M2 Modélisation et Analyse Numérique
UE Programmation 2 - 2022/2023

CC 2 - Classes (I)

les fichiers à rendre à l'issue de cette évaluation sont indiqués en rouge, *alors que les fichiers fournis sont indiqués en* bleu *(dans le dossier* /CC2*). merci de respecter la dénomination demandée.*

Several methods may be used to solve nonlinear equations of the form $f(x) = 0$. When f is regular enough and such that f' does not vanishes (at least in the vicinity of the sought root), the *Newton-Raphson* method may be successfully applied, depending on the value of the initial guess x_0: successive iterates satisfy the following relations:

$$x_i = x_{i-1} - f'(x_{i-1})^{-1} f(x_{i-1}), \qquad i = 1, 2, 3 \ldots$$

This method has already been implemented in CC1 and you can (should ?) re-use it. **In this evaluation, we want to design a simple class dedicated to the computation of such roots** for regular enough functions:

(1) define a class RootFinder with the following private members:

```
double (*sf) (double x); //  function f
double (*sfp) (double x); //  function f'
```

together with:

(a) a constructor that initializes these members,

(b) a method NewtonRaphson with the following prototype:

```
double NewtonRaphson(double x0, double epsilon, unsigned int Nmax) const;
// Newton method with Nmax iterations
```

that returns the approximate value $x^\star$ such that $f(x^\star) \approx 0$,

where x0 refers to the initial guess, epsilon is the threshold value that stops the iterations and Nmax is an integer that stands for the maximum number of allowed iterations before ending the algorithm with a failure message. Hence, your initial algorithm should be slightly modified to account for the maximum number of iterations Nmax.

Your class should be placed in files class_RootFinder.hpp and class_RootFinder.cpp.

(2) compile and link your class with the provided main1.cpp program,

(3) If f is not derivable, then we loosely have to consider a simpler method, like for instance the *bissection* method.

[Here is a brief summary of the bissection algorithm: assuming that f is continuous on $[a, b]$ and $f(a)f(b) < 0$, first computes the middle point $c = (a + b)/2$ and then tests if $f(a)f(c) < 0$. If it is true, then f must have a root in the smaller interval $[a, c]$. If it is false, then f must have a root in $[c, b]$. In either case, rename the smaller interval as $[a, b]$, which contains a root but whose size is reduced by half. Repeat this process until $|f(c)|$ is small enough: the middle point c is taken as an approximate root of f.]

This algorithm is provided as a free function in file bissection.cpp. You do not have to implement it, just use it to answer the following questions.

Add a new method in your class, called Bissection, with the following prototype:

```
double Bissection(double a, double b, double epsilon, int Nmax) const;
```

that returns the approximate solution values $x^\star$ computed with the bissection method, and such that $f(x^\star) \approx 0$.

(4) compile and link your class with the provided main2.cpp program.

Fin du sujet.