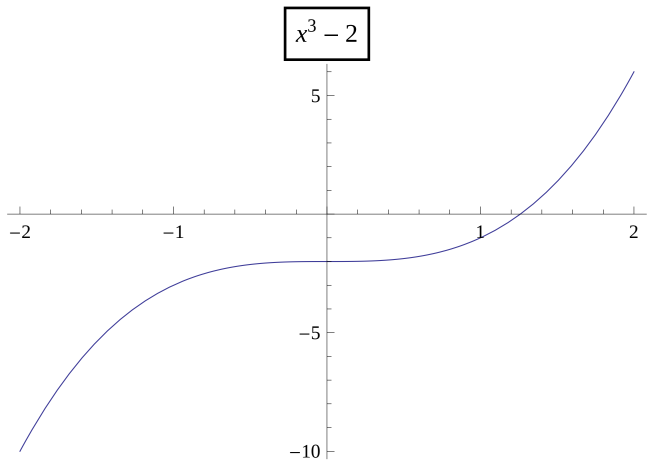
INTRODUCTION TO QUASI-NEWTON METHODS

ALEX SCHIFF

Abstract. Newton's method is a way to find roots of equations. It can also be used to find critical points, local minimums, maximums, or saddle-points, of functions if it is performed on the function's first derivative. This can be generalized to finding critical points of functions with multiple variables, that map $\mathbb{R}^n$ to R. In this paper, we will carefully define Newton's method, along with several other methods and illustrate them with some well chosen examples. In higher dimensions, there are numerous methods for approximating roots called Quasi-Newton methods. Many of these methods are modifications of Broyden's method, which is explained in this paper. We will also address the drawbacks of Newton's method and the other methods discussed.

Contents

Date: July 30, 2010.



1. Introduction

We are interested in developing techniques for approximating critical points for functions that can not be solved analytically easily. The examples shown in this paper were carefully selected such that their critical points can be found by hand. We will begin by using Newton's method to find roots of a function in $\mathbb{R}$, and then the secant method. We will then move on to another example in $\mathbb{R}^2$.

2. Example in $\mathbb{R}^1$

We will examine a fairly trivial function in $\mathbb{R}^1$ to illustrate Newton's method and the secant method.

2.1. **Function definition.**

Let $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) = x^3 - 2$. Below is a graph of the function:

2.1.1. *Finding roots.*

The only real root of this function, $2^{1/3}$, is trivially found. By selecting such a simple example, we can understand what each root-finding algorithm does step by step.

3. Root finding algorithms in $\mathbb{R}^1$

In this particular example, we are going to use two algorithms to try and approximate the root we seek.

3.1. **Newton's method.**

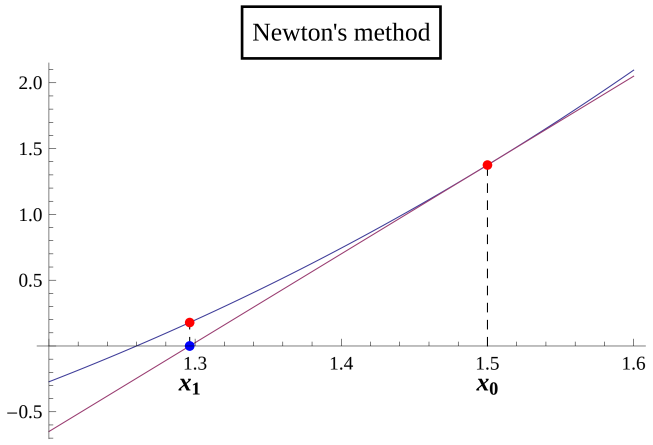
Newton's method is an algorithm for finding zeros through numerous iterations. If certain conditions are satisfied, it can converge to the true value of the root. It starts off with a guess x_0, and finds the next guess through this algorithm:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

3.1.1. *Sample calculation of Newton's method.* Let $x_0 = 1.5$, and we will continue iterating until $\frac{f(x_n)}{f'(x_n)} < 10^{-12}$.

$$\begin{aligned}
x_1 &= 1.5 - \frac{1.375}{6.75} \\
x_2 &= 1.2963 - \frac{.178276}{5.04115} \\
x_3 &= 1.26093 - \frac{.00481929}{4.76985} \\
x_4 &= 1.25992 - \frac{3.86058 \times 10^{-6}}{4.76221}
\end{aligned}$$

The points are plotted along side of the original graph below:



Two points that do not work in this method are $x_0 = -1$ and $x_0 = 0$. These are just two examples of bad points that do not converge.

3.2. **Secant method.** The secant method is another root-finding algorithm that requires two initial guesses instead of one. This method does not always converge, but it can be a better choice over Newton's method if calculating the derivative of f is more time consuming than computing two values of f at each step. Below is the algorithm:

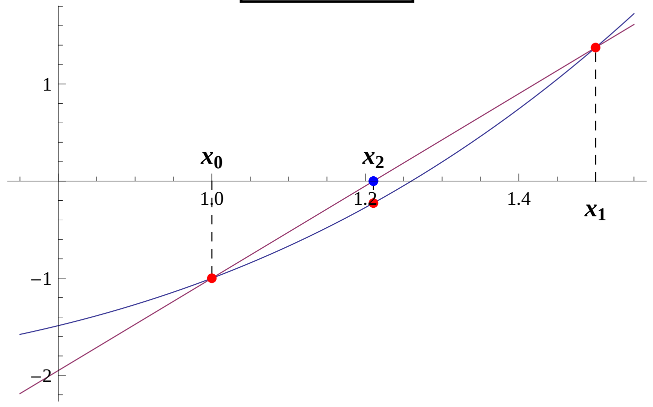
$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

3.2.1. *Sample calculation of the secant method.* We will now show a sample calculation using starting values of $x_0 = 1$ and $x_1 = 1.5$:

$$\begin{aligned} x_2 &= 1.5 - 1.375(0.210526) \\ x_3 &= 1.21053 + 0.226126(0.180794) \\ x_4 &= 1.25141 + 0.040265(0.219961) \\ x_5 &= 1.26027 - 0.00163972(0.211354) \\ x_6 &= 1.25992 + 0.0000111236(0.20993) \\ x_7 &= 1.25992 + 3.03855 \times 10^{-9}(0.209987) \end{aligned}$$

The points are plotted along side of the original graph below:

Secant method



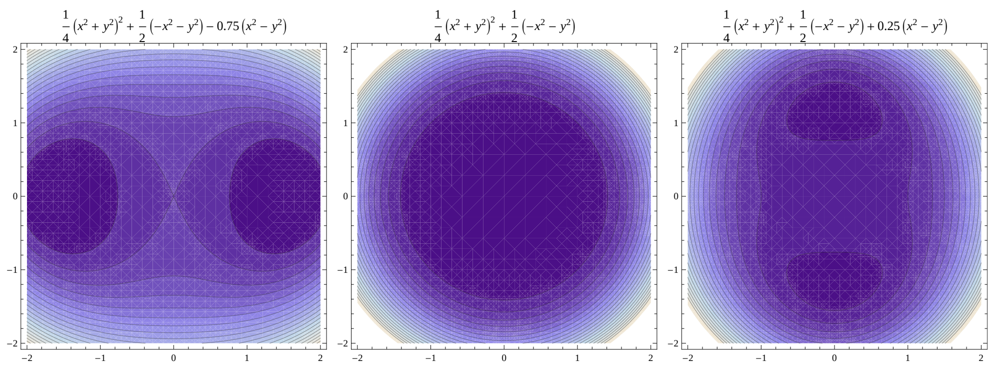
4. Example in $\mathbb{R}^2$

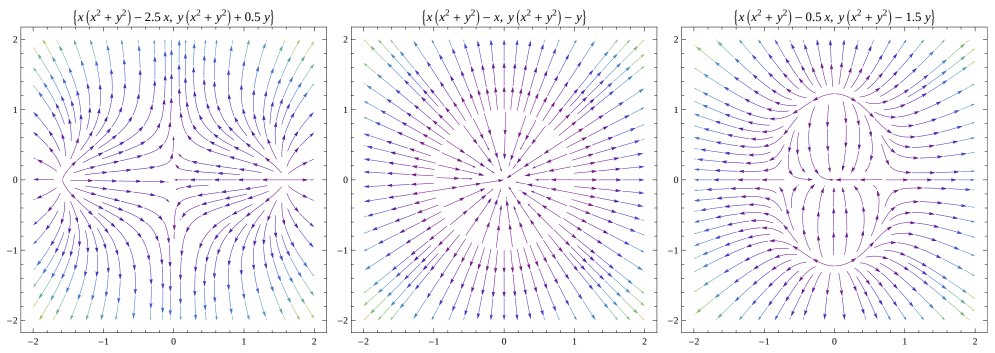
In this section, we are going to focus on a function that maps $\mathbb{R}^2$ to $\mathbb{R}$. This function has a parameter α that can be varied. We are going to find the critical points of f, using its gradient. The reason we choose this function is because different values of α produce a different number of critical points and this is interesting, also because we can easily find the critical points in this function, allowing us to measure the success of the algorithms.

4.1. **Definitions and graphs.** We will define the function f, the gradient g, and the Jacobian matrix H in this section. H is needed for the algorithms later on, but not for finding the critical points analytically.

4.1.1. *Definition of f.* Let $f : \mathbb{R}^2 \to \mathbb{R}$ with $x, y, \alpha \in \mathbb{R}$ such that:

$$f(x,y) = -\frac{x^2+y^2}{2} + \left(\frac{x^2+y^2}{2}\right)^2 + \frac{\alpha(x^2-y^2)}{2}$$





4.1.2. *Graphs of f.* Here are some contour plots of f with different values of α. Notice the symmetry of the graph when $\alpha = 0$, and when $\alpha = -1.5$, there are only two saddle points.

4.1.3. *Definition of g.* Since we are trying to find the critical points of the function f, we need to find the gradient, $\nabla f = g$, and set both components equal to zero. Let $g : \mathbb{R}^2 \to \mathbb{R}^2$ with $x, y, \alpha \in \mathbb{R}$ such that:

$$g(x, y) = \nabla f = (x^3 + xy^2 + \alpha x - x, y^3 + x^2 y - \alpha y - y)$$

4.1.4. *Graphs of g.* Here are some stream plots of g with different values of α:

4.1.5. *Definition of H.* In this example the Jacobian matrix of g is also the Hessian matrix of f. Let H be the Jacobian matrix of g:

$$H = \begin{bmatrix} 3x^2 + y^2 + \alpha - 1 & 2xy \\ 2xy & 3y^2 + x^2 - \alpha - 1 \end{bmatrix}$$

4.2. **Finding critical points analytically.** To find the critical points of f, we need to set the gradient $g = 0$ and solve the system of equations:

$$\begin{aligned} x^3 + xy^2 + \alpha x - x &= 0 \\ y^3 + x^2 y - \alpha y - y &= 0 \end{aligned}$$

To find these points, we will first set both $(x, y) = (0, 0)$, then just $y = 0$, then just $x = 0$, and finally the last case when neither x nor y equal zero.

4.2.1. *Case when* $(x, y) = (0, 0)$. When both x and y equal zero, both components of g equal zero, so the origin is a critical point for any value of α. This can be seen in the graphs later on.

4.2.2. *Case when* $y = 0$. If $y = 0$, the first component of g becomes

$$\begin{aligned} x^3 + \alpha x - x &= 0 \\ x^2 + \alpha - 1 &= 0 \\ x &= \pm\sqrt{1 - \alpha} \end{aligned}$$

So when $\alpha < 1$, there are two critical points at $(\pm\sqrt{1-\alpha}, 0)$.

4.2.3. *Case when* $x = 0$. If $x = 0$, the second component of g becomes

$$\begin{aligned} y^3 - \alpha y - y &= 0 \\ y^2 - \alpha - 1 &= 0 \\ y &= \pm\sqrt{1 + \alpha} \end{aligned}$$

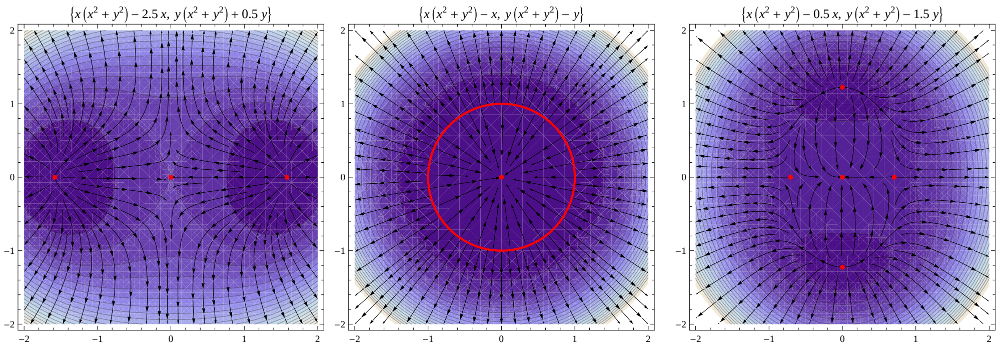
So when $\alpha > -1$, there are two critical points at $(0, \pm\sqrt{1+\alpha})$.

4.2.4. *Case when* $(x, y) \neq (0, 0)$. If $(x, y) \neq (0, 0)$, the components of g become

$$\begin{aligned} x^3 + xy^2 + \alpha x - x &= 0 \qquad & y^3 + x^2 y - \alpha y - y &= 0 \\ x^2 + y^2 + \alpha - 1 &= 0 \qquad & x^2 + y^2 - \alpha - 1 &= 0 \end{aligned}$$

Thus, when $\alpha = 0$ there are infinitely many critical points along the unit circle.

4.2.5. *Example graphs.* Below are the graphs of f and g with the critical points highlighted for various values of α. Notice when $\alpha = 0$, the critical points lie on the unit circle.



5. Critical point finding algorithms in $\mathbb{R}^2$

We will utilize three methods here to approximate the critical values of f. These methods are generalizations of Newton's method and the secant method to higher dimensions; that is, they perform operations on vector functions. We will see how these algorithms work with our example.

5.1. **Newton's method in $\mathbb{R}^2$.** As we saw earlier, Newton's method in one dimension is $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. Since we are trying to approximate the zeros of the gradient g, the relation becomes $H(\mathbf{v}_n)(\mathbf{v}_{n+1} - \mathbf{v}_n) = -g(\mathbf{v}_n)$, where $\mathbf{v}_n = (x_n, y_n)$. This system of equations can either be solved by finding the inverse Jacobian H_n^{-1} at each step or by solving linearly. Either of these operations can take up quite a bit of processing time. Also, the Jacobian is not always invertible, so to be on the safe side it is better to find the pseudo inverse. If the system does not have an efficient linear solver, other methods should be considered.

5.1.1. *Example calculation.* Let $\mathbf{v}_0 = (1.3, .05)$, and $\alpha = -.5$. We find the next iteration by the formula $\mathbf{v}_{n+1} = \mathbf{v}_n - H^{-1}(\mathbf{v}_n)g(\mathbf{v}_n)$

$$\mathbf{v}_1 = \begin{bmatrix} 1.3 \\ 0.05 \end{bmatrix} - \begin{bmatrix} 0.25025 \\ 0.059625 \end{bmatrix} \begin{bmatrix} 0.281026 & -0.0305081 \\ -0.0305081 & 0.838385 \end{bmatrix}$$

$$\mathbf{v}_2 = \begin{bmatrix} 1.23149 \\ 0.00764594 \end{bmatrix} - \begin{bmatrix} 0.0204817 \\ 0.0077731 \end{bmatrix} \begin{bmatrix} 0.32793 & -0.0060738 \\ -0.0060738 & 0.98364 \end{bmatrix}$$

$$\mathbf{v}_3 = \begin{bmatrix} 1.22482 \\ 0.000124406 \end{bmatrix} - \begin{bmatrix} 0.000234094 \\ 0.00012443 \end{bmatrix} \begin{bmatrix} 0.33327 & -0.000101545 \\ -0.000101545 & 0.999809 \end{bmatrix}$$

$$\mathbf{v}_4 = \begin{bmatrix} 1.22474 \\ 2.37711 \times 10^{-8} \end{bmatrix} - \begin{bmatrix} 4.13077 \times 10^{-8} \\ 2.37711 \times 10^{-8} \end{bmatrix} \begin{bmatrix} 0.333333 & -1.9409 \times 10^{-8} \\ -1.9409 \times 10^{-8} & 1. \end{bmatrix}$$

We are left with $\mathbf{v}_4 = (1.22474, 8.01741 \times 10^{-16})$ after 4 iterations, which corresponds to the critical point $(\sqrt{1.5}, 0)$.

5.2. **Quasi-Newton methods.** The Quasi-Newton methods we examine here work by finding the inverse Jacobian at an initial point $H^{-1}(\mathbf{v}_0)$. Then the methods update the inverse at each iteration, and proceeds in the Newton direction. The methods can also be used to update the Jacobian, however we are more interested in the inverse Jacobian here. For each method we will utilize the same notation. We will first outline the method of updating the inverse, then give a sample calculation using the same initial point given earlier.

5.2.1. *Notation.* Let $\mathbf{v}_n = (x_n, y_n)$ be the current guess of the critical point. Let $\Delta\mathbf{v}_n = -H^{-1}(\mathbf{v}_n)g(\mathbf{v}_n)$, and $\mathbf{v}_{n+1} = \mathbf{v}_n + \Delta\mathbf{v}_n$. Let $\mathbf{u}_n = g(\mathbf{v}_{n+1}) - g(\mathbf{v}_n)$.

5.3. **Broyden's method.** This method was not as successful as the BFGS, but if the guess was a good one, it would converge in a reasonable amount of iterations. Broyden was the first to suggest updating the inverse Jacobian directly when attempting to approximate critical points. His formula for updating the inverse is:

$$H^{-1}(\mathbf{v}_{n+1}) = H^{-1}(\mathbf{v}_n) + \frac{(\Delta\mathbf{v}_n - H^{-1}(\mathbf{v}_n)\mathbf{u}_n)\mathbf{u}_n^T H^{-1}(\mathbf{v}_n)}{\mathbf{u}_n^T H^{-1}(\mathbf{v}_n)\Delta\mathbf{v}_n}$$

5.4. **BFGS.** BFGS stands for Broyden-Fletcher-Goldfarb-Shanno, and it does not require constraints. In writing this paper, the BFGS seemed to be the most reliable and efficient alternative to Newton's method, in that it converged often even with a bad starting point, and it required less iterations than the most of the Quasi-Newton methods. The method we will use to update the inverse of the Jacobian is:

$$H^{-1}(\mathbf{v}_{n+1}) = \left(I - \frac{\mathbf{u}_n\Delta\mathbf{v}_n^T}{\mathbf{u}_n^T\Delta\mathbf{v}_n}\right)^T H^{-1}(\mathbf{v}_n)\left(I - \frac{\mathbf{u}_n\Delta\mathbf{v}_n^T}{\mathbf{u}_n^T\Delta\mathbf{v}_n}\right) + \frac{\Delta\mathbf{v}_n\Delta\mathbf{v}_n^T}{\mathbf{u}_n^T\Delta\mathbf{v}_n}$$

After updating H^{-1}, we find the next $\mathbf{v}_{n+1}$ by using the definition given above. We continue the updating process until the number of iterations is reached or until the norm of the current gradient is less than a given tolerance.

5.5. **SR1.** SR1 stands for Symmetric Rank 1, and is named because it maintains the symmetry of the matrix, but does not guarantee the update of the matrix is positive definite. The formula for updating the inverse of the Jacobian is:

$$H^{-1}(\mathbf{v}_{n+1}) = H^{-1}(\mathbf{v}_n) + \frac{(\Delta\mathbf{v}_n - H^{-1}(\mathbf{v}_n)\mathbf{u}_n)(\Delta\mathbf{v}_n - H^{-1}(\mathbf{v}_n)\mathbf{u}_n)^T}{(\Delta\mathbf{v}_n - H^{-1}(\mathbf{v}_n)\mathbf{u}_n)^T\mathbf{u}_n}$$

5.6. **DFP.** DFP stands for Davidson-Fletcher-Powell, and like the SR1, it relies on the Jacobian being symmetric. Unlike the SR1, it does maintain the positive definiteness of the Jacobian matrix. The implementation of the DFP method used in this paper was one of the poorest performing methods; it often took the most iterations, and converged the slowest. The formula for updating the inverse of the Jacobian is:

$$H^{-1}(\mathbf{v}_{n+1}) = H^{-1}(\mathbf{v}_n) + \frac{\Delta\mathbf{v}_n \Delta\mathbf{v}_n^T}{\mathbf{u}_n^T \Delta\mathbf{v}_n} - \frac{H^{-1}(\mathbf{v}_n)\mathbf{u}_n\mathbf{u}_n^T H^{-1}(\mathbf{v}_n)^T}{\mathbf{u}_n^T H^{-1}(\mathbf{v}_n)\mathbf{u}_n}$$

5.7. **Comparison of methods.** In this table, we start out with the same initial points used in the sample calculation of Newton's method, with $\mathbf{v}_0 = (1.3, .05)$ and $\alpha = -.5$. The values being compared are the norms of $g(\mathbf{v}_n)$.

n	Newton	Broyden	BFGS	SR1	DFP
0	0.257255	0.257255	0.257255	0.257255	0.257255
1	0.0219071	0.0219071	0.0219071	0.0219071	0.0219071
2	0.000265109	0.000779062	0.00232741	0.0023282	0.00232733
3	4.76591×10^{-8}	0.0000543848	0.000054104	0.0000496025	0.0000546315
4	1.74891×10^{-15}	4.11352×10^{-6}	6.63914×10^{-4}	4.19604×10^{-6}	6.80593×10^{-6}
5	1.74891×10^{-15}	5.38751×10^{-7}	2.33609×10^{-7}	8.99118×10^{-9}	2.56817×10^{-7}
6	1.74891×10^{-15}	4.43177×10^{-9}	4.49212×10^{-9}	1.30971×10^{-13}	5.46648×10^{-9}
7	1.74891×10^{-15}	3.93248×10^{-10}	2.09223×10^{-11}	1.30971×10^{-13}	2.83271×10^{-11}
8	1.74891×10^{-15}	3.21458×10^{-11}	1.54048×10^{-13}	1.30971×10^{-13}	2.35487×10^{-13}
9	1.74891×10^{-15}	2.37028×10^{-14}	1.54048×10^{-13}	1.30971×10^{-13}	2.35487×10^{-13}

6. Conclusions and further research

There are many interesting questions that were raised during the research behind this paper. What is the effect of a asymmetric Jacobian matrix on the methods? How would one modify the methods to allow for constraints (such as Lagrangian multipliers)? Is there a way to prove under certain conditions that one method is more or less effective than another? These are questions that I will continue to address in the coming months, and try to get some more results.