Matrix Tricks

August 18, 2019

1 Introduction

This document is my informal reference sheet for linear algebra topics that I should stop forgetting.

Remember that the whole idea of this stuff is that $Ax = b$. Given linear equations of the form

$$ax + by = c \tag{1}$$

and

$$dx + ey = f \tag{2}$$

we construct matrices as a shorthand for their coefficients:

$$\begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x \begin{bmatrix} a \\ d \end{bmatrix} + y \begin{bmatrix} b \\ e \end{bmatrix} = \begin{bmatrix} c \\ f \end{bmatrix}. \tag{3}$$

See [https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/lecture-1-the-geometry-of-linear-equations/].

[1]:
```
v = [i for i=1:3]
```

[1]:
```
3-element Array{Int64,1}:
 1
 2
 3
```

[2]:
```
m = [3(i-1) + j for i=1:3, j=1:3]
```

[2]:
```
3×3 Array{Int64,2}:
 1  2  3
 4  5  6
 7  8  9
```

[3]:
```
ones(Int,1,3)
```

[3]:
```
1×3 Array{Int64,2}:
 1  1  1
```

2 Rotate or transpose a matrix

According to [https://math.stackexchange.com/questions/1945329/can-you-transpose-a-matrix-using-matrix-multiplication] and [https://math.stackexchange.com/questions/2816073/does-there-exist-2-matricies-such-that-they-can-be-used-to-transpose-any-n-by-n] this cannot be done with a cross product. For this, you need the help of your programming language.

[4]:
```
transpose(m)
```

[4]:
```
3×3 LinearAlgebra.Transpose{Int64,Array{Int64,2}}:
 1  4  7
 2  5  8
 3  6  9
```

[5]:
```
rotl90(m)
```

[5]:
```
3×3 Array{Int64,2}:
 3  6  9
 2  5  8
 1  4  7
```

[6]:
```
rotl90(rotl90(m)) == rot180(m)
```

[6]:
```
true
```

[7]:
```
rotl90(rotl90(rotl90(m))) == rotr90(m)
```

[7]:
```
true
```

3 Copy vectors

3.1 Duplicate a vector as columns

[8]:
```
v * ones(Int,1,3)
```

[8]:
```
3×3 Array{Int64,2}:
 1  1  1
 2  2  2
 3  3  3
```

3.2 Duplicate a vector as rows

[9]:
```
ones(Int,3,1) * transpose(v)
```

[9]:
```
3×3 Array{Int64,2}:
 1  2  3
 1  2  3
```

```
 1  2  3
```

3.3 Extract column 1 from a matrix

[10]:
```
m * [1, 0, 0]
```

[10]:
```
3-element Array{Int64,1}:
 1
 4
 7
```

3.4 Extract row 2 from a matrix

[11]:
```
[0 1 0;] * m
```

[11]:
```
1×3 Array{Int64,2}:
 4  5  6
```

4 Substitutions

4.1 Swap x and y of a vector

[12]:
```
[0 1 0; 1 0 0; 0 0 1] * v
```

[12]:
```
3-element Array{Int64,1}:
 2
 1
 3
```

4.2 Swap rows 1 and 2 of a matrix

[13]:
```
[0 1 0; 1 0 0; 0 0 1] * m
```

[13]:
```
3×3 Array{Int64,2}:
 4  5  6
 1  2  3
 7  8  9
```

4.3 Reverse rows

[14]:
```
r = [0 0 1; 0 1 0; 1 0 0]
```

[14]:
```
3×3 Array{Int64,2}:
 0  0  1
 0  1  0
 1  0  0
```

[15]:
```
r * m
```

[15]:
```
3×3 Array{Int64,2}:
 7  8  9
 4  5  6
 1  2  3
```

4.4 Reverse columns

[16]:
```
m * r
```

[16]:
```
3×3 Array{Int64,2}:
 3  2  1
 6  5  4
 9  8  7
```

4.5 Swap columns 1 and 2 of a matrix

(Remember when they said matrix multiplciation is not commutative?)

[17]:
```
m * [0 1 0; 1 0 0; 0 0 1]
```

[17]:
```
3×3 Array{Int64,2}:
 2  1  3
 5  4  6
 8  7  9
```

5 Permutations

Generically, all of the above exchanges are called *permutations*. For a $n \times n$ matrix there are $n!$ such permutation ("P") matrices. For $n = 3$, these are:

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

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

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

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

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

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

For all permutation matrices,

$$P^{-1} = P^T. \tag{4}$$

Also notice that these matrices compose to each other.

6 Insertions

6.1 Append an empty row to the bottom

[18]:
```
[1 0 0; 0 1 0; 0 0 1; 0 0 0] * m
```

[18]:
```
4×3 Array{Int64,2}:
 1  2  3
 4  5  6
 7  8  9
 0  0  0
```

6.2 Insert an empty row to the top

[19]:
```
[0 0 0; 1 0 0; 0 1 0; 0 0 1] * m
```

[19]:
```
4×3 Array{Int64,2}:
 0  0  0
 1  2  3
 4  5  6
 7  8  9
```

6.3 Append an empty column to the right

[20]:
```
m * [1 0 0 0; 0 1 0 0; 0 0 1 0]
```

[20]:
```
3×4 Array{Int64,2}:
 1  2  3  0
 4  5  6  0
 7  8  9  0
```

6.4 Insert an empty column to the left

[21]:
```
m * [0 1 0 0; 0 0 1 0; 0 0 0 1]
```

[21]:
```
3×4 Array{Int64,2}:
 0  1  2  3
 0  4  5  6
 0  7  8  9
```

7 Deletions

7.1 Clear rows 2 and 3 from a matrix

[22]:
```
[1 0 0; 0 0 0; 0 0 0] * m
```

[22]:
```
3×3 Array{Int64,2}:
 1  2  3
 0  0  0
 0  0  0
```

7.2 Clear columns 2 and 3 from a matrix

[23]:
```
m * [1 0 0; 0 0 0; 0 0 0]
```

[23]:
```
3×3 Array{Int64,2}:
 1  0  0
 4  0  0
 7  0  0
```

7.3 Drop the right column

[24]:
```
m * [1 0; 0 1; 0 0]
```

[24]:
```
3×2 Array{Int64,2}:
 1  2
 4  5
 7  8
```

7.4 Drop the center column

[25]:
```
m * [1 0; 0 0; 0 1]
```

[25]:
```
3×2 Array{Int64,2}:
 1  3
 4  6
 7  9
```

7.5 Drop the left column

[26]:
```
m * [0 0; 1 0; 0 1]
```

[26]:
```
3×2 Array{Int64,2}:
 2  3
 5  6
 8  9
```

7.6 Drop the bottom row

[27]:
```
[1 0 0; 0 1 0] * m
```

[27]:
```
2×3 Array{Int64,2}:
 1  2  3
 4  5  6
```

7.7 Drop the center row

[28]:
```
[1 0 0; 0 0 1] * m
```

[28]:
```
2×3 Array{Int64,2}:
 1  2  3
 7  8  9
```

7.8 Drop the top row

[29]:
```
[0 1 0; 0 0 1] * m
```

[29]:
```
2×3 Array{Int64,2}:
 4  5  6
 7  8  9
```

You can achieve the same results by composing transposition with a single drop function. For example, transpose rows 2 and 3, then drop row 3. This has the same effect as dropping the center row. Remember, order of operations matters.

[30]:
```
[1 0 0; 0 1 0] * [1 0 0; 0 0 1; 0 1 0] * m
```

[30]:
```
2×3 Array{Int64,2}:
 1  2  3
 7  8  9
```

Another way to think about this is that the "drop row 3" and "transpose rows 2 and 3" matrices compose, which is identical to the "drop row 2" matrix.

[31]:
```
[1 0 0; 0 1 0] * [1 0 0; 0 0 1; 0 1 0]
```

```
[31]: 2×3 Array{Int64,2}:
 1  0  0
 0  0  1
```

8 Affine Transforms (2D)

Suppose you begin with Bv. Then an affine transform ABv occurs in global coordinate space, and an affine transform BCv occurs in the object's local coordinate space. See [http://math.hws.edu/graphicsbook/c2/s3.html].

8.1 Scaling

$$S_{a,b} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

8.2 Rotation

We use Ptolemy's Sum and Difference Formulae

$$\cos\alpha \pm \beta = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta \tag{6}$$
$$\sin\alpha \pm \beta = \cos\alpha\sin\beta \pm \sin\alpha\cos\beta \tag{7}$$

in order to increase the rotation of a vector from some initial angle α by β:

$$R_\beta \begin{bmatrix} k\cdot\cos\alpha \\ k\cdot\sin\alpha \\ 1 \end{bmatrix} = R_\beta k \begin{bmatrix} k\cdot\cos\alpha \\ k\cdot\sin\alpha \\ 1/k \end{bmatrix} = k \begin{bmatrix} \cos\alpha+\beta \\ \sin\alpha+\beta \\ 1/k \end{bmatrix} = \begin{bmatrix} k\cdot\cos\alpha+\beta \\ k\cdot\sin\alpha+\beta \\ 1 \end{bmatrix}. \tag{8}$$

The matrix which delivers this behavior is

$$R_\Theta = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{9}$$

8.3 Translation

$$T_{a,b} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

9 Gaussian Elimination

Augment the coefficient matrix with the b column vector.

$$Ax = b \rightarrow \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_n \end{bmatrix} \tag{11}$$

Subtract rows from one another, rearranging rows as needed, until the matrix is in upper triangular form. See [http://mathworld.wolfram.com/GaussianElimination.html].

[32]:
```
g = [9 3 4 7; 4 3 4 8; 1 1 1 3]
```

[32]:
```
3×4 Array{Int64,2}:
 9  3  4  7
 4  3  4  8
 1  1  1  3
```

The bottom row is very convenient, so switch the first and third rows.

[33]:
```
[0 0 1; 0 1 0; 1 0 0] * g
```

[33]:
```
3×4 Array{Int64,2}:
 1  1  1  3
 4  3  4  8
 9  3  4  7
```

Subtract four times the first row from the second.

[34]:
```
[1 0 0; -4 1 0; 0 0 1] * [0 0 1; 0 1 0; 1 0 0] * g
```

[34]:
```
3×4 Array{Int64,2}:
 1   1  1   3
 0  -1  0  -4
 9   3  4   7
```

This gives a "nice" result of one variable that we could have used to reduce our problem immediately, but we would still have to solve for x and z.

Subtract nine times the first row from the third.

[35]:
```
[1 0 0; 0 1 0; -9 0 1] * [1 0 0; -4 1 0; 0 0 1] * [0 0 1; 0 1 0; 1 0 0] * g
```

[35]:
```
3×4 Array{Int64,2}:
 1   1   1    3
 0  -1   0   -4
 0  -6  -5  -20
```

Subtract six times the second row from the third.

[36]:
```
[1 0 0; 0 1 0; 0 -6 1] * [1 0 0; 0 1 0; -9 0 1] * [1 0 0; -4 1 0; 0 0 1] * [0 0␣
↪1; 0 1 0; 1 0 0] * g
```

[36]:
```
3×4 Array{Int64,2}:
 1   1   1   3
 0  -1   0  -4
 0   0  -5   4
```

Now we see $-5z = 4$, $-y + 0z = -4$, and $x + y + z = 3$, which give $z = -4/5$, $y = 4$, and $x = -1/5$.

9.1 Left Division Operator

Julia contains a built-in "left division operator" (\) for this purpose.

[37]:
```julia
round.([9 3 4; 4 3 4; 1 1 1] \ [7; 8; 3],digits=2)
```

[37]:
```
3-element Array{Float64,1}:
 -0.2
  4.0
 -0.8
```

9.2 Units

This is a general reminder to be very cautious with units. Example TMP at [http://linear.ups.edu/html/section-SSLE.html] gives a system of linear equations

$$\begin{bmatrix} \text{7 kg/batch of raisins} & \text{6 kg/batch of raisins} & \text{2 kg/batch of raisins} \\ \text{6 kg/batch of peanuts} & \text{4 kg/batch of peanuts} & \text{5 kg/batch of peanuts} \\ \text{2 kg/batch of chocolate} & \text{5 kg/batch of chocolate} & \text{8 kg/batch of chocolate} \end{bmatrix} \begin{bmatrix} b \text{ batches of bulk trail mix} \\ s \text{ batches of standard trail mix} \\ f \text{ batches of fancy trail mix} \end{bmatrix} = \begin{bmatrix} \text{380 kg of raisins} \\ \text{500 kg of peanuts} \\ \text{620 kg of chocolate} \end{bmatrix} \tag{12}$$

This is straightfoward to solve, but the question is looking for answers in kilograms.

[38]:
```julia
round.([7 6 2; 6 4 5; 2 5 8] \ [380; 500; 620])
```

[38]:
```
3-element Array{Float64,1}:
 20.0
 20.0
 60.0
```

To get to the answer, we needed to know 15 kg = 1 batch, so $\begin{bmatrix} b \\ s \\ f \end{bmatrix}$ should actually be 15× the amount calculated.

Again, what we solved was

$$[\text{kg/batch}]\ [\text{batches}] = [\text{kg}] \tag{13}$$

when the question really needs

$$[\text{units}]\,[\text{kg}] = [\text{kg}]\,. \tag{14}$$

[39]:
```
round.(([7 6 2; 6 4 5; 2 5 8]) / 15 \ [380; 500; 620])
```

[39]:
```
3-element Array{Float64,1}:
 300.0
 300.0
 900.0
```

10 Inversions

10.1 Not all matrices are invertible

Let us assume, for the sake of contradiction, that there exists some matrix A^{-1} for the matrix $A = \begin{bmatrix} 1 & 3 \\ 2 & 6 \end{bmatrix}$. Observe that $\begin{bmatrix} 1 & 3 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 3 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. If A^{-1} exists, then $A^{-1}A \begin{bmatrix} 3 \\ -1 \end{bmatrix} = A^{-1} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, which implies $\begin{bmatrix} 3 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. This is a contradiction, therefore it must not be true that all matrices are invertible.

(The key to this proof is to find some x such that $Ax = 0$).

See [https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/lecture-3-multiplication-and-inverse-matrices/].

10.2 Invert a Matrix

Let $A = \begin{bmatrix} 1 & 3 \\ 2 & 7 \end{bmatrix}$. We want to find $A^{-1} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$ such that $AA^{-1} = I$. Then

$$\begin{bmatrix} 1 & 3 \\ 2 & 7 \end{bmatrix} \begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{15}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 7 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{16}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 7 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{17}$$

which produces the linear equations

$$1a + 3b = 1 \tag{18}$$

$$2a + 7b = 0 \tag{19}$$

$$1c + 3d = 0 \tag{20}$$

$$2c + 7d = 1 \tag{21}$$

which we can solve!

[40]:
```
[1 3; 2 7] \ [1; 0]
```

[40]:
```
2-element Array{Float64,1}:
  7.0
 -2.0
```

[41]:
```
[1 3; 2 7] \ [0; 1]
```

[41]:
```
2-element Array{Float64,1}:
 -3.0
  1.0
```

[42]:
```
[1 3; 2 7] * hcat([7; -2], [-3, 1])
```

[42]:
```
2×2 Array{Int64,2}:
 1  0
 0  1
```

10.3 Gauss-Jordan

Gauss-Jordan lets you do this by hand by performing elimination "downwards" and then again "upwards" with your coefficient matrix augumented with a complete identity. It looks like:

$$E\begin{bmatrix} a & c & 1 & 0 \\ b & d & 0 & 1 \end{bmatrix} = E\begin{bmatrix} A & I \end{bmatrix} = \begin{bmatrix} I & A^{-1} \end{bmatrix} \tag{22}$$