1. Counting Solutions.

(b). We first transfer this system of linear equations to an augmented matrix since it's equivalent to:

$$\begin{bmatrix} 0 & -1 & 2 \\ 2 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \Rightarrow \left[\begin{array}{ccc|c} 0 & -1 & 2 & 1 \\ 2 & 0 & 1 & 2 \end{array}\right]$$

Switch the two rows, and since we have more columns than rows, so we add a row of 0.

$$\Rightarrow \left[\begin{array}{ccc|c} 2 & 0 & 1 & 2 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

which means that I ended up with: $\begin{cases} 2x+z=2 \\ -y+2z=1 \\ 0=0 \end{cases} \Rightarrow \begin{cases} x = 1-\frac{z}{2} \\ y=-1+2z \\ 0=0 \end{cases}$

(and since we have a row of 0s, so we have infinite solutions.)

So, we have an **infinite number** of solutions, since I can't uniquely solve for x,y,z. The space of solutions:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} + \begin{bmatrix} -\frac{1}{2} \\ 2 \\ 1 \end{bmatrix} z.$$

(d). The system of linear equations is equivalent to: $\begin{bmatrix} 1 & 2 \\ 2 & -1 \\ 1 & -3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ -5 \end{bmatrix}$

So, it's equivalent to the augmented matrix: $\left[\begin{array}{cc|c} 1 & 2 & 3 \\ 2 & -1 & 1 \\ 1 & -3 & -5 \end{array}\right]$ R_1: Multiply by 2 and Subtract R_2. R_3: Multiply by 2 and Subtract R_2.

$$\Rightarrow \left[\begin{array}{cc|c} 0 & 5 & 5 \\ 2 & -1 & 1 \\ 0 & -5 & -11 \end{array}\right]. \quad R_3: \text{Add } R_1 \quad \Rightarrow \left[\begin{array}{cc|c} 0 & 5 & 5 \\ 2 & -1 & 1 \\ 0 & 0 & -6 \end{array}\right]$$

Since we end up with a row of 0s on the left and the right side is not 0, so (as discussed in class), there is no solution.

(e). The system of linear equations is equivalent to: $\begin{bmatrix} 1 & -1 \\ 5 & -5 \\ 3 & -3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 10 \\ 6 \end{bmatrix}$

So, it's equivalent to this augmented matrix: $\left[\begin{array}{cc|c} 1 & -1 & 2 \\ 5 & -5 & 10 \\ 3 & -3 & 6 \end{array}\right]$

R_2: Subtract $(5 \cdot R_1)$. and R_3: Subtract $(3 \cdot R_1)$ and then we have that. $\left[\begin{array}{cc|c} 1 & -1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right]$

Since we have two rows of 0s, so only 1 linear equation with 2 variables, so we have infinite solutions. Alternatively, I end up with: $\begin{cases} x-y=2 \\ 0=0 \\ 0=0 \end{cases} \Rightarrow \begin{cases} x=2+y \\ 0=0 \\ 0=0 \end{cases}$ which I can't uniquely solve for x, y with.

Thus, there is an infinite number of solutions. The space of solutions is:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} y.$$

2. Elementary Matrices.

(a). i.

$$E_i = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ii.

$$E_{ii} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

iii

$$E_{iii} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{bmatrix}$$

(b) First, reduce R_4 to the form of $[0\ 0\ 0\ 1]$:

$$E_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad \left(\text{Since } [0\,1\,0\,3] - [0\,1\,0\,2] = [0\,0\,0\,1]\right)$$

Then, to reduce R_3 of the resulting matrix to $[0\ 0\ 1\ 0]$:

$$E_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 2 & 7 & 1 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \left(\begin{array}{l} \text{Since } 2\cdot[1\,-2\,0\,-5] \\ \quad +7\cdot[0\,1\,0\,3] \\ \quad +1\cdot[-2\,-3\,1\,-6] \\ \quad -5\,[0\,0\,0\,1] \end{array} = [0\,0\,1\,0]\right)$$

Then, to reduce R_2 of the resulting matrix to $[0\ 1\ 0\ 0]$.

$$E_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \left(\text{Since } [0\,1\,0\,3] - 3\cdot[0\,0\,0\,1] = [0\,1\,0\,0]\right)$$

Lastly, to reduce R_1 to desired $[1\ 0\ 0\ 0]$, similarly so:

$$E_4 = \begin{bmatrix} 1 & 2 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using IPython, we then have:

i.

$$E = E_4 \cdot E_3 \cdot E_2 \cdot E_1 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & -2 & 0 & 3 \\ 2 & 2 & 1 & 5 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

ii.

$$EA = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & -2 & 0 & 3 \\ 2 & 2 & 1 & 5 \\ 0 & 1 & 0 & -1 \end{bmatrix} \cdot \left[\begin{array}{cccc|c} 1 & -2 & 0 & -5 & 15 \\ 0 & 1 & 0 & 3 & -7 \\ -2 & -3 & 1 & -6 & 9 \\ 0 & 1 & 0 & 2 & -5 \end{array}\right] = \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 \end{array}\right]$$

is an identity matrix with constants Verified Q.E.D.

3. Mechanical Inverses.

(a) Yes.

By definition A's inverse, P_1, have that $AP_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{11} & P_{12} \\ P_{13} & P_{14} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

So, $\Rightarrow \left[\begin{array}{cc|cc} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right]$ By swapping R_1 and R_2, we have that $\Rightarrow \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right]$ with Gauss-Jordan Elimination,

So, $P_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $\underbrace{\ }_{\text{if } A\vec{b} = \vec{c}, \text{ then}}$ A changes a vector $\vec{b}$ by reflecting $\vec{b}$ across the line $x = y$.

(d). Yes.

Let P_4 be A's inverse, so we have that $AP_4 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} P_{41} & P_{42} \\ P_{43} & P_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Since the determinant of A is $\cos\theta \cdot \cos\theta - (-\sin\theta \cdot \sin\theta) = \cos^2\theta + \sin^2\theta = 1$,

So, its inverse, $P_4 = \begin{bmatrix} \frac{\cos\theta}{1} & -\frac{-\sin\theta}{1} \\ -\frac{\sin\theta}{1} & \frac{\cos\theta}{1} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$

Apply the original matrix. if $A\vec{b} = \vec{c}$, then A would rotate $\vec{b}$ by $(-90°)$.
(rotate 90° counterclockwise).

(e). Yes.

Let P_5 be A's inverse, so we have that $AP_5 = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{51} & P_{52} \\ P_{53} & P_{54} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

So, $\Rightarrow \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{array}\right]$ R_1: Subtract $(R_2/2)$. and R_2: Divide by 2. then, swap R_1 and R_2 $\Rightarrow \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 1 & -0.5 \end{array}\right]$

So, $P_5 = \begin{bmatrix} 0 & 0.5 \\ 1 & -0.5 \end{bmatrix}$

(f) No.

Let P_6 be A's inverse, so by definition. $AP_6 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 2 \\ 1 & 4 & 4 \end{bmatrix} \cdot \begin{bmatrix} P_{61} & P_{62} & P_{63} \\ P_{64} & P_{65} & P_{66} \\ P_{67} & P_{68} & P_{69} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

So, $\Rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 & 0 \\ 1 & 4 & 4 & 0 & 0 & 1 \end{array}\right]$ R_3: Subtract R_1, Subtract $2 \times R_2$.

$\Rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & -2 & 1 \end{array}\right]$ Since we end up with a row of 0s, and the right side is not 0, so there is no solution to the system of equations.

Thus, no inverse of A exists.

Scanned with CamScanner

5. Properties of Pump Systems.

(a). Using the information from the graph, we have $\begin{cases} \vec{x_1}[n+1] = \vec{x_1}[n] + \vec{x_2}[n] \\ \vec{x_2}[n+1] = 0. \end{cases}$

(b). Thus, $A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$

$\boxed{(c)}$ For both initial states, $\vec{x}[1] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

In Universe 1, where $x_1[0] = 0.5$, $x_2[0] = 0.5$,

so $\vec{x}[1] = A\vec{x}[0] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$;

In Universe 2. where $x_1[0] = 0.3$, $x_2[0] = 0.7$

So similarly, $\vec{x} = A\vec{x}[0] = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Thus, no matter what, the water levels at timestep 1 is $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

(d) No, I can't. This is because that as we proved, both initial states ($x_1[0] = x_2[0] = 0.5$ and $x_1[0] = 0.3$, $x_2[0] = 0.7$) lead to the same result at timestep 1, so I can't figure out the initial water levels.

(e) No, I can't. Proof by Contradiction.

Proof: Assume, for a contradiction, that there exists a state transition matrix A^* such that two different initial state vectors lead to the same water levels / state vectors at timestep k, and that I can recover the unique initial water levels $\vec{x}[0]$. Let $A^* \cdot \vec{x}[i] = \vec{x}[i+1]$

Since we can recover an unique initial water levels, so it means that A^* is invertible, and then we can recover $\vec{x}[0]$ from $\vec{x}[k]$ by multiplying it to A^{*-1} for k times. Now, consider the state vectors at timestep $(k-1)$, $\vec{x}[k-1]$. Since we have proved in the lecture notes that if M is an invertible matrix, then its inverse must be unique. Thus, there is only one state vector possible, $\vec{x}[k-1]$. Similarly, we can deduce this for each timestep's state vector.

Therefore, it's not possible to have two different initial state vectors.

Q.E.D.

f. Consider, for k reservoirs, the state vector at time n,

$$\vec{x}[n] = \begin{bmatrix} x_1[n] \\ x_2[n] \\ \cdots \\ x_k[n] \end{bmatrix}$$

Since we know that the entries of each column vector of the state transition matrix A sum to one, so this implies that for any reservoir i, $1 \le i \le k$, all of its water goes to reservoirs 1 through k. In other words, the amount of water collectively, $x_1[n] + x_2[n] + \cdots + x_k[n] = s$, would be preserved for timestep $(n+1)$.

Thus, let $\vec{x}[n+1] = \begin{bmatrix} x_1[n+1] \\ x_2[n+1] \\ \vdots \\ x_k[n+1] \end{bmatrix}$, so $x_1[n+1] + \cdots + x_k[n+1] = x_1[n] + \cdots + x_k[n] = s$.

which means that the total amount of water at timestep $(n+1)$ is still s for k reservoirs.

(When $k=3$, this generalization provides the case for the first half of the problem).

Q.E.D.

6. Audio File Matching.

(a). → When $\vec{X_1} = [1\ 1 \cdots 1]^T$, $\vec{X_2} = [1\ 1 \cdots 1]^T$, with length n.

so $\vec{X_1}^T \vec{X_2} = [1 1 \cdots 1] \cdot [1 1 \cdots 1]^T = n \cdot 1^2 = n$.

→ When $\vec{X_1} = [1 1 \cdots 1]^T$. $\vec{X_2} = [1\ -1\ 1\ -1 \cdots 1\ -1]^T$ with an even length,

so $\vec{X_1}^T \vec{X_2} = [1 1 \cdots 1] \cdot [1\ -1 \cdots 1\ -1]^T = 1 + (-1) + \cdots + 1 + (-1) = 0$.

→ A larger dot product implies that the vectors are more similar.

This is because if the vectors are less similar, the dot product would cancel out more, which would lead to a smaller dot product, and vice versa.

(b) My approach is to pull out 3 consecutive digits from $\vec{X}$ each time, and then dot multiply each 3-bit string/vector with $\vec{Y}$.

Yes, I can write this as a matrix vector multiplication:

$$A = \begin{bmatrix} 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 \end{bmatrix}, \vec{x} = \vec{X} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

Finishing my approach above, the dot product of A and $\vec{x}$, $A\vec{x}$, would be a 6×1 vector, and the row with largest dot product is the index.

In other words, if row i has the largest dot product, then i is the answer, representing $[x_i \ x_{i+1} \ x_{i+2}]^T$ as closest to $\vec{Y}$.

Now, to figure out what's closest to $\vec{Y} = [1 \ 1 \ 1]^T$,

$$\Rightarrow A\vec{x} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 3 \\ 1 \\ 1 \end{bmatrix}$$

Thus, row 4 has the largest dot product, 3, so $i = 4$ is what we're looking for. (which is indeed $[1 \ 1 \ 1]^T$.)

(d)

- Run the following cell. Do your results here make sense given your answers to previous parts of the problem? What is the function vector_compare doing?

The results here make sense since the closer the vectors are, the larger the result (a normalized version of dot products), which is coherent with my previous answers.

In essence, the function vector_compare first takes the dot product of two vectors we wish to compare, and then divides the results by the magnitude of the two vectors, allowing us to negate the effect of their length. Thus, the result would give a “fairer” comparison.

- Run the following code that runs vector_compare on every subsequence in the song- it will probably take at least 5 minutes. How do you interpret this plot to find where the clip is in the song?

We would look at the magnitude (absolute value) of the resulting plot. The larger the absolute value on a given time, the closer the target signal is to the given signal beginning at that moment. (I got some help from the GSIs and their answers on Piazza, and I was convinced that the positive/negative signs of the value doesn’t matter, and only the absolute values matter.)

(e)

- The code below uses song_compare to print the index of given_signal where target_signal begins. Can you interpret how the code finds index? Verify that the code is correct by playing the song at that index using the play_clip function.

Yes, I can. The code functions by enumerating all absolute values of the variable song_compare, which has given us the plot in part (d). The code then prints out the index where the largest absolute value occurs. As I said above, we could find the clip in the given signal by by finding out the largest absolute value when we run the function vector_compare and run_comparison on each slice of the given signal with the target signal.

I can verify that the code is correct as I compared the target signal and the final answer given by play_clip; they are exactly the same.

7.

Question: Solve the inverse for $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$. If there is, solve; if there isn't, explain.

Solution. Let P be A's inverse, so we have that $AP = I$.

So we have.
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which can be represented as this augmented matrix:

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 4 & 5 & 6 & 0 & 1 & 0 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right]$$

R_3: Add R_1 and Subtract $2 \times R_2$.

and we have:

$$\Rightarrow \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 4 & 5 & 6 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \end{array}\right]$$

Now, we have a row of 0s, but the right side is not 0.

Thus, we reached an ending condition, and we've shown/proved that A does not have an inverse.

12-2-17.

8. Homework Process and Study Group

I worked alone without getting any help, except asking questions and reading posts (especially answers from the GSIs) on Piazza as well as reading the Notes of the course.

EE16A: Homework 3 ¶

In [17]:
```
%matplotlib inline
from numpy import zeros, cos, sin, arange, around, hstack
from matplotlib import pyplot as plt
from matplotlib import animation
from matplotlib.patches import Rectangle
import numpy as np
from scipy.interpolate import interp1d
import scipy as sp
import wave
import scipy.io.wavfile
import operator
from IPython.display import Audio
```

Problem 2: Elementary Matrices

Part (b)

In [19]:
```
## YOUR CODE HERE

E_1 = np.array([
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 1, 0, -1]
])

E_2 = np.array([
    [1, 0, 0, 0],
    [0, 1, 0, 0],
    [2, 7, 1, -5],
    [0, 0, 0, 1]
])

E_3 = np.array([
    [1, 0, 0, 0],
    [0, 1, 0, -3],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
])

E_4 = np.array([
    [1, 2, 0, 5],
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
])

E = np.matmul(np.matmul(E_4, E_3), np.matmul(E_2, E_1))

print(E)
```

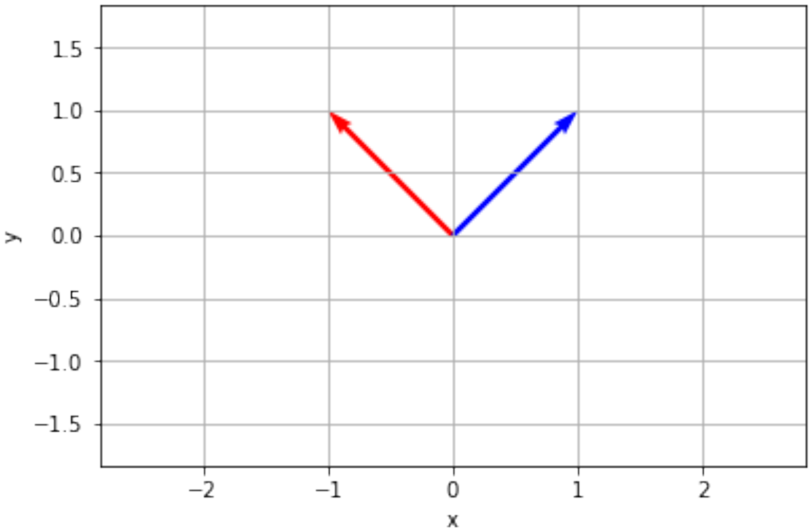
```
[[ 1  1  0  1]
 [ 0 -2  0  3]
 [ 2  2  1  5]
 [ 0  1  0 -1]]
```

Problem 3: Mechanical Inverses

Part (d)

In [3]:
```
def rotation_matrix(v, theta):
    """
    Inputs:
        v: Numpy array with an x- and y-component.
        theta: Float.
    Returns:
        Numpy array with an x- and y-component.
    """
    A = np.array([[np.cos(theta), -np.sin(theta)],
                  [np.sin(theta), np.cos(theta)]])
    return A.dot(v)

def plot_rotation_matrix(v, theta):
    """
    Inputs:
        v: Numpy array with an x- and y-component.
        theta: Float.
    Returns:
        None.
    """
    # plotting the transformation
    origin = [0], [0]
    u = rotation_matrix(v, theta)
    plt.axis('equal')
    plt.quiver(*origin, [u[0], v[0]], [u[1], v[1]], color=['r', 'b'], sc

    # setting appropriate plot boundaries
    boundary = np.linalg.norm(v)*2
    plt.xlim(-boundary, boundary)
    plt.ylim(-boundary, boundary)

    # plot cleanliness
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid()
    return
```

In [4]:
```
# Change v and theta to see how the rotation operation affects it
v = np.array([1, 1])
theta = np.pi/2

plot_rotation_matrix(v, theta)
```



Problem 6: Audio File Matching

This notebook continues the audio file matching problem. Be sure to have song.wav and clip.wav in the same directory as the notebook.

In this notebook, we will look at the problem of searching for a small audio clip inside a song.

The song "Mandelbrot Set" by Jonathan Coulton is licensed under [CC BY-NC 3.0 (http://creativecommons.org/licenses/by-nc/3.0/)](http://creativecommons.org/licenses/by-nc/3.0/)

In [6]:
```
given_file = 'song.wav'
target_file = 'clip.wav'
rate_given,  given_signal  = scipy.io.wavfile.read(given_file)
rate_target, target_signal = scipy.io.wavfile.read(target_file)
given_signal  = given_signal[:2000000].astype(float)
target_signal = target_signal.astype(float)
def play_clip(start, end, signal=given_signal):
    return Audio(data=signal[start:end], rate=rate_given)

def run_comparison(target_signal, given_signal, idxs=None):
    # Run everything if not called with idxs set to something
    if idxs is None:
        idxs = [i for i in range(len(given_signal)-len(target_signal))]
    return idxs, [vector_compare(target_signal, given_signal[i:i+len(ta
            for i in idxs]

play_clip(0, len(given_signal),given_signal)

#scipy.io.wavfile.write(target_file, rate_given, (-0.125*given_signal[1.
```

Out[6]: 0:00 -0:45

We will load the song into the variable `given_signal` and load the short clip into the variable `target_signal`. Your job is to finish code that will identify the short clip's location in the song. The clip we are trying to find will play after executing the following block.

In [8]:
```
play_clip(0, len(target_signal),signal=target_signal)
```

Out[8]: 0:00 -0:01

Part (d)

Run the following cell. Do your results here make sense given your answers to previous parts of the problem? What is the function `vector_compare` doing?

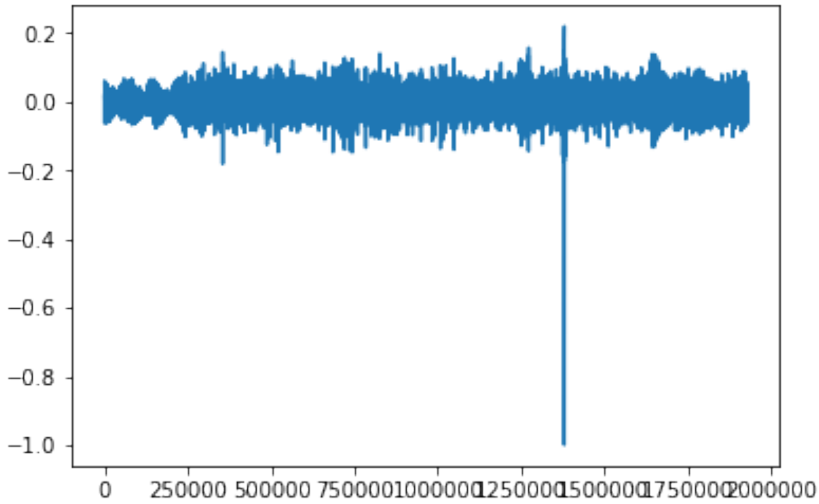
In [9]:
```
def vector_compare(desired_vec, test_vec):
    """This function compares two vectors, returning a number.
    The test vector with the highest return value is regarded as being c
    return np.dot(desired_vec.T, test_vec)/(np.linalg.norm(desired_vec)

print("PART A:")
print(vector_compare(np.array([1,1,1]), np.array([1,1,1])))
print(vector_compare(np.array([1,1,1]), np.array([-1,-1,-1])))
print("PART C:")
print(vector_compare(np.array([1,2,3]), np.array([1,2,3])))
print(vector_compare(np.array([1,2,3]), np.array([2,3,4])))
print(vector_compare(np.array([1,2,3]), np.array([3,4,5])))
print(vector_compare(np.array([1,2,3]), np.array([4,5,6])))
print(vector_compare(np.array([1,2,3]), np.array([5,6,7])))
print(vector_compare(np.array([1,2,3]), np.array([6,7,8])))
```

```
PART A:
0.9999999999666668
-0.9999999999666668
PART C:
0.9999999999928572
0.9925833339660043
0.9827076298202766
0.9746318461941077
0.968329663729021
0.9633753381636556
```

Run the following code that runs `vector_compare` on every subsequence in the song- it will probably take at least 5 minutes. How do you interpret this plot to find where the clip is in the song?

In [10]:
```
import time

t0 = time.time()
idxs, song_compare = run_comparison(target_signal, given_signal)
t1 = time.time()
plt.plot(idxs, song_compare)
print ("That took %(time).2f minutes to run" % {'time':(t1-t0)/60.0} )
```

```
That took 2.06 minutes to run
```



Part (e)

The code below uses `song_compare` to print the index of `given_signal` where `target_signal` begins. Can you interpret how the code finds `index` ? Verify that the code is correct by playing the song at that index using the `play_clip` function.

In [20]:
```
index, value = max(enumerate([abs(i) for i in song_compare]), key=operat
print (index)
play_clip(index,index+len(target_signal))
```

```
1380000
```

Out[20]: 0:01 -0:00

In [ ]: