 **ashuell10** Added page number, ready for check

b484d7f 3 hours ago


1 contributor

<>📄

RawBlameHistory


🖥️✎️🗑️

289 lines (288 sloc) | 15.8 KB



Welcome to LinearAlgebraforStatics!

This is a **basic** notebook to learn how to use linear algebra³ to help solve common problems you see in physics and statics. Linear algebra is a branch of mathematics that allows us to think about linear equations in different ways and **is specifically** helpful to us, as a physicists and engineers, because we can solve for many unknowns at one time, through matrices.



Linear Systems with Matrices

Matrices are a tool in linear algebra that can be used to represent and manipulate systems of linear **systems** (linear systems). **Matrices** are collection of numbers that are arranged in arrays, like tables in a rectangle format enclosed by brackets. Specifically, we will use matrices to represent and solve linear systems.

Linear Systems must be in this form to solve:

$$x - y + z = 0$$

$$-x + y - z = 0$$

$$10y + 25z = 90$$


$$20x + 10y = 80$$



* Notice that the equations are organized with variables with their **coefficients on one side and constants on the other**.



Let's look at this system represented in a couple of different kinds of matrices:



$$\begin{bmatrix} 1 & -1 & 1 & 0 \\ -1 & 1 & -1 & 0 \\ 0 & 10 & 25 & 90 \\ 20 & 10 & 0 & 80 \end{bmatrix}$$

This first matrix is called an augmented matrix. **Augmented matrices** contain the **coefficients of the variables for each equation in order of variable name** **and what the expressions are equal to (constants)**. So, if you make the first column the x coefficient, the second the y coefficient, and the third the z coefficient, then you must keep the matrix in that format for the entirety of the matrix. We can see this by looking at the first row that contains 1, -1 and 1 because x, y, and z have coefficients of 1. Then, the second row contains -1, 1, -1 in relation to x, y, and z, so did not change the order of coefficients. Further note that the third row begins with a 0 because there is no x variable in that equation. The rows are concluded by a 0, 0 and 90, the constants the expressions are equal to. It is standard that the last column holds the constants. Augmented matrices are most commonly used to represent linear systems, since augmented matrices have the power to hold both the coefficients **and the constants but because of the code we are using, we will represent linear systems in a different way which will be explained next**.

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 0 & 10 & 25 \\ 20 & 10 & 0 \end{bmatrix}$$

This second matrix is a **coefficient matrix** of size 4 by 3 (rows by columns). A coefficient matrix is a matrix that solely holds the coefficients of the variables in the equation. We also can categorize this matrix by its shape and call it a rectangular matrix. **Similarly if it was, for example, a 3 by 3 matrix, we could call it a square matrix.**

$$\begin{bmatrix} 0 \\ 0 \\ 90 \\ 80 \end{bmatrix}$$

The third matrix is a special kind of matrix called a **vector**. We call the matrix a vector because it has a singular row (also can have a singular column instead). Note that **this matrix is all of the constants broken off of the augmented matrix showed earlier, but in general does not have to be.** We will use **this technique** in our coding as well and will call them "constant" vectors for the purposes of this tutorial.

In this program, we will use coefficient matrices and constant vectors because that is what the code requires.

Lastly, here is an example of how matrices will look printed from the library we will use in Python:

```
[[ 1 -1  1  0]
 [-1  1 -1  0]
 [ 0 10 25 90]
 [20 10  0 80]]
```

Now, let's learn about the code.

The Code

We will go through the code line by line so you get basic terminology and understand how the program is structured.

`import numpy as np`: This is called an import statement. Python comes with a certain range of knowledge which we can extend by importing libraries. Libraries are a collection of knowledge that we can reference so we do not need to code it ourselves. Libraries are a way that we can share knowledge with other coders. Numpy is a well known library to do scientific computing. We import it as `np` to give it an alias, so in code we do not have to keep typing out numpy in its entirety.

`coemat = np.array([])`
`vecmat = np.array([])`: These statements are called assignment statements. In Python, we also initialize variables in assignment statements. Here, we are setting each of our variables `coemat` and `vecmat` to numpy "styled" arrays. `np.array([])` will return an array that will be written by the user. The basic format for these dot operations is the library and then the routine you would like to access (`library.routine()`). So, in the numpy (`np`) library there is a routine that creates an array, specifically for this routine from an existing array that you will learn how to supply.

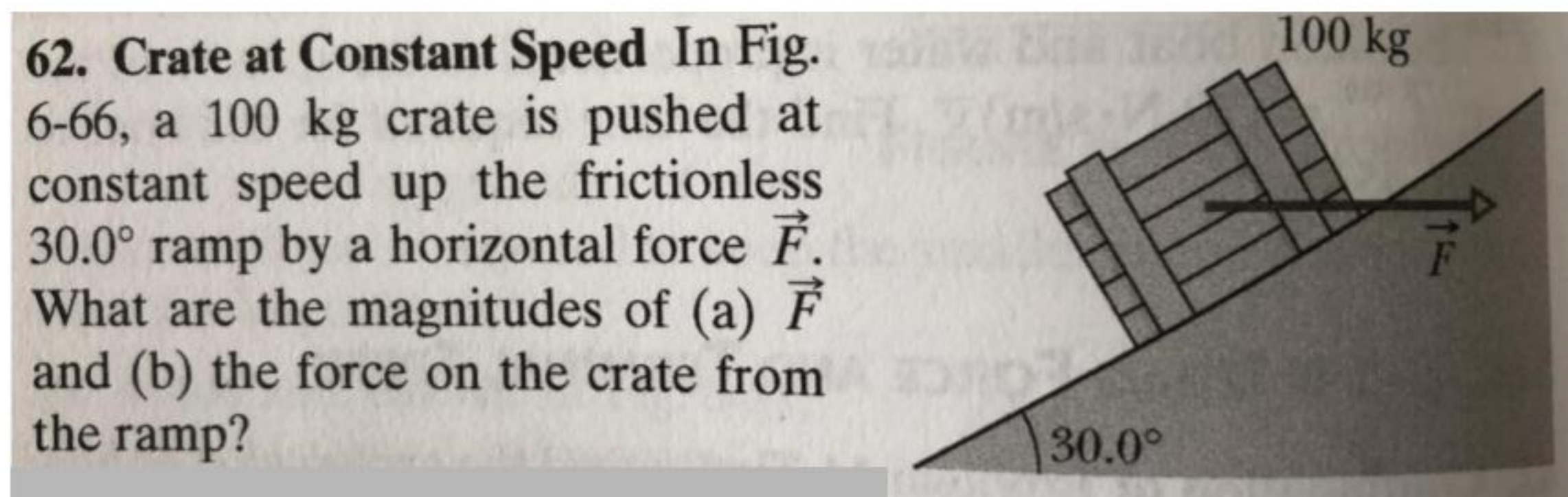
`print(coemat)`
`print(vecmat)`: These are called print statements. The Print function comes preinstalled with Python and prints something to the output device (the screen). A function is a collection of code that does something. You will notice that inside of the parentheses you have the variables `coemat` and `vecmat` inside. These conditions inside of parentheses of a function (or a method later defined) are called parameters. Parameters are passed in and can be used inside of the function. You can even learn later to write your own functions. After you enter your arrays, these statements are printed so you can see your arrays in the format you saw above. These are added for user ease.

`sol = np.linalg.lstsq(coemat, vecmat)`
`print(sol[0])`
`print(Done)`: The first line is another example of an assignment statement. You will see we access the numpy library again and use a method from `linalg` called `lstsq`. This stands for the least squares method and an example of a method buried in a routine from a library. A method is a collection of code that is stored in class. Classes define, in code, a certain kind of object and methods make changes to or accesses that kind of object. So, we will use the numpy library's routine called `linalg` to make changes to it by the `lstsq` method on the object of class numpy array. The general syntax is `library.routine.method()`. This method returns an array of useful data. For our purposes, the answer to the system is put in the first entry of the list. We access and print it on the next line by `sol[0]`. In lists and arrays we count each object starting at 0, which is why we have `sol[0]` written. Finally, we print `Done` to indicate to the user that the program is done, which is purely for the user.

Now we know about matrices and the code, let's look at a familiar example to get us acquainted with the code and entering matrices.

Example 1 - Physics:

Let's look at an example of a problem¹ we can use linear algebra from Workshop Physics I:





The goal of this question is to write 2 force equations that will allow you to solve for the 2 unknowns. The key to do this problem is that we can make both $\sum F_x$ and $\sum F_y$ equal to 0 because the crate is not accelerating on the ramp in any direction (so there is no net force on the crate). Remember a force is related to acceleration, not motion. It is also helpful to use a standard coordinate system since the applied force and the force of gravity would be entirely in the x and y direction respectively in these coordinates, so we only need to break the normal force into its components. Attempt to write down the 2 force equations needed to solve the problem.

You should get $\sum F_x = F - N_x$ and $\sum F_y = N_y - G$, where F = applied force, N = normal force, G = gravitational force

Now, usually you would solve the $\sum F_y$ first for N and then solve the $\sum F_x$ for the F, but we can do this in one step with matrices. Of course, this seems silly since we can solve very easily for N in the 2nd equation but for the purposes of our learning we can easily check our answer.

The key is to look at the values of $\cos(30^\circ)$ and $\sin(30^\circ)$ and G as coefficients, keeping F and N as unknowns. Here is what you are left with: $\sum F_x = F - .5N$ and $\sum F_y = .87N - 980$

Now, knowing $\sum F_x$ and $\sum F_y$ are both 0 we can write the equations like this: $0 = F - .5N$ and $980 = .87N$

So, now we have our system we must write it in the form of a matrix. For a coefficient matrix, we have 2 variables and 2 equations so we will have 2 columns and 2 rows. Here is our coefficient matrix with F as the first variable:

```
[[1, .5],  
 [0, -.866]]
```

Then, our constant vector should look like this:

```
[[0],  
 [980]]
```

For the first example, we have entered in the numbers for you. Scroll down to the code in the next cell see what the code outputs.

```
In [6]: import numpy as np  
  
coemat = np.array([[1, -.5], [0, .866]])  
vecmat = np.array([[0], [980]])  
  
print(coemat)  
print(vecmat)  
  
sol = np.linalg.lstsq(coemat, vecmat, rcond=None)  
print(sol[0])  
print("Done")
```

```
[[ 1.    -0.5  ]  
 [ 0.     0.866]]  
[[ 0]  
 [980]]  
[[ 565.81986143]  
 [1131.63972286]]  
Done
```

The output should be:

```
[[ 1.    -0.5  ]  
 [ 0.     0.866]]  
[[ 0]  
 [980]]  
[[ 565.81986143]  
 [1131.63972286]]  
Done
```

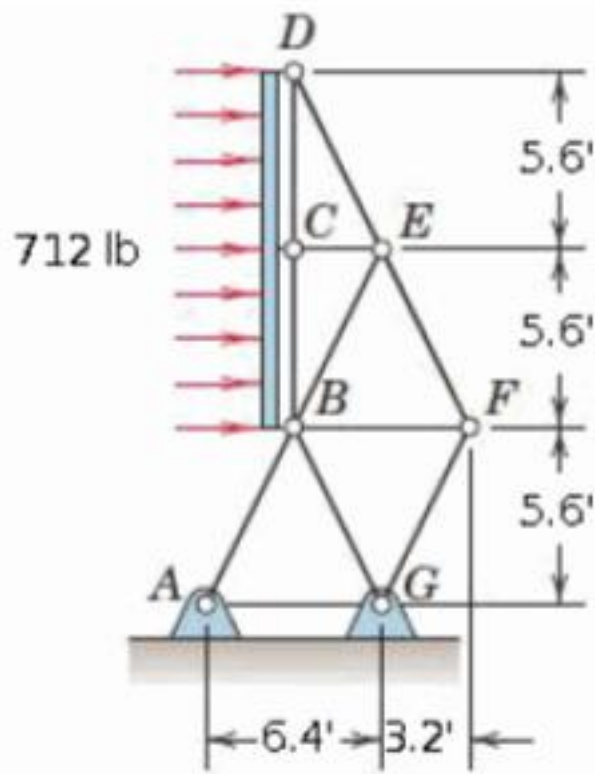
The answer is the array right before Done was printed so it is;

```
[[ 565.81986143]  
 [1131.63972286]]
```

The answers of each variables are printed in their respective columns so the **force is equal to 565.8199 N** and the **normal force is equal to 1131.6397 N**.

Example 2 - Statics:

In statics, one way we can use matrices when we are solving for forces in a structures using the joints method, which you will learn about in more detail later in the year. From *Engineering Mechanics: Statics*, here is problem² about a signboard where we are asked to find the force in members BE and BC when the horizontal wind load is 712 lb. Specifically 5/8 of the force is transmitted to the center connection at C and the rest is equally divided between D and B.



For learning purposes, let's go through this problem so you can start seeing the process to solve structures by the joints method. You do not need to know how to do these problems yet, but we can practice entering their values into the program.

Instead of solving each joint explicitly like you will be taught to do, we can leave the equations unsolved and put the equations in a system which decreases the amount of equations we must solve. First though, we can solve for the D_x and C_x since we are given their values in relation to the full force. We find that $D_x = 133.5lb$, and $C_x = 445lb$. There are also some forces that you can solve very simply (they are the only countering force in a certain direction) and I put in their values explicitly below.

From joint D, you get the equations:

$$133.5 = .4961DE$$

$$0 = .8682DE - CD$$

From joint C, you get the equation:

$$0 = CD - BC$$

* From this joint you also will find $CE = 445$ T (T symbolizes that the member is in tension)

From joint E, you get the equations:

$$0 = .8682DE + .8682DE - .8682EF$$

$$445 = -.4961DE + .4961BE + .4961EF$$

* $.4961 = \sin(29.7449)$ or $\cos(60.2551)$ and $.8682 = \cos(29.7449)$ or $\sin(60.2551)$

Remember when you enter your equations into a matrix to order your variables and place 0's where necessary. With the member variables in the order DE, CD, BC, BE and EF, here are the coefficient matrix and constant vector:

Coefficient matrix:

```
[ [.4961, 0, 0, 0, 0]
  [.8682, -1, 0, 0, 0]
  [0, 1, -1, 0, 0]
  [.8682, 0, 0, .8682, -.8682]
  [-.4961, 0, 0, .4961, .4961]]
```

Constant vector:

```
[ [133.5],
  [0],
  [0],
  [0],
  [445]]
```


Here is the program again with the last example's matrix entered. Change it so it will solve this example.

```
In [7]: import numpy as np

coemat = np.array([[1, -.5], [0, .866]])
vecmat = np.array([[0], [980]])

print(coemat)
print(vecmat)

sol = np.linalg.lstsq(coemat, vecmat, rcond=None)
print(sol[0])
print("Done")

[[ 1.    -0.5   ]
 [ 0.     0.866]]
[[ 0]
 [980]]
[[ 565.81986143]
 [1131.63972286]]
Done
```

Here is what you should get for the answer:

```
[ [269.09897198],
  [233.63172747],
  [233.63172747],
  [448.49828664],
  [717.59725862]]
```

or $DE = 269.0989lb$, $CD = 233.6317lb$, $BC = 233.6317lb$, $BE = 448.4983lb$, $EF = 717.5972lb$, $D_x = 133.5lb$, $C_x = 445lb$ and $CE = 445lb$



Now, you can use this program whenever you want!



References:

¹Cummings, K. *Understanding Physics 2015 F/PacificU* (15th ed., Wiley, 2015), p.172

²Meriam, J. L., et al. *Engineering Mechanics: Statics*. (9th ed., Wiley, 2018).

³Kreyszig E. et. al. *Advanced Engineering Mathematics*, 10th ed., John Wiley & Sons, Inc, 2011, p. 256.

```
In [ ]: 
```