CALCULUS III: PROJECT 1A

1. Approximating Minima of a Function

1.1. **Gradient Descent.** For this project we will explore the method of finding a local minimum[1] of a differentiable function called *gradient descent*. We have learned in class that if a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ attains a relative minimum at a point $p = (x_1, x_2, \dots, x_n)$, then the gradient ∇f necessarily vanishes at p. But, solving the equation $\nabla f(p) = 0$ analytically might not be possible. For example, consider the function

$$f(x, y) = x^6 + y^6 + xy + (xy)^4 + 1$$

The gradient of f is

$$\nabla f(x, y) = \langle 6x^5 + y + 4x^3y^4, 6y^5 + x + 4x^4y^3 \rangle$$

so you would have to solve the system of equations

$$\begin{aligned} 6x^5 + y + 4x^3y^4 &= 0 \\ 6y^5 + x + 4x^4y^3 &= 0 \end{aligned}$$

attempting which is not advised.

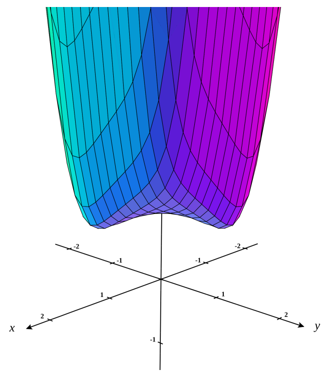
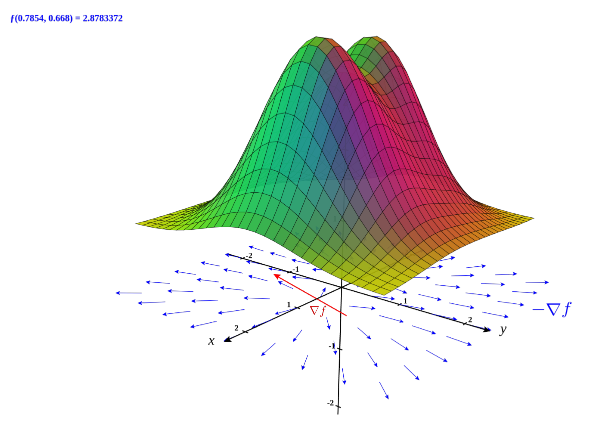


Figure 1.1. The graph of f.

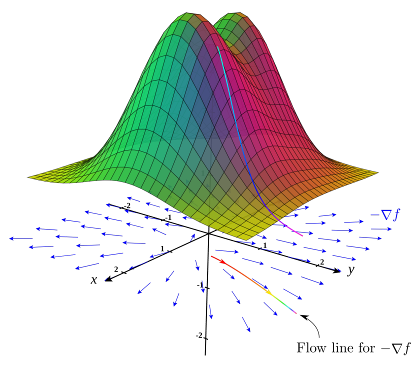
The function f does attain a minimum though, as can be seen in figure 1.1, so we would like to have a method for finding it.

Suppose you were hiking in the mountains and decided that you want to descend in the most direct way. How would you go about doing it? A natural thing to do is to keep walking in the direction of steepest descent. Recall that the gradient ∇f of a function f points in the direction the function increases at the highest rate and $-\nabla f$ points in the direction the function decreases at the highest rate. Therefore, if the function $f(x, y)$ denotes the height at the point with horizontal coordinates (x, y), the direction of steepest descent at the point with horizontal coordinates (x, y) is $-\nabla f(x, y)$. To descend the mountain we can follow the path whose horizontal projection is always tangent to $-\nabla f$ as in figure 1.2*b*

[1]Finding a local maximum is completely analogous, but since the method is called gradient *descent*, we will stick to finding local minima.



(A) The gradient of f at one point and the vector field $-\nabla f$.



(B) The flow line for $-\nabla f$ and the corresponding path down the mountain.

We can apply the same reasoning to find the local minima of our first example. We start at a random point (x, y) and we find the flow line for the vector field $-\nabla f$ as in figure 1.3.[2] Recall that a flow line for a vector field $\boldsymbol{F}$ is a path $c(t)$ such that for all t, $c'(t) = \boldsymbol{F}(c(t))$. In other words, the velocity vector of the path at each point $c(t)$ equals the value of the vector field at that point.

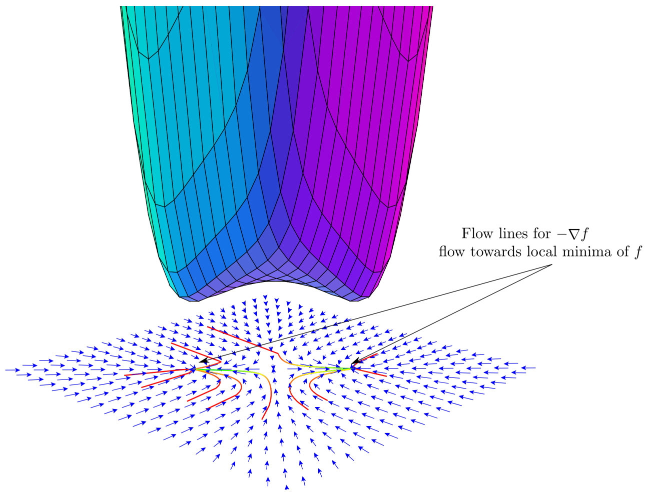
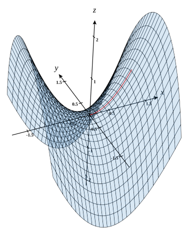


FIGURE 1.3. The flow lines flow to the local minima. In this case, there are two local minima and depending on your starting point you either end up at one or the other. (All vectors are drawn with the same length for ease of viewing)

Viewing the flow lines in the $x - y$ plane, we can estimate the location of the local minima to be approximately at $(0.59, -0.59)$ and $(-0.59, 0.59)$.

[2]It might happen that a flow line ends up at a saddle point instead of a local minimum. For example, consider the function $f(x, y) = x^2 - y^2$ and the flow line starting at $(1, 0)$. This will not happen for a



generic starting point, so we will not consider it in this project.

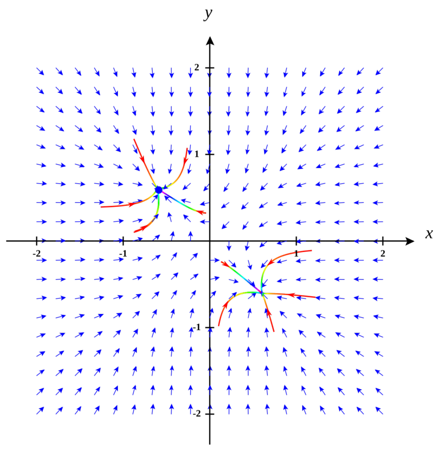


FIGURE 1.4. Flow lines for the vector field $-\nabla f$ where $f = x^6 + y^6 + xy + (xy)^4 + 1$

Task 1. For this task, you will find a local minimum of a function f by using a graphing software in order to plot the vector field $-\nabla f$. You will then trace out several flow lines for the vector field **by hand**.

Choose one function from the following list:

(1) $f(x,y) = x^4 + 2y^4 - 3x^2y + y$
(2) $f(x,y) = \frac{(x^2+7xy)}{e^{x^2+y^2}}$
(3) $f(x,y) = \frac{x^2+y^2}{x^2+1}$
(4) $f(x,y) = \cos(xy) + \frac{x^2+y^2}{2}$

Things to keep in mind:

(1) Rescaling the vector field by a constant will not change the geometric shape of the flow lines but can make viewing the plot easier.
(2) Using fixed length vectors in the plot can make it even easier.
(3) Make sure to adjust the x and y bound of the plot in order to include pertinent features.

Software: You may use any software of your choice. Here are some suggestions:

(1) CalcPlot3D: Probably the easiest to use.
(2) SageMath: You will need to use Sage for the last section of the project, so you might as well use it for graphing the vector field in this section.
(3) FieldPlay: Not for graphing a vector field, but for seeing flow lines.
(4) 3D-XplorMath: Can be used to graph flow lines.

1.2. **Euler's Method.** We have rephrased the problem of finding a local minimum of a function f in terms of finding the flow line for $-\nabla f$, but we haven't yet discussed how to find the flow line in a practical manner. For this project we will pursue a numerical method, called the Euler's Method, in order to find an approximate solution.

Let $\boldsymbol{F}(x,y)$ be a vector field in $\mathbb{R}^2$ and suppose we would like to approximate the flow line $c(t)$ for $0 \leq t \leq 1$ such that $c(0) = (x_0, y_0)$. For c to be the flow line of $\boldsymbol{F}$, we must have $c'(0) = \boldsymbol{F}(c(0))$, i.e., we know the instantaneous velocity of c at $t = 0$. If c was moving in a straight line, we would have $c(t) = c(0) + tc'(0)$, which is the linear approximation of c at $t = 0$. For t small, this is still a good approximation of $c(t)$, which is the approximation we will use.

$$c(t) \approx c(0) + tc'(0) = c(0) + t\boldsymbol{F}(c(0))$$

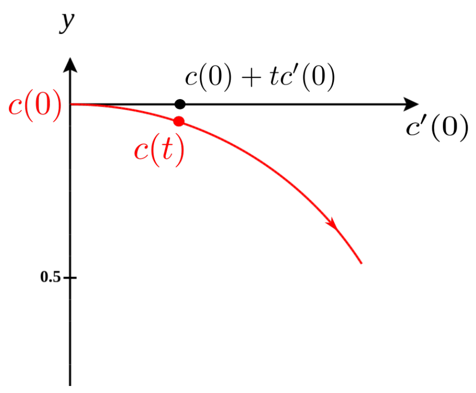


FIGURE 1.5. Linear approximation to a path $c(t)$.

Fixing a step size Δt, we can approximate the value of $c(\Delta t)$ by $c(0) + \Delta t\boldsymbol{F}(c(0))$ and then repeat the process starting with $c(\Delta t)$ instead of $c(0)$. The algorithm then becomes the following:

(1) Fix a step size Δt.
(2) Set $c_0 = c(0) = (x_0, y_0)$.
(3) Compute

$$\begin{aligned} c_1 &= c_0 + \Delta t\boldsymbol{F}(c_0), \\ c_2 &= c_1 + \Delta t\boldsymbol{F}(c_1), \\ \vdots &= \vdots \\ c_n &= c_{n-1} + \Delta t\boldsymbol{F}(c_{n-1}) \end{aligned}$$

(4) The result is the polygonal line with vertices $c_0, \ldots, c_n$.

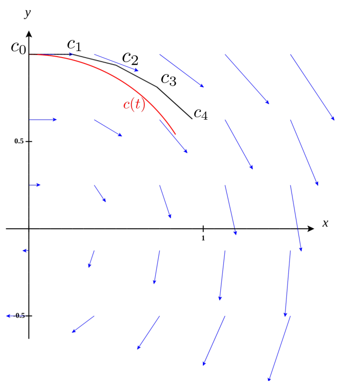


FIGURE 1.6. Example with $\boldsymbol{F} = \langle y, -x\rangle$, $(x_0, y_0) = (0, 1)$, and $\Delta t = \frac{1}{4}$

Note that the linear approximation gets better as Δt gets smaller. But, making the step size too small has the drawback that you need to perform more steps.

Task 2. Pick the following objects of your choice:

(1) A two-dimensional vector field $\boldsymbol{F}$.
(2) A starting point (x_0, y_0).

(3) Step size Δt.

Then, use the Euler method to find the approximate flow line $c(t)$ of $\boldsymbol{F}$ with $c(0) = (x_0, y_0)$. Compute at least 5 steps of the Euler process. Then, plot the vector field and the corresponding polygonal line (either by hand or using a software). One condition that your vector field has to satisfy is that the points c_i have to not lie on a line.

1.3. **Imprecisions.** Figure 1.6 shows that the solution obtained by Euler's method, being only an approximation, can diverge from the actual flow line over time. This might seem to be a problem if we are trying to find a local minimum of a function f by approximating the flow of $-\nabla f$. It turns out to not be a problem: the Euler's method approximation will converge to the same minimal value of f even if along the way it diverges from the actual flow line. [3]

One way to understand why that is the case is the following. Applying the Euler's method to $-\nabla f$, we get a sequence of points $c_0, c_1, c_2, \dots, c_n$. Each subsequent point is obtained by the formula $c_{i+1} = c_i + \Delta t(-\nabla f(c_i))$, i.e., c_{i+1} is a small distance away from c_i in the directions in which f is decreasing. Therefore, as long as Δt is small enough, we have $f(c_{i+1}) < f(c_i)$. In other words, we get a sequence of points with decreasing values of f. Eventually, we will approach the local minimum of f. Another thing to note is that as c_i approaches a local minimum, $-\nabla f(c_i)$ approaches 0, and therefore $c_{i+1} - c_i$ approaches 0. In other words, once we are close enough to the local minimum, each step of the Euler's method only moves the point c_i by a very small amount.

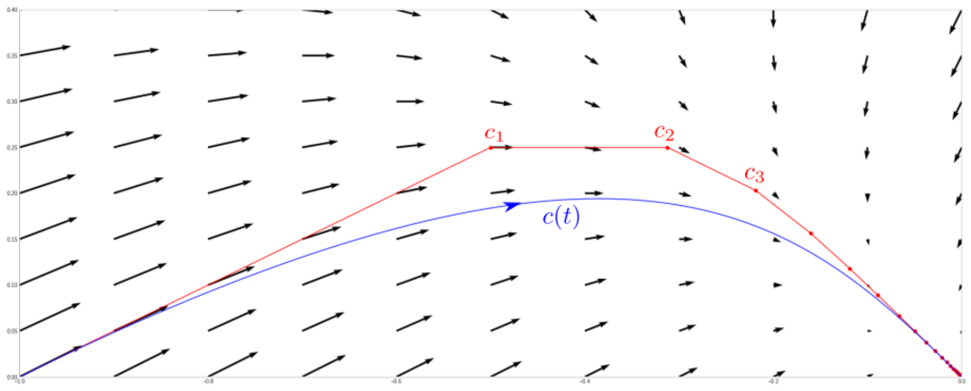


FIGURE 1.7. The flow line and the Euler's method approximation with $\Delta t = \frac{1}{4}$ for $f(x, y) = x^2 + y^2 + xy$ starting at $(-1, 0)$

[3]There are several caveats to this statement. The step size needs to be small enough and it might happen that the approximate flow line will converge to a different local minima than the original flow lines.

1.4. **Step size.** If the step size is too small, then it would take too many iterations for the algorithm to converge to a local minimum. On the other hand, if the step size is too large, then each step of the algorithm might not bring you closer to the minimum, or you can overshoot it.

Figure 1.8 shows the same vector field and starting point as Figure 1.7, but with $\Delta t = 0.65$ and $\Delta t = 0.0001$. In the former case, the algorithm overshoots, but does eventually converge on the minimum. Figure 1.9 shows the algorithm with $\Delta t = .675$. In this case, each subsequent iteration overshoots the minimum more than the previous one, so the process never converges to the minimum. The appropriate choice of Δt is problem dependent. You might need to try various values for Δt when working on computations in section 2.

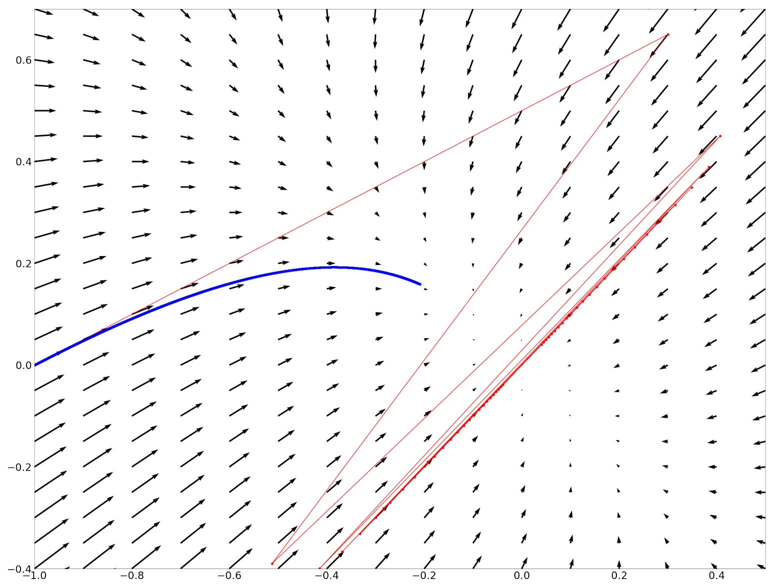


FIGURE 1.8. The blue graph shows 10000 iterations with $\Delta t = 0.0001$. The red graph shows 50 iterations with $\Delta t = 0.65$.

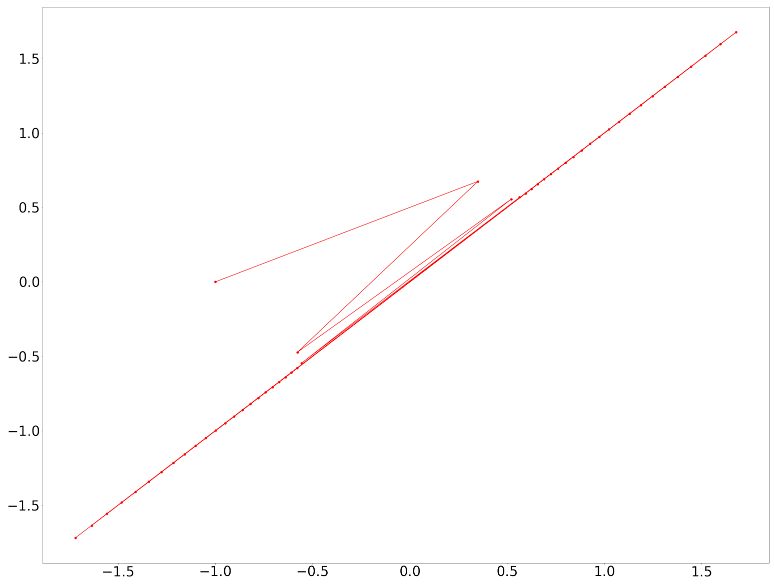


FIGURE 1.9. The graph shows 50 iterations with $\Delta t = 0.675$.

2. Computing with SageMath

In this section you will use a software system in order to compute a minimum of a function. We will use SageMath, which is based on the Python programing language, but is intended for mathematical computations. You will not need to write your own code, but you will need to vary some of the parameters (mostly the step size), in order for the algorithm to succeed.

The most common way to interface with Sage is with Jupyter notebook environment in the browser. You have several options for using Sage:

(1) https://cocalc.com : Has an indefinite free tier (for now). Once you run it in the browser, you'll be able to upload files. If you create new files, make sure you choose Jupyter notebook and Sagemath as the kernel.
(2) https://mybinder.org/v2/gh/sagemath/sage-binder-env/master?filepath=index.ipynb : Ditto
(3) You can also install and run a local copy of SageMath+Jupyter. Instructions can be found on their webpage https://www.sagemath.org/.

Basic starting guides for Sage and Jupyter can be found here and here, but you will mostly like not need them.

The easiest way to start using Sage is to have a functioning code in front of you. For that, download the file graphing.ipynb containing the code for graphing two dimensional vector fields and flow lines. Always choose SageMath as the kernel if prompted.

Task 3. The sage notebook gradient_descent.ipynb has code for computing gradient descent for a function $f(x, y)$ of two variables. Pick a function from Task 1, different from the one you used in Task 1, and use the Sage notebook to find a local minimum of f. Include in your project submission the plot you will obtain in the last cell of the sage notebook with your project submission.

For the last task, we will apply gradient descent to the machine learning problem of finding the best curve fitting a particular set of data points $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ as in Figure 2.2. Suppose we want to find the best degree 2 polynomial $f(x) = a + bx + cx^2$

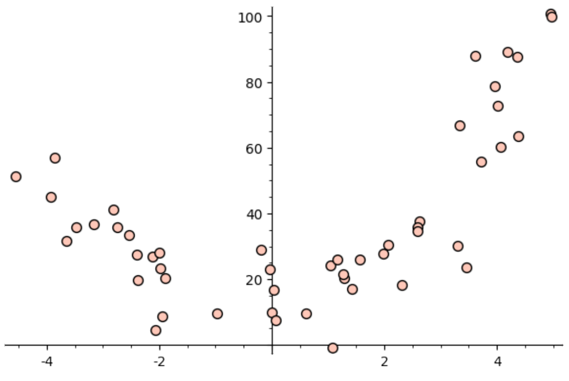


FIGURE 2.1. A set of 50 data points.

which fits the data. In other words, we want to find three numbers (a, b, c) so that $f(x)$ is the "best" fit to the data. We therefore need to quantify how good of a fit a function f is for the data points. If $f(x)$ were to fit the data perfectly, we would have $f(x_i) = y_i$ for all $1 \leq i \leq N$, therefore $|f(x_i) - y_i|$ gives us a measure of how badly f fits to the point (x_i, y_i). Taking the average of the quantities $|f(x_i) - y_i|$ over all i would give us a measure of how bad f fits the data. For numerical reasons, a different measure is used.

Instead of taking the average of $|f(x_i) - y_i|$, we take the average of their squares. The corresponding function is called the mean square error. In other words, we have

$$MSE(a, b, c) = \frac{1}{N} \sum_{i=1}^{N} (a + bx_i + cx_i^2 - y_i)^2$$

This is the function we would like to minimize, and we can use the gradient descent for that purpose. We have

$$\nabla MSE(a, b, c) = \frac{2}{N} \sum_{i=1}^{N} (a + bx_i + cx_i^2 - y_i)\langle 1, x_i, x_i^2 \rangle.$$

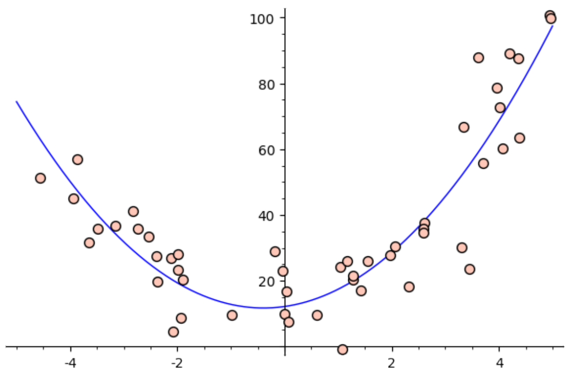


FIGURE 2.2. The best fitting degree 2 after 1000 iterations of gradient descent.

Task 4. The sage notebook best_fit.ipynb has code for computing the best fitting degree 2 polynomial using gradient descent. For this task, you need to use Sage to find the best fitting curve for the set of data generated randomly at `http://math.jhu.edu/~vzakhar2/teaching/spring2021/data.php`. There is a clearly labeled cell in the Sage notebook where you will need to copy and paste the data. You will have to adjust the step size and the number of iteration in order to get a reasonable answer. Include in your project submission the plot of the function with the data points. You will need to adjust the x bounds for your graph.