Отчет по лабораторной работе №3: Решение систем нелинейных уравнений

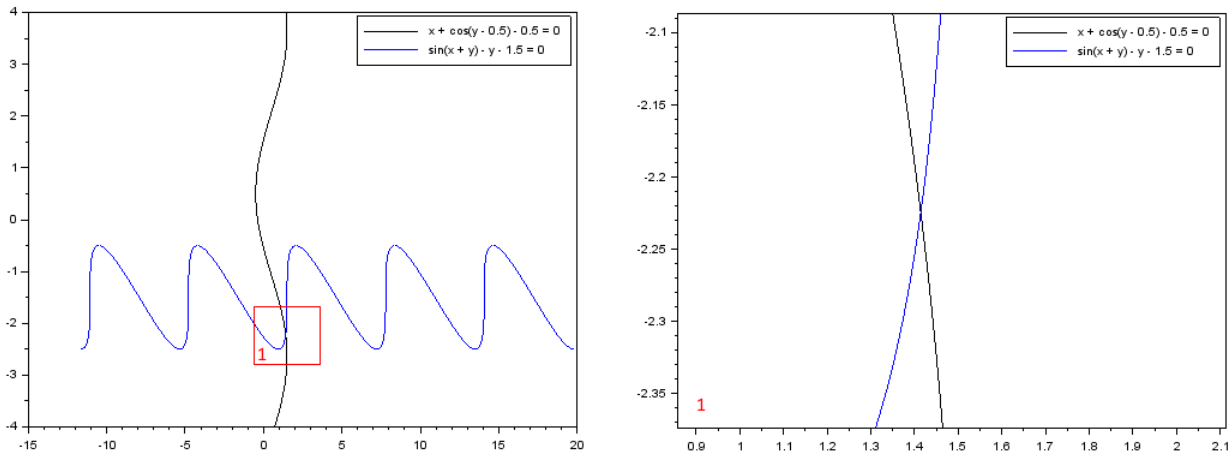
Чигарев Дмитрий 381807-1

1, 2. Графическое исследование

Система уравнений для решения:

$$\begin{cases} \sin(x+y) - y - 1.5 = 0 \\ x + \cos(y - 0.5) - 0.5 = 0 \end{cases}$$

Отобразим оба уравнения системы на графике, точки их пересечения – искомые решения



Из рисунка видно, что у системы имеется как минимум одно решение, его начальным приближением может служить $x_0 = (1.4, -2.2)$

3, 4, 5, 6. Вычисление Якобиана

Поскольку для поиска корней нам скорее всего понадобится Якобиан (нужен для метода Ньютона и модификаций методов оптимизации), то появляется задача о его определении.

Якобиан системы многомерных функций определяется как матрица их частных производных:

$$J(x_0, y_0) = \begin{pmatrix} f'_{(1,x)}(x_0, y_0) & f'_{(1,y)}(x_0, y_0) \\ f'_{(2,x)}(x_0, y_0) & f'_{(2,y)}(x_0, y_0) \end{pmatrix}$$

Заполнить Якобиан можно двумя способами:

1. Аналитически взять все частные производные и составить из них матрицу, в которую мы будет подставлять переданный x, дабы получить значения Якобиана в заданной точке.
$$f'_x(x_0, y_0) = \frac{\partial f}{\partial x}(x_0, y_0)$$
2. По переданному x вычислять численное значение Якобиана по определению производной:
$$f'_x(x_0, y_0) = \frac{f(x_0 + \Delta, y_0) - f(x_0, y_0)}{\Delta}, \text{где } \Delta - \text{некоторый маленький шаг}$$

такой способ заполнения уже реализован в SciLab функцией numderivative, которая возвращает значение Якобиана функции в заданной точке.

Построим Якобиан нашей системы первым способов, взяв все частные производные:

$$J(x_0, y_0) = \begin{pmatrix} \cos(x_0 + y_0) & \cos(x_0 + y_0) - 1 \\ 1 & \sin(0.5 - y_0) \end{pmatrix}$$

Перейдем к коду и выведем информацию о значении функции и якобиана в начальном приближении:

```
>>> disp("f(x) в начальном приближении:", f(x0))
  "f(x) в начальном приближении:"

  -0.017356090899523
  -0.004072142017061
>>> disp("Якобиан в начальном приближении:", jacob_exact(x0))
  "Якобиан в начальном приближении:"

   0.696706709347165  -0.303293290652835
   1.                  0.42737988023383
>>> disp("Обратный Якобиан в начальном приближении:", jacob_exact(x0)^-1)
  "Обратный Якобиан в начальном приближении:"

   0.711053417783105   0.504604313126278
  -1.663750332360218   1.15914601923394
```

7. Метод Ньютона

Метод Ньютона для систем многомерных нелинейных уравнений, имеет такую же форму, как и в одномерном случае, единственное отличие, что производная функции заменяется на её Якобиан:

$$x_{k+1} = x_k - J(x_k)^{-1} * f(x_k)$$

Найдем решение системы используя два разных определения Якобиана: приближенный и точный

```
>>> # Метод Ньютона с приближенным Якобианом
>>> x11 = newtoon_method(f, x01, eps=10e-6, jacob=jacob_approx, verbose=%T)
Шаг 1:
--------
  "x:"                  "f(x):"
  "1.414395928178238"   "0.000034274962342"
  "-2.224155994793144"  "0.000264759663538"

Шаг 2:
--------
  "x:"                  "f(x):"
  "1.414233099729249"   "0.0000000621391"
  "-2.224407416965787"  "0.000000028893608"

Шаг 3:
--------
  "x:"                  "f(x):"
  "1.414233041837887"   "0"
  "-2.224407345399741"  "0.000000000000002"
>>> # Метод Ньютона с точным Якобианом
>>> x12 = newtoon_method(f, x01, eps=10e-6, jacob=jacob_exact, verbose=%T)
```

```
Шаг 1:
--------
  "x:"                  "f(x):"
  "1.414395928178932"   "0.000034274963"
  "-2.224155994793724"  "0.000264759663997"

Шаг 2:
--------
  "x:"                  "f(x):"
  "1.414233099729257"   "0.000000062139109"
  "-2.224407416965801"  "0.00000002889361"

Шаг 3:
--------
  "x:"                  "f(x):"
  "1.414233041837887"   "0"
  "-2.224407345399741"  "0.000000000000002"
```

Найдем разницу между решениями с точным и приближенным Якобианом:

```
>>> norm(x11 - x12)
2.04979500238D-16
```

Как видно различия в решениях начинаются только в 16-том знаке после запятой (тип double в языке C, согласно стандарту, гарантирует лишь 15 знаков после запятой).

8. SciLab fsolve

SciLab в своей стандартной библиотеке предоставляет функцию fsolve, которая находит решение системы нелинейных уравнений по заданному начальному приближению.

Судя по документации, fsolve использует некоторую модификацию метода Пауэлла (какая именно модификация не уточняется). По сути своей метод Пауэлла – это способ отыскания минимума функций.

Использование оптимизационных методов – ещё одна из частых практик отыскании решений систем нелинейных уравнений, которая обусловлена в том числе и простотой сведения исходной задачи к оптимизационной. Пусть у нас есть система уравнений:

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases}$$

Тогда эквивалентную ей экстремальную задачу можно записать как:

$$F(x,y) = f^2(x,y) + g^2(x,y) \to min$$

видно, что глобальным минимумом функции F будет 0, который может быть достигнуто, только при условии равенства нулю обоих уравнений начальной системы.

Главное достоинство оптимизационных методов – глобальная сходимость, оптимизационный процесс всегда приведет нас к какой-либо точке минимума, которая, при определенных условиях, будет решением исходной системы, однако за это мы получаем низкую, по сравнению с методами отыскания корней, скорость сходимости.

Решим систему с помощью функции fsolve:

```
>>> x = fsolve(x01, f, fjac=jacob_exact, tol=10e-6)
>>> disp(["x:", "f(x):"; string(x), string(f(x))])
  "x:"                  "f(x):"
  "1.414233041837886"   "0"
  "-2.224407345399743"  "2.22044604925D-16"
```

9. Сравнение исследованных методов

Метод	x	f(x)	Сложность
Ньютон с приближенным Якобианом	$1.414233042D+00$ $-2.224407345D+00$	$0.000000000D+00$ $2.442490654D-15$	Нужно числено считать Якобиан на каждом шаге. (квадратичная сходимость)
Ньютон с точным Якобианом	$1.414233042D+00$ $-2.224407345D+00$	$0.000000000D+00$ $2.442490654D-15$	Нужно рассчитать Якобиан заранее. (квадратичная сходимость)
fsolve	$1.414233042D+00$ $-2.224407345D+00$	$0.000000000D+00$ $2.220446049D-16$	Нужно как-нибудь рассчитать Якобиан (предположительно линейная сходимость)

Приложения

1. Ссылка на код: https://github.com/proxodilka/numerical-analysis-labs/blob/master/lab3_non_linear_system/lab3.sce