2.003/1053J. Test case for PSET 3.

Version 1 as of September, 27 2005

What's a test case ? A test case is a series of tests that you programs should pass. Of course, it doesn't guarantee that your program is *right* in all situations, but it it very helpful to track basic mistakes.

Is this really a testcase ? Not exactly, since this is a pdf file, not a Matlab program you can execute directly. The reason is that besides doing pure testing, you may also find advice, illustrations etc..., that would be hard to incorporate into a m-file.

How do I copy the pieces of code from this test case into matlab files ? Popular pdf viewers (acroread, xpdf) have support for copy and paste, so you can just copy and paste code from this document directly into matlab.

Will I get full credit if my program passes all these tests ? Not necessarily. Those tests are more intended to help you notice things you could have missed otherwise.

Why do you mention a "version" number near the date ? I may get ideas for tests that I hadn't though of by looking at your code, during office hours, for instance. Hence, I can append them to this list and issue a new version of it. You're encouraged to look at the latest test case before you submit your pset. If you already submitted it, that a new test case release is issued afterwards, and that it makes you see a mistake, you can re-submit your code, as long as the submission period isn't closed.

Test 1 for `simpleNewtonSolver` Try your simple solver with the function `f1` we've seen in class. Here's the function:

```
function y=f1(x)
y=x-(x.^2+exp(x)-10);
```

And here's how you should invoke the solver, and the output you should get. We start with -2.7 as our initial guess, from looking at the plot, and we may want a relative accuracy of 2^{-32}, because many computers represent numbers as 32-bits floating points digits.

```
>> x0 = -2.7;
>> xsol = simpleNewtonSolver('f1',x0,x0*2^-32)

xsol =

   -2.6910
```

Let's check our result:

```
>> f1(xsol)

ans =

  -2.6450e-09
```

As a matter of fact, we're close to the solution.

Test 2 for `simpleNewtonSolver` Test your simple solver with the constant function equal to 1. You can use inline function to avoid having to put your function into a file. Notice that when doing so, you enter directly the variable `f2` (not the string `'f2'`), that contains your function as the argument of `simpleNewtonSolver`.

```
>> f2 = inline('1');
>> xsol = simpleNewtonSolver(f2,0,1e-9)
??? Error using ==> simpleNewtonSolver
The method did not converge
```

As you can see, it can be useful to use the function `error` and give the user a message, rather than never getting out of the `while` loop. Notice also that the program doesn't seem to complain that the derivative of `f2` is zero. You can also try your solver on `f2 = inline('x^2+1');` to see if it gives the same error message.

Test 1 for `my_sqrt` A simple evaluation of $\sqrt{2}$

```
>> x1 = my_sqrt(2)

x1 =

    1.4142
```

Compare to 'original' solution:

```
>> diff1 = my_sqrt(2) - sqrt(2)

diff1 =

     0
```

Test 2 for `my_sqrt` No negative arguments

```
>> x1 = my_sqrt(-2)
??? Error using ==> my_sqrt
my_sqrt: the argument needs to be positive
```