BLG202E Homework № 5

Başar Demir, 150180080 15/06/2020

1 Problem 1

1.1 Error Comparison

For any calculation, given actual result u and approximation v:
Absolute Error:

$$\epsilon = |u - v|$$

Absolute error method calculates error using absolute difference between approximation and actual value.
Relative Error:

$$\epsilon = \frac{|u - v|}{|u|}, u \neq 0$$

Relative error calculates error using ratio between absolute difference and actual value. Thus, we can compare our approximations based on actual value.

The relative error is more meaningful because the difference between the results does not always indicate the error rate. To illustrate, if we are working with large actual results while making calculations, the difference between real result and approximation will likely be high, but for small values, the difference comes probably small. Therefore, absolute error highly depends on the actual value. On the other hand, the relative error method always scales the error using actual value. Thus, we can find how much error occurred based on actual value. Therefore, the relative error method is generally a more reliable technique.

1.2 Discretization Error

$$f''(x_0) \approx \frac{f(x_0) - 2f(x_0 + h) + f(x_0 + 2h)}{h^2}$$

To find discretization error of this formula, we should find how this expression is formulated. Numerical differentiation formulas is based on Taylor's expansion, therefore I expand $f(x_0 + h)$ and $f(x_0 + 2h)$ up to third derivative of $f(x)$.

$$f(x_0 + h) \approx f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) + ...$$

$$f(x_0 + 2h) \approx f(x_0) + 2hf'(x_0) + \frac{4h^2}{2}f''(x_0) + \frac{8h^3}{6}f'''(x_0) + ...$$

To cancel out $f'(x_0)$, I will subtract 2 times first equation from second equation.

$$f(x_0 + 2h) - 2f(x_0 + h) \approx -f(x_0) + h^2 f''(x_0) + h^3 f'''(x_0)$$

If we organize the equation:

$$f(x_0 + 2h) - 2f(x_0 + h) + f(x_0) - h^3 f'''(x_0) \approx h^2 f''(x_0)$$

$$\frac{f(x_0 + 2h) - 2f(x_0 + h) + f(x_0)}{h^2} - \mathbf{hf'''(x_0)} \approx f''(x_0)$$

As seen in the equation truncation error is equal to $\mathbf{hf'''(x_0)}$ that has $\mathbf{O(h)}$ error order. Of course, the Taylor's Expansion continues up to infinity but, for h values that are smaller than 1 $hf'''(x_0)$ is the most dominant term.

1.3 Python Implementation

My python implementation and line by line comments are given below.

```python
import math
import numpy as np
import matplotlib.pyplot as plt

#Required f(x) function and its derivatives
def f(x):
    return math.tan(x)
def f_1st_derivative(x):
    return math.sec(x)**2
def f_2nd_derivative(x):
    return (2*math.tan(x))/(math.cos(x)**2)
def f_3rd_derivative(x):
    return -4*(1/math.cos(x)**2)+6*(1/math.cos(x)**4)

#Function which returns 2nd derivative and O(h) error
def approximate_2nd_derivative(f_3rd_derivative,f,x,h):
    result= (f(x)-(2*f(x+h))+f(x+2*h))/(h**2)
    theoretical_error= abs(f_3rd_derivative(x)*h)
    return result,theoretical_error

#Function which returns absolute values of  actual and theoretical ↩
    error for given x and h values
def error(f,f_2nd_derivative,f_3rd_derivative,x,h):
    result = approximate_2nd_derivative(f_3rd_derivative,f,x,h) #Call ↩
        derivative function
    approximate_result = result[0] #Takes approximate result part
    theoretical_error =  result[1] #Takes theoretical error part
    actual_result = f_2nd_derivative(x) #Calls actual derivative ↩
        function
    actual_error = abs(actual_result-approximate_result) #Error ↩
        between real and theoretical results
    return actual_error,theoretical_error

```

```
def Error_Graph(f,f_2nd_derivative,f_3rd_derivative,x):
    theoretical=[] #list for keeping theoretical errors
    actual=[] #list for keeping actual errors
    h_array=[] #list for keeping h values
    #Fills h array with negative powers of 10 up to -10th power
    for k in range(1,11):
        h1 = math.pow(10,-k)
        h_array.append(h1)
    #For every h value it calculates actual and theoretical error
    for h in h_array:
        actual_error, theoretical_error =error(f,f_2nd_derivative,↩
            f_3rd_derivative,x,h)
        #Adds absolute difference of errors to difference list
        theoretical.append(theoretical_error)
        actual.append(actual_error)

    #Graph for actual and theoretical error observation
    plt.figure()
    plt.xlabel("h")
    plt.loglog( h_array,theoretical,"--", label="Theoretical Error")
    plt.loglog( h_array,actual,"--", label="Actual Error")
    plt.legend()

#Function returns approxiamate second derivate for given x values
#h values are negative powers of 10 up to -10th power
def print_2nd_derivative(f_3rd_derivative,f,x):
    for k in range(1,11):
        h = math.pow(10,-k)#Calculates h
        #Calls derivative function
        print('h=',h,'result=', approximate_2nd_derivative(↩
            f_3rd_derivative,f,x,h)[0])

Error_Graph(f,f_2nd_derivative,f_3rd_derivative,math.pi/4)
print_2nd_derivative(f_3rd_derivative,f,math.pi/4)
```

h	f”(x)
0.1	6.239988622166991
0.01	4.164798117778368
0.001	4.016046794896866
0.0001	4.001600495406876
1e-05	4.000162423523078
1e-06	3.999911513119514
1e-07	4.019007349143068
1e-08	0.0
1e-09	222.04460492503128
1e-10	0.0

Table 1: Calculated f”(x) values for x = $\pi/4$

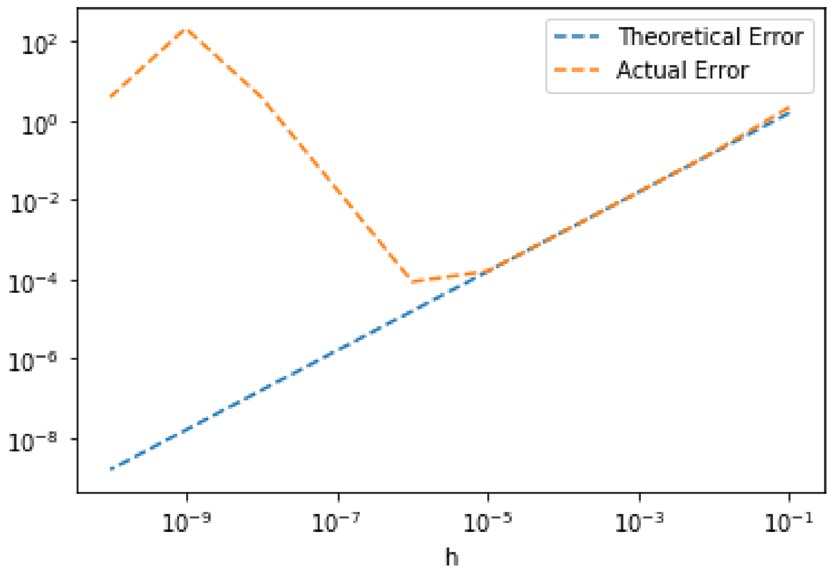

Figure 1: Actual and Theoretical Errors for h values

1.4 Comparison

Theoretical error is calculated by taking an element that has the biggest effect to result from the ignored part of Taylor’s expansion. It gives an error value which is close to actual error but it cannot be equal to actual error as expected. On the other hand, the actual error is a combination of discretization and round off errors. As in the theory, I have observed that theoretical error and actual error are very close to each other until a point (between $10^{-5}, 10^{-6}$). For $[\sim 10^{-6}, 10^{-1}]$ interval, discretization error that occurs as a result of discretization of the Taylor’s Series is the dominant error for the actual error. But after a point, round-off errors started to become dominant. It appears because of the limited floating-point representation capacity of the computer, it cannot calculate values smoothly and it affects the results. As a result after some point, the difference between theoretical and actual errors rises.

1.5 Condition

To determine the condition of our formula for $f''(x)$, I calculate the output for small changes with 0.001 step size in x using python. In addition, I graphed $f''(x)$ between $\pi/6$ and $\pi/3$ interval. The code and output is given below.

```python
def Condition_graph(f_3rd_derivative,f,x):
    results=[] #list for keeping the results
    add_array=[]#list for keeping x values
    add = 0.0001 #Step size
    add_array.append(x)
    for k in range(1,11):
        #Fills list with x values with adding-subtracting small values
        add_array.append(x+add)
        add_array.append(x-add)
        add +=0.0001
    #Sorts array for better graph
    add_array.sort()

    h=0.001 #Constant h value
    for i in add_array:
        #For every value, it calculates second derivative
        result = approximate_2nd_derivative(f_3rd_derivative,f,i,h) #↩
            Call derivative function
        approximate_result = result[0] #Takes approximate result part
        results.append(approximate_result)#Adds to results array

    #It transfroms x values in terms of pi
    for i in range(len(add_array)):
        add_array[i] /= math.pi

    #Figure for short step sizes
    plt.figure()
    plt.xlabel("x (unit of pi)")
    plt.plot( add_array,results,".", label="f''(x)")
    plt.legend()

    #Figure for wider perspective
    plt.figure()
    f_app = np.vectorize(approximate_2nd_derivative)
    x1 = np.linspace(math.pi/6, math.pi/3)
    plt.xlabel("x (unit of pi)")
    plt.plot(x1/math.pi,f_app(f_3rd_derivative,f,x1,h)[0], label="f''(↩
        x)")
    plt.legend()

Condition_graph(f_3rd_derivative,f,math.pi/4)
```

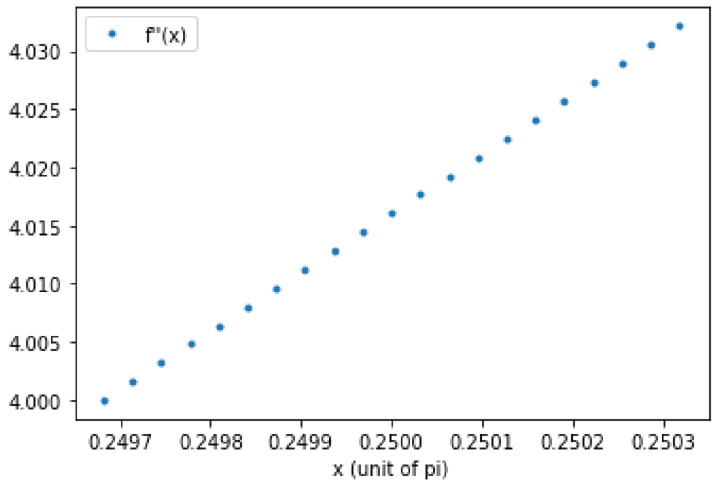

Figure 2: $f''(x)$ with 0.001 step size

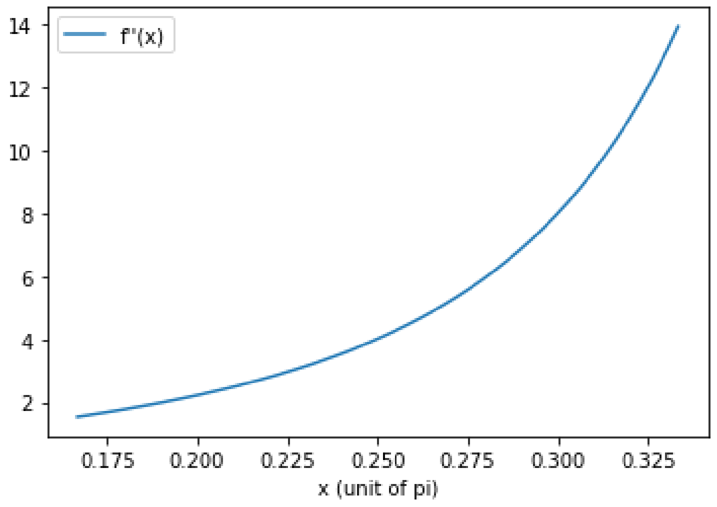

Figure 3: $f''(x)$ between $\pi/6$ and $\pi/3$

I have tried some x values with step sizes near $\pi/4$, the output that is seen in Fig.2 is stable and linear. If this function is ill-conditioned in $\pi/4$ point, we should observe an exponential graph and it rises and falls with large values. But in this scenario, differences between results are acceptable and results are plotted smoothly to graph. Besides, in Fig.3, we can easily observe the stability of $f''(x)$ with wider perspective. In conclusion, at point $\pi/4$, our function is **well-conditioned**.

2 Problem 2

2.1 Floating Point Representation

In computer systems, different types of floating point representation can be applied. But, fundamental principles for conversion are similar. We should represent our float in the form which is given below.

$$float = sign * (1.mantissa) * 2^{exponent}$$

To obtain mantissa, we should find binary representation of our floating number. To obtain fraction part of number, negative exponents of 2 are used in binary systems. Until we get the result, we should try all exponents and count them. But for this question, we do not have to find the exact binary representation, because we cannot represent all floating part in 4 bit mantissa. Therefore, I calculate until 2^{-11}.

Power of 2	**Value**	**Counter**	**Carry**
-1	0.5	0	0.03754
-2	0.25	0	0.03754
-3	0.125	0	0.03754
-4	0.0625	0	0.03754
-5	0.03125	1	0.00629
-6	0.015625	0	0.00629
-7	0.0078125	0	0.00629
-8	0.00390625	1	0.00238375
-9	0.001953125	1	0.000400625
-10	0.0009765625	0	0.000400625
-11	0.00048828125	0	0.000400625

As we see in table our approximate binary form of our number is:

$$0.03754 \approx 0.00001001100$$

$$float = sign * (1.mantissa) * 2^{exponent}$$

Now we should convert our binary number to representation form by shifting.

$$0.00001001100 = 1.001100 * 2^{-5}$$

We should take most significant part of the floating part as mantissa. Therefore,

$$mantissa = 0011$$

Our exponent and sign of our number is clear.

$$\text{exponent} = -5 \rightarrow -0b0101$$

$$\text{sign of exponent} = (-) \rightarrow 0b1$$

$$\text{sign of number} = (+) \rightarrow 0b0$$

As a result, our hypothetical 10-bit binary word representation is:

Sign	Sign of exponent	Exponent	Mantissa
0	1	0101	0011

2.2 Decimal Equivalent

To find decimal equivalent of hypothetical 10-bit binary word, we will do operations that we have done in backwards direction.

$$0101010011 \rightarrow (+1) * (1.0011) * 2^{-0101}$$

$$0101010011 \rightarrow (1.0011) * 2^{-5}$$

Decimal equivalent of 1.0011 is:

$$1.0011 = \frac{1}{2^0} + \frac{0}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{1}{2^4}$$

$$1.0011 = \frac{1}{1} + \frac{1}{8} + \frac{1}{16} = 1.1875$$

If we multiply with the exponent, we get decimal representation of our hypothetical 10-bit binary word:

$$1.1875 * 2^{-5} = \mathbf{0.037109375}$$

2.3 Machine Epsilon

In relative error is guaranteed that is bounded by,

$$\eta = \frac{1}{2} * \beta^{1-t}$$

where relative error is,

$$\epsilon = \frac{|fl(x) - x|}{|x|}$$

For our case relative error is equal to

$$\epsilon = \frac{|0.037109375 - 0.03754|}{|0.03754|} = 0.011471097496004183$$

Boundary for our hypothetical 10-bit binary word (t is bit lenght of mantisssa, so t=4):

$$\eta = \frac{1}{2} * 2^{1-4} = 0.0625$$

These results ensure the relative error for our representation system less than upper error bound for 4-bit mantissa.

$$0.0625 > 0.011471097496004183 \rightarrow \eta > \epsilon$$

3 Problem 3

3.1 Fixed Points

Fixed point is a point which provides $f(x) = x$ condition. Therefore, for our function $h(x)$

$$h(x) = x$$

$$x^2 + \frac{4}{25} = x$$

$$x^2 - x + \frac{4}{25} = 0$$

By quadratic root formula

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

$$x_1 = \frac{1 + \sqrt{1 - \frac{16}{25}}}{2}, x_2 = \frac{1 - \sqrt{1 - \frac{16}{25}}}{2}$$

$$x_1 = \frac{1 + \frac{3}{5}}{2}, x_2 = \frac{1 - \frac{3}{5}}{2}$$

Our fixed points are:

$$x_1 = \frac{4}{5}, x_2 = \frac{2}{10}$$

3.2 Convergence of Fixed Point Iteration

Before the mathematical proof, we should understand the mechanism behind the fixed point iteration. We are trying to find point which holds $x = f(x)$ condition while iterating over $x_{k+1} = f(x_k)$ for k=0,1,2,3... Therefore, we are trying to minimize difference between x_k and root x^* in each step.
If we assume that $f(x)$ function is continuous and differentiable in interval [a,b] and there is a unique fixed point x^*. There is a ρ that satisfies for all x values in [a,b] interval

$$|f'(x)| \leq \rho$$

By mean value theorem, there is a c value in [a,b] such that

$$f'(c) = \frac{f(b) - f(a)}{b - a}$$

We know $f'(c)$ is bounded by ρ. If we adapt the mean value theorem to our fixed point iteration:

$$\rho \geq \frac{|f(x_k) - f(x^*)|}{|x_k - x^*|} \rightarrow \rho|x_k - x^*| \geq |f(x_k) - f(x^*)|$$

We know that from the fixed point mechanism:

$$x_{k+1} = f(x_k), k = 0, 1, 2...$$

$$f(x^*) = x^*$$

So,

$$\rho|x_k - x^*| \geq |x_{k+1} - x^*|$$

If we iterate 0 to k:

$$\rho^k|x_0 - x^*| \geq ... \geq \rho^2|x_{k-1} - x^*| \geq \rho|x_k - x^*| \geq |x_{k+1} - x^*|$$

$$\rho^k|x_0 - x^*| \geq |x_{k+1} - x^*|$$

We are trying to minimize $|x_{k+1} - x^*|$. Therefore $\lim_{k\to\infty} \rho^k$ should equal to 0.

$$\lim_{k\to\infty} \rho^k \to \rho < 1$$

In conclusion, under our assumptions if $|f'(x^*)| \leq 1$ it is possible find proper [a,b] interval and fixed point iteration converges to x^* point.
For our case:

$$h(x) = x^2 + \frac{4}{25} \to h'(x) = 2x$$

For fixed point $x_1 = \frac{4}{5}$:

$$h'(\frac{4}{5}) = \frac{8}{5} > 1$$

h(x) will **not converge** to $x_1 = \frac{4}{5}$
For fixed point $x_2 = \frac{2}{10}$:

$$h'(\frac{2}{10}) = \frac{4}{10} < 1$$

h(x) will **converge** to $x_2 = \frac{2}{10}$

3.3 Reducing the Convergence Error

Under the same assumptions that are in previous part, we should get a condition for reducing error by factor 10:

$$\rho^k|x_0 - x^*| \approx |x_k - x^*|, \rho^k = 0.1$$

$$\rho^k = 0.1 \to k = \log_{0.1}\rho \to -\frac{1}{k} = \log_{10}\rho$$

To obtain approximate result, we can consider $\rho = h'(\frac{2}{10})$. Then,

$$-\frac{1}{k} = \log_{10} h'(\frac{2}{10}) \to -\frac{1}{k} = \log_{10}\frac{4}{10} \to k \approx 2.5129415947320606$$

To reduce the convergence error by a factor of 10, we should perform approximately **2.5129** iterations.

3.4 Newton's Method-Fixed Point Iteration Relationship

$$x_{k+1} = f(x_k) \qquad\qquad x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Fixed Point Iteration Newton's Method

As we see above, methods have similar formulas. If we suppose that $f(x_k)$ is equal to $x_k - \frac{f(x_k)}{f'(x_k)}$ for fixed point iteration, two methods will have same representation. Therefore, we can say that Newtons method can be written in fixed point iteration form with $f(x) = x - \frac{f(x)}{f'(x)}$. Accordingly, Newtons method is specialized version of fixed point iteration.

$$\text{Nextons Method } \rightarrow x_{k+1} = f(x_k), f(x) = x - \frac{f(x)}{f'(x)}$$

Why it is useful?

We can use our all theoretical knowledge about fixed point iteration for Newton's method. Most used advantage is convergence checking for Newton's method. It facilitates the checking process and it is based on fixed point convergence formula.

4 Problem 4

To find the price of the each product, we should construct a matrix which contains quantities of products for each case. Then, we should solve linear equation for payments.

$$200p_1 + 600p_2 + 250p_3 = 55500$$

$$250p_1 + 500p_2 + 200p_3 = 51000$$

$$300p_1 + 350p_2 + 400p_3 = 51000$$

In matrix form:

$$\begin{bmatrix} 200 & 600 & 250 \\ 250 & 500 & 200 \\ 300 & 350 & 400 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 55500 \\ 51000 \\ 51000 \end{bmatrix}$$

Gaussian Elimination:

$$\frac{1}{200}R1 \rightarrow R1 \begin{bmatrix} 1 & 3 & \frac{5}{4} \\ 250 & 500 & 200 \\ 300 & 350 & 400 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \frac{555}{2} \\ 51000 \\ 51000 \end{bmatrix}$$

$$R2 - 250R1 \rightarrow R2 \begin{bmatrix} 1 & 3 & \frac{5}{4} \\ 0 & -250 & -\frac{225}{2} \\ 300 & 350 & 400 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \frac{555}{2} \\ -18375 \\ 51000 \end{bmatrix}$$

$$R3 - 300R1 \rightarrow R3 \begin{bmatrix} 1 & 3 & \frac{5}{4} \\ 0 & -250 & -\frac{225}{2} \\ 0 & -550 & 25 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \frac{555}{2} \\ -18375 \\ -32250 \end{bmatrix}$$

$$-\frac{1}{250}R2 \to R2 \begin{bmatrix} 1 & 3 & \frac{5}{4} \\ 0 & 1 & \frac{9}{20} \\ 0 & -550 & 25 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \frac{555}{2} \\ \frac{147}{2} \\ -32250 \end{bmatrix}$$

$$R1 - 3R2 \to R1 \begin{bmatrix} 1 & 0 & -\frac{1}{10} \\ 0 & 1 & \frac{9}{20} \\ 0 & -550 & 25 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 57 \\ \frac{147}{2} \\ -32250 \end{bmatrix}$$

$$R3 + 550R2 \to R1 \begin{bmatrix} 1 & 0 & -\frac{1}{10} \\ 0 & 1 & \frac{9}{20} \\ 0 & 0 & \frac{545}{2} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 57 \\ \frac{147}{2} \\ 8175 \end{bmatrix}$$

$$\frac{2}{545}R3 \to R3 \begin{bmatrix} 1 & 0 & -\frac{1}{10} \\ 0 & 1 & \frac{9}{20} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 57 \\ \frac{147}{2} \\ 30 \end{bmatrix}$$

$$R1 + \frac{1}{10}R3 \to R1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{9}{20} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 60 \\ \frac{147}{2} \\ 30 \end{bmatrix}$$

$$R1 - \frac{9}{20}R3 \to R1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 60 \\ 60 \\ 30 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 60 \\ 60 \\ 30 \end{bmatrix}$$

As seen in final equation, prices are given below:

$$p_1 = \mathbf{60}, p_2 = \mathbf{60}, p_3 = \mathbf{30}$$