CS 530 Fall 2025 Homework 1

Calvin Stahoviak – cstahoviak@unm.edu

November 24, 2025

Collaborators: N/A

By turning in this assignment, I agree to the UNM Student Code of Conduct and declare that all of this is my own work.

Problem 1

(Taylor Expansions)

(a) Use the Taylor expansions of e^x, $\sin x$, and $\cos x$ to prove $e^{j\theta} = \cos\theta + j\sin\theta$.

(Hint, it is in the published lecture notes already. Please do not cut paste and go through the process. It's important to understand Taylor expansion and the use of polynomials and derivatives to approximate functions.)

Solution: The Taylor expansions of e^x, $\sin x$, and $\cos x$ are as follows:

$$e^x = \sum_{n=1}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \tag{1}$$

$$\sin(x) = \sum_{n=1}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots \tag{2}$$

$$\cos(x) = \sum_{n=1}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = x - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^4}{4!} + \dots \tag{3}$$

To prove $e^{j\theta} = \cos\theta + j\sin\theta$, first substitute $x = j\theta$ in Equation (1), where j is the imaginary unit:

$$e^{j\theta} = \sum_{n=1}^{\infty} \frac{(j\theta)^n}{n!} = 1 + j\theta + \frac{(j\theta)^2}{2!} + \frac{(j\theta)^3}{3!} + \dots \tag{4}$$

Next, via the *Power of a Product Rule*, simplify the powers of j:

$$e^{j\theta} = 1 + j\theta + \frac{(-1)\theta^2}{2!} + \frac{(-j)\theta^3}{3!} + \frac{(1)\theta^4}{4!} + \dots \tag{5}$$

Separating the real terms and the imaginary terms produces the following series:

$$\text{(Real Terms)} \quad 1 + \frac{(-1)\theta^2}{2!} + \frac{(1)\theta^4}{4!} + \dots \tag{6}$$

$$\text{(Complex Terms)} \quad j\theta + \frac{(-j)\theta^3}{3!} + \frac{(j)\theta^5}{5!} + \dots \tag{7}$$

Note that Equation (6) is the same series as Equation (3) and Equation (7) is the same series as Equation (2) multiplied by j. Therefore:

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{8}$$

(b) Construct the Taylor expansions for the following functions at $x_0 = 0$:

$$\frac{1}{1-x} \tag{9}$$

and

$$\ln(1+x) \tag{10}$$

Solution: The Maclaurin series, an application of the Taylor series at $x = 0$, of a function $f(x)$ has the following form:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n \tag{11}$$

Using the chain rule, the first three derivatives of Equation (9) are as follows:

$$\begin{aligned} f^{(1)}(x) &= (-1)(1-x)^{-2}(-1) &&= \frac{1}{(1-x)^2} \\ f^{(2)}(x) &= (-2)(1-x)^{-3}(-1) &&= \frac{2}{(1-x)^3} \\ f^{(3)}(x) &= (-6)(1-x)^{-4}(-1) &&= \frac{6}{(1-x)^4} \end{aligned} \tag{12}$$

When evaluated at $x = 0$:

$$\begin{aligned} f^{(1)}(0) &= \frac{1}{(1-0)^2} &&= 1 \\ f^{(2)}(0) &= \frac{2}{(1-0)^3} &&= 2 \\ f^{(3)}(0) &= \frac{6}{(1-0)^4} &&= 6 \end{aligned} \tag{13}$$

Notice that the result of the nth derivative at $x = 0$ is equal to n factorial. Also note that the zeroth derivative, $f^{(0)}(0) = 1 = 0!$, follows the same pattern. Plugging this back into Equation (11) yields the final Maclaurin series:

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \frac{n!}{n!}(x)^n \\ &= \sum_{n=0}^{\infty} x^n \\ &= 1 + x + x^2 + x^3 + \dots \end{aligned} \tag{14}$$

For Equation (10), a similar process is taken to find the Maclaurin series. To start the first few derivatives are as follows:

$$\begin{aligned} f^{(0)}(x) &= \ln(1+x) \\ f^{(1)}(x) &= (1+x)^{-1} &&= \frac{1}{1+x} \\ f^{(2)}(x) &= (-1)(1+x)^{-2} &&= \frac{-1}{(1+x)^2} \\ f^{(3)}(x) &= (-2)(-1)(1+x)^{-3} &&= \frac{2}{(1+x)^3} \\ f^{(4)}(x) &= (-3)(2)(1+x)^{-3} &&= \frac{-6}{(1+x)^4} \end{aligned} \tag{15}$$

When evaluated at $x = 0$:

$$\begin{aligned} f^{(0)}(0) &= \ln(1+0) &&= 0 \\ f^{(1)}(0) &= \frac{1}{1+0} &&= 1 \\ f^{(2)}(0) &= \frac{-1}{(1+0)^2} &&= -1 \\ f^{(3)}(0) &= \frac{2}{(1+0)^3} &&= 2 \\ f^{(4)}(0) &= \frac{-6}{(1+0)^4} &&= -6 \end{aligned} \tag{16}$$

Notice a similar pattern, that the nth derivative is equal to $(-1)^{n+1}(n-1)!$ with the exception of $f^{(0)} = 0$. Plugging this pattern back into Equation (11) yields the final Maclaurin series:

$$\begin{aligned} f(x) &= 0 + \sum_{n=1}^{\infty} \frac{(-1)^{n+1}(n-1)!}{n!} x^n \\ &= \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} \\ &= x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots \end{aligned} \tag{17}$$

Problem 2

(The Numerical Square Root Finding Algorithm)

The goal of this problem is to give you a taste of iterative numerical conversion. The square root $\sqrt{a}$ for most positive real numbers a are irrational numbers and is not easy to calculate. Since 1,500 BC, ancient Babylonians have been using the following iterative method to find square roots. The method works as follows:

Assume $x^{(0)}$ is the initial solution, say some random positive number.

We then iteratively calculate the following sequence of solutions $x^{(1)}, x^{(2)}, \ldots$, where each solution:

$$x^{(j+1)} = \frac{1}{2}(x^{(j)} + \frac{a}{x^{(j)}}) \tag{18}$$

for $j = 0, 1, \ldots$

(a) Please implement the above simple root finding algorithm using your favorite programming language (e.g., python, MATLAB, C, Java, etc.) and see if the algorithm works with some examples and compare the results from a calculator.

Solution: The following method was implemented in Python to execute the series in Equation (18):

```
import math

def babylonian_sqrt(a,
                    x0=1.0,
                    tolerance=1e-10,
                    max_iterations=100):
    x = x0
    for j in range(max_iterations):
        x_next = 0.5 * (x + a / x)
        if abs(x_next - x) < tolerance:
            return x_next, j + 1
        x = x_next
    return x, max_iterations
```

This Python method executed on the the values $2, 3, 5, 10, 25, 50, 100$ with an initial value $a = 1$ yields the results in Figure 1.

(b) Can you explain why the algorithm can find the square root?

Hint: For the algorithm to find a square it needs to converge to $\sqrt{a}$ i.e.,

$$\lim_{j \to \infty} |x^{(j)} - \sqrt{a}| = 0 \tag{19}$$

Solution: To start, let's look at what happens when $x^{(j)} = \sqrt{a}$, i.e. we start at the solution. The term $\frac{a}{\sqrt{x^{(j)}}}$ yields $x^{(j)}$, and then $\frac{1}{2}(x^{(j)} + x^{(j)})$ also yields $x^{(j)}$. So the

Number	Babylonian	Calculator	Error	Iterations
2	1.41421356	1.41421356	2.22e-16	5
3	1.73205081	1.73205081	0.00e+00	6
5	2.23606798	2.23606798	0.00e+00	6
10	3.16227766	3.16227766	4.44e-16	7
25	5.00000000	5.00000000	0.00e+00	7
50	7.07106781	7.07106781	0.00e+00	8
100	10.00000000	10.00000000	0.00e+00	9

Figure 1: Comparison of calculator square root and the Babylonian method.

following term $x^{(j+1)} = x^{(j)}$, there is no error. The sequence will stay at $\sqrt{a}$ when it reaches that value, i.e. $\sqrt{a}$ is a *fixed point*.

We can also show that error also decreases with each iteration. First let's show that the sequence is bounded. The AM-GM inequality states that for any $u > 0, v > 0$:

$$\frac{u+v}{2} \geq \sqrt{uv} \tag{20}$$

Applying this to Equation (18) results in:

$$x^{(j+1)} = \frac{1}{2}(x^{(j)} + \frac{a}{x^{(j)}}) \geq \sqrt{x^{(j)}\frac{a}{x^{(j)}}} = \sqrt{a} \tag{21}$$

This tells us that $\forall j \geq 0$ that $x^{(j+1)} \geq \sqrt{a}$, i.e. the sequence is bounded below by $\sqrt{a}$. If we can show that $x^{(j)} > \sqrt{a}$ and $x^{(j+1)} < x^{(j)}$ then we demonstrate the the sequence is bounded and decreasing. First, assume that $x^{(j)} > \sqrt{a}$. Rearranging this formula results in:

$$(x^{(j)})^2 > a \implies x^{(j)} > \frac{a}{x^{(j)}} \tag{22}$$

Using Equation (22) in Equation (18) yields the following inequality:

$$x^{(j+1)} = \frac{1}{2}(x^{(j)} + \frac{a}{x^{(j)}}) < \frac{1}{2}(x^{(j)} + x^{(j)}) = x^{(j)} \tag{23}$$

Which states that whenever $x^{(j)} > \sqrt{a}$ then $x^{(j+1)} < x^{(j)}$.

Now that we know error decreases, let the limit be $L = \lim_{j\to\infty} x^{(j)}$, substituting Equation (18) we get:

$$L = \lim_{j\to\infty} x^{(j+1)} = \lim_{j\to\infty} \frac{1}{2}\left(x^{(j)} + \frac{a}{x^{(j)}}\right) = \frac{1}{2}\left(L + \frac{a}{L}\right) \tag{24}$$

Evaluating Equation (24) yields $L = \sqrt{a}$. Using Equation (19) results in the following:

$$\lim_{j\to\infty} |x^{(j)} - \sqrt{a}| = |\lim_{j\to\infty} x^{(j)} - \sqrt{a}| = |\sqrt{a} - \sqrt{a}| = 0 \tag{25}$$

A fixed point, decreasing error, and a limit at $\sqrt{a}$ shows that the series converges with zero error.

(c) It is said that the Babylonian root finding algorithm has a quadratic convergence. Can you explain why?

Hint: The rate of convergence is usually defined as follows.

Assume that a sequence $\{x^{(j)}\}$ converges to x^*, i.e., $\lim_{j\to\infty} x^{(j)} = x^*$.

The sequence is said to have an order of convergence $q(q \geq 1)$ at a rate of μ if:

$$\lim_{f\to\infty} \frac{|x^{(j+1)} - x^*|}{|x^{(j)} - x^*|^q} = \mu \tag{26}$$

When $q = 1$, the convergence is referred to as linear convergence. When $q = 2$, the convergence is referred to as quadratic convergence.

Solution: Let the error at step j be:

$$e^{(j)} = x^{(j)} - \sqrt{a} \tag{27}$$

Combined with Equation (26) where $\sqrt{a} = x^*$ and $q = 2$:

$$\lim_{f\to\infty} \frac{|e^{(j+1)}|}{|e^{(j)}|^2} = \mu \tag{28}$$

Let's find a form that relates consecutive errors $e^{(j)}$ and $e^{(j+1)}$. Substituting the error equation Equation (27) into Equation (18) results in:

$$e^{(j+1)} = \frac{1}{2}(x^{(j)} + \frac{a}{x^{(j)}}) - \sqrt{a} \tag{29}$$

After factoring:

$$e^{(j+1)} = \frac{x^{(j)}}{2} + \frac{a}{2x^{(j)}} - \sqrt{a} \tag{30}$$

Substituting the error equation Equation (27):

$$e^{(j+1)} = \frac{e^{(j)} + \sqrt{a}}{2} + \frac{a}{2(e^{(j)} + \sqrt{a})} - \sqrt{a} \tag{31}$$

Factoring steps:

$$e^{(j+1)} = \frac{e^{(j)}}{2} + \frac{\sqrt{a}}{2} + \frac{a}{2(e^{(j)} + \sqrt{a})} - \sqrt{a} \tag{32}$$

$$e^{(j+1)} = \frac{1}{2}\big(e^{(j)} - \sqrt{a} + \frac{a}{e^{(j)} + \sqrt{a}}\big) \tag{33}$$

$$e^{(j+1)} = \frac{1}{2}\big(\frac{e^{(j)}(e^{(j)} + \sqrt{a})}{e^{(j)} + \sqrt{a}} - \frac{\sqrt{a}(e^{(j)} + \sqrt{a})}{e^{(j)} + \sqrt{a}} + \frac{\sqrt{a}^2}{e^{(j)} + \sqrt{a}}\big) \tag{34}$$

$$e^{(j+1)} = \frac{1}{2}\big(\frac{{e^{(j)}}^2 + e^{(j)}\sqrt{a}}{e^{(j)} + \sqrt{a}} - \frac{e^{(j)}\sqrt{a} + \sqrt{a}^2}{e^{(j)} + \sqrt{a}} + \frac{\sqrt{a}^2}{e^{(j)} + \sqrt{a}}\big) \tag{35}$$

$$e^{(j+1)} = \frac{{e^{(j)}}^2}{2(e^{(j)} + \sqrt{a})} \tag{36}$$

This form is almost in a form similar to Equation (28). A few more steps:

$$\begin{aligned} \frac{|e^{(j+1)}|}{|e^{(j)}|^2} &= \frac{\frac{{e^{(j)}}^2}{2(e^{(j)}+\sqrt{a})}}{|e^{(j)}|^2} \\ &= \frac{|e^{(j)}|^2}{|e^{(j)}|^2 2|\sqrt{a} + e^{(j)}|} \\ &= \frac{1}{2|\sqrt{a} + e^{(j)}|} \end{aligned} \tag{37}$$

If we take we substitute this back into Equation (28) and solve:

$$\lim_{f\to\infty} \frac{|e^{(j+1)}|}{|e^{(j)}|^2} = \lim_{f\to\infty} \frac{1}{2|\sqrt{a} + e^{(j)}|} = \frac{1}{2|\sqrt{a}+0|} = \frac{1}{2\sqrt{a}} \tag{38}$$

Which shows that we have quadratic convergence with a rate of $\mu = \frac{1}{2\sqrt{a}}$

(d) Can you extend this approach to find cubic root or more generally nth root?

(Hint: The Babylonian square root finding algorithm is in fact a special case of the Newton's method for solving equations.)

Can you explain why the Babylonian Root Finding algorithm is a special case of Newton's method for solving equations? What equation is it solving?

Solution: Recall Newton's method for $f(x) = 0$:

$$x^{(j+1)} = x^{(j)} - \frac{f(x^{(j)})}{f'(x^{(j)})} \tag{39}$$

If we take let $f(x) = x^2 - a = 0$ (there will be a zero where $x = \sqrt{a}$) and substitute into Equation (39):

$$\begin{aligned} x^{(j+1)} &= x^{(j)} - \frac{f(x^{(j)})}{f'(x^{(j)})} \\ &= x^{(j)} - \frac{(x^{(j)})^2 - a}{2x^{(j)}} \\ &= x^{(j)} - \frac{x^{(j)}}{2} + \frac{a}{2x^{(j)}} \\ &= \frac{x^{(j)}}{2} + \frac{a}{2x^{(j)}} \\ &= \frac{1}{2}\left(x^{(j)} + \frac{a}{x^{(j)}}\right) \end{aligned} \tag{40}$$

Which is exactly the same as Equation (18). The Babylonian method is a an expression of Newton's method that solves the function $f(x) = x^2 - a$. For a general method that approximates the nth root, let our function be $f(x) = x^n - a = 0$. Substituting this into Equation (39) results in:

$$\begin{aligned} x^{(j+1)} &= x^{(j)} - \frac{f(x^{(j)})}{f'(x^{(j)})} \\ &= x^{(j)} - \frac{(x^{(j)})^n - a}{n(x^{(j)})^{n-1}} \\ &= x^{(j)} - \frac{x^{(j)}}{n} + \frac{a}{n(x^{(j)})^{n-1}} \\ &= \frac{(n-1)x^{(j)}}{n} + \frac{a}{n(x^{(j)})^{n-1}} \\ &= \frac{1}{n}\big((n-1)x^{(j)} + \frac{a}{(x^{(j)})^{n-1}}\big) \end{aligned} \tag{41}$$

which is the general method for approximating the nth root.

Problem 3

(Orthogonality implies linear independence)

(a) Let $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$ be an orthogonal set of non-zero vectors. Prove that the set is also linearly independent.

Solution: To prove linear independence of the set of vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$, suppose we have scalars $c_1, c_2, \ldots, c_N$ such that:

$$c_1\mathbf{u}_1 + c_2\mathbf{u}_2 + \cdots + c_N\mathbf{u}_N = 0 \tag{42}$$

We need to show that $c_1 = c_2 = \cdots = c_N = 0$ is the only solution, i.e. the definition of linear independence holds up. When we take the inner product of both sides of Equation (42) with $\mathbf{u}_1$ we get:

$$\langle \mathbf{u}_1, c_1\mathbf{u}_1 + c_2\mathbf{u}_2 + \cdots + c_N\mathbf{u}_N \rangle = \langle \mathbf{u}_1, 0 \rangle \tag{43}$$

using *linearity of inner product* results in:

$$c_1\langle \mathbf{u}_1, \mathbf{u}_1 \rangle + c_2\langle \mathbf{u}_1, \mathbf{u}_2 \rangle + \cdots + c_N\langle \mathbf{u}_1, \mathbf{u}_N \rangle = 0 \tag{44}$$

Notice that because $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$ is an orthogonal set of non-zero vector, then all terms on the left-hand side except $c_1\langle \mathbf{u}_1, \mathbf{u}_1 \rangle$ become zero:

$$c_1\langle \mathbf{u}_1, \mathbf{u}_1 \rangle = 0 \tag{45}$$

Since $\mathbf{u_1}$ is a non-zero vector, then it must be $c_1 = 0$ to satisfy this equality. If we take the same steps in Equations (43) to (45) for $\mathbf{u}_2, \mathbf{u}_3, \ldots, \mathbf{u}_N$ we arrive at the conclusion that $c_1 = c_2 = \cdots = c_N = 0$. This proves that the orthogonal set of non-zero vectors, $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$, is also linearly independent.

Problem 4

(Change of Basis)

(a) Consider the vector space $\mathbb{E}^2$, i.e., a Euclidean plane. The usual basis that we use is the x-axis and y-axis's, i.e.:

$$U = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \tag{46}$$

Now, assume that we rotate the coordinate system by 45°. Now the new axis becomes:

$$W = \left\{ \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \right\} \tag{47}$$

(Note that we have normalized these base vectors, i.e., their norms are 1.) Now consider a vector $v = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$, i.e., $v = 4 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 5 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, i.e., the coefficients of v under U is $\begin{bmatrix} 4 \\ 5 \end{bmatrix}$. What is the representation of v in W?

Solution: We can represent v in terms of the basis U using a set of coefficients α_1, α_2, that is:

$$v = \alpha_1 U_1 + \alpha_2 U_2 \tag{48}$$

To perform a change of basis, we want to find constants β_1, β_2 that allow us to express v in terms of W:

$$v = \beta_1 W_1 + \beta_2 W_2 \tag{49}$$

Let A be the matrix whose columns are the basis vectors of W:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \tag{50}$$

Multiplying the matrix A by a vector in the W basis will change it to the U basis:

$$[v]_U = A[v]_W \tag{51}$$

The inverse matrix, A^{-1}, will do the opposite:

$$[v]_W = A^{-1}[v]_U \tag{52}$$

Before we determine what A^{-1} is, check the determinant of A:

$$\det(A) = a_{11} \cdot a_{22} - a_{12} \cdot a_{21} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2}}{2} - \left(-\frac{\sqrt{2}}{2} \right) \cdot \frac{\sqrt{2}}{2} = \frac{1}{2} + \frac{1}{2} = 1 \tag{53}$$

which confirms that A is invertible. Inverting A results in:

$$A^{-1} = \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \tag{54}$$

Now we can calculate the coordinates of v in basis W:

$$[v]_W = A^{-1}[v]_U = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 4 \\ 5 \end{bmatrix} \tag{55}$$

$$[v]_W = \begin{bmatrix} \frac{\sqrt{2}}{2} \cdot 4 + \frac{\sqrt{2}}{2} \cdot 5 \\ -\frac{\sqrt{2}}{2} \cdot 4 + \frac{\sqrt{2}}{2} \cdot 5 \end{bmatrix} = \begin{bmatrix} \frac{9\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \tag{56}$$

Therefore, the representation of v in basis W is $\begin{bmatrix} \frac{9\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$.

Problem 5

(SVM via Matlab)

(a) Consider the following simple labeled data sets in 2D space:

Point	Label
(5,0)	+1
(5,2)	+1
(7,7)	−1
(9,6)	−1
(2,0)	−1

Find the optimal linear SVM for the datasets using Matlab and plot the data points and the classifier you find.

Solution: The following solution was implemented in Python instead of Matlab:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# Define the data points and labels
X = np.array([[5, 0], [5, 2], [7, 7], [9, 6], [2, 0]])
y = np.array([1, 1, -1, -1, -1])

# Create and train the SVM classifier
clf = svm.SVC(kernel='linear', C=1000)
clf.fit(X, y)
```

The SVM plot is shown in Figure 2. The SVM identified three support vectors: $(9, 6)$, $(2, 0)$, and $(5, 2)$, which define the maximum margin hyperplane. The decision boundary that the SVM converged to is as follows:

$$2.999x_1 - 3.499x_2 - 6.997 = 0 \tag{57}$$

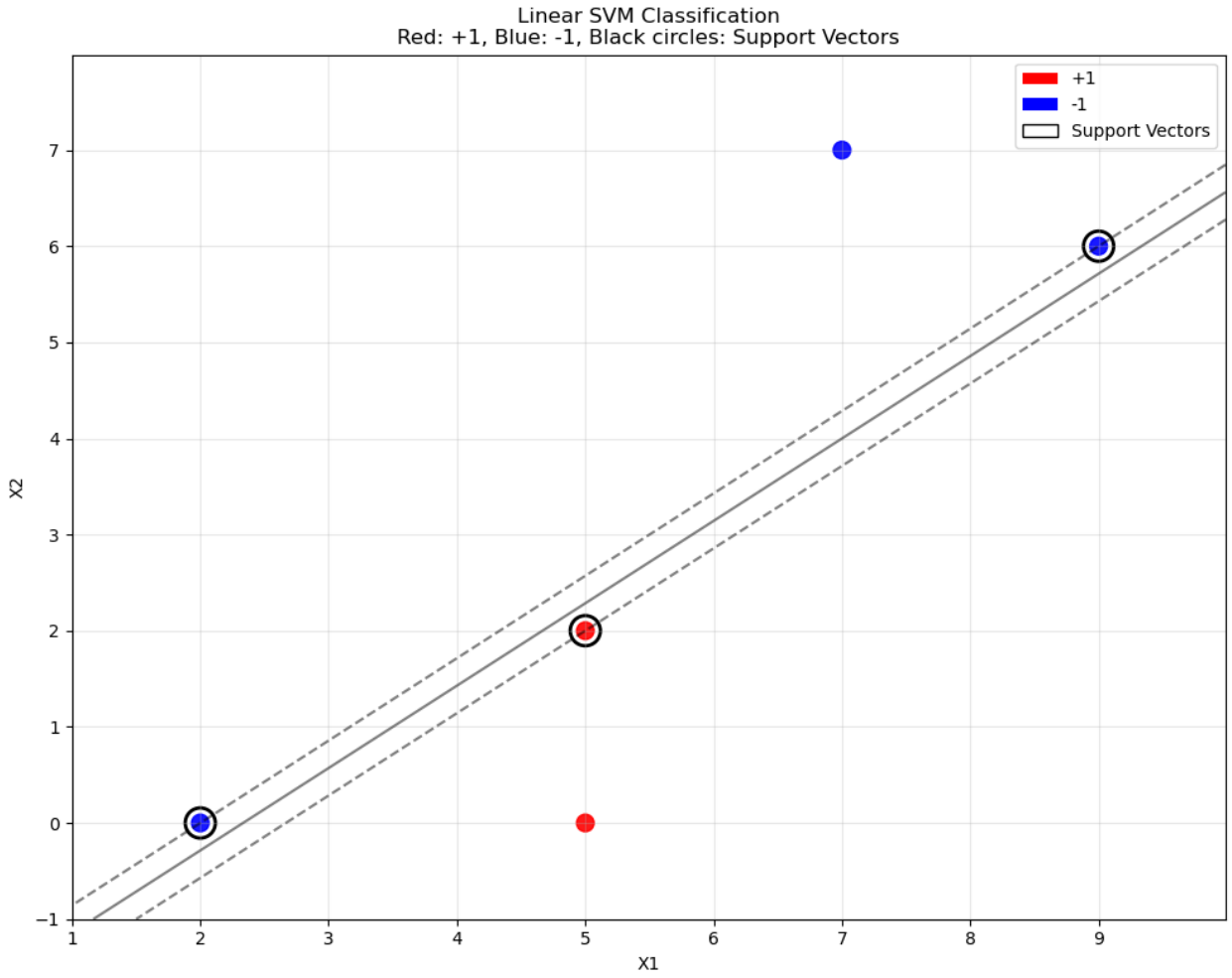

Figure 2: SVM classification results showing data points, decision boundary, and support vectors.

Problem 6

(Least Square Solvers via Python)

Use Python to solve the following optimization problems:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 13 \\ 14 \\ 15 \end{pmatrix} \tag{58}$$

(a) $\min(\mathbf{Ax} - \mathbf{b})^\top(\mathbf{Ax} - \mathbf{b})$ or $\min |\mathbf{Ax} - \mathbf{b}|^2$.

Solution: The following solution was implemented in Python:

```
import numpy as np
```

```
# Define matrix A and vector b
A = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])

b = np.array([13, 14, 15])

# Solve least squares by hand:
# x = (A^T A)^(-1) A^T b
AtA = A.T @ A
Atb = A.T @ b
x = np.linalg.inv(AtA) @ Atb
```

The output to this code is the solution $x = [-10, 4, 2, 4]^\top$ with a squared residual norm of $|\mathbf{Ax} - \mathbf{b}|^2 = 110$.

(b) $\min(\mathbf{Ax} - \mathbf{b})^\top(\mathbf{Ax} - \mathbf{b})$ such that $x \geq 0$.

Solution: The following solution was implemented in Python:

```
import numpy as np
from scipy.optimize import nnls

# Define matrix A and vector b
A = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])

b = np.array([13, 14, 15])

# Method 1: scipy's nnls (Lawson–Hanson algorithm)
x_scipy, residual_scipy = nnls(A, b)

# Method 2: Projected gradient descent (by hand)
x = np.zeros(A.shape[1])
AtA = A.T @ A
Atb = A.T @ b

for i in range(10000):  # More iterations
    grad = 2 * (AtA @ x - Atb)
    alpha = 0.001  # Smaller step size
    x_new = x - alpha * grad
    x_new = np.maximum(0, x_new)
```

```
    if np.linalg.norm(x_new - x) < 1e-10:
        break
    x = x_new
```

The output to this code is the solution $x = [0, 0, 0, 1.5357]^\top$ with a squared residual norm of $|\mathbf{Ax} - \mathbf{b}|^2 = 61.7142$.

Problem 7

(Linear programming)

(a) UNM CS Lab orders computers from two vendors Apple and Dell. Each semester, at least 10 computers must be ordered. A computer from Dell costs 2300 USD each, and a computer from Apple costs 600 USD each. (Yep, Apple is cheaper because it is using its own silicon. Dell is more expensive because it must equip with a discrete graphics card.) Costs must be kept to less than 10,000 USD (because it comes from the course fees). Moreover, UNM requires that the number of computers from each Vendor can't exceed twice the number from another. How many computers UNM CS must order from each vendor to minimize the total cost subject to the above constraints? Show that how this problem can be modeled using linear programming. Solve your linear program either via Matlab or Python. Did you notice any issues? Can you explain what causes the issue?

Solution: The following solution was implemented in Python:

```
import numpy as np
from scipy.optimize import linprog

# Variables: x = Apple computers, y = Dell computers
# Objective: minimize 600x + 2300y

# Coefficients for objective function (minimize)
c = [600, 2300]

# Inequality constraints: Ax <= b
# Convert constraints to standard form (Ax <= b):
# 1. -(x + y) <= -10  (at least 10 computers)
# 2. 600x + 2300y <= 9999  (less than 10,000)
# 3. x - 2y <= 0  (Apple <= 2*Dell)
# 4. y - 2x <= 0  (Dell <= 2*Apple)

A = [[-1, -1],      # -(x + y) <= -10
     [600, 2300],   # 600x + 2300y <= 9999
     [1, -2],       # x - 2y <= 0
```

```
        [-2,  1]]          # -2x + y <= 0

b = [-10, 9999, 0, 0]

# Bounds for variables (non-negative)
x_bounds = (0, None)
y_bounds = (0, None)

# Solve
result = linprog(c, A_ub=A, b_ub=b,
                 bounds=[x_bounds, y_bounds],
                 method='highs')
```

The output of this linear program is that no answer is feasible. This is visualized in Figure 3. No area in in x-y space shares the region of all four constraints. One possible fix is to increase the budget to \$10,150 such that it intersects with the blue and green constraints in Figure 3.

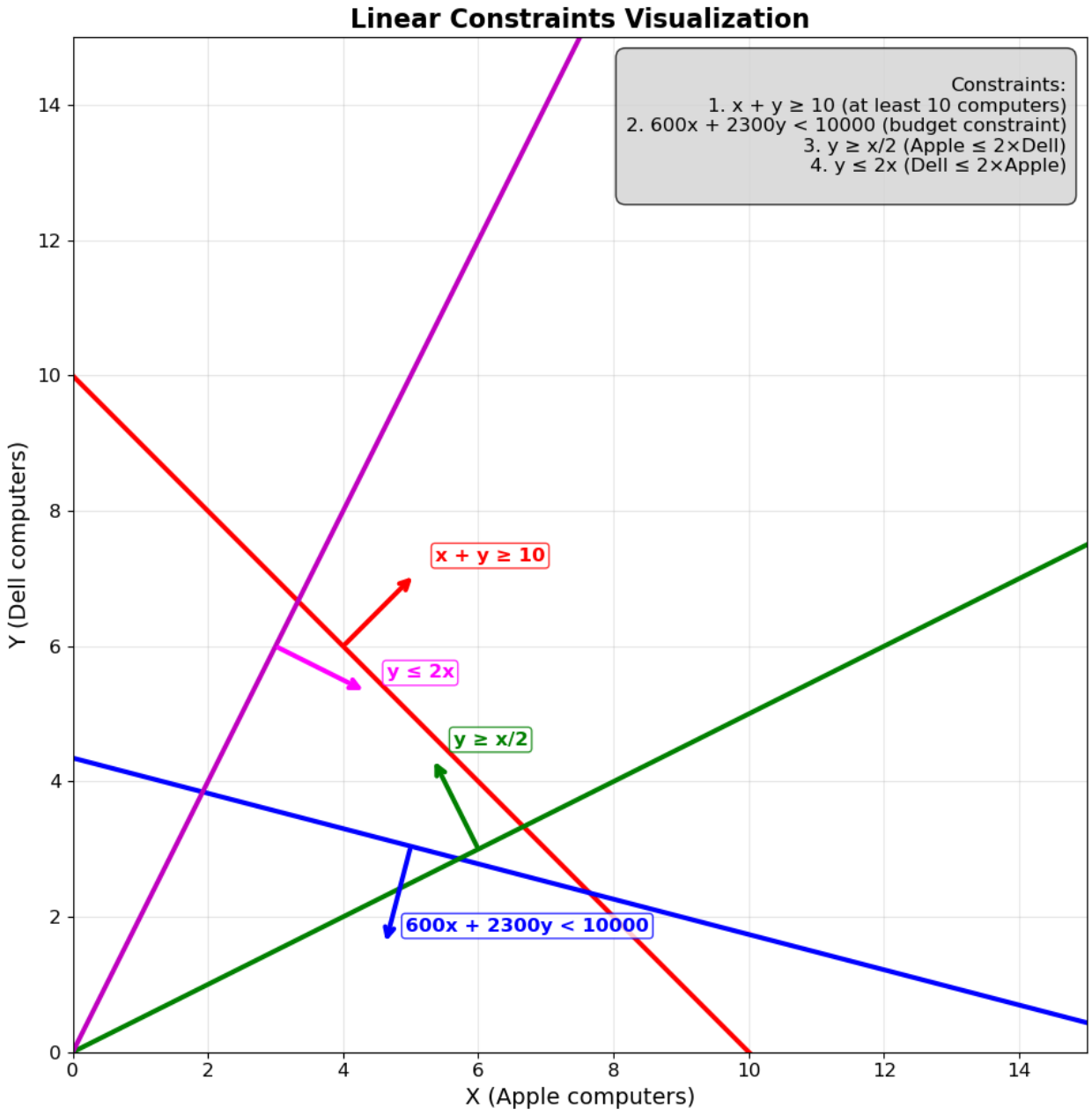

Figure 3: A visualization of each constraint with an arrow pointing to its feasible region. Notice no area shares all four constraints.