Week 3 Activities - The derivative

● Graded

Student

Harper Chen

Total Points

97 / 100 pts

Question 1

#1 p. 115 **10** / 10 pts

✔ **– 0 pts** Correct

– 0 pts Excellent Graphics!

– 0 pts interesting approach.Really shows the difference in estimates.

Question 2

#10 pp118/119 **10** / 10 pts

– 0 pts Correct

✔ **– 0 pts** Good!

– 2.5 pts Number 1 is false. g'(x) is ****+ means the SLOPE is always positive. The function value itself can be negative. For example if g(x) is depth underwater of a rising diver, g' can be positive while position below sea level is still negative.

– 0 pts Nice use of a counterexample.

Question 3

#2 (ab,c,d) p 131 **10** / 10 pts

✔ **– 0 pts** Correct

– 0 pts Good

– 0 pts Great clear format

Question 4

#3 p. 131 **30** / 30 pts

✔ **– 0 pts** Correct

Question 5

#4 p.131 **10** / 10 pts

✔ **– 0 pts** Correct

Question 6

#6 -132 **10** / 10 pts

✔ **– 0 pts** Correct

✔ **– 0 pts** Good

Question 7

#15 p134 **10** / 10 pts

✔ **– 0 pts** b - 3 k. Correct

– 0 pts Great explanation!

Question 8

#16 -134 **7** / 10 pts

– 0 pts Correct

✔ **– 3 pts** (f(a + h) - f(a - h))/h and (f(a + h) - f(h))/h are NOT reasonable estimates **when h is small**. Try graphing to see why! Notice the question is not asking for the BEST estimate.

– 0 pts Excellent demonstration with code.

– 0 pts Which work?

– 10 pts no submission

Question assigned to the following page: 1

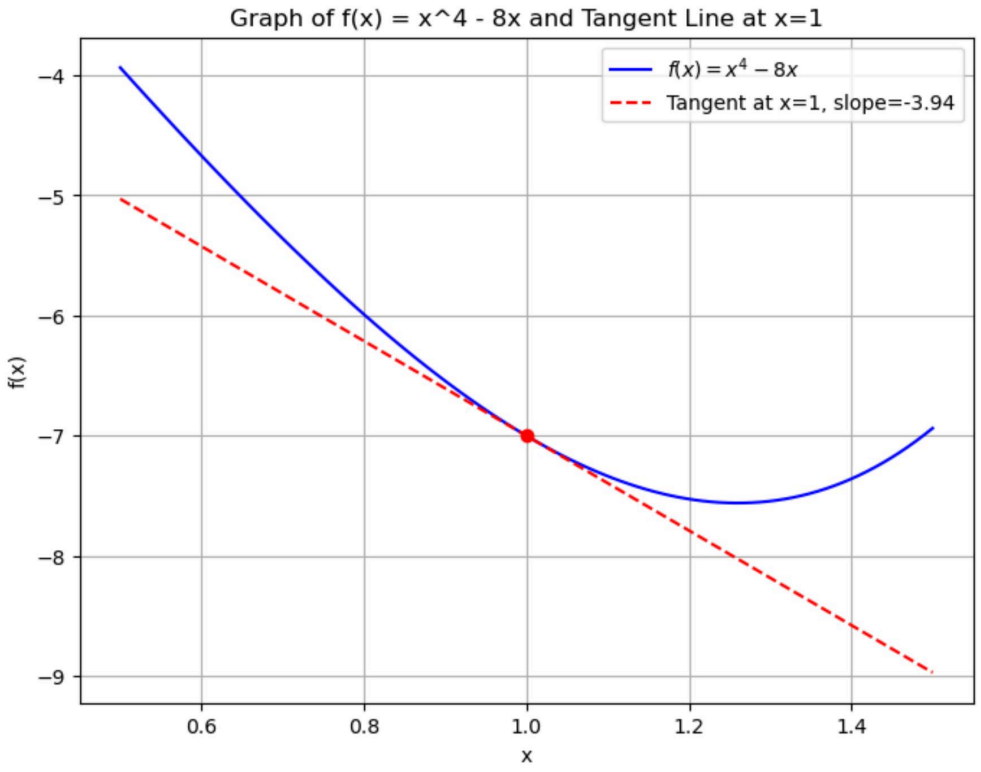
week3

September 16, 2024

#1

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
def f(x):
    return x**4 - 8*x

x_values = np.linspace(0.5, 1.5, 100)
y_values = f(x_values)

h = 0.01
x_point = 1
slope = (f(x_point + h) - f(x_point)) / h

tangent_line = slope * (x_values - x_point) + f(x_point)

plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, label=r'$f(x) = x^4 - 8x$', color='blue')
plt.plot(x_values, tangent_line, label=f'Tangent at x=1, slope={slope:.2f}',␣
 ↪linestyle='--', color='red')
plt.scatter([x_point], [f(x_point)], color='red', zorder=5)
plt.title("Graph of f(x) = x^4 - 8x and Tangent Line at x=1")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()
```

Questions assigned to the following page: 1 and 2



For part (a), I calculated the derivative of $f(x) = x^4 - 8x$ at $x = 1$ using the numerical difference quotient method. I narrowed the interval h and obtained a sequence of estimates. Starting with $h = 0.1$, I calculated the slope as -3.36. As I refined h to 0.01, 0.001, and 0.0001, the slope stabilized around -3.94 and then -3.99. Finally, rounding to one decimal place, the derivative is approximately -4.0. This matches the slope of the tangent line plotted at $x = 1$.

[ ]:

[ ]:

#2

a) This statement is **false**. Even if the derivative $g'(t)$ is positive for all t, it only guarantees that $g(t)$ is increasing. It does not guarantee that $g(214)$ is positive. For example, if $g(t) = t - 500$, the derivative is positive, but $g(214) = -286$, which is negative.

b) This statement is **true**. Since $g'(t)$ is positive for all t, $g(t)$ is increasing, which means $g(214) > g(17)$ because 214 is greater than 17.

c) This statement is **true**. If Bill and Samantha maintain the same speed throughout their drive, the distance between them remains constant. Since Samantha started 1 mile ahead, she will stay 1 mile ahead after 20 minutes.

Questions assigned to the following page: 2 and 3

d) This statement is **true**

Bill and Samantha start their journey at the same time, from the same point, and arrive at the same destination at the same time. Even if they travel at different speeds during the trip, at some point during the four hours, their speeds must have been exactly the same.

imagine they start driving, and sometimes Bill is driving faster than Samantha, and other times Samantha is driving faster than Bill. Since they both start and end at the same place, at some point, they have to "cross over" each other's speed. The IVT tells us that if their speeds are continuous, there must be at least one moment where their speeds match exactly.

So, even if Bill and Samantha drove at different speeds most of the time, there was some point during the four hours when they were driving at exactly the same speed.

[ ]:

[ ]:

#3

[2]:
```python
# for parts (a), (b), (c), and (d)
def f_a(x):
    return 1 / x

def f_b(x):
    return np.sin(7 * x)

def f_c(x):
    return x**3

def f_d(x):
    return 2**x

def derivative_approximation(f, a, h):
    return (f(a + h) - f(a - h)) / (2 * h)

a_a = 2  # For part (a)
a_b = 3  # For part (b)
a_c = 200  # For part (c)
a_d = 5  # For part (d)

h_values = [0.1, 0.01, 0.001, 0.0001, 0.000001]

derivatives_a = [(h, derivative_approximation(f_a, a_a, h)) for h in h_values]
derivatives_b = [(h, derivative_approximation(f_b, a_b, h)) for h in h_values]
derivatives_c = [(h, derivative_approximation(f_c, a_c, h)) for h in h_values]
derivatives_d = [(h, derivative_approximation(f_d, a_d, h)) for h in h_values]

data = {
```

Questions assigned to the following page: 3 and 4

```
    'h': h_values,
    'f\'(2) (1/x)': [d[1] for d in derivatives_a],
    'f\'(3) (sin(7x))': [d[1] for d in derivatives_b],
    'f\'(200) (x^3)': [d[1] for d in derivatives_c],
    'f\'(5) (2^x)': [d[1] for d in derivatives_d],
}

df = pd.DataFrame(data)
print(df)
```

```
          h  f'(2) (1/x)  f'(3) (sin(7x))  f'(200) (x^3)  f'(5) (2^x)
0  0.100000    -0.250627        -3.528569  120000.010000    22.198475
1  0.010000    -0.250006        -3.830974  120000.000100    22.180887
2  0.001000    -0.250000        -3.834074  120000.000002    22.180712
3  0.000100    -0.250000        -3.834105  120000.000005    22.180710
4  0.000001    -0.250000        -3.834105  119999.999646    22.180710
```

[ ]:

[ ]:

#4

[4]:
```
# part a

def f(x):
    return x**3

a = 1
k_values = np.arange(0, 9)  # k = 0 to 8
h_values = [1 / (2**k) for k in k_values]

Q1_values = []
Q2_values = []
for h in h_values:
    Q1 = (f(a + h) - f(a - h)) / (2 * h)  # Central difference
    Q2 = (f(a + h) - f(a)) / h  # Forward difference
    Q1_values.append(Q1)
    Q2_values.append(Q2)

data = {
    'k': k_values,
    'h': h_values,
    'Q1': Q1_values,
    'Q2': Q2_values
}
```

Question assigned to the following page: 4

```
df = pd.DataFrame(data)
print(df)
```

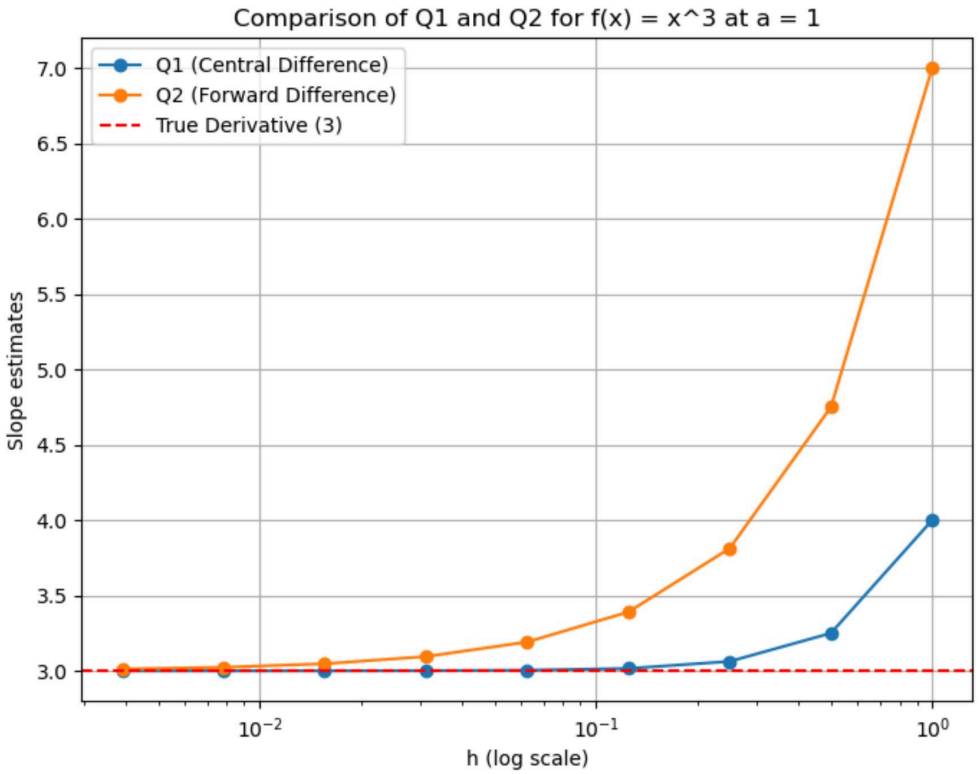
```
   k         h        Q1        Q2
0  0  1.000000  4.000000  7.000000
1  1  0.500000  3.250000  4.750000
2  2  0.250000  3.062500  3.812500
3  3  0.125000  3.015625  3.390625
4  4  0.062500  3.003906  3.191406
5  5  0.031250  3.000977  3.094727
6  6  0.015625  3.000244  3.047119
7  7  0.007812  3.000061  3.023499
8  8  0.003906  3.000015  3.011734
```

Part (b), I analyzed how many digits of Q_1 and Q_2 stabilize near the true derivative value, which is exactly 3. As I reduce h, Q_1 stabilizes much more quickly than Q_2. By looking at the table, I noticed that Q_1 reaches 3.000 to three decimal places by the time $h = 0.0625$ (when $k = 4$). On the other hand, Q_2 does not stabilize as quickly—it reaches 3.191 by $h = 0.0625$, but continues to approach 3 more slowly. So, Q_1 stabilizes to three decimal places earlier, while Q_2 takes more time to converge.

Part (c), I compared the two estimators, Q_1 (central difference) and Q_2 (forward difference), to see which one is better. Q_1 is the better estimator because it consistently approaches the true derivative value of 3 more quickly and more accurately than Q_2. By examining the table, I found that Q_1 is closer to 3 at smaller values of h, and it stabilizes faster. Q_2, on the other hand, starts off farther from 3 and converges more slowly. This shows that using the central difference method (Q_1) provides a more accurate and stable estimate of the derivative compared to the forward difference method (Q_2).

[12]:
```
# part d
plt.figure(figsize=(8, 6))
plt.plot(h_values, Q1_values, label='Q1 (Central Difference)', marker='o')
plt.plot(h_values, Q2_values, label='Q2 (Forward Difference)', marker='o')
plt.axhline(y=3, color='r', linestyle='--', label='True Derivative (3)')
plt.xscale('log')
plt.xlabel('h (log scale)')
plt.ylabel('Slope estimates')
plt.title('Comparison of Q1 and Q2 for f(x) = x^3 at a = 1')
plt.legend()
plt.grid(True)
plt.show()
```

Questions assigned to the following page: 4 and 5



The visual clues from the graph show that Q_1 (the central difference) converges to the true derivative value of 3 much faster and more smoothly as h decreases, while Q_2 (the forward difference) starts farther away and converges more slowly. The smoother and quicker approach of Q_1 to the correct value highlights why it's a better estimator—it more accurately captures the slope at smaller h values without overshooting or undershooting as much as Q_2.

[ ]:

[ ]:

#5

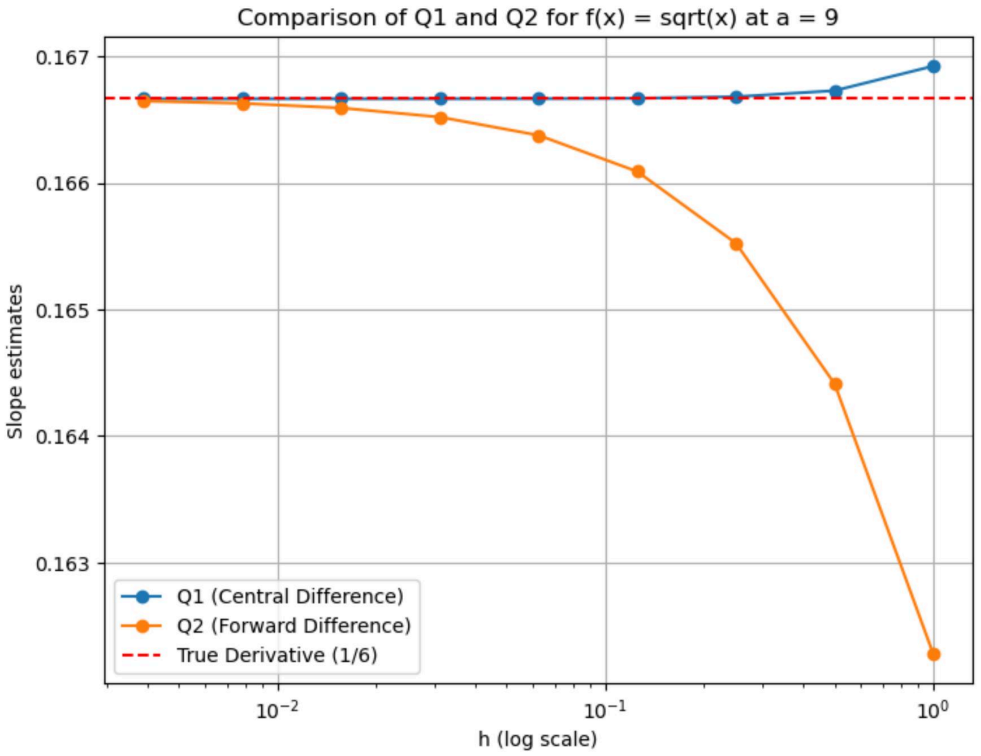
[13]:
```
def f(x):
    return np.sqrt(x)

a = 9
k_values = np.arange(0, 9)  # k = 0 to 8
h_values = [1 / (2**k) for k in k_values]

Q1_values = []
```

Question assigned to the following page: 5

```
Q2_values = []
for h in h_values:
    Q1 = (f(a + h) - f(a - h)) / (2 * h)  # Central difference
    Q2 = (f(a + h) - f(a)) / h  # Forward difference
    Q1_values.append(Q1)
    Q2_values.append(Q2)

data = {
    'k': k_values,
    'h': h_values,
    'Q1': Q1_values,
    'Q2': Q2_values
}

df = pd.DataFrame(data)

print(df)

plt.figure(figsize=(8, 6))
plt.plot(h_values, Q1_values, label='Q1 (Central Difference)', marker='o')
plt.plot(h_values, Q2_values, label='Q2 (Forward Difference)', marker='o')
plt.axhline(y=1/6, color='r', linestyle='--', label='True Derivative (1/6)')
plt.xscale('log')
plt.xlabel('h (log scale)')
plt.ylabel('Slope estimates')
plt.title('Comparison of Q1 and Q2 for f(x) = sqrt(x) at a = 9')
plt.legend()
plt.grid(True)
plt.show()
```

```
   k         h        Q1        Q2
0  0  1.000000  0.166925  0.162278
1  1  0.500000  0.166731  0.164414
2  2  0.250000  0.166683  0.165525
3  3  0.125000  0.166671  0.166092
4  4  0.062500  0.166668  0.166378
5  5  0.031250  0.166667  0.166522
6  6  0.015625  0.166667  0.166594
7  7  0.007812  0.166667  0.166631
8  8  0.003906  0.166667  0.166649
```

Questions assigned to the following page: 5 and 6



[ ]:

[ ]:

#6

[15]:
```
def f_2x(x):
    return 2**x

def f_3x(x):
    return 3**x

def f_10x(x):
    return 10**x

def f_half_x(x):
    return (1/2)**x

def derivative_approximation(f, a, h):
    return (f(a + h) - f(a - h)) / (2 * h)
```

Questions assigned to the following page: 6, 7, and 8

```
a = 0
h_values = [1, 0.1, 0.01, 0.001, 0.0001, 0.00001]

derivatives_2x = [derivative_approximation(f_2x, a, h) for h in h_values]
derivatives_3x = [derivative_approximation(f_3x, a, h) for h in h_values]
derivatives_10x = [derivative_approximation(f_10x, a, h) for h in h_values]
derivatives_half_x = [derivative_approximation(f_half_x, a, h) for h in␣
 ↪h_values]

data = {
    'h': h_values,
    '2^x Derivative': derivatives_2x,
    '3^x Derivative': derivatives_3x,
    '10^x Derivative': derivatives_10x,
    '(1/2)^x Derivative': derivatives_half_x
}

df = pd.DataFrame(data)
print(df)
```

```
         h  2^x Derivative  3^x Derivative  10^x Derivative  \
0  1.00000        0.750000        1.333333         4.950000
1  0.10000        0.693702        1.100824         2.322986
2  0.01000        0.693153        1.098634         2.302789
3  0.00100        0.693147        1.098613         2.302587
4  0.00010        0.693147        1.098612         2.302585
5  0.00001        0.693147        1.098612         2.302585

   (1/2)^x Derivative
0           -0.750000
1           -0.693702
2           -0.693153
3           -0.693147
4           -0.693147
5           -0.693147
```

[ ]:

[ ]:

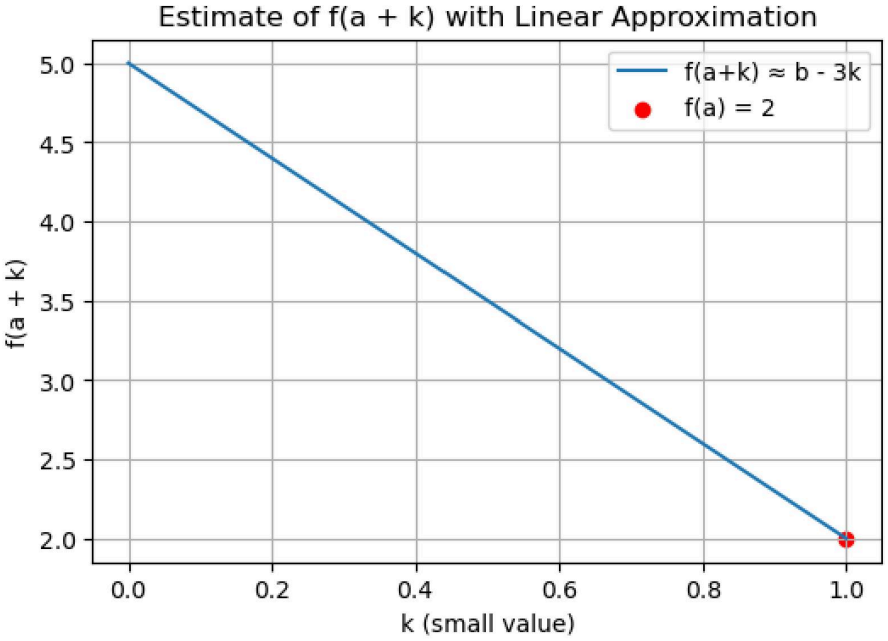
#7 & #8

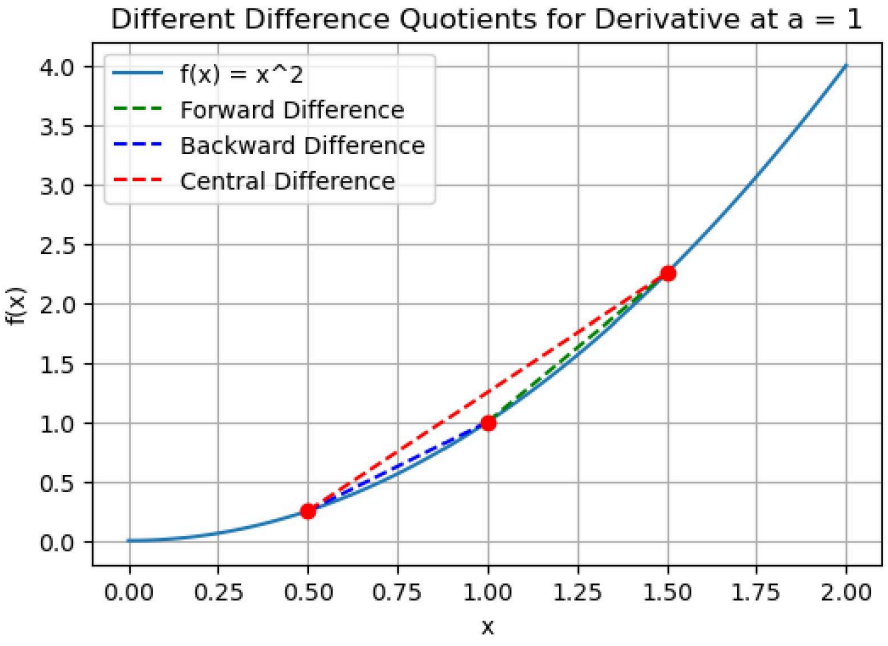
[16]:
```
def linear_function(x, a, b, slope):
    return slope * (x - a) + b

a = 1
```

Questions assigned to the following page: 7 and 8

```
b = 2
slope = -3
k_values = np.linspace(0, 1, 100)

plt.figure(figsize=(6, 4))
plt.plot(k_values, linear_function(k_values, a, b, slope), label="f(a+k)   b -␣
 ↪3k")
plt.scatter(a, b, color="red", label=f"f(a) = {b}")
plt.title("Estimate of f(a + k) with Linear Approximation")
plt.xlabel("k (small value)")
plt.ylabel("f(a + k)")
plt.grid(True)
plt.legend()
plt.show()

def f(x):
    return x**2

h = 0.5
x_values = np.linspace(0, 2, 100)
a = 1

forward_difference = (f(a + h) - f(a)) / h
backward_difference = (f(a) - f(a - h)) / h
central_difference = (f(a + h) - f(a - h)) / (2 * h)

plt.figure(figsize=(6, 4))
plt.plot(x_values, f(x_values), label="f(x) = x^2")
plt.scatter([a + h, a - h, a], [f(a + h), f(a - h), f(a)], color='red',␣
 ↪zorder=5)


plt.plot([a, a + h], [f(a), f(a) + forward_difference * h], 'g--',␣
 ↪label="Forward Difference")
plt.plot([a - h, a], [f(a - h), f(a)], 'b--', label="Backward Difference")
plt.plot([a - h, a + h], [f(a - h), f(a + h)], 'r--', label="Central␣
 ↪Difference")
plt.title("Different Difference Quotients for Derivative at a = 1")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()
```

Questions assigned to the following page: 7 and 8


Estimate of f(a + k) with Linear Approximation
f(a+k) ≈ b - 3k
f(a) = 2
f(a + k)
k (small value)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
0.0
0.2
0.4
0.6
0.8
1.0


Different Difference Quotients for Derivative at a = 1
f(x) = x^2
Forward Difference
Backward Difference
Central Difference
f(x)
x
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Questions assigned to the following page: 7 and 8

[ ]:

For problem 15, It asked to estimate $f(a+k)$ given that $f(a)=b$ and $f'(a)=-3$, where k is small. To do this, I use a linear approximation based on the derivative, which tells me how fast the function is changing at a. The approximation is:

$f(a+k) \approx f(a) + f'(a) \cdot k$

Since $f'(a) = -3$, the estimate becomes:

$f(a+k) \approx b - 3k$

This is a linear function with slope -3, which means as k increases, $f(a+k)$ decreases. On the graph, I plotted this as a straight line passing through (a, b), with the red point representing $f(a) = b$. The graph clearly shows that for small values of k, the linear approximation holds, and the function decreases linearly as predicted by the slope of -3.

[ ]:

For problem 16, I am evaluating different methods for estimating the derivative of a function at a, particularly for small values of h. The central difference method is the most accurate in these cases and is given by:

$\frac{f(a+h)-f(a-h)}{2h}$

This method is better because it uses information from both sides of a to average the rates of change, which gives a more balanced estimate. In contrast, the forward difference and backward difference methods only use information from one side, making them less reliable for small h.

To visualize this, I plotted $f(x) = x^2$ and used the different methods to estimate the derivative at $a = 1$. The central difference, shown as a red dashed line, closely matches the true slope of the curve. The forward and backward differences, represented by the green and blue dashed lines, deviate more from the true slope, demonstrating why they are less accurate. The graph helps to clearly show why the central difference is the best method for estimating the derivative in this situation.