Lab 12: Eigen

1. Quickstart [15 minutes]: We will go over the following Eigen quides as a class
    a. https://eigen.tuxfamily.org/dox-3.3/group__QuickRefPage.html
    b. https://eigen.tuxfamily.org/dox-3.3/group__TutorialMatrixArithmetic.html
    c. https://eigen.tuxfamily.org/dox-3.3/group__TutorialReductionsVisitorsBroadcasting.html
    d. We will now setup a GDB plugin that will make debugging with Eigen a lot better! Open a terminal and run the following commands:

```
git clone https://github.com/dmillard/eigengdb  # downloads the code we need
cd eigengdb
pip install --user .  # installs the python code which will let you print Eigen matrices
python bin/eigengdb_register_printers
```

2. Basic operations [20 minutes]: Copy the code below into a file called `lab12_eigen.cpp` and fill it in as described in the comments

```
#include <eigen3/Eigen/Dense>
#include <iostream>

int main()
{
    // Constructors, create the following, using the letter as the variable name
    //  a = vector of 2 integers [0, 1]
    //  b = vector of 3 doubles [0.1, -0.1, 3.0]
    //  c = vector of 4 doubles [0.2, -0.2, 0.0, 1.0]
    //  d = 2x2 matrix of integers [[0, 1], [2, 3]]
    //  e = 3x3 matrix of doubles where t = pi/2
    //    [[cos(t), -sin(t), 0],
    //     [sin(t),  cos(t),  0],
    //     [0,       0,       1]]
    //  f = 5x2 matrix of doubles
    //    [[0.1,  0.1],
    //     [0.2,  0.0],
    //     [0.3, -0.1],
    //     [0.4,  0.0],
    //     [0.5,  0.1]]
```

```
    // these assert statements will check that you initilized 'e' and 'f' correctly
    assert(std::abs(e.sum() - 1) < 0.01);
    assert(std::abs(f.sum() - 1.6) < 0.01);

    // create a 3x3 identity matrix called g of type double

    // Now, translate the following into code. Make sure you use the specified variables!

    //   h = g + e


    //   i = d^T
    //   here ^T means transpose


    //   j = |b|, the element-wise absolute value


    //  k = h + j


    // set l to be the mean of each column in 'f'


    // m = e * e^T


    // n = a + 5; Make sure you think about whether to use "arrays" or "matrices"


    // o = k + m;


    // get the first row o and store it in the variable o_first_row


    // use head to get the first 3 elements of the first column in f,
    // and store it in the variable f_head


    // compute the dot product of  f_head and o_first_row and store it in a variable p


    // you will be asked about the values of some of these variables on the quiz!
    std::cout << "p: " << p << '\n';

    return 0;
}
```

3. Rollouts of Pendulum Dynamics [30 minutes]: In this problem you will use Eigen to represent the dynamics of a swinging pendulum, and roll-out the dynamics. The state space is the position $x = [\theta, \dot{\theta}]$ and the dynamics are (for small θ):

$$x_{t+1} = x_t + \Delta t \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{bmatrix} x_t$$

Copy the following code into a file called `lab12_pendulum.cpp` and fill in the "your code here" sections.

```
#include <fstream>
#include <eigen3/Eigen/Dense>
#include <cmath>

int main()
{
    Eigen::IOFormat CSVFormat(Eigen::StreamPrecision, Eigen::DontAlignCols, ", ", ", "\n");
    std::ofstream outfile{"pendulum_output.csv"};
    if (!outfile.good())
    {
        throw std::runtime_error("failed to open file for writing!");
    }

    float const g = 9.8;
    float const l = 0.1;
    float const dt = 0.005; // this is $\Delta t$
    // Create a vector for [x, xdot] and initialize it to [pi/4, 0]


    // Create the matrix here, it should be 2x2, and should have the values:
    // [ 0     1 ]
    // [ -g/l  0 ]
    // --- Your code here

    // ---

    // roll out the dynamics using the equation in the lab document
    for (int i{0}; i < 100; ++i)
    {
        // write the current x to the outfile
        outfile << x.format(CSVFormat) << '\n';

        // now update x based on the dynamics
        // --- Your code here
        // ---
    }

    // to visualize the output, run `python viz_pendulum.py` using F5 in VSCode.
```

```
    return 0;
}
```