Partial Derivatives (SKP)

In [1]:
```
import sympy as smp
from sympy import *
```

In [2]:
```
x, y, z = smp.symbols('x y z')
# input the function
fxy = y**2 * smp.sin(x+y)

display('function, f', fxy)
display('f_x', fxy.diff(x))
display('f_y', fxy.diff(y))
```

'function, f'

$$y^2 \sin(x + y)$$

'f_x'

$$y^2 \cos(x + y)$$

'f_y'

$$y^2 \cos(x + y) + 2y \sin(x + y)$$

The Chain Rule: Suppose x,y and z are functions of t and $w = w(x, y, z)$. Find dw/dt.

In [3]:
```
t = smp.symbols('t')
x, y, z, w = smp.symbols('x y z w', cls = smp.Function)
x = x(t)
y = y(t)
z = z(t)
w = w(x,y,z)
display(w)
display('dw/dt', w.diff(t))
```

$$w(x(t), y(t), z(t))$$

'dw/dt'

$$\frac{d}{dx(t)} w(x(t), y(t), z(t)) \frac{d}{dt} x(t) + \frac{d}{dy(t)} w(x(t), y(t), z(t)) \frac{d}{dt} y(t) + \frac{d}{dz(t)} w(x(t), y(t), z(t)) \frac{d}{dt} z(t)$$

For some particular functions;

In [4]:
```
w1 = x* smp.sin(y)* smp.exp(-z**2) # input w(x,y,z)
display('function w1',w1,'dw1/dt',smp.diff(w1,t))
# substitute x = x(t), y = y(t) and z = z(t)
dw1dt = smp.diff(w1,t).subs([(x, 1/t**2), (y,14*t), (z, 2*t)])
display('for a given x(t), y(t) and z(t),', dw1dt.doit())
```

'function w1'

$$x(t)e^{-z^2(t)} \sin(y(t))$$

'dw1/dt'

$$-2x(t)z(t)e^{-z^2(t)} \sin(y(t))\frac{d}{dt}z(t) + x(t)e^{-z^2(t)} \cos(y(t))\frac{d}{dt}y(t) + e^{-z^2(t)} \sin(y(t))\frac{d}{dt}x(t)$$

'for a given x(t), y(t) and z(t),'

$$-\frac{8e^{-4t^2} \sin(14t)}{t} + \frac{14e^{-4t^2} \cos(14t)}{t^2} - \frac{2e^{-4t^2} \sin(14t)}{t^3}$$

In [ ]:
```

```

Maxima and Minima of a 2D function

Extreme values of $f(x, y)$ can occur at;

1. Boundary points of the domain of $f(x, y)$.
2. Critical points ($f_x = f_y = 0$)

At a point(a,b);

1. Local maxima: $f_{xx} < 0$ and $f_{xx}f_{yy} - f_{xy}^2 > 0$.
2. Local minima: $f_{xx} > 0$ and $f_{xx}f_{yy} - f_{xy}^2 > 0$.
3. Saddle point: $f_{xx}f_{yy} - f_{xy}^2 < 0$.
4. Inconclusive: $f_{xx}f_{yy} - f_{xy}^2 = 0$.

In [5]:
```python
x, y = smp.symbols('x y', real=True)
f = x**2 -y**3 +x*y**2  # input f(x,y)
display('function', f)

fxx = f.diff(x,x)
fyy = f.diff(y,y)
fxy = f.diff(x,y)

# solving df/dx = df/dy = 0   (check it)
display('critical points',smp.solve([f.diff(x),f.diff(y)]))

x1, y1 = 1, -1   # input the point
fxx1 = fxx.subs([(x,x1),(y,y1)]).evalf()
D1 = (fxx*fyy-fxy**2).subs([(x,x1),(y,y1)]).evalf()
print('Given point is', (x1,y1))
display('fxx', fxx1)
display('fxx*fyy - fxy**2', D1)

if fxx1 < 0 and D1 > 0:
    print('local maxima')
elif fxx1 > 0 and D1 > 0:
    print('local minima')
elif D1 < 0:
    print('saddle point')
else:
    print('nothing can be said')
```

```
'function'
```

$x^2 + xy^2 - y^3$

```
'critical points'

[{x: -9/2, y: -3}, {x: 0, y: 0}]

Given point is (1, -1)

'fxx'
```

2.0

```
'fxx*fyy - fxy**2'
```

12.0

```
local minima
```

In [ ]:
```python
```

Lagrange Multipliers

Minimize $f(x, y, z)$ subject to the constraint $g(x, y, z) = 0$. It requires to solve 2 equations $\nabla f = \lambda \nabla g$ and $g(x, y, z) = 0$.

The function is $f = T = 8x^2 + 4yz - 16z + 600$ and the constraint is $g = 4x^2 + y^2 + 4z^2 - 16 = 0$.

In [6]:
```python
from sympy.vector import *
C = CoordSys3D('')

lam = smp.symbols('\lambda')
# input the function
f = 8*C.x**2 +4*C.y*C.z -16*C.z +600
# input the constraint
g = 4*C.x**2 +C.y**2 +4*C.z**2 -16

eq1 = gradient(f) - lam*gradient(g)
eq1m = eq1.to_matrix(C)
eq2 = g
display('f',f,'g',g, 'equation 1',eq1,eq1m, 'equation 2',eq2)
```

'f'

$$8\mathbf{x}^2 + 4\mathbf{y}\mathbf{z} - 16\mathbf{z} + 600$$

'g'

$$4\mathbf{x}^2 + \mathbf{y}^2 + 4\mathbf{z}^2 - 16$$

'equation 1'

$$(-8\mathbf{x}\lambda + 16\mathbf{x})\hat{\mathbf{i}} + (-2\mathbf{y}\lambda + 4\mathbf{z})\hat{\mathbf{j}} + (4\mathbf{y} - 8\mathbf{z}\lambda - 16)\hat{\mathbf{k}}$$

$$\begin{bmatrix} -8\mathbf{x}\lambda + 16\mathbf{x} \\ -2\mathbf{y}\lambda + 4\mathbf{z} \\ 4\mathbf{y} - 8\mathbf{z}\lambda - 16 \end{bmatrix}$$

'equation 2'

$$4\mathbf{x}^2 + \mathbf{y}^2 + 4\mathbf{z}^2 - 16$$

In [7]:
```python
sols = smp.solve([eq1m,eq2]) # use the matrix to solve
for sol in sols:
    print('\n (x,y,z,lambda) =', sol)
    print('value of the function =',f.subs(sol).evalf())
```

```
 (x,y,z,lambda) = {.x: -4/3, .y: -4/3, .z: -4/3, \lambda: 2}
value of the function = 642.666666666667

 (x,y,z,lambda) = {.x: 0, .y: -2, .z: -sqrt(3), \lambda: sqrt(3)}
value of the function = 641.569219381653

 (x,y,z,lambda) = {.x: 0, .y: -2, .z: sqrt(3), \lambda: -sqrt(3)}
value of the function = 558.430780618347

 (x,y,z,lambda) = {.x: 0, .y: 4, .z: 0, \lambda: 0}
value of the function = 600.000000000000

 (x,y,z,lambda) = {.x: 4/3, .y: -4/3, .z: -4/3, \lambda: 2}
value of the function = 642.666666666667
```

In [ ]: