Queens College, CUNY, Department of Computer Science

Numerical Methods

CSCI 361 / 761

Spring 2018

Instructor: Dr. Sateesh Mane

3 Homework set 3

- Please email your solution, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
- Please submit one zip archive with all your files in it.
  1. The zip archive should have either of the names (CS361 or CS761):

     `StudentId_first_last_CS361_hw3.zip`

     `StudentId_first_last_CS761_hw3.zip`
  2. The archive should contain one "text file" named "hw3.[txt/docx/pdf]" and one cpp file per question named "Q1.cpp" and "Q2.cpp" etc.
  3. Note that not all homework assignments may require a text file.
  4. Note that not all questions may require a cpp file.

3.1 Secant method

- The secant method is simply the Newton–Raphson algorithm but we replace the function derivative $f'(x)$ by a numerical approximation

$$f'(x_i) \simeq \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} . \tag{3.1}$$

- Hence we require *two* initial iterates x_0 and x_1 to get started.
- The function signature for the secant function is very similar to Newton–Raphson:

```
int root_secant(double target, double tol_f, double tol_x, int max_iter,
                double x0, double x1,
                double & x, int & num_iter);
```

- The inputs contain two initial iterates x_0 and x_1.
- **Everything else is almost the same as Newton–Raphson.**

The following pseudocode describes the steps. You can use it as the basis to write a working function.

1. Declare some useful variables at the start of the function (same as NR).

```
const double tol_fprime = 1.0e-12;
double f = 0;
double fprime = 0;
```

2. We need variables to hold data for the "previous iteration" x_{i-1} and $f(x_{i-1})$ in eq. (3.1).
   (a) At the start, the "previous iterate" is x_0, so declare and initialize a variable `x_prev`

   ```
   double x_prev = x0;
   ```

   (b) Then obviously we need a variable `f_prev` for $f(x_{i-1})$. At the start this is $f(x_0)$.

   ```
   double f_prev = func(x0);
   ```

3. The obvious validation test is to check if $f(x_0)$ is already good enough to meet the tolerance.
4. If `std::abs(f_prev - target) <= tol_f`, set $x = x_0$ and return 0 (= success) and exit.
5. Otherwise we must iterate.
6. The "initial iterate" is x_1, **so set `x = x1`** and begin the main iteration loop.

```
x = x1;
for (num_iter = 1; num_iter < max_iter; ++num_iter) {
   // (to be filled in below)
}
```

7. In the loop, *we require only* $f(x)$, so we call `func(double x)`, same as for bisection.

```
f = func(x);
```

8. As before, calculate `double diff_f = f - target`.

9. If `|diff_f|` $\leq$ `tol_f`, *then we are done.*
   The value of `f` is within the tolerance.
   We have found a "good enough" value for x.
   Hence "return 0" (= success) and exit.

10. **Secant method:**

    (a) Calculate the numerical derivative as follows.

    ```
    fprime = (f - f_prev)/(x - x_prev);
    ```

    (b) **This is the key difference between secant and Newton–Raphson.**

    (c) We require a "division by zero" check. Test if `|f_prime|` $\leq$ `tol_fprime`.

    (d) If yes, this will cause a division by zero and ***the secant method fails.***

    (e) Set `x = 0` and **"return 1" (= fail)** and exit.

11. Calculate `double delta_x = diff_f/fprime` (same as NR).

12. Now test for convergence in x (same as NR).

13. If `|delta_x|` $\leq$ `tol_x`, *then the algorithm has converged (up to the tolerance).*
    Note that the **tolerance is `tol_x`** in this test (convergence in x).
    Hence "return 0" (= success) and exit.

14. If the convergence tests have not passed, then we must do *two* things.

    (a) **Update the values of `x_prev` and `f_prev`. *This is important.***

    ```
    x_prev = x;
    f_prev = f;
    ```

    (b) *After the above,* we update the value of x (same as NR).

    ```
    x -= delta_x;
    ```

15. Continue with the iteration loop.

16. If we exit the iteration loop after `max_iter` steps and the calculation still has not converged, then set $x = 0$ and `num_iter = max_iter` and "return 1" (= fail) and exit.

17. We have reached the end of the function. By now either we have a "good enough" answer ("return 0" = success) or not ("return 1" = fail).

18. **Testing:** Use the same tests employed for bisection and Newton–Raphson. The answer should be the same in all cases (up to tolerance). Also print the number of iterations. The secant method usually requires a few more iterations than Newton–Raphson.

3.2 Relative tolerances

- Up to now we have implemented only absolute tolerances for f and x.
- Suppose we also have relative tolerance parameters `rel_tol_f` and `rel_tol_x`.
    1. The relative tolerance test for f is
    $$\frac{|f - f_{\text{target}}|}{|f_{\text{target}}|} \leq \texttt{rel\_tol\_f}\,. \tag{3.1}$$
    2. To avoid a divide by zero weakness, we reformulate the test as
    $$|f - f_{\text{target}}| \leq \texttt{rel\_tol\_f} * |f_{\text{target}}|\,. \tag{3.2}$$
- For x there are different tests for bisection, Newton–Raphson and secant.
- **Bisection**
    1. The relative tolerance test for x is
    $$\frac{|x_{\text{high}} - x_{\text{low}}|}{|x|} \leq \texttt{rel\_tol\_x}\,. \tag{3.3}$$
    2. To avoid a divide by zero weakness, we reformulate the test as
    $$|x_{\text{high}} - x_{\text{low}}| \leq \texttt{rel\_tol\_x} * |x|\,. \tag{3.4}$$
- **Newton–Raphson and secant**
    1. The relative tolerance test for x is
    $$\frac{|\delta x|}{|x|} \leq \texttt{rel\_tol\_x}\,. \tag{3.5}$$
    2. To avoid a divide by zero weakness, we reformulate the test as
    $$|\delta x| \leq \texttt{rel\_tol\_x} * |x|\,. \tag{3.6}$$

3.3 Absolute & relative tolerances

- The overall set of tests is (I suppose I should write "`abs`" for absolute tolerance)

$$
\begin{aligned}
|f - f_{\rm target}| &\le \texttt{tol\_f}\,, && (3.7a)\\
|f - f_{\rm target}| &\le \texttt{rel\_tol\_f} * |f_{\rm target}|\,, && (3.7b)\\
|x_{\rm high} - x_{\rm low}| &\le \texttt{tol\_x}\,, && (3.7c)\\
|x_{\rm high} - x_{\rm low}| &\le \texttt{rel\_tol\_x} * |x|\,, && (3.7d)\\
|\delta x| &\le \texttt{tol\_x}\,, && (3.7e)\\
|\delta x| &\le \texttt{rel\_tol\_x} * |x|\,. && (3.7f)
\end{aligned}
$$

- If any of the tests pass, we say "converged" and return with a success status.
- This is an "either/or" implementation: if either the absolute or relative tolerance tests pass, we declare success.
- We can customize our tolerance tests.
  1. If we set the relative tolerances to zero, they will never pass (except by accident), so our convergence tests will be based on the absolute tolerance.
  2. If we set the absolute tolerances to zero, they will never pass (except by accident), so our convergence tests will be based on the relative tolerance.
  3. We can employ absolute tolerance for x and relative tolerance for f, etc.
- **What if we want *both* the absolute and relative conditions to be met?**
  1. We would require a logical AND for the tests:

     ```
     if ((absolute tolerance) && (relative tolerance)) then (success)
     ```
  2. This is a stricter condition and a different way of doing things.
  3. In this course, we shall not demand both an absolute and relative tolerance.

Class library

Does it occur to any of you that we could package all of these root finding functions into a "RootFinder" class library?

- All the tolerance parameters and "`max_iter`" are common to all the algorithms.
- I also added two "relative tolerance" parameters.
- I also included two function pointers, with the appropriate function signatures.

```
class RootFinder {
public:
  RootFinder() : tol_f(0.0), rel_tol_f(0.0), tol_x(0.0), rel_tol_x(0.0), max_iter(0) {}
  ~RootFinder() {}

  int root_bisection(double target, double x_low, double x_high,
                     double & x, int & num_iter) const;
  int root_NR(double target, double x0, double & x, int & num_iter) const;
  int root_secant(double target, double x0, double x1, double & x, int & num_iter) const;

  // data
  double tol_f;
  double rel_tol_f;
  double tol_x;
  double rel_tol_x;
  int max_iter;

  // function pointers
  double (*func1)(double x);
  void (*func2)(double x, double &f, double &fprime);
};
```

Do you think you could write it?