MN2: Assignment 2

Taras Yarema

December 19, 2020

1 Pre-defined constraints

We define (using the macro definition `#define`) the following constrains for the whole program execution:

- `TOLERANCE = 1e-12`: error bound.
- `MAX_ITERATIONS = 1000`: maximum Newton method iterations.
- `H_STEP = 1e-3`: value of h, used in the last section.
- `PLOT_STEPS = 100000`: steps for the plotting values.

2 C functions implemented

As in section (a), we implemented the C function to compute values of $f(x, y)$ and $\nabla f(x, y) = (\frac{\partial}{\partial x} f, \frac{\partial}{\partial y} f)$. The function signatures for those implementations are `double f(double, double)` and `double gradient(double, double)`.

Note that the gradient function depends on the partial derivatives of f which have signatures `double partial_x(double, double)` and `double partial_y(double, double)`.

In section (b) we need to implement the simple Newton method for one dimension. For that, we suppose that one of the coordinates x or y is zero. We then apply the Newton method as described in [1, Note 3.2.1 (a)].

The implementation is for both variables, with signatures `int newton_zero_x(double*)` and `int newton_zero_y(double*)`. Note that these method return the minimum of the `MAX_ITERATIONS` and the needed iterations by the method based on the pre-defined tolerance `TOLERANCE`. We then store the value in the pointer given as argument.

3 Solving the system of equations using Newton's method

We are given a system of equations which can be rewritten as

$$\begin{cases} f(x, y) = 0 \\ g(x, y) := (x - x_0)^2 + (y - y_0)^2 - h^2 = 0 \end{cases} \tag{1}$$

We can now define the map $H(x, y) := (f(x, y), g(x, y))$ such that

$$DH(x, y) = \begin{pmatrix} \frac{\partial}{\partial x} f & \frac{\partial}{\partial y} f \\ 2(x - x_0) & 2(y - y_0) \end{pmatrix} \tag{2}$$

Suppose that we have a solution $\overline{x}$ of the system 1, so if we do the Taylor development of H in a point x_k sufficiently near to $\overline{x}$ we have

$$0 = H(\overline{x}) \approx H(x_k) + DH(x_k)(\overline{x} - x_k) \tag{3}$$

Now, we know by [1, Equation 3.1] that we can get the following iterative method to get an approximation of the solution from 3 such that, with a predefined x_0, for all $\forall k \geq 0$

$$x_{k+1} = x_k - DH(x_k)(x_k)^{-1}H(x_k) \tag{4}$$

So, as we already have f and g predefined, we can compute the product $DH(x_k)(x_k)^{-1}H(x_k)$. We only need to know the structure of $DH(x_k)(x_k)^{-1}$. Firstly, the determinant of DH is

$$|DH(x,y)| = \begin{vmatrix} \frac{\partial}{\partial x}f(x,y) & \frac{\partial}{\partial y}f(x,y) \\ 2(x-x_0) & 2(y-y_0) \end{vmatrix} = 2(y-y0)\frac{\partial}{\partial x}f(x,y) - 2(x-x0)\frac{\partial}{\partial y}f(x,y) \tag{5}$$

So the inverse of DH is

$$DH(x,y)^{-1} = \frac{1}{|DH(x,y)|}\begin{pmatrix} 2(y-y0) & -\frac{\partial}{\partial y}f(x,y) \\ -2(x-x_0) & \frac{\partial}{\partial x}f(x,y) \end{pmatrix} \tag{6}$$

Finally, if we put together 6 and 4, we get the wanted method. This method has signature `int newton_iterator(double, double, double, double, double, double*)`. This function is passed a first guess x_0 and a point at a distance h, x_1. The result of the method is stored in the pointer passed as last argument.

Also, an integer exit code is returned to know if there were any errors during the method computation.

4 Plotting the curve

To plot the curve we use the Newton iterator method from the last section to compute the next point to plot. We start at the point given by `partial_x(0)` (we could pick `partial_y` too) and do `PLOT_STEPS` iterations while saving the computed points in a text file. Finally we use `gnuplot` to plot the curve, getting as result the desired curve from figure 1.

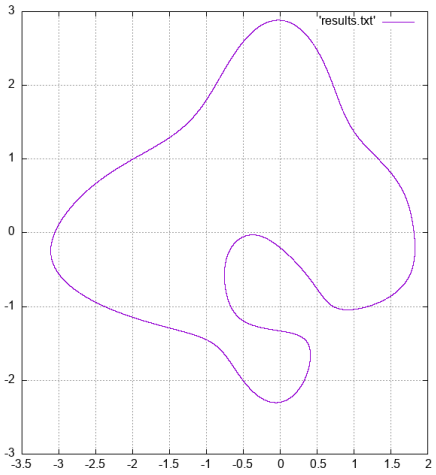

Figure 1: Plot of the curve such that $f(C) = 0$

References

[1] Joan Carles Tatjer. *Mètodes Numèrics II*. UB, 2020.