Homework Assignment - No tool is perfect. Solutions.

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). For example, $f(x) = x^2$ is a nice function, but $sin(1/x^2)$ or x^3 are not. Suppose you use a numerical procedure to find the minimum x^{approx} of a nice function. We call such numerical solution *right* if $|x^{approx} - x^{exact}| < 10^{-3}$, 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 $\sim 10^{-7}$.

0.1 Part I. Explore. 10 pts

Use *Mathematica* to explore the straightforward Newton's method for finding local minimum. 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 (see examples on the class site, you may use any of them as a template). 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.

0.1.1 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 becoming very small, which slows down the iterations. 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.

0.2 Part II. Break. 20 pts

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

0.2.1 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

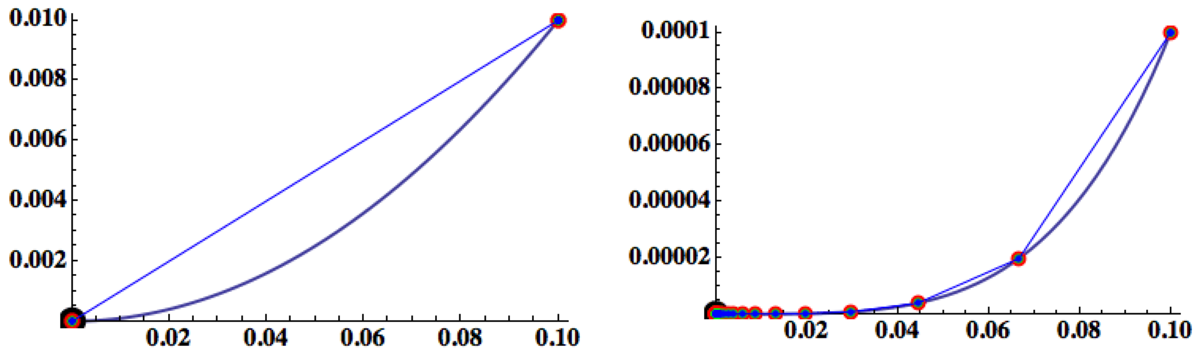

Figure 1: Convergence of Newton's minimization method. Each iteration step is shown by a circle. **Left:** Single step convergence for $f(x) = x^2$. **Right:** Much slower convergence for $f(x) = x^4$.

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. 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.

0.3 Part III. Fix. 10 pts

Reason which method from your C&K textbook may mitigate the problem. Give an intuitive explanation for why the alternative to Newton's works; in a few sentences, give at least one pro and one con for your alternative.

0.3.1 Solution sketch

What you have learned is that performance of Newton's method is great for nearly quadratic functions, but quickly deteriorates for those that are shallower near the minimum. One reason is that the method depends on derivatives that $\to 0$ in the latter case. So you might try your luck with a more "primitive" method that does not use derivatives: it may not be as fast in the best case scenario ($f(x) = x^2$), but may be more robust in the worst case. The golden section method is an obvious choice: the only requirement is that the function is unimodal – nice functions are (see *e.g.* the appropriate chapter in the C & K book). The obvious pro of the golden section is its robustness: it will work for just about any unimodal function, not just nice differentiable ones. The obvious con is that the method converges slowly to the solution (linearly). Brent method (also available in GSL) is another option along these lines: it is a bit more complicated than the Golden section, but might be faster.

0.4 Part IV. Fix again. 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 when you need to quickly find lots of minima as part of a larger code written in a standard language such as C.

0.4.1 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 $MaxIterations \to 10000$ 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 $WorkingPrecision \to 400$ to $FindMinimum[]$ options, and you will get the position of the minimum to within 10^{-199} from the exact answer $x_{exact} = 0$. 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, and even then, accuracy may be poor. This is where Mathematica excells: as a research tool.