Lab 4

Daniel Mehta #n01753264

In [2]:
```python
import sympy as sp
```

Question 1: Find Derivatives

In [32]:
```python
w, y, z = sp.symbols('w y z')

S_w = (w**2 * (2 - w) + w**5) / (3 * w)
h_y = 3*y**-6 - 8*y**-3 + 9*y**-1

G_z = z**2 * (z - 1)**2

dS_dw = sp.simplify(sp.diff(S_w, w))
dh_dy = sp.simplify(sp.diff(h_y, y))
dG_dz = sp.simplify(sp.diff(G_z, z))

print("Derivative of S(w):", dS_dw)
print("Derivative of h(y):", dh_dy)
print("Derivative of G(z):", dG_dz)
```

```
Derivative of S(w): 4*w**3/3 - 2*w/3 + 2/3
Derivative of h(y): 3*(-3*y**5 + 8*y**3 - 6)/y**7
Derivative of G(z): 2*z*(z - 1)*(2*z - 1)
```

Question 2

In [33]:
```python
t, h = sp.symbols('t h')

f_t =(3-2*t**3)**2

f_t_h = f_t.subs(t,t+h)
f_t_h

difference_quotient = (f_t_h -f_t) / h
difference_quotient

derivative = sp.limit(difference_quotient, h, 0)
derivative

slope_at_1 = derivative.subs(t, 1)
slope_at_1

print("Derivative (General):", derivative)
print("Slope at t = 1:", slope_at_1)
```

```
Derivative (General): 24*t**5 - 36*t**2
Slope at t = 1: -12
```

Question 3

In [3]:
```python
t = sp.Symbol('t', real=True)

V = t**3 -24*t**2 + 192*t - 50

#derivative
dV = sp.diff(V,t)
dV


#critical points
critical_points =sp.solve(dV, t)
print("Critical point(s):", critical_points)


test_points =[(0, "t < 8"),(10, "t > 8")]

for pt, label in test_points:
    derivative_value = dV.subs(t,pt)
    if derivative_value > 0:
        print(f"For {label} (e.g., t = {pt}), V'(t) = {derivative_value} so V(t) is increasing")
    elif derivative_value<0:
        print(f"For {label} (e.g., t = {pt}), V'(t) = {derivative_value} so V(t) is decreasing")
    else:
        print(f"For {label} (e.g., t = {pt}), V'(t) = {derivative_value} so t = {pt} is a critical point")
```

```
Critical point(s): [8]
For t < 8 (e.g., t = 0), V'(t) = 192 so V(t) is increasing
For t > 8 (e.g., t = 10), V'(t) = 12 so V(t) is increasing
```

Out[3]: $\displaystyle 12$

Question 4

In [35]:
```python
x, u, o = sp.symbols('x u o', real=True)

#First function
f1 = sp.log(x**4) *sp.sin(x**3)
df1 = sp.diff(f1, x)

#Second function
f2 = 1/(1+ sp.exp(-x))
df2 = sp.diff(f2,x)

#Thrid Function
f3 = sp.exp(-(1/(2* o**2))*(x-u)**2)

df3 = sp.diff(f3,x)

print("Derivative of f1:", df1.simplify())
print("Derivative of f2:", df2.simplify())
print("Derivative of f3:", df3.simplify())
```

```
Derivative of f1: (3*x**3*log(x**4)*cos(x**3) + 4*sin(x**3))/x
Derivative of f2: 1/(4*cosh(x/2)**2)
Derivative of f3: (u - x)*exp(-(u - x)**2/(2*o**2))/o**2
```

Question 5

In [36]:
```python
w = sp.symbols('w')

f = ((1 - 4*w) * (2 + w)) / (3 + 9*w)

df1 = sp.diff(f, w)
df2 = sp.diff(df1,w)
df3 = sp.diff(df2,w)

print("First derivative:", df1.simplify())
print("Second derivative:", df2.simplify())
print("Third derivative:", df3.simplify())
```

```
First derivative: (-12*w**2 - 8*w - 13)/(3*(9*w**2 + 6*w + 1))
Second derivative: 70/(3*(27*w**3 + 27*w**2 + 9*w + 1))
Third derivative: -210/(81*w**4 + 108*w**3 + 54*w**2 + 12*w + 1)
```

Question 6

In [37]:
```
x, y, z = sp.symbols('x y z')
p, r, t = sp.symbols('p r t')

w = sp.cos(x**2 + 2*y) - sp.exp(4*x - z**4 * y) + y**3
dw_dx = sp.diff(w,x)
dw_dy = sp.diff(w,y)
dw_dz = sp.diff(w,z)

z_func = (p**2 * (r + 1)) / t**3 + p*r * sp.exp(2*p + 3*r + 4*t)
z_func
dz_dp = sp.diff(z_func, p)
dz_dr = sp.diff(z_func, r)
dz_dt = sp.diff(z_func, t)

print("Partial derivatives of w:")
print("dw/dx:", dw_dx.simplify())
print("dw/dy:", dw_dy.simplify())
print("dw/dz:", dw_dz.simplify())

print("\nPartial derivatives of z:")
print("dz/dp:", dz_dp.simplify())
print("dz/dr:", dz_dr.simplify())
print("dz/dt:", dz_dt.simplify())
```

```
Partial derivatives of w:
dw/dx: -2*x*sin(x**2 + 2*y) - 4*exp(4*x - y*z**4)
dw/dy: 3*y**2 + z**4*exp(4*x - y*z**4) - 2*sin(x**2 + 2*y)
dw/dz: 4*y*z**3*exp(4*x - y*z**4)

Partial derivatives of z:
dz/dp: (2*p*(r + 1) + r*t**3*(2*p + 1)*exp(2*p + 3*r + 4*t))/t**3
dz/dr: p*(p + t**3*(3*r + 1)*exp(2*p + 3*r + 4*t))/t**3
dz/dt: p*(-3*p*(r + 1) + 4*r*t**4*exp(2*p + 3*r + 4*t))/t**4
```

Question 7

In [38]:
```
x1, x2 = sp.symbols('x1 x2')
f1 = sp.sin(x1) * sp.cos(x2)
f1

J = sp.Matrix([[sp.diff(f1, x1),sp.diff(f1, x2)]])

print("Jacobian Matrix:")
sp.pprint(J)
print("Jacobian Dimension:", J.shape)
```

```
Jacobian Matrix:
[cos(x₁)⋅cos(x₂)  -sin(x₁)⋅sin(x₂)]
Jacobian Dimension: (1, 2)
```

In [ ]: