**Math321 Numerical Analysis**

**Tutorial 1, related to Newton's method.**

Please enter the following Maple program, which is a sample implementation of Newton's method, and then perform the experiments as requested, and answer the questions asked. I find it convenient to create the Maple program with Notepad, and save it on my floppy disk, as *newton.txt* for example, and then to execute it by opening Maple and then typing:

restart;

read "A:/newton.txt";

Most of the statements in my program are terminated with **:** so if you want to see the effect of those particular statements, you should change the ending to a **;**

1. Run the program as entered, and note the rapid convergence of Newton's method on a well behaved function with a good guess. This is why quadratic convergence is such a desirable property of numerical algorithms.

2. Change the definition of $f$ and $fd$ for the function $(x-2)^2$, and rerun. The procedure converges, but not as quickly. How would you characterise the convergence now? Change the definition of $f$ and $fd$ again for the function $(x-2)^3$. What happens this time? Is it what you expected? What is the explanation?

3. Change the definition of $f$ and $fd$ for the function $x^2 - 10^{-41}$. Run the program and see what happens. This shows you the difference between a good and bad guess. Change the value of **xstart** to $10^{-20}$ and note the improved performance.

4. Change the definition of $f$ for the function $x^2 - 10^{41}$, and don't forget to change **xstart** to $10^{20}$. What is happening now? For the first time in this tutorial, you are seeing how roundoff errors in floating point arithmetic are affecting the progress of the calculation.

5. I hope these examples show you how careful you have to be when writing the termination conditions for terminating iterative numerical processes.

   (a) You should always impose a maximum number of iterations, to deal with the unexpected possibility that your other termination conditions are not satisfied.

   (b) You cannot rely on the difference between $x_n$and $x_{n+1}$ being exactly 0.0, or that a function $f(x_n)$ will take on exactly the value that it is supposed to, however many significant digits are used in the calculation.

   (c) Ideally, you want your procedure to behave the same way, whatever the scale of the numbers you are processing, so it is not desirable to have the different behaviours exhibited in (3) and (4) above. The problem here is caused by the test

   ```
   if (abs(num) < ftolerance)
   ```

   which ignores the possible scale of $f$. Can you suggest a better test to replace it, or could we just get rid of it altogether?

6. If a numerical procedure might converge to $x = 0$ you have to be careful that your test on the size of $h$ is meaningful, which is why there are two tests on $h$ given here.

```
# A simple template for Newton's method
f := (x) -> x^2 - 4.0;
fd := (x) -> 2.0 * x;
xstart := 1.0;
ftolerance := 1.0e-6;
htolerance := 1.0e-6;
iterations := 20;
Newton := proc(xstart, f, fd)
    local i, x, xnew, denom, num, h, finish;
    global ftolerance, htolerance, iterations;
    i := 0:
    x := xstart:
    finish := 0;
    while (finish < 1) do
        denom:= evalf(fd(x));
        if (denom <> 0.0) then
            num:= evalf(f(x)):
            printf(" x = %e, f(x) = %e\n", x, num);
            h := evalf(-num/denom):
            xnew := x + h:
            if ( abs(num) < ftolerance ) then
                if ( x <> 0.0 ) then
                    if ( abs(h/x) < htolerance ) then
                        finish := 3;
                    end if;
                elif ( abs(h) < htolerance ) then
                    finish := 3:
                end if:
            end if:
            if (finish < 1) then
                x:= xnew:
                i:= i+1:
                if (i >= iterations) then
                    finish := 2:
                end if:
            end if:
        else
            finish := 1:
        end if:
    end do:
    num := evalf(f(x)):
    if ( finish < 2 ) then
        printf("Zero derivative at x=%e, f(x)=%e\n",x, num);
    elif ( finish < 3) then
        printf("Too many iterations at x = %e, f(x) = %e\n", x, num)
    else
        printf("Convergence after %d iterations, x = %e, f(x) = %e\n", i, x, num);
    end if;
    return (x):
end proc:
x := Newton(xstart, f, fd);
```