1. **Systems of Linear Equations:**

   a. Write a program that takes a matrix, A, and uses Gaussian elimination to decompose the matrix into lower and upper triangular matrices, L and U. You may check this against the built-in function, but need to use your own code to get L and U.

   b. A system of equations is shown below, where A is the 3x3 coefficient matrix. Using your code, solve for x. The solution is $x = [1,1,1]^T$. Is this what you get using your code? Compare using a built-in function.

$$\begin{bmatrix} 3 & 2 & 4 \\ 8 & -6 & -8 \\ -1 & 2 & 3 \end{bmatrix} x = \begin{bmatrix} 9 \\ -6 \\ 4 \end{bmatrix}$$

   c. What is the condition number for the matrix, A? What is the determinant? You may use built-in functions.

   d. Adjust the coefficient matrix slightly. Do you get the same solution? You may use your code or built-in functions.

Ans:

a) Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <lapacke.h>

void LU_T_matrix(int n_rows, int n_cols, double
equation_A[n_rows][n_cols+1]){
   double lower_T_matrix[n_rows][n_cols];
   for(int i=0; i<(n_rows); i++){
       for(int j=0; j<(n_cols); j++){
```

```
            if (j>i){
                double ratio = equation_A[j][i]/equation_A[i][i];
                for(int k=0; k<n_cols+1; k++){
                    equation_A[j][k] = equation_A[j][k] -
ratio*equation_A[i][k];
                }
                lower_T_matrix[j][i] = ratio;
            }
            else if(i==j){
                lower_T_matrix[i][j] = 1;
            }
            else{
                lower_T_matrix[j][i] = 0;
            }
        }
    }
    double det = 1;
    printf("Upper Triangular Matrix\n");
    for(int i=0; i<n_rows; i++){
        for(int j=0; j<n_cols; j++){
            if(i==j){
                det*=equation_A[i][j];
            }
            printf("%lf ", equation_A[i][j]);
        }
        printf("\n");
    }
    printf("Lower Triangular Matrix\n");
    for(int i=0; i<n_rows; i++){
        for(int j=0; j<n_cols; j++){
            printf("%lf", lower_T_matrix[i][j]);
        }
        printf("\n");
    }
    printf("Determinant of the matrix: %lf\n", det);
}

void solve_X(int n_rows, int n_cols, double
equation_A[n_rows][n_cols+1]){
    printf("Values in X\n");
```

```
    double *val_array = (double *)malloc(n_cols * sizeof(int));;
    for(int i=n_rows-1; i>=0; i--){
        double sum=0;
        for(int j=n_cols-1; j>i; j--){
            sum+=val_array[j]*equation_A[i][j];
        }
        val_array[i] = (equation_A[i][n_cols]-sum)/equation_A[i][i];
    }
    for(int i=0; i<n_cols; i++){
        printf("%lf\n", val_array[i]);
    }
}

int main(int argc, char *argv[]){
    int n_rows = argc > 1 ? atoi(argv[1]) : 3, n_cols = argc > 2 ?
atoi(argv[2]) : 3;
    double B[n_rows];
    for(int i=0; i<n_rows; i++){
        scanf("%lf", &B[i]);
    }
    double A[n_rows * n_cols],equation_A[n_rows][n_cols+1];

    char input[n_cols*2], *token;
    for (int i=0; i<n_rows; i++){
        int j = 0;
        scanf(" %[^\n]", input);
        token = strtok(input, " ");

        // Loop through the tokens and print each one
        while (token != NULL) {
            equation_A[i][j] = atof(token);
            A[j*n_rows+i] = atof(token);
            j+=1;
            token = strtok(NULL, " ");  // Get the next token
        }
    }

    for(int i=0; i<n_rows; i++){
        equation_A[i][n_cols] = B[i];
    }
```

```
    LU_T_matrix(n_rows, n_cols, equation_A);
    solve_X(n_rows, n_cols, equation_A);

    int nhrs = 1, ipiv[n_rows], info;

    dgesv_(&n_rows, &nhrs, A, &n_rows, ipiv, B, &n_cols, &info);

    if(info == 0){
        printf("Solution using dgesv from LAPACK\n");
        for(int i=0; i<n_rows; i++){
            printf("%lf\n", B[i]);
        }
    }
    else{
        printf("Solution not found\n");
    }

    double rcond = 0.0, work[4*n_rows], anorm=15;
    int iwork[n_rows];
    char norm = '1';

    dgecon_(&norm, &n_rows, A, &n_rows, &anorm, &rcond, work, iwork,
&info);

    if(info == 0){
        printf("1 Norm Condition number of the matrix: %lf\n",
1/rcond);
    }
    else{
        printf("Condition number not found\n");
    }

    return 0;
}
```

b) Output:

```
(base) ➜  assignment2 git:(main) ✗ gcc MatrixGaussialElemination.c -o executables/MatrixGaussialElemination -llapack -lblas -l
m && ./executables/MatrixGaussialElemination 3 3
9 -6 4
3 2 4
8 -6 -8
-1 2 3
Upper Triangular Matrix
3.000000 2.000000 4.000000
0.000000 -11.333333 -18.666667
0.000000 0.000000 -0.058824
Lower Triangular Matrix
1.0000000.0000000.000000
2.6666671.0000000.000000
-0.333333-0.2352941.000000
Determinant of the matrix: 2.000000
Values in X
1.000000
1.000000
1.000000
Solution using dgesv from LAPACK
1.000000
1.000000
1.000000
```

c)

```
(base) ➜  assignment2 git:(main) ✗ gcc MatrixGaussialElemination.c -o executables/MatrixGaussialElemination -llapack -lblas -l
m && ./executables/MatrixGaussialElemination 3 3
9 -6 4
3 2 4
8 -6 -8
-1 2 3
Upper Triangular Matrix
3.000000 2.000000 4.000000
0.000000 -11.333333 -18.666667
0.000000 0.000000 -0.058824
Lower Triangular Matrix
1.0000000.0000000.000000
2.6666671.0000000.000000
-0.333333-0.2352941.000000
Determinant of the matrix: 2.000000
Values in X
1.000000
1.000000
1.000000
Solution using dgesv from LAPACK
1.000000
1.000000
1.000000
1 Norm Condition number of the matrix: 735.000000
```

d) Since the condition number of the matrix is big, whenever there is a slight change in the coefficient matrix, there will be quite a big change in the solution output.

```
(base) ➜  assignment2 git:(main) ✗ gcc MatrixGaussialElemination.c -o executables/MatrixGaussialElemination -llapack -lblas -l
m && ./executables/MatrixGaussialElemination 3 3
9 -6 4
3.1 2 4
8 -6 -8
-1 2 3
Upper Triangular Matrix
3.100000 2.000000 4.000000
0.000000 -11.161290 -18.322581
0.000000 0.000000 -0.052023
Lower Triangular Matrix
1.0000000.0000000.000000
2.5806451.0000000.000000
-0.322581-0.2369941.000000
Determinant of the matrix: 1.800000
Values in X
1.111111
1.888889
0.444444
Solution using dgesv from LAPACK
1.111111
1.888889
0.444444
```

2. **Systems of Linear Equations:** The equation to create the Hilbert matrix is defined below.

$$H_n = \left(\frac{1}{i+j+1} \right) i, j = 0, 1, \ldots, n-1$$

   a. (3 points) Using this matrix, display a Hilbert matrix of the order of 5.

   b. (4 points) What is the condition number of the 9x9 Hilbert matrix, H_9.

   c. (3 points) Solve $H_9x = [1,1,1,1,1,1,1,1,1]^T$. Then change the first component of the right-hand side to 1.01 and solve again. Which component of x is most changed?

Ans:

Code:

```
import argparse

parser = argparse.ArgumentParser(description='Hilbert matrix')
parser.add_argument('--size', type=int, default=9, help='Size of the
Hilbert matrix')
parser.add_argument('sols', metavar='B', type=float, nargs='+',
help='Solutions of the system of equations')
args = parser.parse_args()

size = 5 if not args.size else args.size

H_custom = np.array([[1 / (i + j + 1) for j in range(size)] for i in
range(size)])

print("Hilbert matrix of size", size)
for row in H_custom:
   for elem in row:
       print("{:.4f}".format(elem), end=" ")
   print()

B = np.array(args.sols)

X_solution_custom = np.linalg.solve(H_custom, B)
```

```
cond_number_custom = np.linalg.cond(H_custom)
print("Condition number of the Hilbert matrix of size", size, "is",
cond_number_custom)

print("Solution of the system of equations:")
for i in X_solution_custom:
   print("{:.4f}".format(i))
```

a)

```
(base) ➜  assignment2 git:(main) ✗ python3 hilbertmatrix.py --size=5 1 1 1 1 1
Hilbert matrix of size 5
1.0000 0.5000 0.3333 0.2500 0.2000
0.5000 0.3333 0.2500 0.2000 0.1667
0.3333 0.2500 0.2000 0.1667 0.1429
0.2500 0.2000 0.1667 0.1429 0.1250
0.2000 0.1667 0.1429 0.1250 0.1111
Condition number of the Hilbert matrix of size 5 is 476607.2502425855
Solution of the system of equations:
5.0000
-120.0000
630.0000
-1120.0000
630.0000
```

b)

```
(base) ➜  assignment2 git:(main) ✗ python3 hilbertmatrix.py --size=9 1 1 1 1 1 1 1 1 1
Hilbert matrix of size 9
1.0000 0.5000 0.3333 0.2500 0.2000 0.1667 0.1429 0.1250 0.1111
0.5000 0.3333 0.2500 0.2000 0.1667 0.1429 0.1250 0.1111 0.1000
0.3333 0.2500 0.2000 0.1667 0.1429 0.1250 0.1111 0.1000 0.0909
0.2500 0.2000 0.1667 0.1429 0.1250 0.1111 0.1000 0.0909 0.0833
0.2000 0.1667 0.1429 0.1250 0.1111 0.1000 0.0909 0.0833 0.0769
0.1667 0.1429 0.1250 0.1111 0.1000 0.0909 0.0833 0.0769 0.0714
0.1429 0.1250 0.1111 0.1000 0.0909 0.0833 0.0769 0.0714 0.0667
0.1250 0.1111 0.1000 0.0909 0.0833 0.0769 0.0714 0.0667 0.0625
0.1111 0.1000 0.0909 0.0833 0.0769 0.0714 0.0667 0.0625 0.0588
Condition number of the Hilbert matrix of size 9 is 493153755941.02344
```

c) output 1:

```
Solution of the system of equations:
9.0000
-719.9973
13859.9532
-110879.6634
450448.7578
-1009005.4503
1261257.0587
-823678.2166
218789.5579
```

When we change the input to 1.01, 1, 1 , 1 , 1, 1, 1, 1, 1;
output 2:

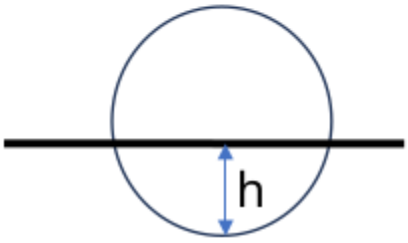
```
Solution of the system of equations:
9.8100
-752.3972
14275.7526
-113374.4593
458556.8426
-1024140.5389
1277473.2225
-832944.5946
220977.4524
```

The X7 component has changed the most with the difference of 16216.16379999998 in magnitude.

3. **Roots:** The weight of a sphere of density, d, and radius, r, is $\frac{4}{3}\pi r^3 d$. When put in water, buoyancy acts to balance the weight of the sphere. The volume of a sphere segment is $\frac{1}{3}\pi(3rh^2-h^3)$. Find the depth to which a sphere of density 0.5kg/m^3 sinks in water as a fraction of its radius.
   The formula for buoyancy is

$$F_b=-\rho gV$$

   Use 1000kg/m^3 as the density of water.



Ans:

Code:

```
/*
When we balance the weight of the ball and the buoyancy force; we end up
with the equation;
1000*(x^3) - 3000(x^2) + 2 = 0; where x = h/r where h is the height of the
ball in the water and r is the radius of the ball.
*/

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
```

```c
double func_eval(double x){
 return 1000*pow(x,3)-3000*pow(x,2)+2;
}

double func_eval_derivative(double x){
 return 3000*pow(x,2)-6000*x;
}

int main(){
   double relative_error_tolerance = 0.00625, x = 4,
abs_relative_approx_error = relative_error_tolerance+1, next_x;

   while(abs_relative_approx_error>relative_error_tolerance){
       double func_x = func_eval(x), func_x_derivative =
func_eval_derivative(x);
       if (func_x_derivative == 0){
           printf("Derivative is zero. Exiting\n");
           exit(0);
       }
       next_x = x - (func_x/func_x_derivative);
       abs_relative_approx_error = fabs((next_x-x)/next_x);
       x = next_x;
   }

   printf("Root calculated in Newton's method with relative error
tolerance of %f is %f\n", relative_error_tolerance, x);

   printf("The height of the ball in the water is %f times the radius of
the ball\n", x);
   printf("h=%f*r\n", x);
}
```

Output:

```
(base) ➜  assignment2 git:(main) ✗ gcc floatingBall.c -o executables/floatingBall -lm && ./executables
/floatingBall
Root calculated in Newton's method with relative error tolerance of 0.006250 is 2.999780
The height of the ball in the water is 2.999780 times the radius of the ball
h=2.999780*r
```

4. **Interpolation:** Use the data and divided differences to answer the following questions:

X = [-1,3,2,-2,4]
Y = [8,0,-1,15,3]

a. Is this data from a polynomial?
b. If so, what degree is the polynomial?
c. If the point (5, -7) is added, what will be the degree of the new approximating polynomial?

Ans:

Code:

```
int main(int argc, char *argv[]){
   int n = (argc>1) ? atoi(argv[1]) : 5;
   double x[n], y[n], divided_diff[n][n];
   int term = 0;
   for(int i=0; i<n; i++){
       scanf("%lf %lf", &x[i], &y[i]);
   }
   int itr = n;
   for(int i=0;i<n;i++){
       divided_diff[i][0] = y[i];
   }
   for(int i=1;i<n;i++){
       if(divided_diff[0][i-1]!=0){
           term=i-1;
       }
       for(int j=0;j<itr-1;j++){
           divided_diff[j][i] =
(divided_diff[j+1][i-1]-divided_diff[j][i-1])/(x[j+i]-x[j]);
       }
       itr--;
   }
   if(divided_diff[0][n-1]!=0){
       term=n-2;
   }
   printf("Given data is a polynomial of degree %d\n", term);
}
```

a) For any polynomial function, a value in X should map to only one value in the Y axis and in those cases, if we are given n points, we can construct a polynomial with degree n-1. Hence, the above data points can be interpolated into a polynomial.

b) Output:

```
(base) ➜  assignment2 git:(main) ✗ gcc interpol_divided_diff.c -o executables/interpol_divided_diff -lm && ./executables/inter
pol_divided_diff 5
-1 8
3 0
2 -1
-2 15
4 3
Given data is a polynomial of degree 2
```

c) Output:

```
(base) ➜  assignment2 git:(main) ✗ gcc interpol_divided_diff.c -o executables/interpol_divided_diff -lm && ./executables/inter
pol_divided_diff 6
-1 8
3 0
2 -1
-2 15
4 3
5 -7
Given data is a polynomial of degree 4
```

5. **Least Squares:** When using least squares to fit a polynomial, we need to create a matrix such that we can solve for our coefficients. To do this, we can use what is called a design matrix, where n is the degree of our polynomial and N is the number of datapoints.

$$A=\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \\ \vdots & \ddots & \cdots & \vdots \\ x_1^n & x_2^n & \cdots & x_N^n \end{bmatrix}$$

To find the least squares fit, we must solve the matrix equation $Ax=b$ for the coefficient vector x. Re-arranging the matrix equation gives:

$$A^T Ax = A^T$$

$$x=(A^T A)^{-1} A^T b$$

Solve for x to find the coefficients of the interpolating polynomial. You may use built-in matrix operations.

a. Solve the least squares fit for the points (1, 2), (3, 7), (4, 15). What is the polynomial?

b. Does the polynomial perfectly fit the points?

c. Given 5 points, what degree polynomial would be required to perfectly fit the data?

Ans:

Code:

```
#include<stdlib.h>
#include<string.h>
#include<lapacke.h>
#include<math.h>

void fit_check(int degree, double A[][degree+1], double XY[][2], int n){
   for(int i=0;i<n;i++){
       double y=0;
```

```
        for(int j=0;j<=degree;j++){
            y+=A[j][0]*pow(XY[i][0], j);
        }
        printf("f(%lf) = %lf\n", XY[i][0], y);

        if(fabs(y-XY[i][1])>0.0001){
            printf("The fit is incorrect\n");
            return;
        }
    }
    printf("The fit is perfect\n");
}

int main(int argc, char *argv[]) {
    int n = argc > 1 ? atoi(argv[1]) : 3;
    int degree = argc > 2 ? atoi(argv[2]) : 2;
    double XY[n][2], X[n][degree+1], Y[n][1], X_T[degree+1][n];

    char input[n*2], *token;
    for (int i=0; i<n; i++){
        int j=0;
        scanf(" %[^\n]", input);
        token = strtok(input, " ");

        while(token != NULL){
            XY[i][j] = atoi(token);
            token = strtok(NULL, " ");
            j++;
        }
    }

    for (int i=0; i<n; i++){
        X[i][0] = 1;
        X_T[0][i] = 1;
        for (int j=1; j<=degree; j++){
            X[i][j] = X[i][j-1] * XY[i][0];
            X_T[j][i] = X[i][j];
        }
        Y[i][0] = XY[i][1];
    }
```

```
    double X_T_X[degree+1][degree+1];
    for(int i=0;i<=degree;i++){
        for(int j=0;j<=degree;j++){
            X_T_X[i][j] = 0;
            for(int k=0;k<n;k++){
                X_T_X[i][j] += X_T[i][k] * X[k][j];
            }
        }
    }

    int ipiv[degree+1];
    int info;
    info = LAPACKE_dgetrf(LAPACK_ROW_MAJOR, degree+1, degree+1, X_T_X[0],
degree+1, ipiv);
    if(info != 0){
        printf("dgetrf failed with error code %d\n", info);
        return 1;
    }

    info = LAPACKE_dgetri(LAPACK_ROW_MAJOR, degree+1, X_T_X[0], degree+1,
ipiv);

    if(info != 0){
        printf("dgetri failed with error code %d\n", info);
        return 1;
    }

    double X_T_X_X_INV_X_T[degree+1][n];
    for(int i=0;i<=degree;i++){
        for(int j=0;j<n;j++){
            X_T_X_X_INV_X_T[i][j] = 0;
            for(int k=0;k<=degree;k++){
                X_T_X_X_INV_X_T[i][j] += X_T_X[i][k] * X_T[k][j];
            }
        }
    }

    double A[degree+1][degree+1];
    for(int i=0;i<=degree;i++){
```

```
        for(int j=0;j<=degree;j++){
            A[i][j] = 0;
            for(int k=0;k<n;k++){
                A[i][j] += X_T_X_X_INV_X_T[i][k] * Y[k][0];
            }
        }
    }

    printf("The polynomial of degree %d is:\n", degree);
    printf("f(x) = ");
    for(int i=0;i<=degree;i++){
        printf("%lf", A[i][0]);
        if(i>0){
            printf("(x^%d)", i);
        }
        if(i<degree){
            printf(" + ");
        }
    }
    printf("\n");

    fit_check(degree, A, XY, n);

    return 0;
}
```

a)

```
(base) ➜  assignment2 git:(main) gcc leastSquares.c -o executables/leastSquares -lm -llapacke -lopenblas && exe
cutables/leastSquares 3 2
1 2
3 7
4 15
The polynomial of degree 2 is:
f(x) = 5.000000 + -4.833333(x^1) + 1.833333(x^2)
```

b)

```
The polynomial of degree 2 is:
f(x) = 5.000000 + -4.833333(x^1) + 1.833333(x^2)
f(1.000000) = 2.000000
f(3.000000) = 7.000000
f(4.000000) = 15.000000
The fit is perfect
```

c) To perfectly fit a given set of n points with a polynomial, we can use a polynomial of degree n−1 with n+1 coefficients at the max.