This document is intended to be a quick reference with some useful Python commands to help with the homework assignments, exams, and labs involved with ECE 470: Introduction to Robotics at the University of Illinois at Urbana-Champaign. For further reference, check out the official documentation for the various packages talked about here.

- Numpy: `https://numpy.org/doc/stable/reference/arrays.html`
- Scipy: `https://docs.scipy.org/doc//scipy/reference/index.html#scipy-api`

Importing Libraries

In order to use the libraries talked about in this document, you'll need to include them in your code. This can be done using the `import` command in Python. Add the following lines to the beginning of your code to import everything the same way I have.

```
import numpy as np
from scipy.linalg import expm, logm
```

Create an Array

This first section describes a few useful methods for creating arrays in Python.

np.array()

Creates an array with desired entries. It's done as a list of lists, where the innermost lists make up the rows and they are stacked in sequential order. Below are a few examples:

$$\texttt{np.array([[a, b, c]])} = \begin{bmatrix} a & b & c \end{bmatrix}$$

$$\texttt{np.array([[a], [b], [c]])} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$\texttt{np.array([[a, b], [c, d]])} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

np.zeros()

Creates an array of all zeros. If the input is just an integer, say n the output will be an array of size $1 \times n$. If the integer is a tuple (or a list of numbers), say $m \times n$, the output will be an array of size $m \times n$.

$$\texttt{np.zeros(4)} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\texttt{np.zeros((3,2))} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

np.eye()

Creates a square identity matrix. The input is an integer that denotes the size of the matrix.

$$\texttt{np.eye(3)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

np.block()

Creates a block matrix, allowing you to build a matrix of other matrices. For example, let's say we wanted to code up a transformation matrix using the following rotation matrix and position vector:

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \quad \text{where} \quad R = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad p = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

We could do this using `np.array()`, but `np.block()` makes life a little easier. The following code is an example of how this can be done:

```
R = np.array([[0, 1, 0], [-1, 0, 0], [0, 0, 1]])
p = np.array([[2], [0], [1]])
zeros = np.zeros((1,3))
T = np.block([[R, p], [zeros, 1]])
```

which would give us the following result.

$$T = \begin{bmatrix} 0 & 1 & 0 & 2 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note: It's up to you to make sure that the dimensions all match up! If you try to create a matrix using `np.block()` with dimensions that don't line up it will cause an error.

Array Operations

The next couple methods can be used to conduct matrix operations. For each of them, I'll use the following example matrices to help explain how they work:

$$A = \begin{bmatrix} 1 & 0 & -3 \\ 2 & 2 & 1 \\ 0 & -1 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 2 & -1 & 1 \\ 5 & 0 & 7 \\ 1 & 0 & -1 \end{bmatrix} \quad C = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$$

np.linalg.inv()

Takes the inverse of a matrix.

$$\texttt{np.linalg.inv(A)} = A^{-1} = \begin{bmatrix} 0.6 & 0.2 & 0.4 \\ -0.5333 & 0.2667 & -0.4667 \\ -0.1333 & 0.0667 & 0.1333 \end{bmatrix}$$

$$\texttt{np.linalg.inv(B)} = B^{-1} = \begin{bmatrix} 0 & 0.8333 & 0.5833 \\ -1 & 0.25 & 0.75 \\ 0 & 0.0833 & -0.4167 \end{bmatrix}$$

np.linalg.norm()

Takes the norm of a vector or matrix.

$$\texttt{np.linalg.norm(A)} = ||A|| = 6$$

$$\texttt{np.linalg.norm(C)} = ||C|| = 2.4495$$

np.linalg.det()

Takes the determinant of a matrix.

$$\texttt{np.linalg.norm(A)} = \det(A) = 15$$

$$\texttt{np.linalg.norm(B)} = \det(B) = -12$$

np.transpose()

Takes the transpose of a vector or matrix.

$$\texttt{np.transpose(B)} = B^T = \begin{bmatrix} 2 & 5 & 1 \\ -1 & 0 & 0 \\ 1 & 7 & -1 \end{bmatrix}$$

$$\texttt{np.transpose(C)} = C^T = \begin{bmatrix} 1 & 2 & -1 \end{bmatrix}$$

This can also be done by adding `.T` to the end of the matrix.

$$\texttt{B.T} = B^T = \begin{bmatrix} 2 & 5 & 1 \\ -1 & 0 & 0 \\ 1 & 7 & -1 \end{bmatrix}$$

$$\texttt{C.T} = C^T = \begin{bmatrix} 1 & 2 & -1 \end{bmatrix}$$

np.matmul()

Multiplies two matrices together.

$$\texttt{np.matmul(A,B)} = AB = \begin{bmatrix} -1 & -1 & 4 \\ 15 & -2 & 15 \\ -1 & 0 & -11 \end{bmatrix}$$

$$\texttt{np.matmul(A,C)} = AC = \begin{bmatrix} 4 \\ 5 \\ -6 \end{bmatrix}$$

This can also be done using the `@` operator.

$$\texttt{A @ B} = AB = \begin{bmatrix} -1 & -1 & 4 \\ 15 & -2 & 15 \\ -1 & 0 & -11 \end{bmatrix}$$

$$\texttt{A @ C} = AC = \begin{bmatrix} 4 \\ 5 \\ -6 \end{bmatrix}$$

np.cross()

Takes the cross product of two vectors or matrices.

$$\texttt{np.cross(A, B)} = A \times B = \begin{bmatrix} -3 & -7 & -1 \\ 14 & -9 & -10 \\ 1 & 4 & 1 \end{bmatrix}$$

For column vectors (tall, not wide), you'll need to specify the axis, otherwise you'll get an error.

$$\texttt{np.cross(A, C, axis=0)} = A \times C = \begin{bmatrix} -2 & 0 & -9 \\ 1 & -1 & 1 \\ 0 & -2 & -7 \end{bmatrix}$$

scipy.linalg.expm()

Solves matrix exponential.

$$\texttt{expm(B)} = e^B = \begin{bmatrix} -0.8904 & -1.0595 & -2.7487 \\ 8.6294 & -3.4853 & -0.2000 \\ 0.5834 & -0.4760 & -0.2606 \end{bmatrix}$$

scipy.linalg.logm()

Solves matrix logarithm.

$$\texttt{logm(A)} = \log(A) = \begin{bmatrix} 0.2657 & -0.2637 & -1.1695 \\ 1.1313 & 0.9193 & 0.8294 \\ 0.1758 & -0.3019 & 1.5231 \end{bmatrix}$$