Homework Assignment: No tool is perfect.

The first objective of this assignment is to become aware that even well-written, commercial, tried-and-true software can produce wrong answers without any warning signs. The second objective is to learn how to recover once you understand what is going on. The moral of the homework is "Never use a numerical method until you understand how it works. No black boxes".

1 Definitions

Let's call function $f(x)$ *nice* if it is defined everywhere on [-1, 1], $-10 < f(x) < 10$, the function is infinitely differentiable on [-1, 1], and has a single minimum on (-1, 1) (trivial case of extrema at the ends are excluded). Suppose you use a numerical procedure to solve for the minimum, that is to find x^{num} for which $f(x^{num}) \to min$. We call a numerical solution *right* if $|x^{num} - x^{exact}| < 10^{-3}$, x^{exact} being the exact answer. Otherwise, the solution is called *wrong*. Note that our definition is very generous: generically, one expects the correct solution to be within $\sqrt{\epsilon}$ of the exact, that is within $\sim 10^{-7}$.

1.1 Part I. Explore. 10 pts

Use Mathematica to explore the straightforward Newton's method for finding local minimum. First, read up and thoroughly understand the method. Wiki has a surprisingly good intro article on it. Follow up with any textbook. Nice functions have only one minimum by definition, so not to worry. Use FindMinimum[]; let Mathematica select all input parameters automatically, except the method "Newton", which you specify explicitly. You may use the template provided. Explore a nice function $f(x) = ax^2 + bx^4$, consider limiting cases such as $a = 0$, $b = 0$, and some intermediates. Present convergence graphs (use FindMinimumPlot[]). Make your conclusions.

1.1.1 Solution sketch

Start with $b = 0, a = 1$, notice how you get the exact solution in one step. This is because Newton's is exact for quadratic function. For nearly quadratic functions, it converges fast (quadratically) to the exact solution. Now set $b = 0.01$ and notice the slowdown. In the $a = 0, b = 1$ case the convergence is even slower: this is because $f(x)$ deviates more and more from ideal quadratic, that is becoming more and more "shallow than a parabola" near its minimum, and so the iteration step $|x_{n+1} - x_n| = f'(x_n)/f''(x_n)$ is getting smaller as N grows. Generically, iterative methods stop iterating if some convergence criteria are satisfied.

e.g. $|x_{n+1} - x_n| < tol$ (with $tol \sim \sqrt{\epsilon}$) or if too many iterations n have been made. You can see how one or both of these can occur for a function that is very shallow (much more shallow than x^2) around its minimum.

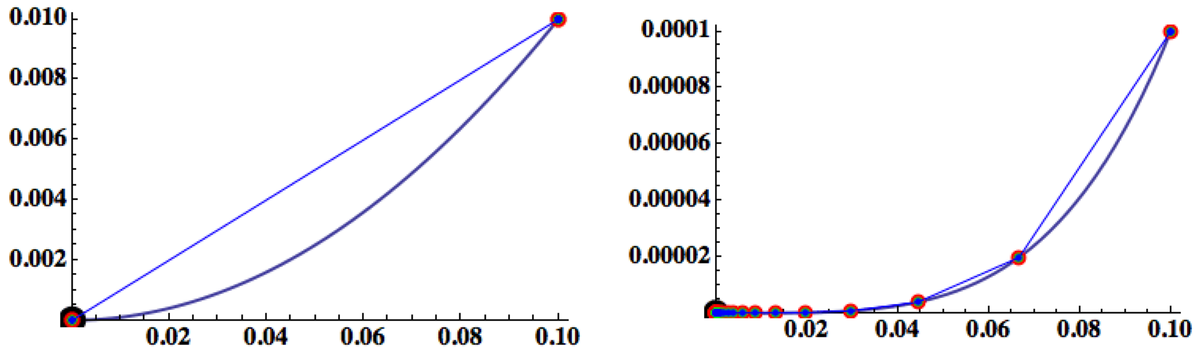

Figure 1: Convergence of Newton's minimization method. Each iteration step is shown by a circle. **Left:** Single step convergence for $f(x) = x^2$. **Right:** Much slower convergence for $f(x) = x^4$.

1.2 Part II. Break. 20 pts

Now that you understand how Newton's method works, show it! **Come up with a nice function that breaks it,** that is Mathematica gives a wrong solution (see above defs.) without so much as a peep (no warnings or errors). Present convergence graphs (use Find-MinimumPlot[]). Explain the failure.

1.2.1 Solution sketch

You have inferred in part II that Newton's works progressively worse as the function deviates from pure parabola: the shallower the minimum, the slower the iterative process. This suggest that a very shallow minimum might break the method: a little experimenting with $f(x) = x^N$ shows that $N = 20$ does the trick (on my machine). Here is the Mathematica output in this case: Value of X that minimizes $F, xmin = -0.0240265, F(xmin) = 4.10968 * 10^{-33}$. The "solution", $x_{num} = -0.0240265$ is wrong by our definition above. Obviously, x^{20} is not a unique $f(x)$ here. Depending on yiur version of Mathematica, you may need a larger N, in the range 20-40. For example, for $N = 40$ you may get $xmin = -0.148953$, $F(xmin) = 8.3564 * 10^{-34}$.

1.3 Part III. Fix. 10 pts

See if you can harness Mathematica's unique functionality and options to make it find the right solution, using the same Newton's. In fact, you may be able to get to it within $\sqrt{\epsilon}$ (Find what machine epsilon is for your machine. Use code on the class site). Explain why the solution, while useful in research, is not very useful in situations where you need to quickly find lots of minima as part of a larger code written in a standard language such as C.

1.3.1 Solution sketch

By now you know what is happening: Newton's converging too slowly for x^{20} near the exact minimum $x = 0$. So, a straightforward, albeit not very effective, remedy is to up the number of iterations. However, setting $MaxIterations \rightarrow 10000$ gives the same wrong answer "Value of X that minimizes F, xmin = -0.0240265".

The problem is that beyond a certain number of iterations, x_{n+1} and x_n become indistinguishable within the standard machine precision, and the algorithm stops. Here is where the power of *Mathematica* comes in: you can set any precision for the numbers and functions you are computing. Add $WorkingPrecision \rightarrow 400$ to $FindMinimum[]$ options, and you will get the position of the minimum to within 10^{-199} from the exact answer $x_{exact} = 0$.

(On a different version fo Mathematica with $N = 40$ you may get $xmin = 10^{-113}$ with the above settings.)

This is an overkill, but makes the point. The iterations will take appreciably longer though. Implementing an arbitrary precision arithmetic in your regular code is cumbersome, and the calculations will take a long time. Not worth it in most case. A good solution is to recognize that the problem is "ill conditioned" in some sense and solve it in a different way. Sometime more primitive but robust approaches help. But, generally, ill-conditioned problems require special, problem-specific treatment. This is where Mathematica excells: as a research tool.