2}$; $x_3 = 0$.

Метод Гаусса-Жордана дает точное решение и обладает хорошей надежностью.

Программная реализация


```python
def gauss_jordan(a, b):
    """
    :param a: матрица коэффициентов системы линейных уравнений
    :param b: вектор-столбец правых частей лененйных уравнений
    :return: вектор решений
    """
    n = len(a)
    arr = [a[i] + [b[i]] for i in range(n)]

    for k in range(n):

        #если разрешающий элемент равен 0, меняем строки местами
        if arr[k][k] == 0:
            arr[k], arr[k-1] = arr[k-1], arr[k]

        d = arr[k][k]

        for i in range(n + 1):
            arr[k][i] /= d

        for i in (i for i in range(n) if i != k): #для i ≠ k
            d = arr[i][k]

            for j in range(n + 1):
                arr[i][j] -= arr[k][j] * d

    return [arr[i][-1] for i in range(n)]
```

Программная реализация


```
28
29
30  sol = gauss_jordan([[1, 1, 1],
31                      [4, 2, 1],
32                      [9, 3, 1]], [0, 1, 3])
33  print(sol)
34
```

```
[0.5, -0.5, 0.0]
```

Минимизация функций нескольких переменных

- Конечная цель моделирования химико-технологического процесса заключается в его более эффективной реализации или *оптимизации*.
- *Оптимизация* – это комплекс целенаправленных действий, заключающихся в получении наилучших результатов (значений параметров объектов) при соответствующих результатах.
- Оптимизация заключается в нахождении точки экстремума рассматриваемой функции или оптимальных технологических условий процесса. Для того, чтобы оценить оптимум, нужно выбрать *критерий оптимизации. Критерий оптимизации (оптимальности)* – это количественная оценка оптимизируемого показателя качества объекта, т.е. главный признак эффективности решения оптимизационной задачи.
- В зависимости от решаемой задачи, в качестве критерия оптимальности может быть выбран *технологический критерий* (к примеру, максимальная селективность или выход продукта) или *экономический критерий* (скажем, минимальные затраты на производство продукта при заданной производительности).
- Основываясь на выбранном критерии оптимальности строится *целевая функция*, выражающая зависимость критерия оптимальности от входных параметров системы:

$$F = f(x_1, x_2, \ldots, x_n) \tag{16}$$

Таким образом, точка оптимума – это экстремум функции (16), а задача оптимизации сводится к нахождению этого экстремума.

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Метод Нелдера-Мида

Метод Нелдера-Мида

Метод Нелдера-Мида представляет собой безусловный метод оптимизации функции нескольких переменных, не требующий вычисления градиентов.

- Сущность данного метода заключается в последовательном перемещении и деформации симплекса в окрестности точки экстремума посредством выполнения трех операций:
  1. Отражение;
  2. Растяжение;
  3. Сжатие.
- Симплекс является геометрической фигурой, представляющей n-мерное обобщение треугольника. То есть в случае одномерного пространства – это отрезок; в случае двумерного пространства – треугольник; для трехмерного пространства – тетраэдр.
- Данный метод осуществляет поиск локальных экстремумов и может остановиться на одном из них, поэтому в случае необходимости найти глобальный экстремум функции нужно пробовать задавать другой начальный симплекс.

Метод Нелдера-Мида имеет следующие параметры:

- коэффициент отражения $\alpha > 0$, как правило, выбирается значение 1;
- коэффициент сжатия $\beta > 0$, как правило, выбирается значение 0.5;
- коэффициент растяжения $\gamma > 0$, как правило, выбирается значение 2.

Метод Нелдера-Мида

Рассмотрим минимизацию функции от n переменных без ограничений. $p_0, p_1, \ldots, p_n$ – это $(n+1)$ точки в n-мерном пространстве, определяющие текущий симплекс. Обозначим y_i значения функции при значениях p_i и определим:

- индекс *high*, при котором $y_{high} = max\,(y_i)$;
- индекс *low*, при котором $y_{low} = min\,(y_i)$.

Далее определим p_c, как центр всех точек при условии $i \neq high$. На каждом этапе процесса p_{high} заменяется новой точкой; используются три операции – *отражение*, *сжатие* и *расширение*. Данные операции определяются следующим образом: отражение p_{high} обозначается p_r и его координаты определяются соотношением:

$$p_r = (1+\alpha)\cdot p_c - \alpha \cdot p_{high} \tag{17}$$

где α – положительная константа, коэффициент отражения. Если y_r лежит между y_{high} и y_{low}, то p_{high} заменяется на p_r и мы начинаем с начала с новым симплексом.
Если $y_r < y_{low}$, т.е. если отражение произвело новый минимум, тогда мы расширяем p_r до p_e, применяя соотношение:

$$p_e = \gamma \cdot p_r + (1-\gamma)\cdot p_c \tag{18}$$

Коэффициент расширения γ имеет значение больше единицы. Если $y_e < y_{low}$, то p_{high} заменяется на p_e и процесс начинается с начала, но если $y_e > y_{low}$, то расширение было неудачным, заменяем p_{high} на p_r перед возвратом в начало цикла.

Метод Нелдера-Мида

Если после отражения p на p_r $y_r > y_{low}$ для всех $i \neq high$, т.е. эта замена p на p_r оставляет y_r максимальным, тогда определяем новое значение p_{high} как предыдущее значение p_{high} или p_r в зависимости от того, что имеет меньшее значение y и определяем:

$$p_s = \beta \cdot p_{high} + (1 - \beta) \cdot p_c \tag{19}$$

Коэффициент сжатия β лежит в диапазоне от 0 до 1. Далее мы принимаем p_s для p_{high} и переходим в начало цикла, если не выполняется условие $y_s > min\left(y_{high}, y_r\right)$, т.е. сжатая точка хуже, чем лучшая из p_{high} и p_r. В случае такого неудачного сжатия заменяем все p_i на $(p_i + p_{low})/2$ и повторяем процесс с начала.

Последний аспект касается критерия, используемого для выхода из цикла. В качестве такого критерия выбрана «стандартная ошибка» величины y_i в следующем виде:

$$error = \sqrt{\frac{\sum_{i=1}^{n} \left(y_i - y_{mean}\right)^2}{n}} \tag{20}$$

где y_{mean} – среднее значение.

Таким образом, величина стандартной ошибки на каждой итерации цикла сравнивается с заранее выбранной величиной, и как только значение ошибки становится меньше – цикл завершается. Успех такого критерия зависит от того, не станет ли симплекс слишком маленьким по отношению к кривизне поверхности до того, как будет достигнут окончательный минимум.

Метод Нелдера-Мида

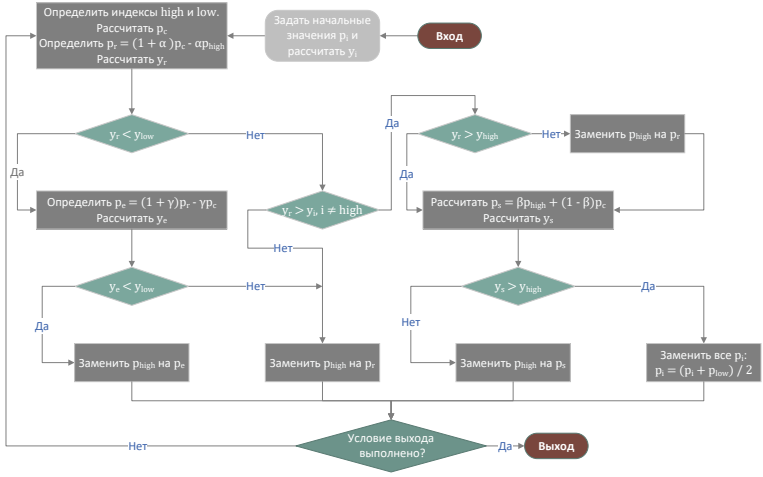

Рисунок 1 – Блок-схема алгоритма Нелдера-Мида

Пример

Пусть требуется найти экстремум следующей функции:

$$f(x, y) = x^2 + xy + y^2 - 6x - 9y$$

Из аналитического решения известно, что экстремум функции достигается в точке $f(1, 4) = 21$.

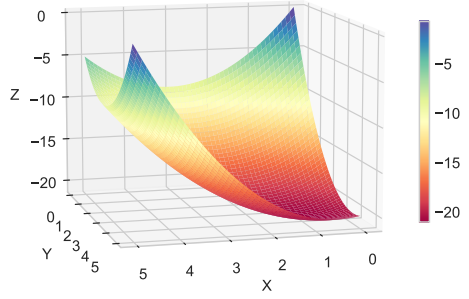

Рисунок 2 – 3-d график функции $x^2 + xy + y^2 - 6x - 9y$

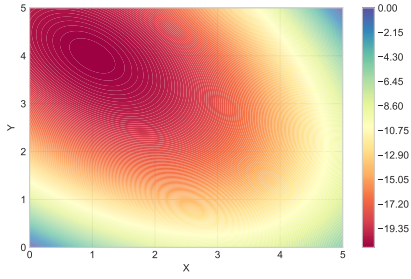

Рисунок 3 – 2-d представление функции $x^2 + xy + y^2 - 6x - 9y$

Примем в качестве начального приближения следующие точки: $p_1(0, 0)$; $p_2(1, 0)$; $p_3(0, 1)$ Примем в качестве критерия выхода из цикла значение стандартной ошибки равное 1×10^{-8}.

Программная реализация


```
def func(x):  #оптимизируемая функция
    return (x[0] ** 2 + x[0] * x[1]
            + x[1] ** 2 - 6 * x[0] - 9 * x[1])


def nelder_mead(func, initial, alpha=1, beta=0.5, gamma=2, eps=1e-8):
    """
    :param func: <function> оптимизируемая функция
    :param initial: <list-object> начальный симплекс из n+1 точек
    :param alpha: <float> коэффициент отражения; по умолчанию alpha = 1
    :param beta: <float> коэффициент сжатия; по умолчанию beta = 0.5
    :param gamma: <float> коэффициент растяжения; по умолчанию gamma = 2
    :return: <list-object> координаты точки, при которой достигается
                           минимум функции.
    """
    n = len(initial)
    p = initial[:]
    condition = False
    y = [func(item) for item in p]
    iteration = 1

    while not condition:
        high, low = y.index(max(y)), y.index(min(y))

        pc = []
        s = 0
```

Программная реализация


```
27          for j in range(n - 1):
28              for i in range(n):
29                  if i != high:
30                      s += p[i][j] / (n - 1)
31              pc.append(s)
32              s = 0
33
34          #отражение
35          pr = [(1 + alpha) * pc[i] - alpha * p[high][i] for i in range(n - 1)]
36          yr = func(pr)
37
38          if yr < y[low]:
39              #растяжение
40              pe = [(1 + gamma) * pr[i] - gamma * pc[i] for i in range(n - 1)]
41              ye = func(pe)
42              if ye < y[low]:
43                  p[high] = pe[:]
44              else:
45                  p[high] = pr[:]
46
47          elif all(yr > y[i] for i in range(n) if i != high):
48              if yr <= y[high]:
49                  p[high] = pr[:]
```

Программная реализация


```
            #сжатие
            ps = [beta * p[high][i] + (1 - beta) * pc[i]
                  for i in range(n - 1)]
            ys = func(ps)

            if ys > y[high]:
                p = [[(p[i][j] + p[low][j]) / 2
                      for j in range(n - 1)]
                     for i in range(n)]
            else:
                p[high] = ps[:]

        else:
            p[high] = pr[:]

        y = [func(item) for item in p]

        y_mean = sum(y) / n
        error = sum((y[i] - y_mean) ** 2 / n for i in range(n)) ** 0.5
        iteration += 1
        condition = error <= eps or iteration >= n * 200

    return p[low]


print(nelder_mead(func, [[0, 0], [1, 0], [0, 1]]))
```

```
[1.0000959008633465, 3.999973542055873]
```

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Генетический алгоритм

Генетический алгоритм

- Генетические алгоритмы в настоящее время широко применяются в науке и технике в качестве адаптивных алгоритмов для решения практических задач.
- Общепризнанно, что генетические алгоритмы особенно подходят для многомерных задач глобального поиска, где пространство поиска потенциально содержит несколько локальных минимумов. Базовый вариант генетического алгоритма не требует обширных знаний о пространстве поиска, таких как вероятные границы решения или функциональные производные.
- Задача, для которой простые генетические алгоритмы плохо подходят, связаны с быстрой локальной оптимизацией, однако объединение генетических алгоритмов с другими методами поиска может быть эффективным решением этой проблемы.
- Ярким примером задачи из области химической технологии, которую можно эффективно решить при помощи генетического алгоритма, является определение кинетических параметров химических реакций по экспериментальным данным (обратная кинетическая задача).
- Понятие «генетического алгоритма» было впервые введено Джоном Холландом для формального исследования механизмов естественной адаптации, но с тех пор данный алгоритм был адаптирован для решения задач вычислительного поиска. Современный вариант генетического алгоритма сильно отличается от первоначальной формы, предложенной Холландом.

Генетический алгоритм

Термины и определения

- В природе все живые организмы содержат набор генетических данных, называемых *геномом*.
- Определенный набор генетической информации является *генотипом*, и точно так же определенный набор физических характеристик, или *признаков*, является *фенотипом*.
- Пригодность данного организма к окружающей среде обычно измеряется как его *приспособленность*. В вычислительном отношении обычно оценивают *приспособленность* организма напрямую, без учета какого-либо явления.
- Идея «Выживания наиболее приспособленных», введенная Дарвином, хорошо известна. Там, где в природе *приспособленность* относится к способности организма выживать и размножаться, в генетических алгоритмах *приспособленность* является числовым значением некоторой *целевой функции*.
- Организмы с лучшим показателем *приспособленности* с большей вероятностью будут отобраны для размножения либо с помощью какого-либо механизма конкуренции за спаривание, либо в результате гибели наименее приспособленных организмов. Таким образом, гены, кодирующие полезные характеристики, передаются последующим поколениям популяции за счет генов, кодирующих вредные характеристики.

Генетический алгоритм

Целевые (фитнесс) функции

- Предполагаемые решения целевой проблемы оцениваются с использованием *функций приспособленности*, так же известных как *целевые функции*.
- Основываясь на результатах таких функций, эволюционный процесс может быть применен к набору решений. Целевая функция будет зависеть от конкретной проблемы.
- Преимущество генетических алгоритмов перед многими алгоритмами поиска или оптимизации заключается в том, что не требуется определять производные целевой функции. Этот факт гарантирует, что генетический алгоритм может быть легко применен к функциям, которые содержат разрывы или другие особенности, без каких-либо специальных обработок.

Операторы селекции

- Операторы селекции в генетических алгоритмах выполняют роль, эквивалентную естественному отбору. Общий эффект заключается в смещении набора генов в следующих поколениях к тем генам, которые принадлежат наиболее подходящим особям в текущем поколении.
- Существует множество схем отбора, описанных в литературе; выбор «колеса рулетки», конкурентный выбор, случайный отбор, стохастическая выборка. Они, по сути, имитируют процессы, связанные с естественным отбором.

Генетический алгоритм

Оператор скрещивания

Существует также множество методов создания согласованного набора генов из двух родительских наборов. В данном случае скрещивание – это процесс, посредством которого объединяются наиболее приспособленные особи. Распространенные схемы скрещивания показаны на рисунке 4.

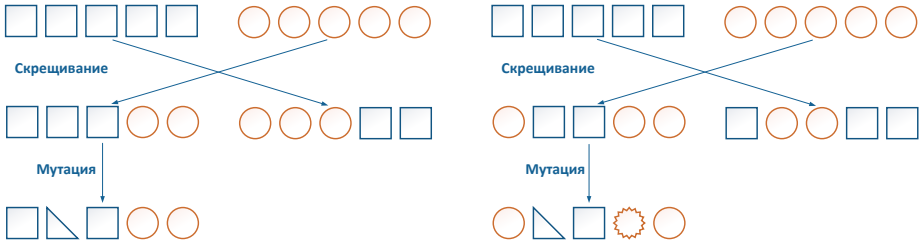

Рисунок 4 – Распространенные схемы скрещивания и мутации

Оператор мутации

Оператор мутации позволяют генетическому алгоритму поддерживать разнообразие, одновременно вводя некоторое поведение случайного поиска. Многие типы операторов мутации могут быть задуманы, как продолжение некоторой стратегии, начинающейся еще при операторе скрещивания.

Программная реализация

- Рассмотрим один из возможных примеров реализации генетического алгоритма (рисунок 5).
- Программная реализация данного алгоритма выполнена в виде функции, при этом каждый оператор генетического алгоритма является отельной функцией.
- Создание начальной популяции происходит случайным образом.
- Каждый такой набор является отдельной особью (решением поставленной задачи).
- Совокупность таких особей и составляет популяцию.
- Если есть начальные приближения к решению задачи, то они добавляются к начальной популяции.

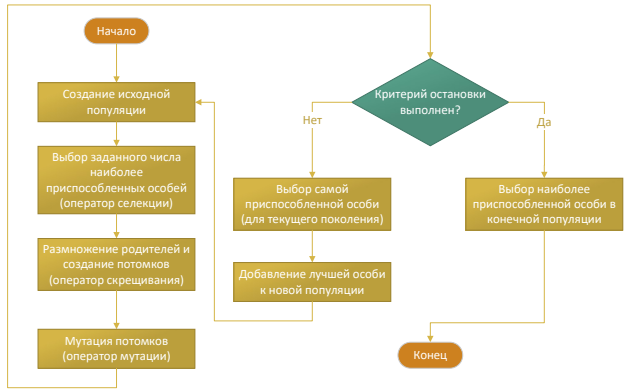

Рисунок 5 – Схема генетического алгоритма

Программная реализация

Создание начальной популяции

```python
from random import uniform


def create_population(bounds, popsize, initial_guess):
    """
    Функция для создания исходной популяции
    :param: bounds <list-object, tuple-object> границы для каждого параметра
                                               в искомом решении
    :param: popsize <int-object> размер популяции
    :param: initial_guess <list-object> список начальных приближений
                                        к решению опционально
    :return: <list-object> исходная популяция размера popsize
    """
    population = [[uniform(bounds[i][0], bounds[i][1])
                   for i in range(len(bounds))]
                  for _ in range(popsize)]

    if initial_guess: #добавляем начальное приближение, если оно было введено
        return population[:-len(initial_guess)] + initial_guess

    return population
```

После создания исходной популяции происходит сортировка всех особей по их приспособленности при помощи *оператора селекции*. Значение приспособленности для каждой особи рассчитывается при помощи целевой функции.

Программная реализация

Реализация оператора селекции

```
def selection(func, population, selected, args):
    """
    Реализация оператора селекции
    :param: func <function-object> целевая функция, значение которой нужно
                                   оптимизировать
    :param: population <list-object> популяция особей, которые нужно
                                     отсортировать
    :param: selected <int-object> количество отобранных особей
    :param: args <tuple-object> дополнительные аргументы целевой функции
                                (опционально)
    :return: <list-object> популяция отобранных особей размера selected
    """
    sorted_population = sorted(population,
                               key=lambda x: func(x, *args))
    return sorted_population[:selected]
```

На каждой итерации генетического алгоритма происходит обновление популяции созданием новых особей и уничтожения наименее приспособленных особей. Создание новых особей происходит путем моделирования процесса размножения при помощи *оператора скрещивания*. Порождающие особи называют родителями, а порожденные – потомками. Выбор родительских особей происходит случайным образом. При этом возможна ситуация, при которой один родитель участвует в нескольких скрещиваниях, и нет гарантированного участия в размножении всех родительских особей. Родительская пара генерирует пару потомков за счет обмена случайно выбранным набором генов.

Программная реализация

Реализация оператора скрещивания

```python
from random import randint


def crossover(selected_pop, popsize):
    """
    Реализация оператора скрещивания
    :param: selected_pop <list-object> наиболее приспособленные особи
    :param: popsize <int-object> исходный размер популяции
    :return: <list-object> популяция наиболее приспособленных особей
                           после скрещивания размера popsize
    """
    length = len(selected_pop)
    crossovered = int((popsize - length / 2)) or 1  #оператор скрещивания
                                                    #выполнится хотябы один раз
    count = 0
    genome1, genome2 = [], []  #вспомогательные списки для мутирующих генов

    while count != crossovered:  #индексы выбираются случайным образом
        index1 = randint(0, length - 1)
        index2 = randint(0, length - 1)

        if index1 == index2:
            continue

        genome1.append(selected_pop[index1])
        genome2.append(selected_pop[index2])
```

Программная реализация


```
28
29          count += 1
30
31      #граница деления генома также выбирается случайным образом
32      new1 = [gen1[:(s := randint(1, len(gen1) - 1))] + gen2[s:]
33              for gen1, gen2 in zip(genome1, genome2)]
34      new2 = [gen1[:(s := randint(1, len(gen1) - 1))] + gen2[s:]
35              for gen1, gen2 in zip(genome2, genome1)]
36      new_population = selected_pop + new1 + new2
37
38      return new_population[:popsize]
```

Моделирование процесса мутации лучших с точки зрения приспособленности особей происходит при помощи оператора мутации, который применяется к случайным особям и изменяет случайный ген (гены) у этих особей.

Программная реализация

Реализация оператора мутации

```python
from random import uniform


def mutate(crossed_pop, limits, m_range):
    """
    Реализация оператора мутации
    :param: selected_pop <list-object> наиболее приспособленные особи
    :param: limits <list-object> пределы мутации, в которых будут
                                 изменяться гены
    :param: m_range <float-object> коэффициент затухания мутации
                                   m_range >= 1
    :return: <list-object> популяция отобранных особей размера selected
    """
    low = 1 - (1 - limits[0]) / m_range #стремится к 1 с ростом поколений
    high = 1 + (limits[1] - 1) / m_range
    length = len(crossed_pop)
    #особи для мутации выбираются случайным образом
    indexes = [randint(0, length - 1) for _ in range(length)]
    mutation_coeff = uniform(low, high)
    mutated = crossed_pop[:]

    for i in indexes:
        mutated[i] = [item * mutation_coeff for item in mutated[i]]

    return mutated
```

Программная реализация

Реализация функции генетического алгоритма

```
1  import math
2
3  def genetic_algorithm(
4      bounds, func, initial_guess=(), args=(),
5      popsize=1000, selection_size=200,
6      mutation_limits=(0.5, 1.2), mutation_range=1,
7      generations_count=100
8  ):
9      """
10     Реализация генетического алгоритма для оптимизации целевой функции
11     :param: bounds <list-object, tuple-object> границы для каждого параметра в искомом решении
12     :param: func <function-object> целевая функция, значение которой нужно оптимизировать
13     :param: initial_guess <list-object> список начальных приближений к решению, по умолчанию
       пустой кортеж
14     :param: args <tuple-object> дополнительные аргументы целевой функции (опционально)
15     :param: popsize <int-object> размер популяции, по умолчанию popsize=1000
16     :param: selection_size <int-object> количество отобранных особей, по
       умолчанию selection_size=200
17     :param: mutation_limits <tuple-object> пределы мутации, в которых будут изменяться гены, по
       умолчанию mutation_limits=(0.5, 1.2)
18     :param: mutation_range <float-object> коэффициент затухания мутации, mutation_range >= 1,
       увеличивается с ростом числа поколений, по умолчанию mutation_range=1
19     :param: generations_count <int-object> количество поколений, по
       умолчанию  generations_count=100
20     :return: <list-object> лучшие особи каждого поколения, отсортированные по уменьшению
       приспособленности
21     """
```

Программная реализация


```
22      best = [None for _ in range(generations_count)]
23
24      for generation in range(generations_count):
25          mutation_decrease = mutation_range + math.log(1 + generation)
26          population = create_population(bounds, popsize, initial_guess)
27          selected = selection(func, population, selection_size, args)
28          crossed = crossover(selected, popsize)
29          mutated = mutate(crossed, mutation_limits, mutation_decrease)
30          population = mutated[:]
31          best[generation], = selection(func, population, 1, args)
32          initial_guess = [best[generation]]
33
34      return selection(func, best, generations_count, args)
```

- Поскольку генетический алгоритм предполагает итерационный процесс, вызов генетических операторов будет происходить внутри цикла.
- В качестве критерия завершения цикла выберем критерий достижения определенного числа поколений.
- Для различных задач оптимальное число поколений может отличаться, однако для обеспечения высокой степени вероятности нахождения хорошего решения зададим требуемое число поколений равное 100.
- Для удобства многократного использования описанной выше реализации генетического алгоритма, ее следует поместить в отдельный модуль и подключать в различные проекты по мере необходимости.

Пример

Пусть дана схема химических реакций:

$$A \underset{k_2}{\overset{k_1}{\rightleftarrows}} B$$

Скорость прямой реакции: $r_1 = k_1 \cdot C_A$; скорость обратной реакции: $r_2 = k_2 \cdot C_B$; C_A и C_B – концентрации компонентов A и B. Изменение концентрации реагирующих веществ во времени описывается следующей системой дифференциальных уравнений:

$$\begin{cases} \dfrac{\partial C_A}{\partial t} = -r_1 + r_2 \\ \dfrac{\partial C_B}{\partial t} = r_1 - r_2 \end{cases}$$

Необходимо определить константы скоростей реакций: k_1 и k_2, если известно, что к моменту времени $t = 1(c)$ $C_A = 0.4513$; $C_B = 0.5487$. Начальные условия: $C_A(0) = 1$ (моль/л); $C_B(0) = 0$ (моль/л).

Пример

- Для получения более точного решения выберем метод Рунге-Кутты и предположим, что его реализация, а также реализация генетического алгоритма, сохранены в отдельных модулях под именами `solve_ode` и `genetic_algorithm`, соответственно.
- Для того, чтобы найти константы скоростей химических реакций по данным лабораторного эксперимента при помощи генетического алгоритма потребуется составить целевую функцию, которую нужно будет минимизировать.
- В данном случае можно использовать сумму квадратов разности расчетных и экспериментальных значений концентраций компонентов, участвующих в химическом превращении:

$$F = \sum_{i=1}^{n} \left(C_i^{calc} - C_i^{act} \right)^2$$

где C_i^{calc} – расчетная концентрация i-го компонента; C_i^{act} – концентрация i-го компонента, определенная экспериментально; n – число компонентов.

Прежде всего необходимо задать систему дифференциальных уравнений в виде отдельной функции:

```
1  def equations(t, c, k):
2      right_parts = [
3          -k[0] * c[0] + k[1] * c[1],
4          k[0] * c[0] - k[1] * c[1]
5      ]
6      return right_parts
```

Программная реализация

Реализация целевой функции

```
def obj_func(k, equations, method, t, h,
             initial_composition, actual_values):
    """
    Целевая функция для минимизации генетическим алгоритмом
    :param: k <list-object> список констант скоростей химических реакций
    :param: equations <function-object> правые части дифференциальных уравнений
    :param: method <function-object> численный метод решения систем дифференциальных уравнений
    :param: t <list-object> пределы интегрирования
    :param: h <float-object> шаг интегрирования
    :param: initial_composition <list-object> начальные условия
    :param: actual_values <list-object> фактические значения концентраций из эксперимента
    :return: <float-object> квадрат суммы октлонений расчетных концентраций от экспериментальных
    """
    #решение системы ОДУ численным методом
    c = method(equations, t[0], t[-1],
               initial_composition, h, args=(k,))

    return sum((c[-1][i] - actual_values[i]) ** 2
               for i in range(len(actual_values)))
```

Программная реализация

Ограничим область поиска для значения констант диапазоном $[0, 1]$.

```
import genetic_algorithm as ga
from solve_ode import rk


def obj_func(k, equations, method, t, h,
             initial_composition, actual_values):
    #решение системы ОДУ численным методом
    c = method(equations, t[0], t[-1],
    initial_composition, h, args=(k,))

    return sum((c[-1][i] - actual_values[i]) ** 2
               for i in range(len(actual_values)))


actual_values = [0.4513, 0.5487]  #экспериментальные значения
k = ga.genetic_algorithm([[0, 1], [0, 1]], obj_func,
                         args=(equations, rk, [0, 1], 0.1,
                               [1, 0], actual_values))
print(k[0])
```

```
[0.8614790508500587, 0.12043760065726794]
```

По аргументу `method` можно передавать различные численные методы решения систем обыкновенных дифференциальных уравнений.

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Контакты

Вячеслав Алексеевич Чузлов,
к.т.н., доцент ОХИ ИШПР

Учебный корпус №2, ауд. 136
chuva@tpu.ru
+7-962-782-66-15

Благодарю за внимание!