Question 1

By Taylor Series:

1. $f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(\zeta_1)h^3$
2. $f(x+2h) = f(x) + f'(x)(2h) + \frac{1}{2}f''(x)(2h)^2 + \frac{1}{6}f'''(\zeta_2)(2h)^3$
   $= f(x) + 2f'(x)h + 2f''(x)h^2 + \frac{4}{3}f'''(\zeta_2)h^3$

$for\ some\ \zeta_1 \in [0,h]\ and\ \zeta_2 \in [0,2h]$

Subtracting *2* - 2 * *1*:

$$f(x+2h) - 2f(x+h) = -f(x) + f''(x)h^2 + (\frac{4}{3}f'''(\zeta_2) - \frac{1}{3}f'''(\zeta_2))h^3$$

$$\implies f''(x) \approx \frac{1}{2h^2}(f(x) - 2f(x+h) + f(x+2h))$$

Thus:

$$f''(x) \approx af(x) + bf(x+h) + cf(x+2h)$$

Where:

$$a = c = \frac{1}{h^2}\ and\ b = \frac{-2}{h^2}$$

Considering the error:

$$|f''(x) - (\frac{1}{2h^2}(f(x) - 2f(x+h) + f(x+2h)))| = |(\frac{4}{3}f'''(\zeta_2) - \frac{1}{3}f'''(\zeta_2))h|$$

$$\text{Thus the error scales with } h \text{ and the error is of size } O(h)$$

This is a second order forward difference approximation, taking points at $x, x+h\ and\ x+2h$ whereas the standard formula:

$$f''(x) \approx \frac{1}{h^2}(f(x+h) - 2f(x) + f(x-h))$$

is a central difference approximation, taking points at $x\ and\ x \pm h$.

Question 2

If a_1 and a_2 are consecutive IEEE single precision numbers then $\epsilon_s = |a_2 - a_1|$ is the machine epsilon for single precision numbers - the smallest positive nunmber representable with single precision.

Thus, in double precision the amount of numbers between a_2 and a_1 is exactly the amount of numbers between 0 and ϵ_s

Let amount of such numbers = N, then:

$$N = |\{x : x \in [0, \epsilon_s] \text{ and x is in IEEE double precision}\}|$$

Now:

$$\text{Single precision machine epsilon } (\epsilon_s) = 2^{-23}$$

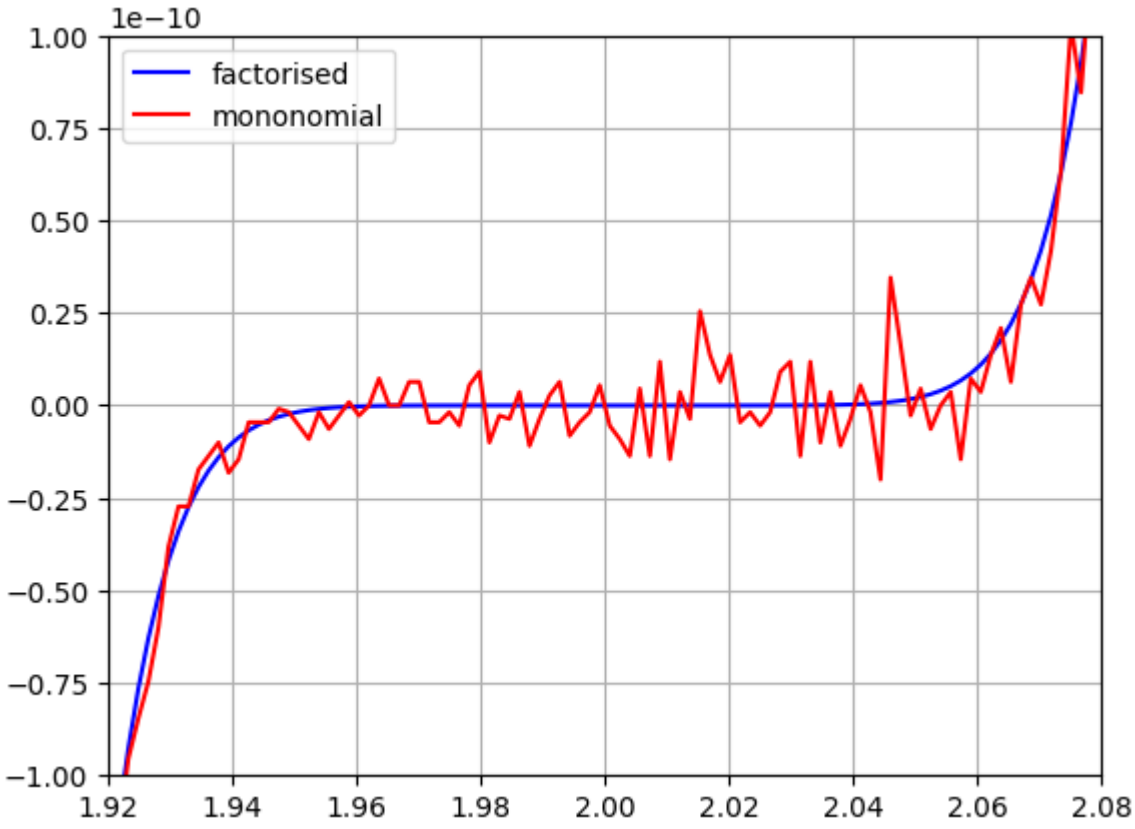
$$\text{Double precision machine epsilon } (\epsilon_d) = 2^{-52}$$

Therefore: $N = \epsilon_s/\epsilon_d \implies N = 2^{52-23} = 2^{29} = 5.37 \times 10^8$

Question 3

Part a:

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(1.92, 2.08, 100)
factorised_fx = (x - 2) ** 9
mononomial_fx = (
          x**9
          - 18*(x**8)
          + 144*(x**7)
          - 672*(x**6)
          + 2016*(x**5)
          - 4032*(x**4)
          + 5376*(x**3)
          - 4608*(x**2)
          + 2304*x
          - 512)

plt.figure()
plt.clf()
plt.plot(x, factorised_fx, 'b-', label='factorised')
plt.plot(x, mononomial_fx, 'r-', label='mononomial')
plt.xlim(1.92, 2.08)
plt.ylim(-0.0000000001, 0.0000000001)
plt.grid()
plt.legend(loc='best')
plt.show()
```

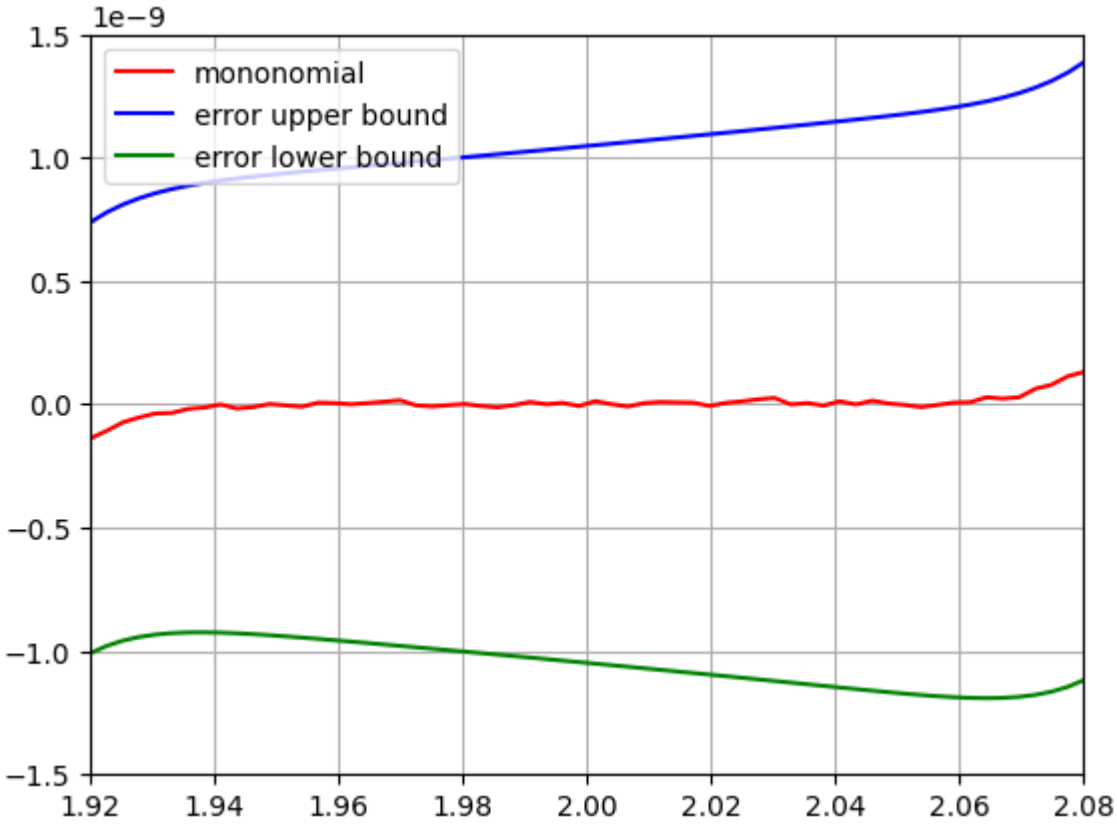


Part b:

The difference in accuracy stems from catestrophic cancellation as subtraction is an ill conditioned operation. Subtracting two numbers whos values are close together can result in a significant loss of precision. While this happens for the factorised graph, there is only one subtraction operation, followed by exponentition. However, the exanded polynomial contains terms of alternating whos overall sum is close to 0. Each term is evaluated separately and during summation the loss of precision is compounded due to the multiple additions and subtractions. This results in a significant loss of precision in comparison to the factorised graph where the loss of precision is minimized as there is only one subtraction and thus less accumulation of rounding errors.

Part c:

In [ ]:
```
#default type for x is float64
eps  = np.finfo(x[0]).eps
d = 9

x = np.linspace(1.87, 2.13, 100)
factorised_fx = (x - 2) ** 9
mononomial_fx = (
         x**9
         - 18*(x**8)
         + 144*(x**7)
         - 672*(x**6)
         + 2016*(x**5)
         - 4032*(x**4)
         + 5376*(x**3)
         - 4608*(x**2)
         + 2304*x
         - 512
         )

error = 2 * d * eps * (
         abs(x**9)
         + abs(18*(x**8))
         + abs(144*(x**7))
         + abs(672*(x**6))
         + abs(2016*(x**5))
         + abs(4032*(x**4))
         + abs(5376*(x**3))
         + abs(4608*(x**2))
         + abs(2304*x)
         + abs(512)
         )

upper_bound = factorised_fx + error
lower_bound = factorised_fx - error
y_lim = 0.0000000015
plt.figure()
plt.clf()
plt.plot(x, mononomial_fx, 'r-', label='mononomial')
plt.plot(x, upper_bound, 'b-', label='error upper bound')
plt.plot(x, lower_bound, 'g-', label='error lower bound')
plt.xlim(1.92, 2.08)
plt.ylim(-y_lim, y_lim)
plt.grid()
plt.legend(loc='best')
plt.show()
```



Question 4

Part a:

Using Gaussian quadrature, we let:

$$\int_a^b f(x)dx = c_0 f(x_0) + c_1 f(x_1) + c_2 f(x_2)$$

Now, let $a = x_i$, $b = x_i + h$, $x_0 = \frac{h}{3}$, $x_1 = \frac{h}{2}$, $x_2 = \frac{2h}{3}$

Using basis functions:

$$f(x) = 1$$

$$f(x) = x$$

$$f(x) = x^2$$

Considering $f(x) = x^n$ for $n > 0$:

$$let\ I_n = \int_{x_i}^{x_i+h} f(x)dx$$

$$= \int_{x_i}^{x_i+h} x^n dx$$

$$= \frac{1}{n+1}[(x_i + h)^{n+1} - x_i^{n+1}]$$

$$= c_0(x_i + \frac{h}{3})^n + c_1(x_i + \frac{h}{2})^n + c_2(x_i + \frac{2h}{3})^n$$

Then:

$$I_0 = h = c_0 + c_1 + c_2$$

$$I_1 = \frac{1}{2}[(x_i + h)^2 - x_i^2] = c_0(x_i + \frac{h}{3}) + c_1(x_i + \frac{h}{2}) + c_2(x_i + \frac{2h}{3})$$

$$I_2 = \frac{1}{3}[(x_i + h)^3 - x_i^3] = c_0(x_i + \frac{h}{3})^2 + c_1(x_i + \frac{h}{2})^2 + c_2(x_i + \frac{2h}{3})^2$$

This gives the following linear system:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & h \\ (x_i + \frac{h}{3}) & (x_i + \frac{h}{2}) & (x_i + \frac{2h}{3}) & \frac{1}{2}[(x_i + h)^2 - x_i^2] \\ (x_i + \frac{h}{3})^2 & (x_i + \frac{h}{2})^2 & (x_i + \frac{2h}{3})^2 & \frac{1}{3}[(x_i + h)^3 - x_i^3] \end{array}\right]$$

Applying

- $R2 \to R2 - (x_i + \frac{h}{3})R1$
- $R3 \to R3 - (x_i + \frac{h}{3})^2 R1$

We get:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & h \\ 0 & (x_i + \frac{h}{2}) - (x_i + \frac{h}{3}) & (x_i + \frac{2h}{3}) - (x_i + \frac{h}{3}) & \frac{1}{2}[(x_i + h)^2 - x_i^2] - h(x_i + \frac{h}{3}) \\ 0 & (x_i + \frac{h}{2})^2 - (x_i + \frac{h}{3})^2 & (x_i + \frac{2h}{3})^2 - (x_i + \frac{h}{3})^2 & \frac{1}{3}[(x_i + h)^3 - x_i^3] - h(x_i + \frac{h}{3})^2 \end{array}\right]$$

$$= \left[\begin{array}{ccc|c} 1 & 1 & 1 & h \\ 0 & \frac{h}{6} & \frac{h}{3} & \frac{h^2}{6} \\ 0 & \frac{1}{3}x_i h + \frac{5}{36}h^2 & \frac{2}{3}x_i h + \frac{1}{3}h^2 & \frac{1}{3}x_i h^2 + \frac{4}{9}h^3 \end{array}\right]$$

Applying

- $R2 \to \frac{6}{h}R2$
- $R3 \to \frac{9}{h}R2$

Yields:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & h \\ 0 & 1 & 2 & h \\ 0 & 3x_i + \frac{5}{4}h & 6x_i + 3h & 3x_i h + 4h^2 \end{array}\right]$$

Applying $R3 \to R3 - (3x_i + \frac{5}{4}h)R2$ yields:

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & h \\ 0 & 1 & 2 & h \\ 0 & 0 & 6x_i + 3h - 2(3x_i + \frac{5}{4}h) & 3x_i h + 4h^2 - h(3x_i + \frac{5}{4}h) \end{array}\right]$$

$$= \left[\begin{array}{ccc|c} 1 & 1 & 1 & h \\ 0 & 1 & 2 & h \\ 0 & 0 & \frac{1}{2}h & \frac{11}{4}h^2 \end{array}\right]$$

Thus:

$$\frac{1}{2}hc_3 = \frac{11}{4}h^2 \implies c_3 = \frac{11}{2}h$$

$$c_1 + 2c_2 = h \implies c_1 = h - 2(\frac{11}{2}h) = -10h$$

$$c_0 + c_1 + c_2 = h \implies c_0 = h - (-10h) - (\frac{11}{2}h) = \frac{11}{2}h$$

Therefore:

$$\int_{x_i}^{x_i+h} f(x)dx \approx \frac{11}{2}hf(x_i + \frac{h}{3}) + -10hf(x_i + \frac{h}{2}) + \frac{11}{2}hf(x_i + \frac{2h}{3})$$

Part b:

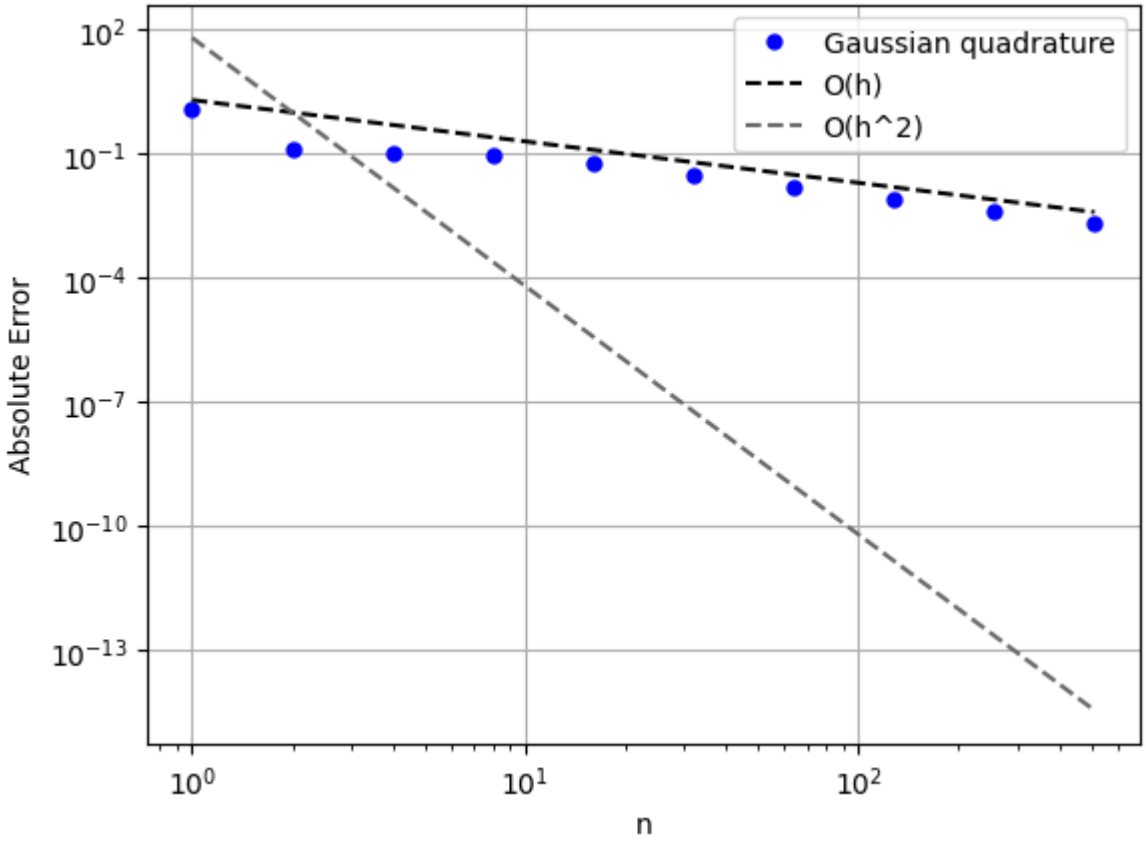
It is known that a quadrature rule with n + 1 nodes cannot be exact for all polynomials of degree ≤ 2n + 2.

However, Gaussian quadrature is exact for all polynomials of degree ≤ 2n - 1.

For the rule above, there are 3 nodes so n = 2. Thus as it is derived using Gaussian quadrature, it is correct for all polynomials of degree ≤ 5.

Part c:

In [ ]:
```
#evaluate error for f(x) = cos(x)
f = lambda x: np.cos(x)
a = -1
b = 1

ns = np.array([2**a for a in range(10)])
hs = (b-a) / ns

true_integral = np.sin(b) - np.sin(a)

approx_ints = np.zeros((len(ns),))
def approx_sub_int(x, h, f):
    #approximate integral from x to x + h using function f
    xs = np.array([x + h/3, x + h/2, x + 2*h/3])
    ys = f(xs)
    coefs = np.array([11/2 * h, -10 * h, 11/2 * h])
    return np.dot(coefs, ys)


for i, (n, h) in enumerate(zip(ns, hs)):
    xs = np.linspace(a,b, n + 1)
    approx_ints[i] = sum([approx_sub_int(x, h, f) for x in xs])

es = np.abs(approx_ints - true_integral)

plt.figure()
plt.clf()
plt.loglog(ns, es, 'b.', markersize=10, label='Gaussian quadrature')
plt.loglog(ns, hs, 'k--', label='O(h)')
plt.loglog(ns, hs**6, '--', label='O(h^6)', color='dimgrey')
plt.grid()
plt.xlabel('n')
plt.ylabel('Absolute Error')
plt.legend(loc='upper right')
plt.show()
```



Errors for Gaussian quadrature scale with $O(h^{2n+2})$. Here n = 2 so the error is expected to scale with $O(h^6)$ but is is upper bounded by O(h)

Question 7

Part a:

$$A = \begin{bmatrix} 2 & 4 & -1 \\ 1 & 1 & -3 \\ 4 & 1 & 2 \end{bmatrix}$$

$$E_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix} \Rightarrow E_1 A_1 = \begin{bmatrix} 2 & 4 & -1 \\ 0 & -1 & -5/2 \\ 0 & -7 & 4 \end{bmatrix}$$

$$E_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -7 & 1 \end{bmatrix} \Rightarrow E_2 E_1 A = \begin{bmatrix} 2 & 4 & -1 \\ 0 & -1 & -5/2 \\ 0 & 0 & 43/2 \end{bmatrix} = U.$$

Now:

$$E_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix} \quad \text{and} \quad E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 7 & 1 \end{bmatrix}$$

And $E_2 E_1 A = U \Rightarrow A = LU$

where $L = E_1^{-1} E_2^{-1}$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 7 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 2 & 7 & 1 \end{bmatrix}$$

And $U = \begin{bmatrix} 2 & 4 & -1 \\ 0 & -1 & -5/2 \\ 0 & 0 & 43/2 \end{bmatrix}$

Part b:

Swap 1 − 3 :

$$P_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -1 \\ 1 & 1 & -3 \\ 4 & 1 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 1 & 2 \\ 1 & 1 & -3 \\ 2 & 4 & -1 \end{bmatrix}$$

$$\Rightarrow E_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow E_1 P_1 A = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 & 2 \\ 1 & 1 & -3 \\ 2 & 4 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 1 & 2 \\ 0 & 3/4 & -7/2 \\ 0 & 7/2 & -2 \end{bmatrix}$$

Swap 2 − 3 :

$$P_2 E_1 P_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 4 & 1 & 2 \\ 0 & 3/4 & -7/2 \\ 0 & 7/2 & -2 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 1 & 2 \\ 0 & 7/2 & -2 \\ 0 & 3/4 & -7/2 \end{bmatrix} \qquad -\frac{3}{4} \cdot \frac{2}{7} = \frac{-3}{14}$$

$$\Rightarrow E_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{-3}{14} & 1 \end{bmatrix}$$

$$\Rightarrow E_2 P_2 E_1 P_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3/14 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 & 2 \\ 0 & 7/2 & -2 \\ 0 & 3/4 & -7/2 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 1 & 2 \\ 0 & 7/2 & -2 \\ 0 & 0 & \frac{-43}{14} \end{bmatrix}$$

$$\Rightarrow U = E_2 P_2 E_1 P_1 A$$

$$= E_2 \tilde{E}_1 P_2 P_1 A$$

where $\tilde{E}_1$ = P_2 applied to below diag entries of E_1

$$= \begin{bmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/4 & 0 & 1 \end{bmatrix}$$

Now: $E_2 \tilde{E}_1 P_2 P_1 A = U$.

$\Rightarrow P_2 P_1 A = \tilde{E}_1^{-1} E_2^{-1} U$.

$$E_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3/14 & 1 \end{bmatrix} \Rightarrow E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3/14 & 1 \end{bmatrix}$$

$$\tilde{E}_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/4 & 0 & 1 \end{bmatrix} \Rightarrow \tilde{E}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/4 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow \tilde{E}_1^{-1} E_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/4 & 3/14 & 1 \end{bmatrix} = L$$

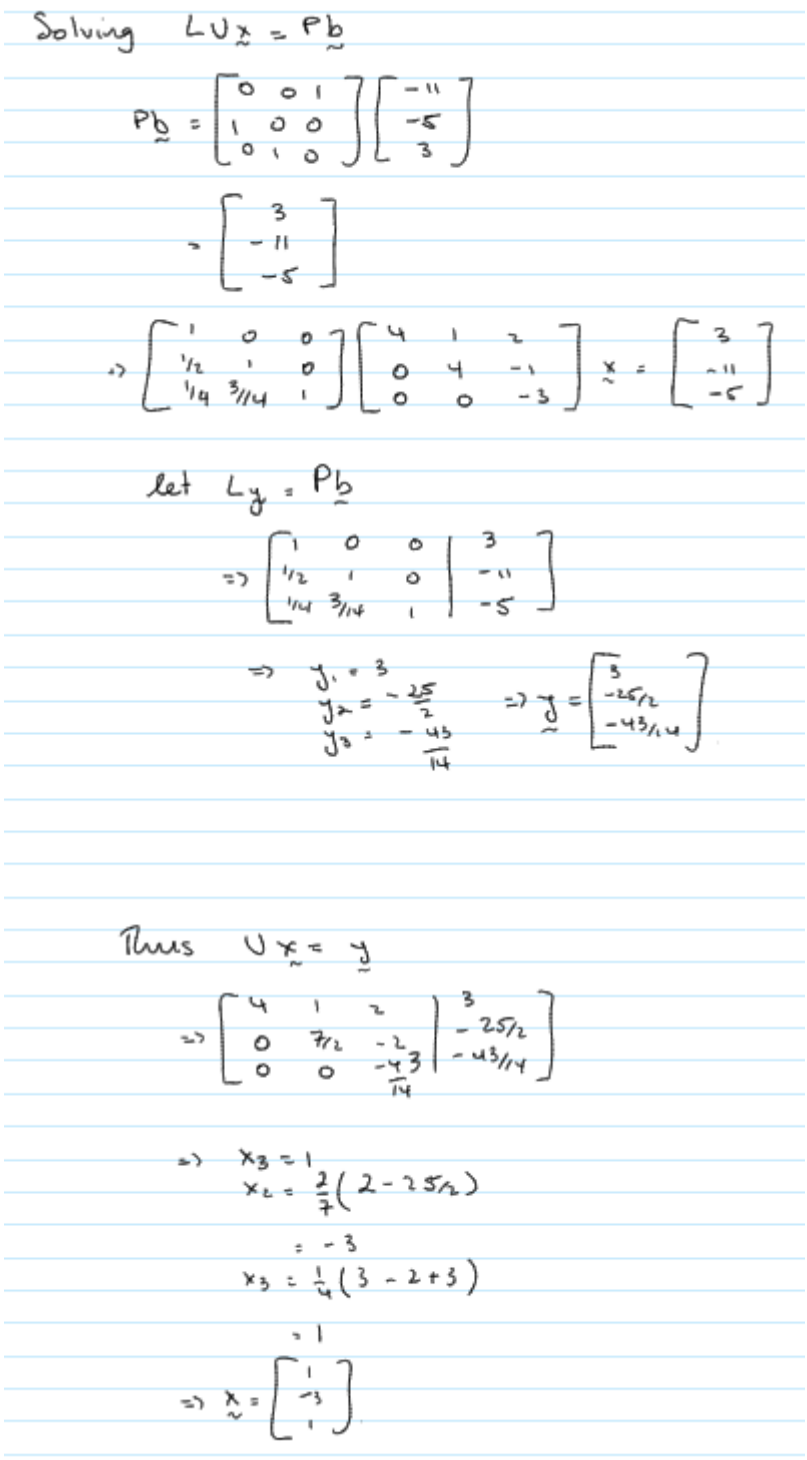
And: $$P_2 P_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = P$$

Thus: $PA = LU$

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 4 & -1 \\ 1 & 1 & -3 \\ 4 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/4 & 3/14 & 1 \end{bmatrix} \begin{bmatrix} 4 & 1 & 2 \\ 0 & 7/2 & -2 \\ 0 & 0 & -43/14 \end{bmatrix}$$

Part c:



Part d:

Standard LU factorisations are very sensitive to rounding errors due to the possibility of small pivot elements. These small pivots can make the decompositions unstable as the numbers being eliminated are divided by it. When the pivot is relatively small, its inverse can be very large and due to the way floating point numbers are stored, rounding errors can occur. Furthermore, the calculation can fail if the pivot element is 0 as it will cause a division by 0 error.

Pivotting solves these problems by selecting the largest element as the pivot, ensuring that the pivot is relatively large compared to the other entries. In doing so, LU factorisations with partial pivotting become backwards stable.

Therefore, the factorisation from part B would have better backwards stability properties as the pivotting ensures relatively large elements are selected and in doing so reduces the impact of rounding errors.