Метод Ньютона для решения систем нелинейных уравнений

Шепрут Илья

26 января 2019 г.

Содержание

1. Введение

Часто в программировании возникает задача решать системы уравнений, причем это не обычные Системы Линейных Алгебраических Уравнений (СЛАУ), для которых существует несчетное число методов решения, а нелинейные системы, где аналитическое решение чаще всего не выражается. Тут на помощь приходят численные методы нахождения решения. Метод Ньютона является одним из них. Если обычный метод Ньютона решает одномерные уравнения, то в данной статье будет описан этот метод для многомерного случая, то есть когда у нас не одно уравнение, а система.

Всё будет описано для размерности $n = 2$, но это легко обобщается на более высокие размерности.

2. Формулировка

Дана Система Нелинейных Уравнений (СНУ) в виде:

$$\begin{cases} F_1(x_1,\ x_2) = 0 \\ F_2(x_1,\ x_2) = 0 \end{cases} \tag{1}$$

Требуется, чтобы она имела непрерывные производные 1 порядка.

Необходимо найти такие $x_1,\ x_2$ при которых система оборачивается в нуль. Это называется решение СНУ.

3. Обозначения

Заменим в (1) аргументы на вектор $\mathbf{x} = (x_1,\ x_2)$:

$$\begin{cases} F_1(\mathbf{x}) = 0 \\ F_2(\mathbf{x}) = 0 \end{cases} \tag{2}$$

Преобразуем (2) к одной вектор-функции $\mathbf{F}$, такой что $\mathbf{F} : \mathbb{R}^2 \to \mathbb{R}^2$:

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \end{pmatrix}$$

Тогда (1) примет вид:

$$\mathbf{F}(\mathbf{x}) = \mathbf{0} \tag{3}$$

где $\mathbf{0} = (0,\ 0)^T$.

Под **невязкой** понимается вектор $\mathbf{F}(\mathbf{x})$.

4. Преобразования

4.1. Разложение в ряд Тейлора для функции многих переменных

Взято с Википедии.

Пусть дана функция $f(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}$, $\mathbf{x} = (x_1,\ x_2)^T$.

Разложение функции $f(\mathbf{x})$ будет произведено в окрестности точки $\mathbf{a} = (a_1,\ a_2)^T$. Требуется, чтобы у функции f имелись непрерывные производные до $(2+1)$-го порядка включительно по всем переменным.

Тогда разложение в ряд Тейлора имеет следующий вид:

$$f(\mathbf{x}) = f(\mathbf{a}) + \sum_{i=1}^{n} \frac{\partial f(\mathbf{a})}{\partial x_i}(x_i - a_i) + \frac{1}{2!}\sum_{i=1}^{2}\sum_{j=1}^{2} \frac{\partial f(\mathbf{a})}{\partial x_i \partial x_j}(x_i - a_i)(x_j - a_j) + R_3(\mathbf{x}) \tag{4}$$

Раскроем сумму первых двух членов предыдущего равенства:

$$f(\mathbf{x}) = f(\mathbf{a}) + \frac{\partial f(\mathbf{a})}{\partial x_1}(x_1 - a_1) + \frac{\partial f(\mathbf{a})}{\partial x_2}(x_2 - a_2) + R_2(\mathbf{x}) \tag{5}$$

Где $R_n(\mathbf{x})$ — остаточный член в форме Лагранжа, $R_n(\mathbf{x}) \xrightarrow{n\to\infty} 0$. (6)

4.2. Разложение в ряд Тейлора СНУ

Обозначим за $\hat{\mathbf{x}}$ искомое решение, за $\mathbf{x}^{[k]}$ решение на k-м шаге и введем ещё обозначение смещения $\Delta\mathbf{x}^{[k]} = \hat{\mathbf{x}} - \mathbf{x}^{[k]}$.

Разложим $F_i(\mathbf{x})$ в окрестности точки $\mathbf{x}^{[k]}$ в ряд Тейлора по формуле (5):

$$F_i(\mathbf{x}) = F_i\left(\mathbf{x}^{[k]}\right) + \frac{\partial F_i\left(\mathbf{x}^{[k]}\right)}{\partial x_1}\left(x_1 - x_1^{[k]}\right) + \frac{\partial F_i\left(\mathbf{x}^{[k]}\right)}{\partial x_2}\left(x_2 - x_2^{[k]}\right) + R_2(\mathbf{x})$$

Подставив в это разложение точку $\hat{\mathbf{x}}$ получаем следующее:

$$F_i(\hat{\mathbf{x}}) = F_i\left(\mathbf{x}^{[k]}\right) + \frac{\partial F_i\left(\mathbf{x}^{[k]}\right)}{\partial x_1}\Delta x_1^{[k]} + \frac{\partial F_i\left(\mathbf{x}^{[k]}\right)}{\partial x_2}\Delta x_2^{[k]} + R_2(\hat{\mathbf{x}})$$

Теперь запишем (3) с учетом предыдущего разложения:

$$\begin{cases} F_1\left(\mathbf{x}^{[k]}\right) + \dfrac{\partial F_1\left(\mathbf{x}^{[k]}\right)}{\partial x_1}\Delta x_1^{[k]} + \dfrac{\partial F_1\left(\mathbf{x}^{[k]}\right)}{\partial x_2}\Delta x_2^{[k]} + R_2^{F_1}(\hat{\mathbf{x}}) = 0 \\ F_2\left(\mathbf{x}^{[k]}\right) + \dfrac{\partial F_2\left(\mathbf{x}^{[k]}\right)}{\partial x_1}\Delta x_1^{[k]} + \dfrac{\partial F_2\left(\mathbf{x}^{[k]}\right)}{\partial x_2}\Delta x_2^{[k]} + R_2^{F_2}(\hat{\mathbf{x}}) = 0 \end{cases}$$

Или в матричном виде:

$$\mathbf{F}\left(\mathbf{x}^{[k]}\right) + \begin{pmatrix} \dfrac{\partial F_1\left(\mathbf{x}^{[k]}\right)}{\partial x_1} & \dfrac{\partial F_1\left(\mathbf{x}^{[k]}\right)}{\partial x_2} \\ \dfrac{\partial F_2\left(\mathbf{x}^{[k]}\right)}{\partial x_1} & \dfrac{\partial F_2\left(\mathbf{x}^{[k]}\right)}{\partial x_2} \end{pmatrix} \cdot \Delta\mathbf{x}^{[k]} + \begin{pmatrix} R_2^{F_1}(\hat{\mathbf{x}}) \\ R_2^{F_2}(\hat{\mathbf{x}}) \end{pmatrix} = \mathbf{0}$$

$$\mathbf{F}\left(\mathbf{x}^{[k]}\right) + \mathbf{J_F}\left(\mathbf{x}^{[k]}\right) \cdot \Delta\mathbf{x}^{[k]} + \mathbf{R}_2(\hat{\mathbf{x}}) = \mathbf{0}$$

$$\mathbf{J_F}\left(\mathbf{x}^{[k]}\right) \cdot \Delta\mathbf{x}^{[k]} = -\mathbf{F}\left(\mathbf{x}^{[k]}\right) - \mathbf{R}_2(\hat{\mathbf{x}})$$

Где $\mathbf{J_F}(\mathbf{x})$ — матрица Якоби функции $\mathbf{F}$ в точке $\mathbf{x}$.

Из этого уравнения можно найти $\Delta\mathbf{x}^{[k]}$, который прибавляется к текущему решению $\mathbf{x}^{[k]}$ чтобы найти $\hat{\mathbf{x}}$.

Мы не не знаем как вычислять $\mathbf{R}_2(\hat{\mathbf{x}})$, но можем им пренебречь, так как он стремится к нулю. Именно из-за этого принебрежения мы не сразу получаем точное решение, а процесс становится итеративным. Итог:

$$\mathbf{J_F}\left(\mathbf{x}^{[k]}\right) \cdot \Delta\mathbf{x}^{[k]} = -\mathbf{F}\left(\mathbf{x}^{[k]}\right) \tag{7}$$

Можно заметить, что это СЛАУ. Её можно легко решить известными методами.

5. Алгоритм

6. Стандартный метод Ньютона

Весь метод заключается в итерационном процессе:

$$\mathbf{x}^{[k+1]} = \mathbf{x}^{[k]} + \Delta\mathbf{x}^{[k]} \tag{8}$$

$$\mathbf{x}^{[k+1]} = \mathbf{x}^{[k]} - \left(\mathbf{J_F}\left(\mathbf{x}^{[k]}\right)\right)^{-1}\mathbf{F}\left(\mathbf{x}^{[k]}\right) \tag{9}$$

Далее за k обозначается количество итераций; за m максимальное число итераций; за ε_1 максимальная допустимая невязка;

Algorithm 1: Метод Ньютона

Input: ε_1, m, $\mathbf{F}$, $\mathbf{x}^{[0]}$
Result: $\mathbf{x}$

```
1  begin
2      x ← x^[0]
3      k ← 0
4      while ||F(x)|| > ε_1 and k < m do
5          A ← J_F(x)
6          b ← −F(x)
7          Решить СЛАУ A · Δx = b, результат поместить в Δx
8          x ← x + Δx
9          k ← k + 1
10     end
11 end
```

Выход из итерационного процесса производится когда невязка достигает необходимой нижней границы, либо превышается максимальное число итераций.

В $||\mathbf{F}(\mathbf{x})|| > \varepsilon_1$ невязка считается через норму, потому что так проще сравнить вектор с одним числом. Норма выбирается по вашему усмотрению.

Замечание: данный алгоритм не зависит от размерности системы, а, следовательно, может быть применен к любой размерности исходной СНУ.

Замечание: в алгоритме используется матрица Якоби в известной точке, что означает, что знать её аналитическое выражение не нужно, её можно вычислять при помощи численного взятия производной в конкретной точке.

6.1. Улучшенный метод Ньютона

Иногда может получаться такой $\Delta\mathbf{x}$, что невязка в векторе $\mathbf{x}^{[k+1]}$ будет больше, чем в $\mathbf{x}^{[k]}$, поэтому вводится коэффициент $\beta \in [0, 1]$, который домножается на $\Delta\mathbf{x}$, причем β подбирается такой, чтобы невязка строго уменьшалась. Тогда (8) будет выглядеть следующим образом:

$$\mathbf{x}^{[k+1]} = \mathbf{x}^{[k]} + \beta \cdot \Delta\mathbf{x}^{[k]}$$

Чтобы найти такой β, при котором невязка будет уменьшаться, используется аналог двоичного поиска: изначально $\beta = 1$, затем, пока невязка нового решения больше, чем предыдущего, β уменьшается вдвое.

Так же, если $\beta \to 0$, то это означает, что невязка не может быть уменьшена на текущей итерации, и это ещё один повод завершить итерационный процесс.

Algorithm 2: Улучшенный Метод Ньютона

Input: ε_1, ε_2, m, $\mathbf{F}$, $\mathbf{x}^{[0]}$
Result: $\mathbf{x}$

begin
  $\mathbf{x} \leftarrow \mathbf{x}^{[0]}$
  $\beta \leftarrow 1$
  $k \leftarrow 0$
  while $||\mathbf{F}(\mathbf{x})|| > \varepsilon_1$ **and** $\beta > \varepsilon_2$ **and** $k < m$ **do**
    $\mathbf{A} \leftarrow \mathbf{J_F}(\mathbf{x})$
    $\mathbf{b} \leftarrow -\mathbf{F}(\mathbf{x})$
    Решить СЛАУ $\mathbf{A} \cdot \Delta\mathbf{x} = \mathbf{b}$, результат поместить в $\Delta\mathbf{x}$
    $\beta \leftarrow 1$
    while $||\mathbf{F}(\mathbf{x} + \beta\Delta\mathbf{x})|| > ||\mathbf{F}(\mathbf{x})||$ **and** $\beta > \varepsilon_2$ **do**
      $\beta \leftarrow 0.5\beta$
    end
    $\mathbf{x} \leftarrow \mathbf{x}^{[k]} + \beta \cdot \Delta\mathbf{x}$
    $k \leftarrow k + 1$
  end
end

6.2. Одномерный случай

В одномерном случае (9) преобразуется к:

$$x^{[k+1]} = x^{[k]} - \left(\frac{df\left(x^{[k]}\right)}{dx}\right)^{-1} f\left(x^{[k]}\right) = x^{[k]} - \frac{f\left(x^{[k]}\right)}{f'\left(x^{[k]}\right)}$$

7. Пример

Например, у нас имеется следующая СНУ:

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 = (r_1 + r)^2 \\ (x - x_2)^2 + (y - y_2)^2 = (r_1 + r)^2 \\ \dfrac{|ax + by + c|}{\sqrt{a^2 + b^2}} = r \end{cases}$$

Решением этой СНУ является окружность с центром в точке $(x,\ y)$ и радиусом r, которая касается внешним образом окружностей $(x_1,\ y_1,\ r_1)$, $(x_2,\ y_2,\ r_2)$ и прямой $ax + by + c = 0$.

Для этой системы получим:

$$\mathbf{F} = \begin{pmatrix} (x-x_1)^2 + (y-y_1)^2 - (r_1+r)^2 \\ (x-x_2)^2 + (y-y_2)^2 - (r_1+r)^2 \\ \dfrac{|ax+by+c|}{\sqrt{a^2+b^2}} - r \end{pmatrix}, \mathbf{J_F} = \begin{pmatrix} 2(x-x_1) & 2(y-y_1) & -2(r_1+r) \\ 2(x-x_2) & 2(y-y_2) & -2(r_2+r) \\ \dfrac{|a|}{\sqrt{a^2+b^2}} & \dfrac{|b|}{\sqrt{a^2+b^2}} & -1 \end{pmatrix}$$

Далее остается только применить алгоритм.

8. Когда размерность СНУ не соответствует размерности решения

Пусть дана СНУ:

$$\begin{cases} F_1(x_1,\ x_2,\ \ldots,\ x_n) = 0 \\ F_2(x_1,\ x_2,\ \ldots,\ x_n) = 0 \\ \ldots \\ F_m(x_1,\ x_2,\ \ldots,\ x_n) = 0 \end{cases}$$

Вышеописанный алгоритм работает только когда $m = n$, но бывают так же случаи когда $m \neq n$. Если попробовать применить алгоритм к такой СНУ, то можно заметить, что проблемы с ней возникают только при вычислении решения СЛАУ. И следующие методы как раз будут давать методы решения СЛАУ конкретно для этой задачи.

И в алгоритме возникают лишь небольшие изменения:

8.1. Расширенный метод Ньютона для $m \neq n$

Algorithm 3: Расширенный Метод Ньютона для $m \neq n$

Input: ε_1, ε_2, m, $\mathbf{F}$, $\mathbf{x}^{[0]}$
Result: $\mathbf{x}$

```
begin
    x ← x^[0]
    β ← 1
    k ← 0
    while ||F(x)|| > ε1 and β > ε2 and k < m do
        A ← J_F(x)
        b ← −F(x)
        Δx ← 𝕊(A, b, F, x)
        while ||F(x + βΔx)|| > ||F(x)|| and β > ε2 do
            β ← 0.5β
        end
        x ← x^[k] + β · Δx
        k ← k + 1
    end
end
```

Где $\mathbb{S}$ — функция, получающая текущую СЛАУ функцию вычисления и текущее решение, возвращающая вектор смещения $\Delta\mathbf{x}$. Данная функция «решает» эту СЛАУ некоторым методом. Этим методам и будет посвящена следующие главы.

Главная цель функции $\mathbb{S}$ — по всем имеющимся данным найти такой подходящий вектор смещения, чтобы решение двигалось в сторону уменьшения невязки.

В случае, когда $m = n$, $\mathbb{S}$ — просто функция, решающая обычную СЛАУ.

8.2. $m < n$

Это означает, что решений, где исходная СНУ оборачивается в ноль, множество.

Так же это означает, что не хватает уравнений, для того, чтобы построить полный вектор смещения $\Delta\mathbf{x}$ при решении СЛАУ. Поэтому делается так, чтобы в итоге некоторые компоненты $\Delta\mathbf{x}$ были равны нулю, предполагается что по этим компонентам уже найдено достаточно хорошее решение, и искать для них смещения не нужно.

Какие именно компоненты занулять — выбирается таким образом, чтобы подтверждалось вышеописанное предположение. В данном случае они выбирается по значениям частных производных, то есть для каждой компоненты под номером j находится число $y_j = \max_i \left|\frac{\partial F_i(\mathbf{x})}{\partial x_j}\right|$, и зануляются те $(n-m)$ компонент, для которых y_j минимально.

Далее можно просто исключить соответствующие строки из матрицы Якоби и компоненты из вектора $\mathbf{b}$.

Именно в этих действиях и заключается функция $\mathbb{S}$ для этого метода.

8.3. $m > n$

Это означает, что решений может не быть, в данном случае метод Ньютона находит точку с минимальной невязкой.

Так же это означает, что уравнений слишком много, чтобы найти вектор смещения $\Delta\mathbf{x}$, поэтому придумываются методы, чтобы либо исключить какие-то уравнения, либо скомбинировать их каким-то образом. Но нельзя сделать эти преобразования один раз. На каждой итерации придется делать это заного.

8.3.1. Метод исключения

Создается новая СНУ, состоящая только из тех n уравнений, для которых $|F_i(\mathbf{x})|$ максимально.

Далее для полученной СНУ заного считается матрица Якоби, вектор $\mathbf{b}$, и уже по новым данным получается СЛАУ корректного размера, по которой находится $\Delta\mathbf{x}$.

Обратите внимание, что можно получить матрицу Якоби нового уравнения и вектор $\mathbf{b}$ лишь исключив нужные строки.

8.3.2. Метод свертки

Создается новая СНУ, которая состоит из $(n-1)$ уравнений исходной СНУ, для которых $|F_i(\mathbf{x})|$ максимально, все оставшиеся $(n-m-1)$ уравнений комбинируются: складывается сумма их квадратов.

Далее для полученной СНУ заного считается матрица Якоби, вектор $\mathbf{b}$, и уже по новым данным получается СЛАУ корректного размера, по которой находится $\Delta\mathbf{x}$.

Обратите внимание, что можно получить матрицу Якоби нового уравнения и вектор $\mathbf{b}$, сложив необходимые строки матрицы Якоби, помноженные на $2\mathbf{b}$, а в векторе $\mathbf{b}$ сложить необходимые строки в квадрате.

В матрице Якоби надо складывать строки именно с такими действиями, потому что матрица Якоби состоит из производных, и данные действия можно получить, взяв производную суммы квадратов уравнений.

8.3.3. Процедура симметризации

Вместо исходной системы должна решаться система:

$$\left(\mathbf{J_F}\left(\mathbf{x}^{[k]}\right)\right)^T \cdot \mathbf{J_F}\left(\mathbf{x}^{[k]}\right) \cdot \Delta\mathbf{x} = \left(\mathbf{J_F}\left(\mathbf{x}^{[k]}\right)\right)^T \cdot \mathbf{F}\left(\mathbf{x}^{[k]}\right)$$