Title: **Augmented Matrix Round Off Error and Least Squares Fit**

Authors: **William Franzen, Noah Harbor, Brandon Mitchell, Logan Reed**

Abstract

Round off error can have a significant effect on the solution of an augmented matrix. To simulate the effects of round off error, the same matrix with differing degrees of precision was used. This showed how even a small round off error in the augmented could cause large errors in the final solution and the importance of partial pivoting in methods like Gauss Elimination. In addition, the least squares linear and quadratic fit methods were explored. Despite relatively large differences in the points, the least squares method was able fit a line and parabola to the data with relative success. However, a negative of this method when compared to previous interpolation methods is that the user needs to make a best guess at the degree needed for the data.

Matrix Comparison

In Python, two-dimensional Numpy arrays were used, as the Numpy library allows for more flexible use of matrix operations. A function called invertMatrix() was programmed, which returns the inverse of a square matrix:

```
def invertMatrix(matrix):
    size = matrix.shape[0]
    #Create an augmented matrix by using an identity matrix.
    aug = numpy.concatenate((matrix, numpy.identity(size)), axis=1)

    #Iterate through each row in the augmented matrix.
    for i in range (size):
        pivot = aug[i][i] #Sets the pivot to each value on the diagonal.
        aug[i, :] /= pivot #Divides each row by the pivot value.
        for j in range(i+1, size): #Eliminates values below each pivot.
            factor = aug[j, i]
            aug[j, :] -= factor * aug[i, :]
    #Iterates through the matrix backwards to do backwards substitution.
    for i in range(size-1, 0, -1):
        for j in range(i-1, -1, -1):
            factor = aug[j, i]
            aug[j, :] -= factor * aug[i, :]

    return aug[:, size:] #Retrieves the inverted matrix from the augmented
matrix.
```

This function takes a square matrix (2D Numpy Array) as a parameter. It then determines the dimensions of the matrix using the shape operator. By using this size, it concatenates the identity matrix to the given matrix. Afterwards, it iterates through each row in the matrix and turns it into a triangular matrix. It then iterates through the matrix bottom-to-top, and applies backwards substitution to find the values for the inverse matrix. The inverted matrix is then returned by truncating the unneeded part of the augmented matrix.

For each matrix equation in problem 1, the invertMatrix() function was used to evaluate each matrix solution. Multiplying A^{-1} with the b matrix (solution matrix) was used to determine the coefficients for the unknown variables.

For problem *1a*, A was defined as:

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

In problem *1a*, A^{-1} was calculated to be:

$$\begin{bmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{bmatrix}$$

The solution of $A^{-1} \times b$ was found to be:

$$\begin{bmatrix} -3 \\ -12 \\ 30 \end{bmatrix}$$

This solution matched exactly with what was expected. In addition, the ill-conditioned value for problem 1a was found to be 0.000327500737630741, which is close to the value of 0.00033 that was expected.

Problem *1b* explored how a small bit of round off error could result in a large difference in the solution. The fractions in the 1a matrix were rounded to two decimal points and gave the following matrix:

$$\begin{bmatrix} 1 & 0.5 & 0.33 \\ 0.5 & 0.33 & 0.25 \\ 0.33 & 0.25 & 0.2 \end{bmatrix}$$

For problem *1b*, A^{-1} was calculated to be:

$$\begin{bmatrix} 55.56 & -277.78 & 255.56 \\ -277.78 & 1446.03 & -1349.21 \\ 255.56 & -1349.21 & 1269.84 \end{bmatrix}$$

The solution of $A^{-1} \times b$ was found to be:

$$\begin{bmatrix} 11.11 \\ -84.13 \\ 96.83 \end{bmatrix}$$

As can be seen, the solution is way off of the solution found for 1a. This is a result of the round off error, though it is more extreme than expected. The ill-conditioned value for this matrix was found to be 4.464045245605494e-05. If the value is much less than 1, the matrix is considered ill-conditioned. In this case, the value is much smaller than one and smaller than the ill-conditioned value found for 1a. As such, this matrix can be considered to be more ill-conditioned than 1a.

In problem *1c*, the same matrix as 1a was used but rounded to three decimals this time, which gave the following matrix:

$$\begin{bmatrix} 1 & 0.5 & 0.333 \\ 0.5 & 0.333 & 0.25 \\ 0.333 & 0.25 & 0.2 \end{bmatrix}$$

For problem *1c*, A^{-1} was calculated to be:

$$\begin{bmatrix} 9.67 & -39.51 & 33.28 \\ -39.51 & 210.19 & -196 \\ 33.28 & -196.95 & 195.77 \end{bmatrix}$$

The solution of $A^{-1} \times b$ was found to be:

$$\begin{bmatrix} -2.78 \\ -13.04 \\ 30.92 \end{bmatrix}$$

The solution is still off of the exact solution found in problem *1a*, but it is much closer than the solution found in problem *1b*. Rounding to three decimals appeared to result in an answer that was still off, though. This shows how rounding to one or two more digits of precision could have major effects on the accuracy of the solution. The ill-conditioned value for this matrix was found to be 0.0002999621162269025. This is less than the ill-conditioned value found in problem *1a*, which makes sense. However, it is a lot greater than the value found in problem 1b.

Least-Squares Linear Fit Analysis

To explore the use of the least-squares linear fit method, data consisting of test scores and grades were used. This was to see if the test scores could be used to predict the grade for the class. The following data was used and is in the format of (test score, grade):

(5, 0), (8, 1.3), (10, 2), (12, 1.7), (14, 2.3), (18, 3), (22, 4), (24, 3.3)

This produced the coefficients a = -0.27842209072978313 and b = 0.17546351084812625. Using the formula $y = a + bx$, we get the graph below.

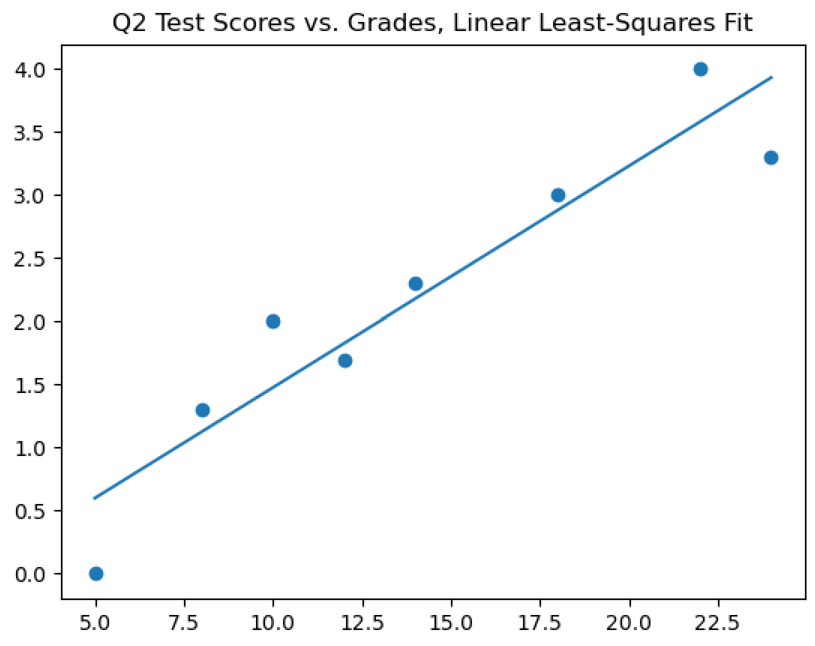

Figure 1: Question 2, Linear Least-Squares Fit

As can be seen in the graph, the resulting line formed from these coefficients manages to fit the data rather well and helps to make the correlation between test score and grade more clear. Using the coefficients, we can also easily predict what test score predicts what grade. For example, a test score of about 18.68 (specifically, 18.68435251798561) would predict a B, or 3.0, in the course.

Quadratic Least-Squares Fit Analysis

The quadratic least-squares fit is another way to determine the line of best fit for a given data set. This method uses the standard formula $y = a + bx + cx^2$, and values of a, b, and c must be determined using the data set to generate the line of best fit. For this problem, the following data was used:

(2.3, -3.1), (3.0, 0.2), (5.8, 1.5), (6.4, 0), (7.2, -2.3)

The constants satisfying the quadratic least-squares fit standard formula were determined to be: a = -17.088275907865057, b = 8.182768265473623, and c = -0.8553470945631577. The data points and resulting line of best fit are shown in the figure below.

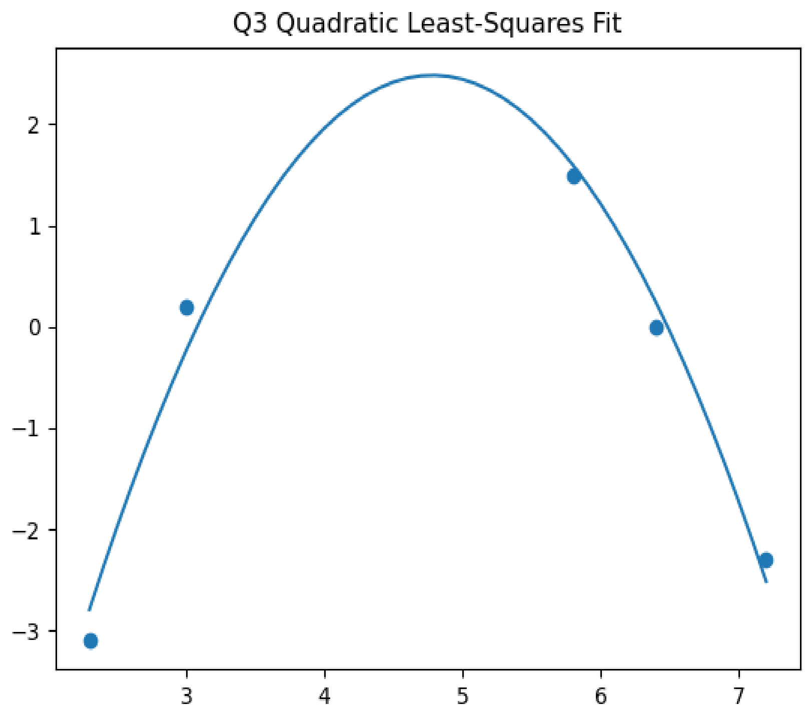

Figure 2: Question 3, Quadratic Least-Squares Fit

As is shown in Figure 2, the resulting line matches the data set very closely. This line of best fit can be used to help estimate outputs given another input within the limits of this data set. This method is useful when it is known that the data has a quadratic relationship. Plotting the points first makes it easier to determine the degree of order. However, this could get more complicated with larger data sets, or data sets representative of higher degrees of order.