Name, SID, Date .....

Midterm Project: Integer Linear Programming in Java

Benjamin Sanders, MS October 29, 2020

1 Instructions

Read the description, and complete the tasks detailed in the description, sections 2.1, 2.2, 2.3 and 2.4. You may work in teams of three to complete this project.

2 Description

You may have fond memories learning about straight lines in High School: $y = mx + b$. (Or maybe you don't, lol.) I personally love the slope-intercept formula because it is the most coarse way to take a derivative in Calculus. Imagine that you knew y, m, and b to be 3.0, 5.7 and 8.9, respectively. Can you solve for x? In code:

```
double y = 3.0, m = 5.7, b = 8.9;

double x = (y - b) / m;

System.out.println( x );
```

Now imagine that you have a set of linear equations: three straight lines, if you graph them. You can solve for x in all of these as well.

- $5 = 4.25x + 3.7$
- $12.9 = 1.1x + 3.8$
- $2 = 0.97x + 1$

Let's do two things:

1. Let's group these equations into convenient collections, called matrices.
2. Let's allow ourselves to solve for x as a decimal value.

First, we have a matrix for all of our left-hand side numbers, including 5, 12.9 and 1. A special case of a matrix, when it is only one-dimensional, is called a vector. Therefore, this first example will actually be a vector. We will call this vector **b** to differentiate it from other things, such as the y-intercept b.

$$\mathbf{b} = \begin{bmatrix} 5 \\ 12.9 \\ 2 \end{bmatrix}$$

Next, we can have a vector for all of our m numbers, including 4.25, 1.1 and 0.97. Let's call this **m**.

$$\mathbf{m} = \begin{bmatrix} 4.25 \\ 1.1 \\ 0.97 \end{bmatrix}$$

Further, we can compose a vector for all of our right-hand side numbers, including 3.7, 3.8 and 2. Let's call this **s**.

$$\mathbf{s} = \begin{bmatrix} 3.7 \\ 3.8 \\ 1 \end{bmatrix}$$

Since every x will be different for each linear equation, let's collect those three different values in a vector, called **x**. Now, we can more compactly describe the problem we are trying to solve as: $\mathbf{b} = \mathbf{mx} + \mathbf{s}$ instead of three different examples of $y = mx + b$. Hopefully this all makes sense so far!

Presumably, you would be able to easily program a solver for this set of three independent equations in code, as well. (Don't use any matrix multiplications yet!)

2.1 Programming in Polynomial Complexity (P)

Let's change one small piece. Instead of a vector **m** which represents the three numbers 4.25, 1.1, and 0.97, let's replace it with a matrix that is $n \times n$, where $n = 3$ in this case. We will call this matrix **A**. Therefore, we can arbitrarily define **A** in this way:

$$\mathbf{A} = \begin{bmatrix} 3.2 & 8.7 & 5.9 \\ 2.4 & 3.1 & 1.1 \\ 9.7 & 6.1 & 0.3 \end{bmatrix}$$

Now, consider our statement: $\mathbf{b} = \mathbf{Ax} + \mathbf{s}$ instead of $\mathbf{b} = \mathbf{mx} + \mathbf{s}$. We can spell this out:

$$\begin{bmatrix} 5 \\ 12.9 \\ 2 \end{bmatrix} = \begin{bmatrix} 3.2 & 8.7 & 5.9 \\ 2.4 & 3.1 & 1.1 \\ 9.7 & 6.1 & 0.3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 3.7 \\ 3.8 \\ 1 \end{bmatrix}$$

Now, this means something different. The original three equations are no longer independent in this representation. Further, we cannot just solve this using the simple code above. Now, we must implement a *dot-product* in order to solve. Remember that a dot-product is defined as an element-by-element sum of multiples between two adjacent matrices of matching interior dimensions. In other words, since **A** is a 3×3 matrix and **x** is a 3×1 matrix, the fact that the second 3 of **A** matches with the first 3 of **x** allows a dot-product to be appropriate for **A** and **x**.

Let's introduce one more component. Let's say that we have a goal with this problem. Our goal is to choose **x** values such that we maximize the single decimal number result of this dot product:

$\mathbf{c}^T\mathbf{x}$

What is **c**? Just another $n \times 1$ vector, similar to **b** or **s**. Let's define **c** therefore as:

$$\mathbf{c} = \begin{bmatrix} 8.2 \\ 9.7 \\ 1.1 \end{bmatrix}$$

The symbol ^{T} refers to the transpose of a matrix. Consequently, $\mathbf{c}^T = [8.2\ 9.7\ 1.1]$. Ultimately the transpose allows the dot-product to make sense here.

Therefore, our goal is to maximize $\mathbf{c}^T\mathbf{x}$, while solving for **x** in the midst of $\mathbf{b} = \mathbf{Ax} + \mathbf{s}$. Here is one more simple constraint: all numbers in **s** and **x** must be non-negative.

Therefore, your first assignment is to write a solver for **x** given **c**, an $n \times 1$ vector, **b**, an $n \times 1$ vector, **A**, an $n \times n$ matrix, and **s**, an $n \times 1$ vector, which returns values for **x**, an $n \times 1$ vector that maximize the dot product: $\mathbf{c}^T\mathbf{x}$ such that $\mathbf{Ax} + \mathbf{s} = \mathbf{b}$. Write this in Java for n of any size, $n \geq 3$.

Known solutions for this exist in Polynomial time complexity, which means that you may be able to write this as an $O(n^3)$ solution, or an $O(n^2)$ solution, or an $O(n \cdot log(n))$ solution, or even an $O(n)$ solution.

2.2 Programming in Nondeterministic Polynomial Complexity (NP)

Let's add one small constraint: you can no longer use decimal values. Back up your code. Then, take all decimal values and convert them into integer values in your code.

No-one actually knows whether or not this can be performed in Polynomial time. We can say that if it is performed in polynomial time, it must be done so using a nondeterministic algorithm.

For your next assignment, I am not asking you to write a nondeterministic algorithm. I am however, asking you to write an algorithm which may be exponential in its execution time, such as $O(2^n)$. Notice that this is different from $O(n^2)$. Just plug in $n = 5$ to see the smallest amount of data points (five data points) that exhibits a difference in exponential vs polynomial algorithms.

Therefore, your second assignment is to write a solver for **x** given **c**, an $n \times 1$ vector of integers, **b**, an $n \times 1$ vector of integers, **A**, an $n \times n$ matrix of integers, and **s**, an $n \times 1$ vector of integers, which returns values for **x**, an $n \times 1$ vector of integers that maximize the dot product: $\mathbf{c}^T\mathbf{x}$ such that $\mathbf{Ax} + \mathbf{s} = \mathbf{b}$. All numbers in **s** and **x** must be non-negative. Write this in Java for n of any size, $n \geq 3$.

2.3 NP-Hard

The above NP problem is called "Integer Linear Programming." Find a proof online that demonstrates that Integer Linear Programming is NP-Hard, or that it is not. Write out that proof here. Cite your reference below.

- Reference: .....

2.4 NP-Complete

Is "Integer Linear Programming" NP-Complete? Circle the correct answer: yes / no.
Explain why: .....
.....
.....

- Reference: .....

3 What to Turn In

Turn in one PDF or Word document on Blackboard, containing the following items.

1. All code you and your team generated for sections 2.1 and 2.2. All the code must be your team's own original code. You may use libraries for data structures such as matrices and vectors, but you may not use libraries for solvers. If in doubt, ask Professor Sanders.
2. All pages scanned or photographed of the Midterm completed document.
3. Any additional pages you used to complete the assignment.