P4: Matrices and Vectors

Random Vector - 23/02/21

To create a vector of random double I had to use the random and vector libraries, this gave me the functions I required.

When initially initialising dre I used the default_random. However, due to issues with my MinGW installation this was deterministic. To fix this I instead initialised it with the time to get a true random string.

```
#include <random>
#include <ctime>
#include <vector>

int main()
{
    std::default_random_engine dre(std::time(NULL));
    std::uniform_real_distribution<double> dr(10, 20);

    std::vector<double> v;
    int vec_size;

    cout << "Enter the size of your vector." << endl;
    cin >> vec_size;
    for (int i = 0; i < vec_size; i++)
    {
        v.push_back(dr(dre));
    }

    for (int i = 0; i < vec_size; i++)
    {
        cout << v[i] << ", ";
    }
}
```

Matrix - 23/02/21

To create a matrix I used a vector of vectors. This allowed me to make it easily scalable. My implementation means it has to be a square matrix but as this is intended for a tridiagonal matrix this is not an issue.

To correctly initialise the matrix I had to change the code. I used a helpful suggestion I found on stackoverflow to initialise it.

```
    int mat_size;
    cout << "Enter the size of your Matrix." << endl;
    cin >> mat_size;

    std::vector<std::vector<double>> M(mat_size); //Luchian Grigore at
https://stackoverflow.com/questions/12375591/vector-of-vectors-to-create-matrix
    for (int i = 0; i < mat_size; i++)
    {
        M[i].resize(mat_size);
    }

    for (int i = 0; i < mat_size; i++)
    {
        for (int j = 0; j < mat_size; j++)
        {
            M[i][j] = dr(dre);
        }
    }

    for (int i = 0; i < mat_size; i++)
    {
        for (int j = 0; j < mat_size; j++)
        {
            cout << M[i][j] << " ";
        }
        cout << endl;
    }
```

Linear Equation Solver - 23/02/21

To convert the function from c to c++ I converted all the the pointers to vectors. I also found it useful to change the naming convention to something that is more intuitive.

```
std::vector<double> TridiagonalSolve(double c1, std::vector<double> &diag,
std::vector<double> &vect, int n)
{
    int i;
    std::vector<double> off_diag(n);
    std::vector<double> output(n);
    double id;

    // Set the off diagonal elements
    for (i = 0; i < n; i++)
    {
        off_diag[i] = c1;
    }

    //forward bit
    off_diag[0] /= diag[0]; //if /0 rearrange equations
    vect[0] /= diag[0];

    for (i = 1; i < n; i++)
    {
        id = diag[i] - off_diag[i - 1] * c1;
        off_diag[i] /= id; // Last value calculated is redundant.
        vect[i] = (vect[i] - vect[i - 1] * c1) / id;
    }

    // Now back substitute.
    output[n - 1] = vect[n - 1];

    for (i = n - 2; i >= 0; i--)
    {
        output[i] = vect[i] - off_diag[i] * output[i + 1];
    }

    return output;
}
```