P1.

First, I construct a divided difference table for f(x) = 1+log(x). Use the corresponding two values at the last column and the difference of two corresponding x to compute each entry. The first column is exactly the y value of each x.

```
x_vals = [0.15, 0.21, 0.23, 0.27, 0.32, 0.35]
n = len(x_vals)
y_vals = [f(x) for x in x_vals]

# Calculate the divided differences
def divided_difference(x, y):
    n = len(x)
    coef = np.zeros((n, n))
    coef[:, 0] = y
    for j in range(1, n):
        for i in range(n - j):
            coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1]) / (x[i + j] - x[i])

    return coef
```

The result table: The value in yellow box is needed for later use

```
[[  0.17609   2.43547  -5.7505   15.3474  -38.81159  94.57949]
 [  0.32222   1.97543  -3.90881   8.74943 -19.89569   0.     ]
 [  0.36173   1.7409   -2.94638   5.96403   0.        0.     ]
 [  0.43136   1.47572  -2.23069   0.        0.        0.     ]
 [  0.50515   1.29727   0.        0.        0.        0.     ]
 [  0.54407   0.        0.        0.        0.        0.     ]]
```

We need to find the best three consecutive points to have best approximation

$$P_n'(x) = f[x_0, x_1] + f[x_0, x_1, x_2][(x - x_1) + (x - x_0)]$$

I travel all the possible starting points and compute its correspond approximation value and update the best value closest to Ture value

```
def apporximate_value(idx, x0):
    return coeff[idx][1] + coeff[idx][2] * ((x0 - x_vals[idx]) + (x0 - x_vals[idx + 1]))
```

```
True_value = f_prime(0.268)
min_error = float('inf')
best_approx = None
Used_idx = None
for i in range(n-2):
    x0 = 0.268
    approx_value = apporximate_value(i, x0)
    error = abs(approx_value - True_value)
    if error < min_error:
        min_error = error
        best_approx = approx_value
        Used_idx = i
```

The result: Choose x = 0.23, 0.27, 0.32 will get the best f'(0.268) = 1.63483

```
Used x for approximation: 0.23, 0.27, 0.32
Best approximate value: 1.6348286354082655, True value: 1.6205017981464618, Error: 0.014326837261803727
```

P2.

Similar to p1, I first compute the function difference table for f(x). We just need to use the corresponding value in the last column to compute each value. The values in the first column are exactly f(x).

```
def function_difference(x, y):
    n = len(x)
    coef = np.zeros((n, n))
    coef[:, 0] = y
    for j in range(1, n):
        for i in range(n - j):
            coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1])

    return coef
```

```
Divided differences coefficients:
[[ 0.39851  0.2613  -0.00637 -0.00219  0.00034  0.00007 -0.00002]
 [ 0.65981  0.25493 -0.00856 -0.00185  0.00041  0.00006  0.     ]
 [ 0.91474  0.24637 -0.01041 -0.00143  0.00047  0.       0.     ]
 [ 1.16111  0.23596 -0.01184 -0.00096  0.       0.       0.     ]
 [ 1.39707  0.22412 -0.0128   0.       0.       0.       0.     ]
 [ 1.62119  0.21131  0.       0.       0.       0.       0.     ]
 [ 1.8325   0.       0.       0.       0.       0.       0.     ]]
```

P(X) is the polynomial using function difference and I compute the correspond p'(x) by myself.

$$P(X) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{6}\Delta^3 f_0 + \frac{s(s-1)(s-2)(s-3)}{24}\Delta^4 f_0$$

$$P'(X) = \frac{\Delta f_0}{h} + \frac{s+(s-1)}{2h}\Delta^2 f_0 + \frac{s(s-1)+s(s-2)+(s-1)(s-2)}{6h}\Delta^3 f_0$$

$$+ \frac{s(s-1)(s-2)+s(s-1)(s-3)+s(s-2)(s-3)+(s-1)(s-2)(s-3)}{24h}\Delta^4 f_0$$

Next, we are able to find the answer for (a)(b)(c). For each possible starting point, we choose different p'(x) according to what each subproblem requires and compute the approximate differential value using p'(x) . Try to find the best approximate closest to true value.

The following code is for (a) and (b). Subproblem (c) is very similar, just change a different p'(x).

```
target = 0.72
h = 0.2
min_error = float('inf')
best_approx = None
Used_idx = None
for i in range(len(x)-3):
    s = (target - x[i]) / h
    p = coeff[i][0] + coeff[i][1] * s + coeff[i][2] * s * (s - 1) / 2 + coeff[i][3] * s * (s - 1) * (s - 2) / 6
    p_prime = (coeff[i][1] + coeff[i][2] * (s + (s-1))/2 + coeff[i][3] * (s*(s-1) + (s-1)*(s-2) + s*(s-2))/6)/h
    error = abs(p_prime - f_prime(target))
    if error < min_error:
        min_error = error
        best_approx = p_prime
        Used_idx = i
print(f"Used x for approximation: {x[Used_idx]}, {x[Used_idx + 1]}, {x[Used_idx + 2]}")
print(f"Approximate value: {best_approx}, True value: {f_prime(target)}, Error: {abs(best_approx - f_prime(target))}")
print("-------------------------------------------------")

target = 1.33
h = 0.2
min_error = float('inf')
best_approx = None
Used_idx = None
for i in range(len(x)-2):
    s = (target - x[i]) / h
    p = coeff[i][0] + coeff[i][1] * s + coeff[i][2] * s * (s - 1) / 2
    p_prime = (coeff[i][1] + coeff[i][2] * (s + (s-1))/2)/h
    error = abs(p_prime - f_prime(target))
    if error < min_error:
        min_error = error
        best_approx = p_prime
        Used_idx = i

print(f"Used x for approximation: {x[Used_idx]}, {x[Used_idx + 1]}")
print(f"Approximate value: {best_approx}, True value: {f_prime(target)}, Error: {abs(best_approx - f_prime(target))}")
print("-------------------------------------------------")
```

Result:

```
Used x for approximation: 0.5, 0.7, 0.9
Approximate value: 1.2504649521290017, True value: 1.2506019097136316, Error: 0.0001369575846299486
7
-------------------------------------------------
Used x for approximation: 1.1, 1.3
Approximate value: 1.0789695160766248, True value: 1.079492017811241, Error: 0.000522501734616209
7
-------------------------------------------------
Used x for approximation: 0.7, 0.9, 1.1, 1.3
Approximate value: 1.023537393744599, True value: 1.0235790672225675, Error: 4.16734779684624
36e-05
```

3.

$$f''(x_0) = C_{-2}f(x_0-2h) + C_{-1}f(x_0-h) + C_0 f(x_0) + C_1 f(x_0+h) + C_2 f(x_0+2h)$$
$$= C_{-2}f_{-2} + C_{-1}f_{-1} + C_0 f_0 + C_1 f_1 + C_2 f_2$$

$$P(u) = au^4 + bu^3 + cu^2 + du + e$$

Case 1: $P(u) = 1$

$$f_{-2} = f_{-1} = f_0 = f_1 = f_2 = 1$$
$$f''(x_0) = C_{-2} + C_{-1} + C_0 + C_1 + C_2 = P''(0) = 0$$

Case 2: $P(u) = u$

$$f_{-2} = -2h,\ f_{-1} = -h,\ f_0 = 0,\ f_1 = h,\ f_2 = 2h$$
$$f''(x_0) = C_{-2}\cdot(-2h) + C_{-1}\cdot(-h) + C_1\cdot h + C_2\cdot 2h = P''(0) = 0$$

Case 3: $P(u) = u^2$

$$f_{-2} = 4h^2,\ f_{-1} = h^2,\ f_0 = 0,\ f_1 = h^2,\ f_2 = 4h^2$$
$$f''(x_0) = C_{-2}\cdot 4h^2 + C_{-1}\cdot h^2 + C_1\cdot h^2 + C_2\cdot 4h^2 = P''(0) = 2$$

Case 4: $P(u) = u^3$

$$f_{-2} = -8h^3,\ f_{-1} = -h^3,\ f_0 = 0,\ f_1 = h^3,\ f_2 = 8h^3$$
$$f''(x_0) = C_{-2}\cdot(-8h^3) + C_{-1}(-h^3) + C_1\cdot(h^3) + C_2(8h^3) = P''(0) = 0$$

Case 5: $P(u) = u^4$

$$f_{-2} = 16h^4,\ f_{-1} = h^4,\ f_0 = 0,\ f_1 = h^4,\ f_2 = 16h^4$$
$$f''(x_0) = C_{-2}(16h^4) + C_{-1}(h^4) + C_1(h^4) + C_2(16h^4) = P''(0) = 0$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2h & -h & 0 & h & 2h \\ 4h^2 & h^2 & 0 & h^2 & 4h^2 \\ -8h^3 & -h^3 & 0 & h^3 & 8h^3 \\ 16h^4 & h^4 & 0 & h^4 & 16h^4 \end{bmatrix} \begin{bmatrix} C_{-2} \\ C_{-1} \\ C_0 \\ C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix} \Rightarrow C = \begin{bmatrix} -1/12 \\ 4/3 \\ -5/2 \\ 4/3 \\ -1/12 \end{bmatrix} \frac{1}{h^2}$$

$$f''(x_0) \doteq \frac{-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2}{12h^2}$$

$$\text{error} = (x-x_{-2})(x-x_{-1})(x-x_0)(x-x_1)(x-x_2)\frac{f^5(\xi)}{5!}$$

$$\Rightarrow \frac{d}{dx}\left(\frac{d}{dx}\text{error}\right) = \frac{d}{dx}\left(\sum_{k=-2}^{2}\left(\prod_{i\neq k}(x-x_i)\cdot\frac{f^5(\xi)}{5!}\right) + \prod_i (x-x_i)\cdot\frac{d}{dx}\frac{f^5(\xi)}{5!}\right)$$

$$= \sum_{k=-2}^{2}\sum_{\substack{j=-2 \\ j\neq k}}^{2}\prod_{i\neq j,k}(x-x_i)\cdot\frac{f^5(\xi)}{5!} + \sum_{k=-2}^{2}\prod_{i\neq k}(x-x_i)\cdot\frac{d}{dx}\frac{f^5(\xi)}{5!} + \sum_{k=-2}^{2}\prod_{i\neq k}(x-x_i)\cdot\frac{d}{dx}\frac{f^5(\xi)}{5!} + \prod_i(x-x_i)\cdot\frac{d}{dx}\frac{d}{dx}\frac{f^5(\xi)}{5!}$$

$$\Rightarrow \frac{d}{dx}\left(\frac{d}{dx}\text{error}\right)\Big|_{x=x_0} = \frac{f^5(\xi)}{5!}\cdot\Big[(x_0-x_{-2})(x_0-x_{-1})(x_0-x_1) + (x_0-x_{-2})(x_0-x_{-1})(x_0-x_2) + (x_0-x_{-2})(x_0-x_1)(x_0-x_2) + (x_0-x_{-1})(x_0-x_1)(x_0-x_2)\Big]$$

$$+2\times(x_0-x_{-2})(x_0-x_{-1})(x_0-x_1)(x_0-x_2)\cdot\frac{d}{dx}\frac{f^5(\xi)}{5!} \quad \#$$

Similarly, for $f'''(x_0) = C_{-2}f_{-2} + C_{-1}f_{-1} + C_0f_0 + C_1f_1 + C_2f_2$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2h & -h & 0 & h & 2h \\ 4h^2 & h^2 & 0 & h^2 & 4h^2 \\ -8h^3 & -h^3 & 0 & h^3 & 8h^3 \\ 16h^4 & h^4 & 0 & h^4 & 16h^4 \end{bmatrix}\begin{bmatrix} C_{-2} \\ C_{-1} \\ 0 \\ C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 6 \\ 0 \end{bmatrix} \Rightarrow C = \begin{bmatrix} -\frac{1}{2} \\ 1 \\ 0 \\ -1 \\ \frac{1}{2} \end{bmatrix}\times\frac{1}{h^2}$$

$$f'''(x_0) \doteq \frac{-f_{-2} + 2f_{-1} - 2f_1 + f_2}{2h^2}$$

$$\frac{d}{dx}\frac{d}{dx}\left(\frac{d}{dx}\text{error}\right)\Big|_{x=x_0} \doteq \frac{f^5(\xi)}{5!}\times 2\times\Big[(x_0-x_{-2})(x_0-x_{-1}) + (x_0-x_{-2})(x_0-x_1) + (x_0-x_{-2})(x_0-x_2) + (x_0-x_{-1})(x_0-x_1) + (x_0-x_{-1})(x_0-x_2) + (x_0-x_1)(x_0-x_2)\Big]$$

(Ignore $\frac{d}{dx}\frac{f^5(\xi)}{5!}$ and $\frac{d}{dx}\left(\frac{d}{dx}\frac{f^5(\xi)}{5!}\right)$ in this error)

P4

I construct two functions to compute Simpson's 1/3 rule and Simpson's 3/8 rule.

The first function's input is all the x and it will compute the sum of each segment.

The second function's input is a segment of x used to compute Simpson's 3/8 rule.

```python
def simpson_one_third(fx, h):

    n = len(fx) - 1
    result = fx[0] + fx[-1]
    for i in range(1, n):
        coeff = 4 if i % 2 == 1 else 2
        result += coeff * fx[i]
    return (h / 3) * result
```

```python
def simpson_three_eighth(fx, h):
    return (3 * h / 8) * (fx[0] + 3 * fx[1] + 3 * fx[2] + fx[3])
```

We let the true value be using the 1/3 rule for each segment. Find all possible ways to combine 3/8 rule with 1/3 rule. We found that there are only three possible combinations. Finally, I compute all the approximate value for each case.

```python
# 1. True value using all Simpson's 1/3 rule
true_value = simpson_one_third(f_vals, h)
print(f"True value using 1/3 rule over all panels: {true_value:.20f}")

# 2. Try using 1/3 on first 2 panels (0 to 2), 3/8 on next 6 (3 to 8)
val1 = simpson_one_third(f_vals[0:3], h)
val1 += simpson_three_eighth(f_vals[2:6], h)
val1 += simpson_three_eighth(f_vals[5:9], h)
print(f"1/3 on [1.0-1.2], 3/8 on [1.2-1.5] & [1.5-1.8]: {val1:.20f}, Error: {abs(val1 - true_value):.20f}")

# 3. Try using 1/3 on last 2 panels (6 to 8), 3/8 on first 6
val2 = simpson_three_eighth(f_vals[0:4], h)
val2 += simpson_three_eighth(f_vals[3:7], h)
val2 += simpson_one_third(f_vals[6:9], h)
print(f"3/8 on [1.0-1.3] & [1.3-1.6], 1/3 on [1.6-1.8]: {val2:.20f}, Error: {abs(val2 - true_value):.20f}")

# 4. Try using 1/3 in the middle (2 panels in middle)
val3 = simpson_three_eighth(f_vals[0:4], h)
val3 += simpson_one_third(f_vals[3:6], h)
val3 += simpson_three_eighth(f_vals[5:9], h)
print(f"3/8 on [1.0-1.3] & [1.5-1.8], 1/3 on [1.3-1.5]: {val3:.20f}, Error: {abs(val3 - true_value):.20f}")
```

Using 3/8 rule on 1.0-1.3 and 1.3-1.6 and using 1/3 rule on 1.6-1.8 will get the most accurate result to true value.

```
True value using 1/3 rule over all panels: 1.76693333333333324475
1/3 on [1.0-1.2], 3/8 on [1.2-1.5] & [1.5-1.8]: 1.76694583333333365971, Error: 0.00001250000000041496
3/8 on [1.0-1.3] & [1.3-1.6], 1/3 on [1.6-1.8]: 1.76694583333333343766, Error: 0.00001250000000019291
3/8 on [1.0-1.3] & [1.5-1.8], 1/3 on [1.3-1.5]: 1.76695000000000002061, Error: 0.00001666666666677585
```

P5

Set the initial h to be 1 – 0.2 = 0.8 and compute the integration of f(x) over [0.2, 1] with h using the trapezoidal rule. Update h to be h/2 for each iteration and stop if the integral value does not differ more than 0.02 between two iterations.

The first if statement is for floating point error, cur + h should always equal to 1 at the end in this algorithm. Floating point error might occur and didn't compute the last segment.

```python
def f(x):
    return 1/(x**2)

last = float('inf')

h = 1 - 0.2
while True:
    sum = 0
    cur = 0.2
    while cur + h <= 1:
        sum += (f(cur) + f(cur + h)) * h / 2
        cur += h
    if abs(cur-1) > 1e-3:
        sum += (f(cur) + f(1)) * (1 - cur) / 2

    if abs(sum - last) < 0.02:
        print(f"Approximate value: {sum}, h = {h}, Error: {abs(sum - last)}")
        break

    last = sum
    h /= 2
```

Result: It stops at h = 0.0125

```
Approximate value: 4.003226631406194, h = 0.0125, Error between two iterations: 0.009649802703419752
```

P6

We can spilt the double integral and compute it respectively.

$$\int_{-0.2}^{1.4}\int_{0.4}^{2.6} e^x \sin(2y)\,dy\,dx = \int_{-0.2}^{1.4} e^x\,dx \cdot \int_{0.4}^{2.6} \sin(2y)\,dy$$

To compute each integral, we split the interval with same space h = 0.1 and compute each integral using Gaussian Quadrature, three-term formulas.

$$\int_{-0.2}^{1.4} e^x dx = \int_{-0.2}^{-0.1} e^x dx + \int_{-0.1}^{0} e^x dx + \cdots + \int_{1.3}^{1.4} e^x dx$$

Using the method from the slide, we first need to map each interval to [-1, 1] to apply Gaussian quadrature. Change the form to the below equation and compute the integral using the coefficient for 3-term. Noted that the function f has changed since the mapping from (a, b) to (-1, 1)

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{(b-a)t+b+a}{2}\right)dt$$

```python
xi = np.array([-np.sqrt(3/5), 0.0, np.sqrt(3/5)])
wi = np.array([5/9, 8/9, 5/9])

def f_x(x):
    return np.exp(x)

def f_y(y):
    return np.sin(2*y)

def gaussian_quad_x(x):
    mul = 0.1/2 # (b-a)/2
    tmp_sum = 0
    for i in range(3):
        tmp_sum += wi[i] * f_x(mul * xi[i] + (x + x + 0.1)/2)
    tmp_sum *= mul
    return tmp_sum

def gaussian_quad_y(y):
    mul = 0.1/2 # (b-a)/2
    tmp_sum = 0
    for i in range(3):
        tmp_sum += wi[i] * f_y(mul * xi[i] + (y + y + 0.1)/2)
    tmp_sum *= mul
    return tmp_sum
```

Compute the sum of value of each integral and get the result integral on x direction and y direction. Multiply them and get the result double integral.

```python
x_start, x_end = -0.2, 1.4
y_start, y_end = 0.4, 2.6
h = 0.1

x_total = 0
y_total = 0
x_vals = np.arange(x_start, x_end, h)
y_vals = np.arange(y_start, y_end, h)
for x in x_vals:
    x_total += gaussian_quad_x(x)
for y in y_vals:
    y_total += gaussian_quad_y(y)

print(x_total)
print(y_total)

total = x_total * y_total
print(f"Total integral: {total:.20f}")
```

The result is very close to the true value of double integral computed by Desmos.

```
3.2364692137650883
0.1140950190270217
Total integral: 0.36926501652489768235
```

$$\int_{-0.2}^{1.4}\int_{0.4}^{2.6} e^x \sin(2\cdot y)dydx = 0.369265016513$$

Q1: $f(x) = e^{-x}\sin(2x-1)$

Find fourier coefficient for even expansion of $f(x)$ at $[0,2]$, $p=4$

$$f(x) = \frac{A_0}{2} + \sum_{n=1}^{\infty}\left[A_n \cdot \cos\left(\frac{2n\pi x}{p}\right) + B_n \sin\left(\frac{2n\pi x}{p}\right)\right], \quad p=4$$

Goal: find A_0, A_n and B_n, for $n = 1, 2, 3, \cdots$

1° $$\int_{-2}^{2} f(x)\cdot\cos\left(\frac{2m\pi x}{4}\right)dx = 0 + \sum_{n=1}^{\infty}\int_{-2}^{2} A_n\cdot\cos\left(\frac{2n\pi x}{p}\right)\cdot\cos\left(\frac{2m\pi x}{p}\right)dx + 0$$

$$= 2\cdot B_m$$

$$\Rightarrow A_m = \frac{1}{2}\int_{-2}^{2} f(x)\cdot\cos\left(\frac{2m\pi x}{4}\right)dx$$

$$= \frac{2}{2}\int_{0}^{2} f(x)\cdot\cos\left(\frac{m\pi x}{2}\right)dx \quad (\text{since } f(x) \text{ is even})$$

$$= \int_{0}^{2} e^{-x}\sin(2x-1)\cdot\cos\left(\frac{m\pi x}{2}\right)dx, \quad \text{for } m = 1, 2, \cdots$$

2° $$\int_{-2}^{2} f(x)dx = \int_{-2}^{2}\frac{A_0}{2}dx + 0 + 0$$

$$= 2A_0$$

$$\Rightarrow A_0 = \frac{2}{2}\int_{0}^{2} f(x)dx = \int_{0}^{2} e^{-x}\sin(2x-1)dx \approx 0.0996$$

3° $$\int_{-2}^{2} f(x)\sin\left(\frac{2m\pi x}{4}\right) = 0 = 2\cdot B_m \Rightarrow B_m = 0 \quad \text{for } m = 1, 2, \cdots$$

Therefore, $$f(x) = \frac{0.0996}{2} + \sum_{n=1}^{\infty}\left[\int_{0}^{2} e^{-x}\sin(2x-1)\cdot\cos\left(\frac{n\pi x}{2}\right)dx \times \cos\left(\frac{n\pi x}{2}\right)\right]$$