1 Multivariable calculus

primer-computational-mathematics.github.io/book/c_mathematics/

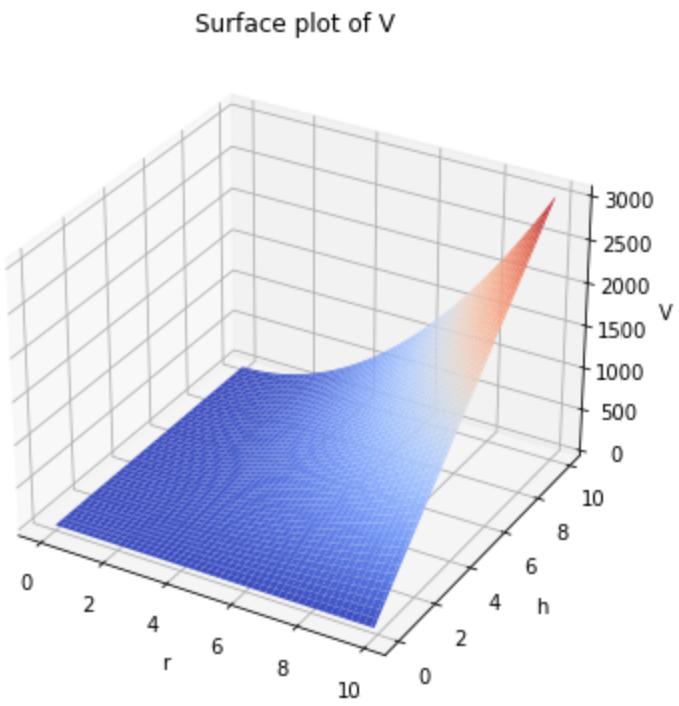
In [ ]:
```
## This cell just imports necessary modules
import numpy as np
from sympy import sin, cos, Function, Symbol, symbols, diff, integrate, exp, pi
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
```

First partial derivatives

In [ ]:
```
# Define the independent variables using Symbol
r = Symbol('r')
h = Symbol('h')

# Define the function V(r,h)
V = pi*(r**2)*h

# The first partial derivative of V w.r.t h (i.e. r is kept constant)
print("The first partial derivative of V w.r.t. h is: ", diff(V, h))
# The first partial derivative of V w.r.t r (i.e. h is kept constant)
print("The first partial derivative of V w.r.t. r is: ", diff(V, r))
```

```
The first partial derivative of V w.r.t. h is:  pi*r**2
The first partial derivative of V w.r.t. r is:  2*pi*h*r
```

In [ ]:
```
r = np.arange(0, 10, 0.1)
h = np.arange(0, 10, 0.1)
R, H = np.meshgrid(r, h)
V = np.pi * R**2 * H

fig = plt.figure(figsize=(14,6))
ax1 = fig.add_subplot(111, projection='3d')

ax1.plot_surface(R, H, V, cmap='coolwarm', edgecolor='none')
ax1.set_xlabel('r')
ax1.set_ylabel('h')
ax1.set_zlabel('V')
ax1.set_title('Surface plot of V')
plt.show()
```



Second partial derivatives

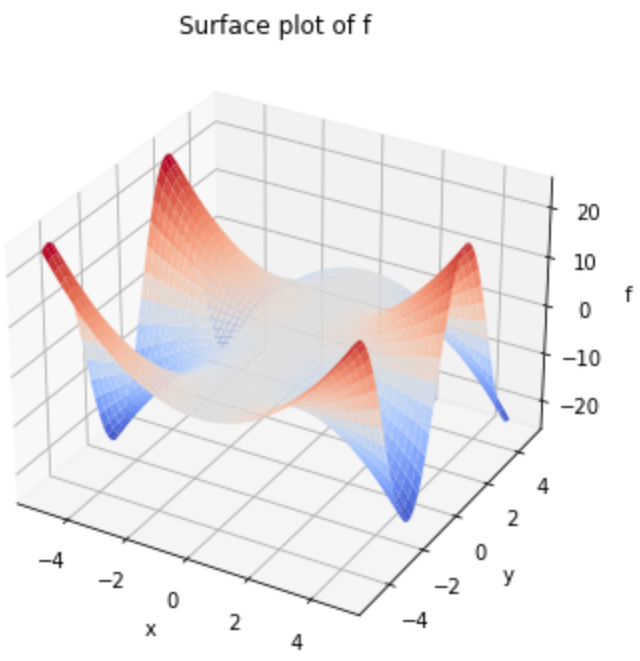
In [ ]:
```
x = Symbol('x')
y = Symbol('y')
f = (x**2)*sin(y)

f_x = diff(f, x)
f_y = diff(f, y)

print("The first partial derivatives of f = (x**2)*sin(y) are: ")
print("f_x = ", f_x)
print("f_y = ", f_y)

f_xx = diff(f_x, x)
f_xy = diff(f_x, y)
f_yx = diff(f_y, x)
f_yy = diff(f_y, y)

print("The second partial derivatives of f = (x**2)*sin(y) are: ")
print("f_xx = ", f_xx)
print("f_xy = ", f_xy)
print("f_yy = ", f_yy)
print("f_yx = ", f_yx)

if f_xy == f_yx:
```

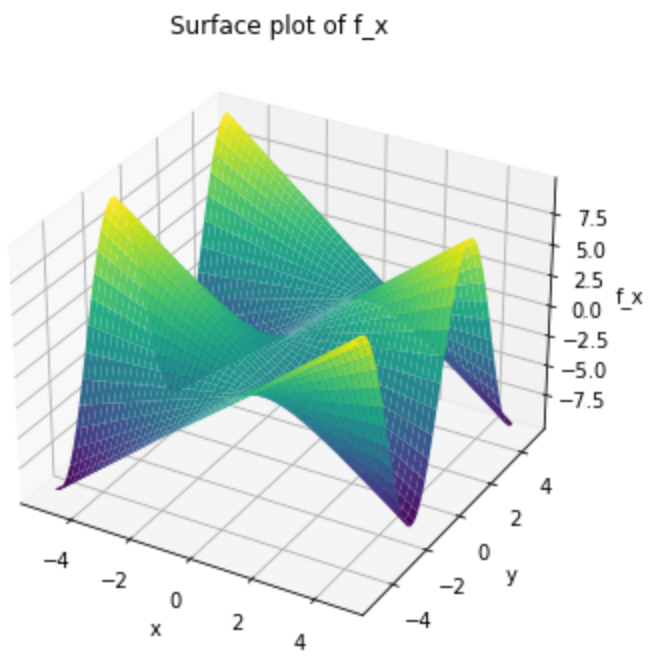
```
    print("\nf_xy = f_yx")
else:
    print("Error")
```

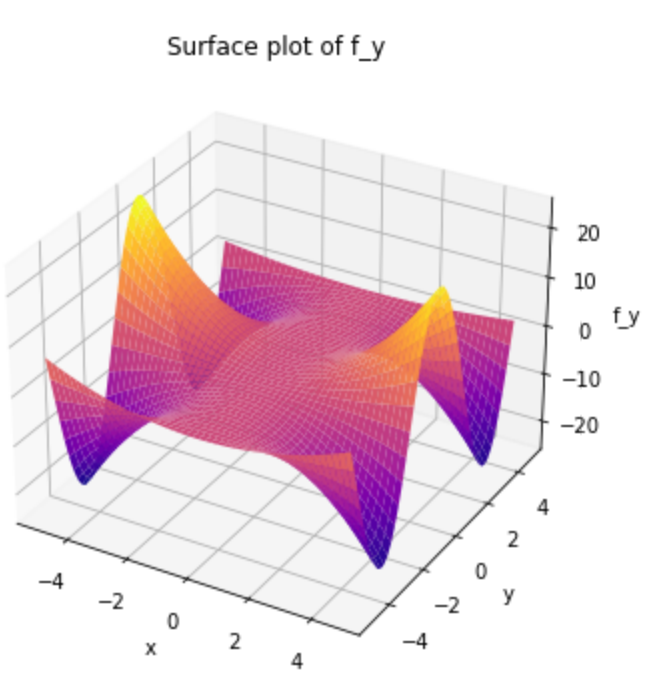
```
The first partial derivatives of f = (x**2)*sin(y) are:
f_x =  2*x*sin(y)
f_y =  x**2*cos(y)
The second partial derivatives of f = (x**2)*sin(y) are:
f_xx =  2*sin(y)
f_xy =  2*x*cos(y)
f_yy =  -x**2*sin(y)
f_yx =  2*x*cos(y)

f_xy = f_yx
```

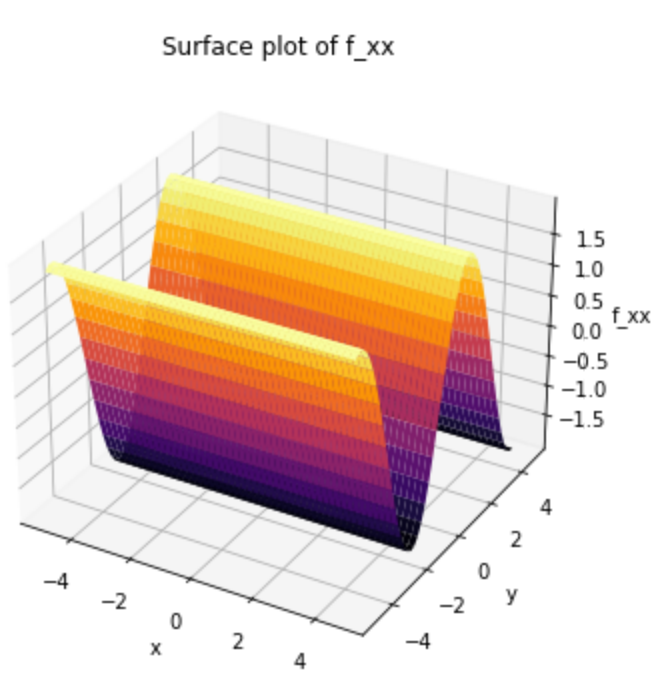
In [ ]:

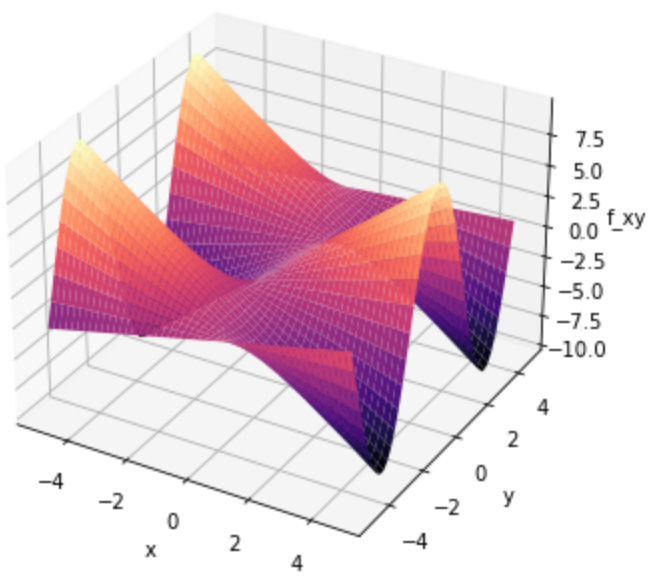
```
x = np.arange(-5, 5, 0.2)
X, Y = np.meshgrid(x, x)

f = X**2 * np.sin(Y)
f_x = 2 * X * np.sin(Y)
f_y = X**2 * np.cos(Y)
f_xx = 2 * np.sin(Y)
f_xy = 2 * X * np.cos(Y)
f_yy = -X**2 * np.sin(Y)

graphs = [f, f_x, f_y, f_xx, f_xy, f_yy]
titles = ['f', 'f_x', 'f_y', 'f_xx', 'f_xy', 'f_yy']
cmaps = ['coolwarm', 'viridis', 'plasma', 'inferno', 'magma', 'cividis']

fig, ax = plt.subplots(3, 2, figsize=(12, 24), subplot_kw={'projection':'3d'})
ax = ax.flatten()

for i in range(len(ax)):
    ax[i].plot_surface(X, Y, graphs[i], cmap=cmaps[i], edgecolor='none')
    ax[i].set_xlabel('x')
    ax[i].set_ylabel('y')
    ax[i].set_zlabel(titles[i])
    ax[i].set_title(f'Surface plot of {titles[i]}')

plt.show()
```

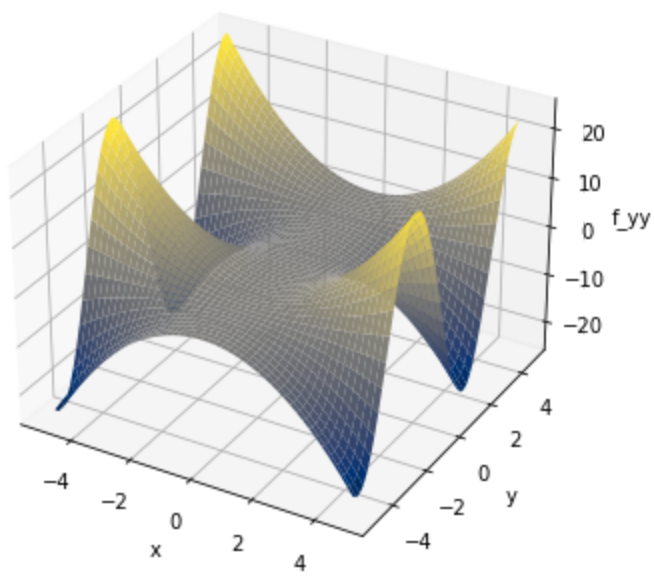












Chain rule

Slide 19

In [ ]:
```
a, b, t = symbols('a b t')
x = a*t
y = b*t
T = 3*y*sin(x)
# SymPy automatically applies the chain rule here:
print("diff(T, t) = ",diff(T, t))
```

```
diff(T, t) =  3*a*b*t*cos(a*t) + 3*b*sin(a*t)
```

Definite and indefinite integrals

In [ ]:
```
x, y = symbols('x y')

# Remember: Indefinite integrals result in a constant 'c'. SymPy sets this to zero.
```

```
# f is the function we want to integrate.
f = cos(x)

# The second argument is the variable we want to integrate with respect to.
# (in this example, it is 'x').
print("Integrating cos(x) yields:", integrate(f, x))

# Using integrate(2*x, (x, a, b)) evaluates a DEFINITE integral between x=a and x=b
a = 0
b = 2
print("Integrating 2*x between x=0 and x=2 yields:", integrate(2*x, (x, a, b)))
```

```
Integrating cos(x) yields: sin(x)
Integrating 2*x between x=0 and x=2 yields: 4
```

Double integrals

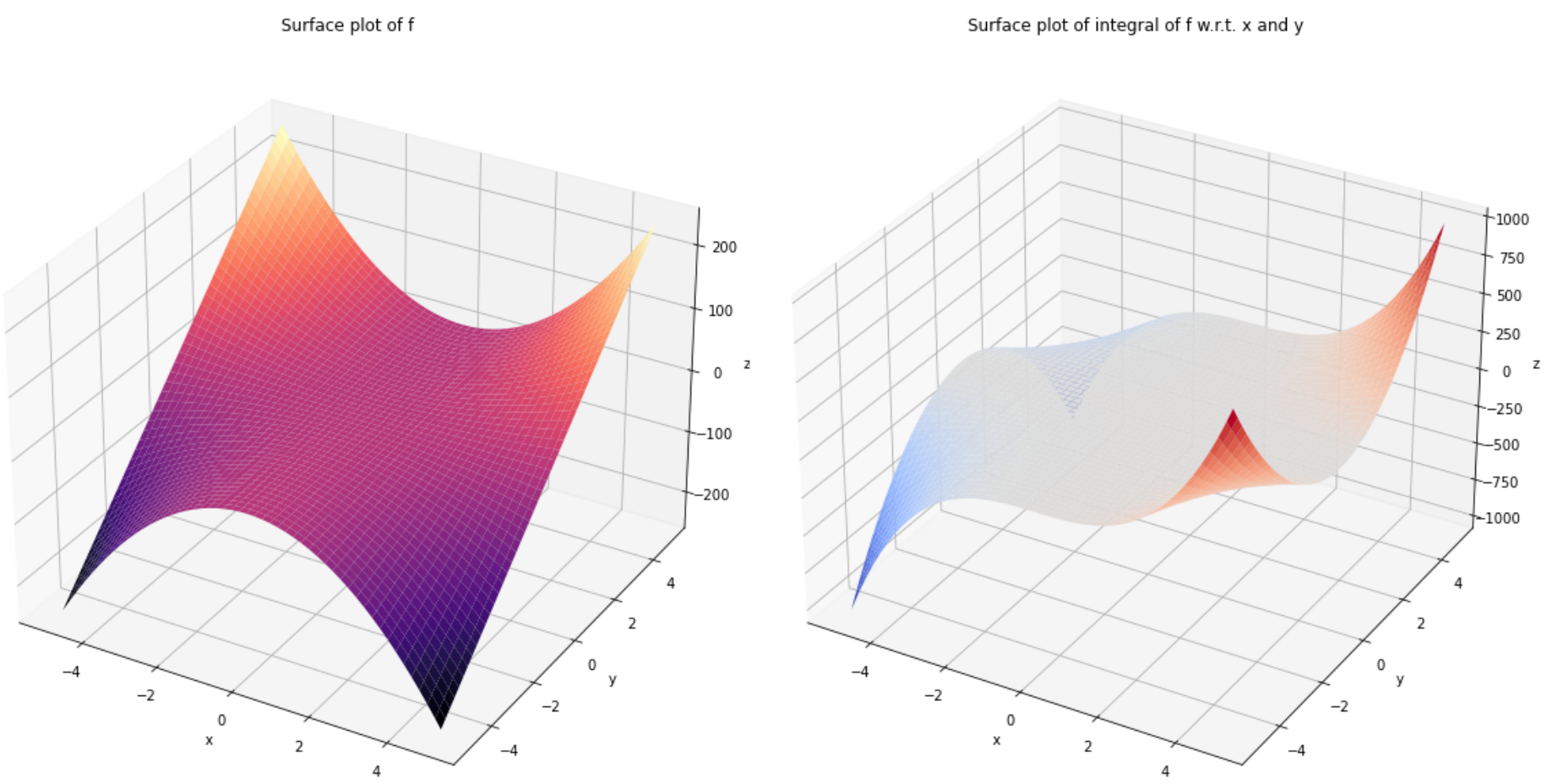
In [ ]:
```
# The function we want to integrate.
f = 2*(x**2)*y

# First integrate f wrt y between y=0 and y=2.
inner_integral = integrate(f, (y, 0, 2))
print("The inner integral is: ", inner_integral)

# Then integrate the inner_integral wrt x.
outer_integral = integrate(inner_integral, (x, 0, 2))
print("The outer integral is: ", outer_integral)
```

```
The inner integral is:  4*x**2
The outer integral is:  32/3
```

In [ ]:
```
x = Symbol('x')
y = Symbol('y')
f = Function('f')

f = 2 * x**2 * y

print("Integral of f w.r.t x and y is: ", integrate(f, x, y))
```

```
Integral of f w.r.t x and y is:  x**3*y**2/3
```

In [ ]:
```
x = np.arange(-5, 5, 0.05)
y = np.arange(-5, 5, 0.05)
X, Y = np.meshgrid(x, y)

f = 2 * X**2 * Y
integral = 1/3 * X**3 * Y**2

fig = plt.figure(figsize=(16, 10))

ax1 = fig.add_subplot(121, projection='3d')
ax1.plot_surface(X, Y, f, cmap='magma', edgecolor='none')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('z')
ax1.set_title('Surface plot of f')

ax2 = fig.add_subplot(122, projection='3d')
ax2.plot_surface(X, Y, integral, cmap='coolwarm', edgecolor='none')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('z')
ax2.set_title('Surface plot of integral of f w.r.t. x and y')

fig.tight_layout()
plt.show()
```



In [ ]:

2 Vector calculus (MM1)

{index}

In [ ]:
```
## This cell just imports necessary modules
from sympy import sin, cos, exp, Function, Symbol, diff, integrate, solve, pi
from mpl_toolkits import mplot3d
import numpy
import matplotlib.pyplot as plt
```

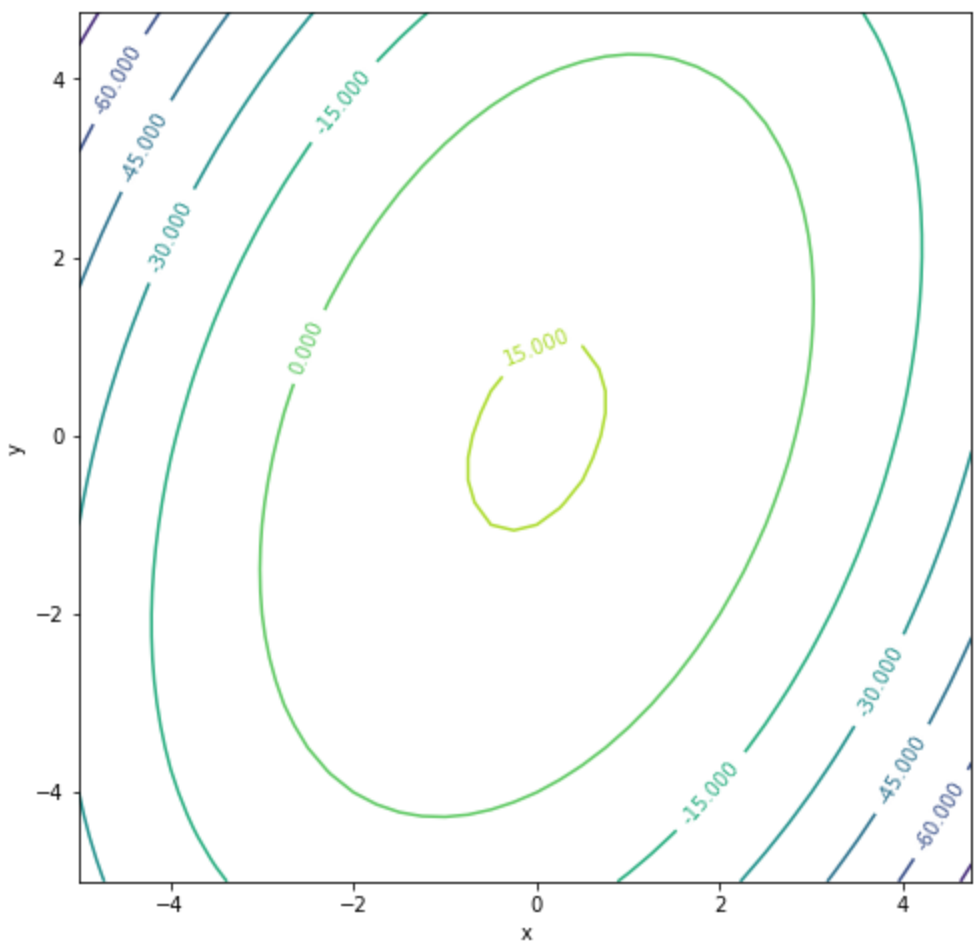
Scalar fields

Slide 3

In [ ]:
```
# Create a mesh of 2D Cartesian coordinates, where -5 <= x <= 5 and -5 <= y <= 5
x = numpy.arange(-5., 5., 0.25)
y = numpy.arange(-5., 5., 0.25)
X, Y = numpy.meshgrid(x, y)

# Computes the value of the scalar field at each (x,y) coordinate, and stores it in Z.
f = 16 - 2*X**2 - Y**2 + X*Y
```

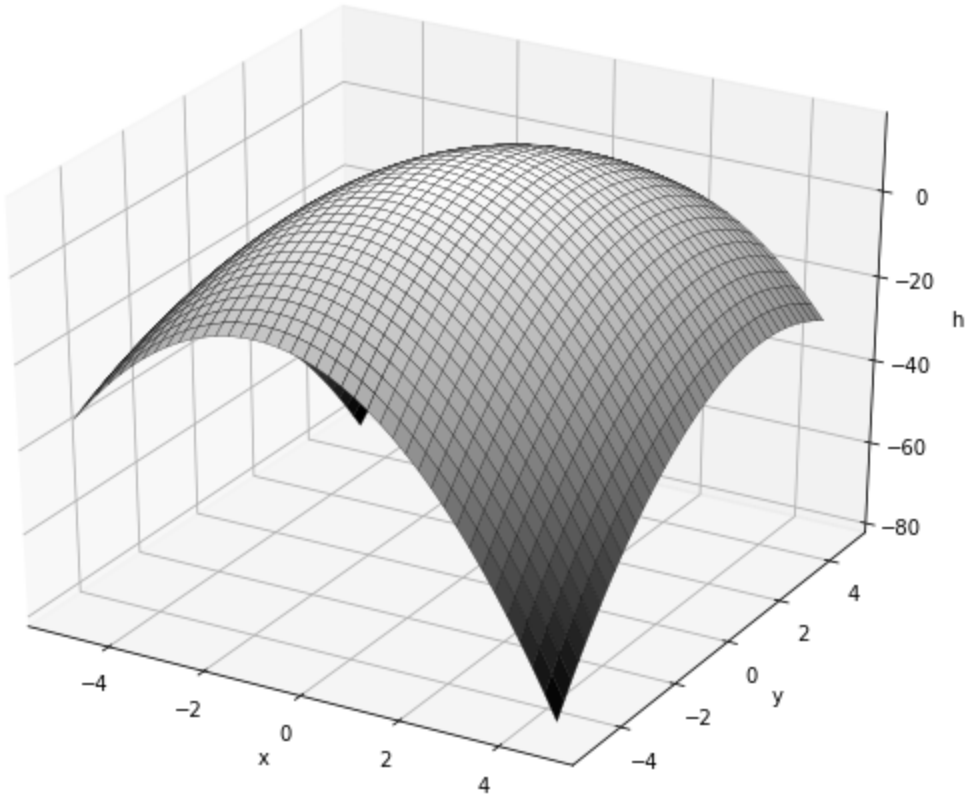
Contour plot of the scalar field $f(x, y) = 16 - 2x^2 - y^2 + xy$

In [ ]:
```
fig = plt.figure(figsize=(8, 8))
contour_plot = plt.contour(X, Y, f)
plt.clabel(contour_plot, inline=True)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



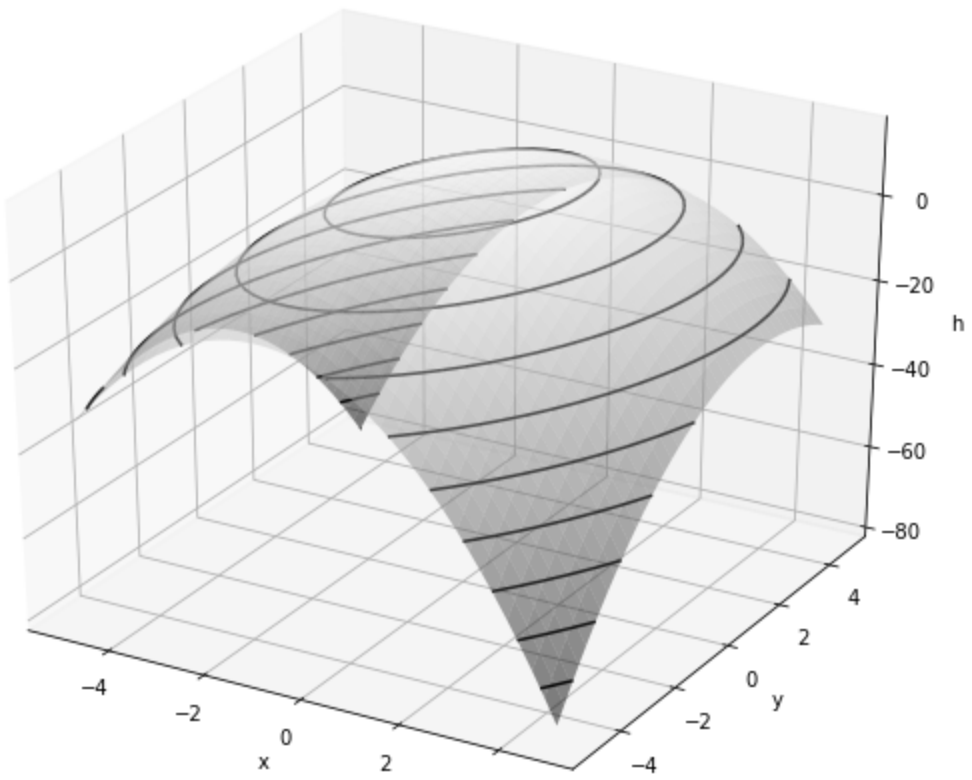
Surface plot of the scalar field $f(x, y) = 16 - 2x^2 - y^2 + xy$

In [ ]:
```
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, f, cmap='gray', edgecolor = 'k', lw=0.25)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("h")
plt.show()
```



Surface and contour plot of the scalar field $f(x, y) = 16 - 2x^2 - y^2 + xy$

In [ ]:
```
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, f, cmap='gray', alpha=0.5)
ax.contour3D(X, Y, f, 10, colors='k', linestyles='-')
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("h")
plt.show()
```

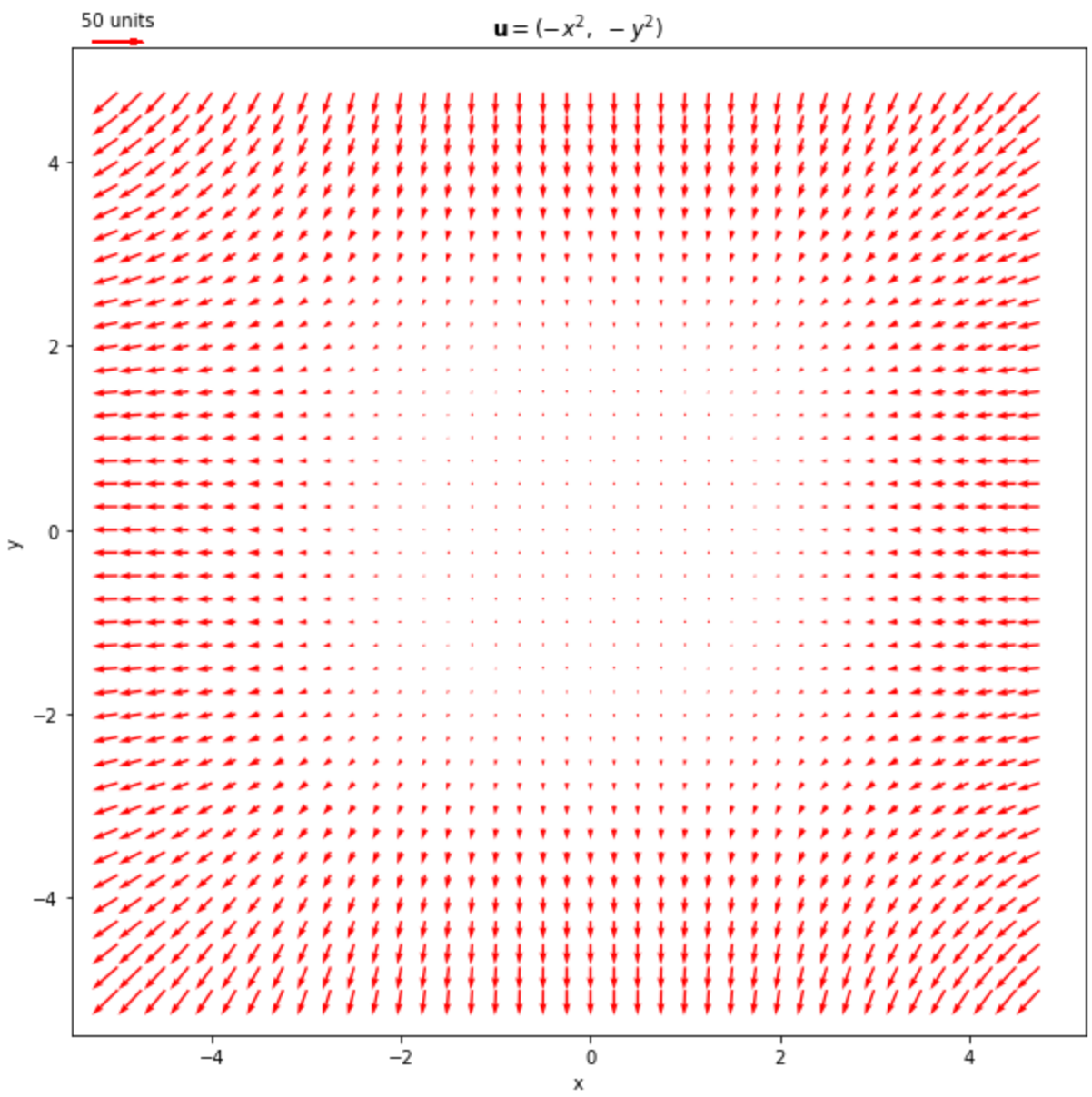


Vector fields

Slide 4

Define and plot the vector field $u = (-x^2, \ -y^2)$ on a quiver plot

In [ ]:
```
# Create a mesh of 2D Cartesian coordinates, where -5 <= x <= 5 and -5 <= y <= 5
x = numpy.arange(-5.0, 5.0, 0.25)
y = numpy.arange(-5.0, 5.0, 0.25)
X, Y = numpy.meshgrid(x, y)

# Computes the value of the vector field at each (x,y) coordinate.
# Z1 and Z2 hold the i and j component of the vector field respectively.
Z1 = -(X**2)
Z2 = -(Y**2)

fig = plt.figure(figsize=(10, 10))
quiver_plot = plt.quiver(X, Y, Z1, Z2, angles='xy', scale=1000, color='r')
plt.quiverkey(quiver_plot, -5, 5.3, 50, "50 units", coordinates='data', color='r')
plt.xlabel("x")
plt.ylabel("y")
plt.title(r'$\mathbf{u}=(-x^2, \ -y^2)$')
plt.show()
```



Gradients

Slide 6

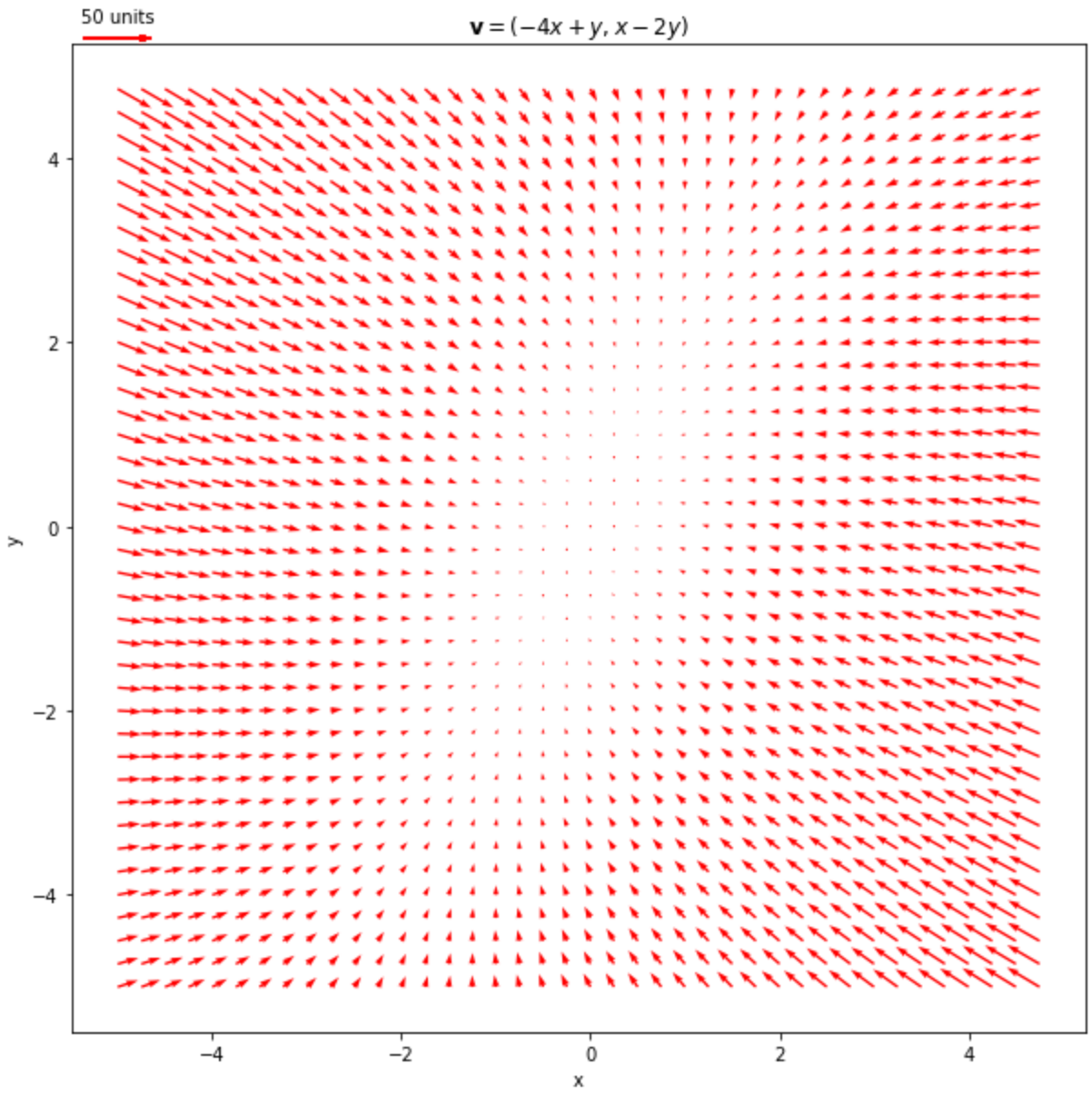
Computing the gradient on scalar field using `sympy.diff`, and finding where `gradient=0` using `sympy.solve`:

In [ ]:
```
# Define the independent variables using Symbol
x = Symbol('x')
y = Symbol('y')
# Define the function f(x,y)
f = 16 - 2*(x**2) - y**2 + x*y

# The gradient of f (a scalar field) is a vector field:
grad_f = [diff(f,x), diff(f,y)]
print("The gradient of the scalar field f(x,y) = 16 - 2*(x**2) - y**2 + x*y is:", grad_f)

print("The point where the gradient is zero is:",
      solve([grad_f[0], grad_f[1]], [x, y]))
```

```
The gradient of the scalar field f(x,y) = 16 - 2*(x**2) - y**2 + x*y is: [-4*x + y, x - 2*y]
The point where the gradient is zero is: {y: 0, x: 0}
```

Let us now plot $v = (-4x + y,\ x - 2y)$

In [ ]:
```
x = numpy.arange(-5., 5., 0.25)
y = numpy.arange(-5., 5., 0.25)
X, Y = numpy.meshgrid(x, y)

Z1 = -4*X + Y
Z2 = X - 2*Y

fig = plt.figure(figsize=(10, 10))
quiver_plot = plt.quiver(X, Y, Z1, Z2, angles='xy', scale=750, color='r')
plt.quiverkey(quiver_plot, -5, 5.3, 50, "50 units", coordinates='data', color='r')
plt.xlabel("x")
plt.ylabel("y")
plt.title(r'$\mathbf{v} = (-4x + y, x - 2y)$')
plt.show()
```



Directional derivatives

Slide 12

Here we will compute the gradient of a scalar field using `sympy.diff`, the gradient at a specific point using `evalf` method, and the gradient at a specific point towards the direction of a unit vector using `numpy.dot`.

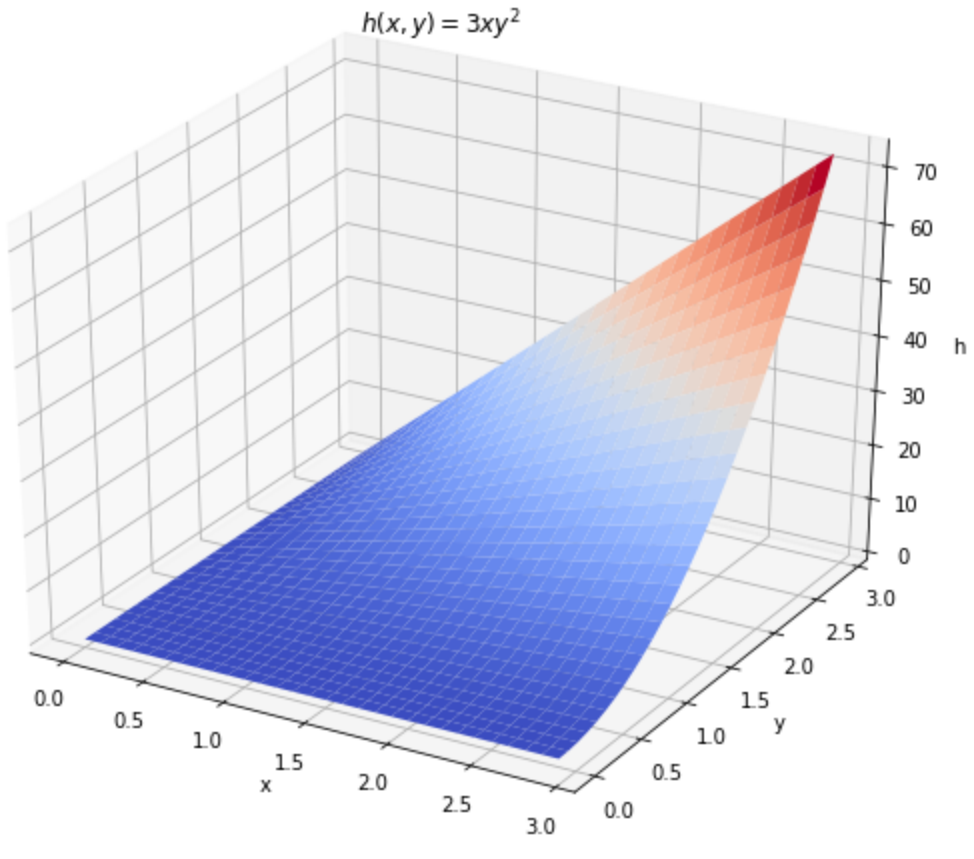
Let us consider a scalar field $h(x, y) = 3xy^2$.

In [ ]:
```
# Define the independent variables using Symbol
x = Symbol('x')
y = Symbol('y')
# Define the function h(x,y)
h = 3*x*(y**2)

# The gradient of h
grad_h = [diff(h,x), diff(h,y)]
print("The gradient of h(x,y) = 3*x*(y**2) is: ")
print(grad_h, "\n")

# Use evalf to evaluate a function, with subs to substitute in specific values for x and y
grad_h_at_point = [grad_h[0].evalf(subs={x:1, y:2}), grad_h[1].evalf(subs={x:1, y:2})]
print("At the point (1,2), the gradient is:", grad_h_at_point, "\n")

# Find the unit vector in the direction 3i + 4j
a = numpy.array([3, 4])
a_magnitude = numpy.linalg.norm(a, ord=2)
unit_a = a/a_magnitude

print("The unit vector in the direction 3i + 4j is:")
print(unit_a, "\n")

# Dot product to get the directional derivative
# (i.e. the gradient of h in the direction of the vector unit_a)
slope = numpy.dot(grad_h_at_point, unit_a)
print("The slope of h in the direction", unit_a, "at (1,2) is:", slope)
```

```
The gradient of h(x,y) = 3*x*(y**2) is:
[3*y**2, 6*x*y]

At the point (1,2), the gradient is: [12.0000000000000, 12.0000000000000]

The unit vector in the direction 3i + 4j is:
[0.6 0.8]

The slope of h in the direction [0.6 0.8] at (1,2) is: 16.8000000000000
```

In [ ]:
```
x = numpy.arange(0., 3., 0.1)
y = numpy.arange(0., 3., 0.1)
X, Y = numpy.meshgrid(x, y)
h = 3 * X * Y**2

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, h, cmap='coolwarm', lw=0.25)
ax.set_xlabel("x")
```

```
ax.set_ylabel("y")
ax.set_zlabel("h")
ax.set_title(r'$h(x,y)=3xy^2$')
plt.show()
```



Divergence

Slide 15

We will compute the divergence of a vector field using `sympy.diff`.

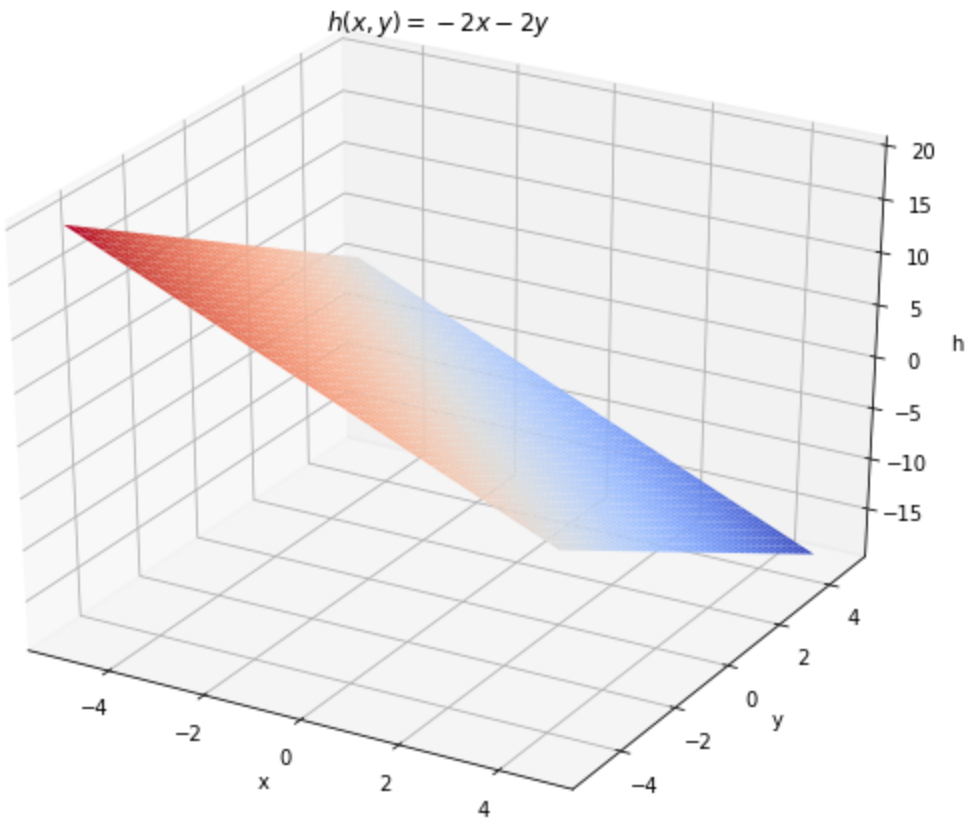
In [ ]:
```
# Define the independent variables using Symbol
x = Symbol('x')
y = Symbol('y')
# Define the vector field v(x,y)
v = [-(x**2), -(y**2)]

# Compute the divergence using diff.
divergence = diff(v[0],x) + diff(v[1],y)
print("The divergence of the vector field", v, "is:", divergence)
```

```
The divergence of the vector field [-x**2, -y**2] is: -2*x - 2*y
```

{note}
NOTE 1: A neater way would be to use SymPy's dot function. However, there doesn't seem to be a way of defining a gradient vector in SymPy without specifying the function we wish to operate on, so we'll compute the divergence the long way.

NOTE 2: this is the dot product of the gradient vector and v, which will always result in a scalar. d/dx is applied to the first component of v (i.e. v[0]), d/dy is applied to the second component of v (i.e. v[1])

In [ ]:
```
# Create a mesh of 2D Cartesian coordinates, where -5 <= x <= 5 and -5 <= y <= 5
x = numpy.arange(-5., 5., 0.25)
y = numpy.arange(-5., 5., 0.25)
X, Y = numpy.meshgrid(x, y)

h = -2 * X - 2 * Y

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, h, cmap='coolwarm', edgecolor = 'none', lw=0.25)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("h")
ax.set_title(r'$h(x,y)=-2x-2y$')
plt.show()
```



Curl

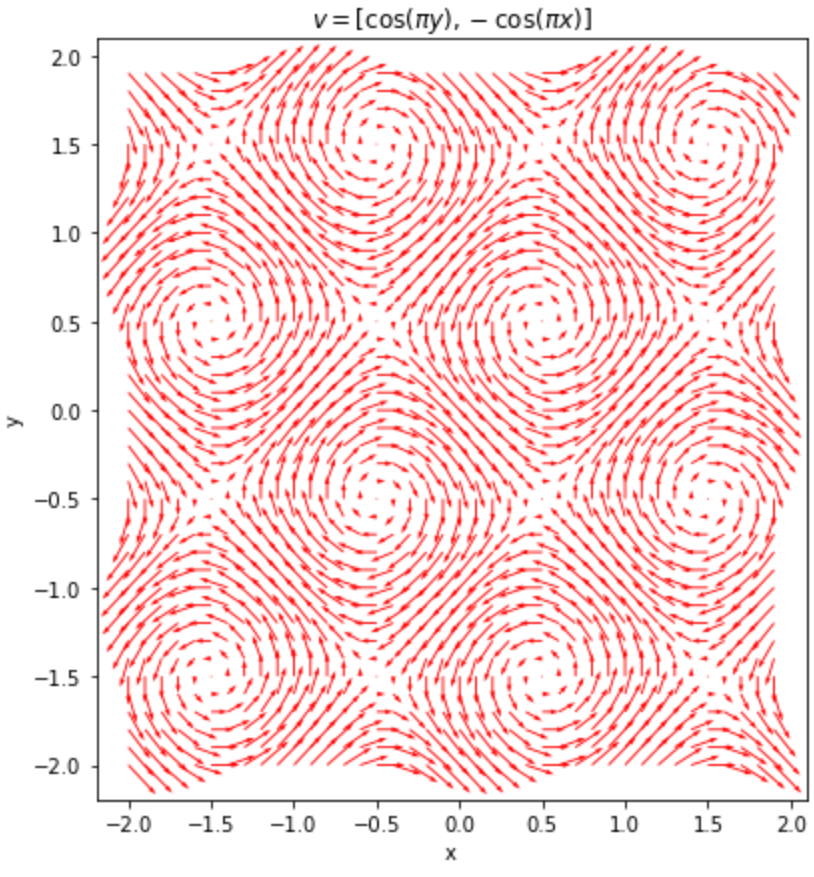
Slide 19

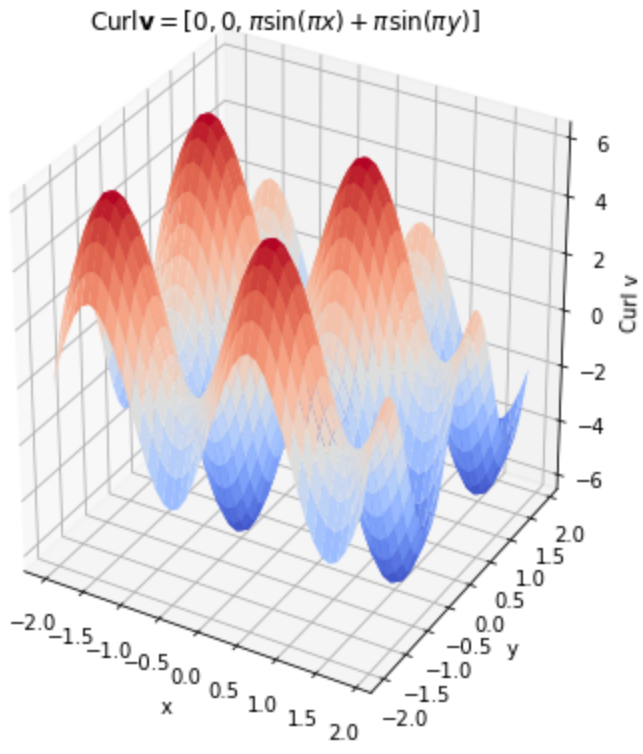
Compute the curl of a vector field using `sympy.diff`, and the curl at a specific point using `evalf`. Remember: the curl of a vector always results in another vector.

In [ ]:
```
# Define the independent variables using Symbol
x = Symbol('x')
y = Symbol('y')
# Define the vector field v(x,y)
v = [cos(pi*y), -cos(pi*x)]

# Compute the curl using diff.
# The first two components of the curl are zero because v has a zero k-component.
curl = [0, 0, diff(v[1], x) - diff(v[0], y)]
print("The curl of the vector field", v, "is:", curl)
print("At the point (0, -0.5), the curl is:",
      [0, 0, curl[2].evalf(subs={x:0, y:-0.5})])
```

```
The curl of the vector field [cos(pi*y), -cos(pi*x)] is: [0, 0, pi*sin(pi*x) + pi*sin(pi*y)]
At the point (0, -0.5), the curl is: [0, 0, -3.14159265358979]
```

In [ ]:
```
x = numpy.arange(-2., 2., 0.1)
y = numpy.arange(-2., 2., 0.1)
X, Y = numpy.meshgrid(x, y)

# Computes the value of the vector field at each (x,y) coordinate.
# Z1 and Z2 hold the i and j component of the vector field respectively.
Z1 = numpy.cos(numpy.pi*Y)
Z2 = -numpy.cos(numpy.pi*X)

# Curl v
Z = numpy.pi*numpy.sin(numpy.pi*X) + numpy.pi*numpy.sin(numpy.pi*Y)

ax = [0, 0]
fig = plt.figure(figsize=(14, 7))
ax[0] = fig.add_subplot(121)
ax[1] = fig.add_subplot(122, projection='3d')

ax[0].quiver(X, Y, Z1, Z2, angles='xy', scale=25, color='r')
ax[1].plot_surface(X, Y, Z, cmap='coolwarm', lw=0.25)

for i in range(2):
    ax[i].set_xlabel("x")
    ax[i].set_ylabel("y")

ax[0].set_title(r'$v=[\cos(\pi y), -\cos(\pi x)]$')
ax[1].set_zlabel("Curl v")
ax[1].set_title('Curl' + r'$\mathbf{v} = [0, 0, \pi \sin(\pi x)+\pi \sin(\pi y)]$')

plt.show()
```





Laplacian

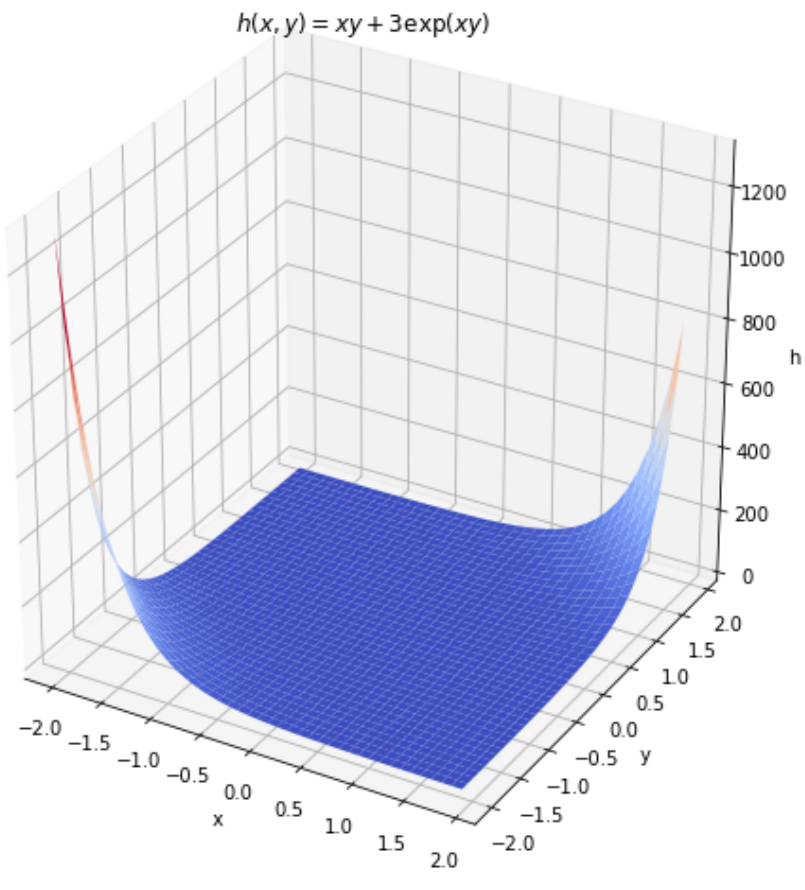
Slide 22

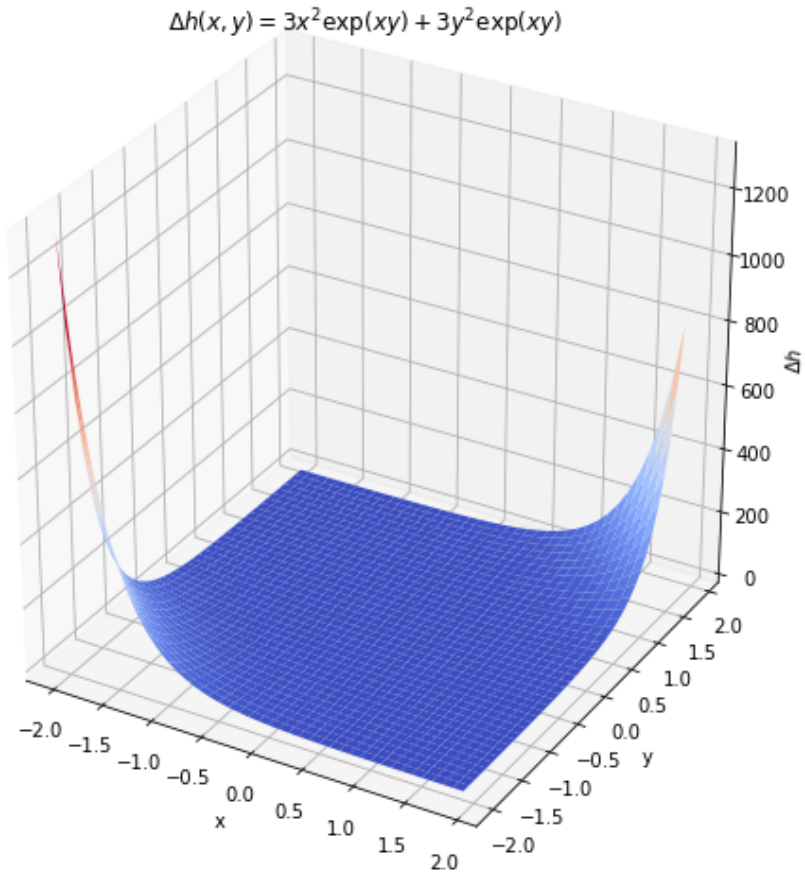
Here we compute the Laplacian of a scalar field using `sympy.diff`. Consider a scalar field $h(x, y) = xy + 3\exp(xy)$:

In [ ]:
```
# Define the independent variables using Symbol
x = Symbol('x')
y = Symbol('y')
# Define the scalar field f(x,y)
f = x*y + 3*exp(x*y)

# In SymPy we can specify the order of the derivative as an optional argument
# (in this case, it is '2' to get the second derivative).
laplacian = diff(f, x, 2) + diff(f, y, 2)
print("The Laplacian of", f, "is:", laplacian)
```

```
The Laplacian of x*y + 3*exp(x*y) is: 3*x**2*exp(x*y) + 3*y**2*exp(x*y)
```

In [ ]:
```
x = numpy.arange(-2, 2, 0.1)
y = numpy.arange(-2, 2, 0.1)
X, Y = numpy.meshgrid(x, y)

h = X * Y + 3 * numpy.exp(X*Y)

# Laplacian
h = 3*X**2*numpy.exp(X*Y) + 3*Y**2*numpy.exp(X*Y)

ax = [0, 0]
fig = plt.figure(figsize=(14, 7))
ax[0] = fig.add_subplot(121, projection='3d')
ax[1] = fig.add_subplot(122, projection='3d')

ax[0].plot_surface(X, Y, h, cmap='coolwarm', lw=0.25)
ax[1].plot_surface(X, Y, h, cmap='coolwarm', lw=0.25)

for i in range(2):
    ax[i].set_xlabel("x")
    ax[i].set_ylabel("y")

ax[0].set_zlabel("h")
ax[0].set_title(r'$h(x,y)=xy+3\exp(xy)$')
ax[1].set_zlabel(r"$\Delta h$")
ax[1].set_title(r'$\Delta h(x,y)=3x^2\exp(xy)+3y^2\exp(xy)$')

fig.tight_layout()
plt.show()
```





In [ ]:

3 Vector calculus (MM3)

{index}

Vector functions

{index}

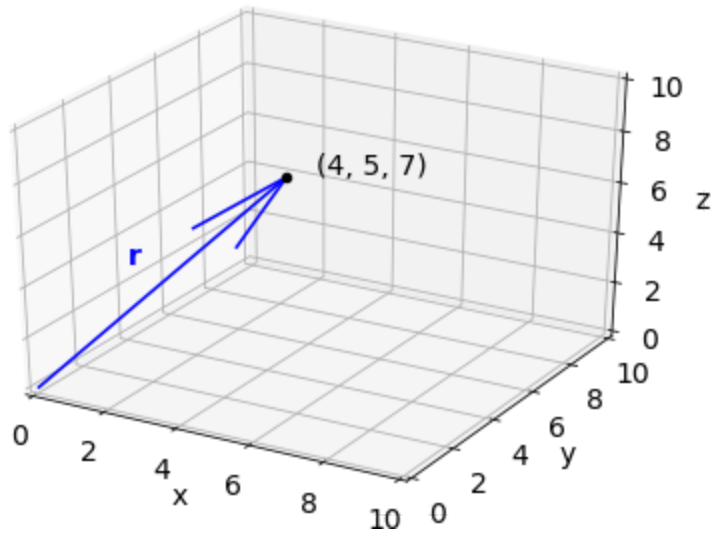
Vector is a quantity that is described with magnitude and direction. A *vector function* describes a vector whole value is a function of some independent variable.

For example, if $f(t)$ represents a real function of t, then vector function of t would be:

$$\mathbf{F}(t) = [f_x(t), f_y(t), f_z(t)].$$

We can visualise a vector in Cartesian (x, y, z) cooridante system as an arrow beginning at the origin and ending at point specified by the vector:

In [ ]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# Create plot
fig = plt.figure(figsize=(7,5))

# Add subplot with 3D axes projection
ax = fig.add_subplot(111, projection='3d')

ax.quiver(0, 0, 0, 4, 5, 7,
          color="blue")

ax.scatter(4, 5, 7, c="black")

ax.text(4.5, 5.5, 7, "(4, 5, 7)", fontsize=14)
ax.text(2, 1, 5, "$\mathbf{r}$", fontsize=14, color="blue")

ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")

ax.set_xlim(0,10)
ax.set_ylim(0,10)
ax.set_zlim(0,10)

plt.show()
```



Vector (4, 5, 7) can be represented with *position vector* $\mathbf{r}$ plotted on the graph.

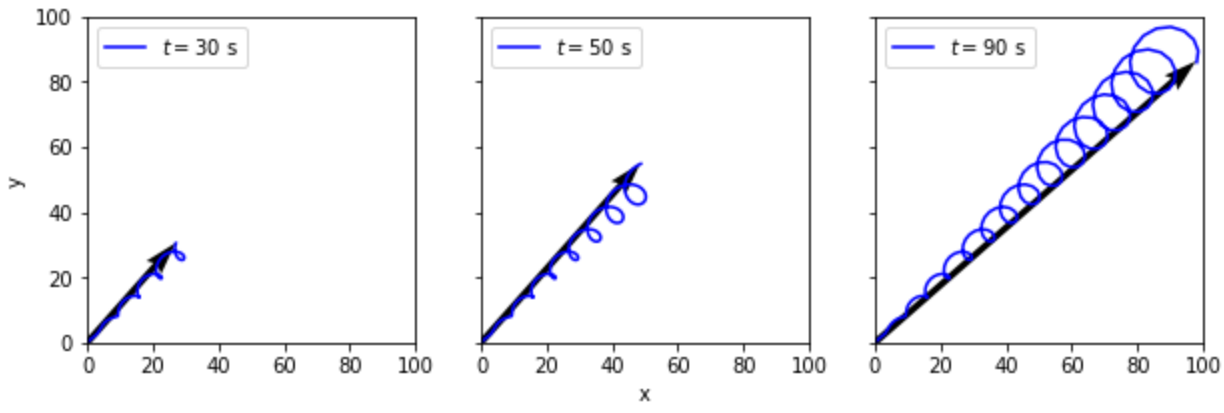
Example:

Describe the shape of the geometric representation of the vector function given by

$$\mathbf{F}(t) = [t + (t/10)\sin t, t + (t/10)\cos t].$$

In [ ]:
```
t = np.linspace(0,100, 301)

def F(t):
    return np.array([t+t/10*np.sin(t), t+(t/10.)*np.cos(t)])

t1 = np.linspace(0, 30, 200)
t2 = np.linspace(0, 50, 200)
t3 = np.linspace(0, 90, 200)
```

In [ ]:
```
fig, axes = plt.subplots(1, 3, figsize=(10,3), sharey=True)

ax1 = axes[0]
ax2 = axes[1]
ax3 = axes[2]

ax1.plot(F(t1)[0], F(t1)[1], label="$t=30$ s", color="blue")
ax2.plot(F(t2)[0], F(t2)[1], label="$t=50$ s", color="blue")
ax3.plot(F(t3)[0], F(t3)[1], label="$t=90$ s", color="blue")

ax1.quiver(0, 0, F(t1)[0,-1], F(t1)[1,-1],
           scale=1, scale_units="xy", width=0.02)

ax2.quiver(0, 0, F(t2)[0,-1], F(t2)[1,-1],
           scale=1, scale_units="xy", width=0.02)

ax3.quiver(0, 0, F(t3)[0,-1], F(t3)[1,-1],
           scale=1, scale_units="xy", width=0.02)

for ax in axes:
    ax.set_xlim(0,100)
    ax.set_ylim(0,100)
    ax.legend(loc="upper left")

ax1.set_ylabel("y")
ax2.set_xlabel("x")


plt.show()
```



Differentiation

{index}

To differentiate vector functions, we need to take derivatives of its components:

$$\frac{d\mathbf{F}}{dt} = \left(\frac{df_x}{dt}, \frac{df_y}{dt}, \frac{df_z}{dt}\right).$$

Example:

Let's find the derivative of $\mathbf{F}(t) = [t + (t/10)\sin t, t + (t/10)\cos t]$ with respect to t. We will differentiate each component to get:

$$\frac{d\mathbf{F}}{dt} = \left[\left(1 + \frac{1}{10}\sin t + \frac{t}{10}\cos t\right), \left(1 + \frac{1}{10}\cos t - \frac{t}{10}\sin t\right)\right].$$

We can also use SymPy package for Python that allows symbolic mathematics. With this package we can get analytical solutions to derivatives in this example. At first we can create separate derivatives:

{index}

In [ ]:
```
import sympy as sym
from IPython.display import display
from sympy import init_printing

init_printing(use_latex='mathjax')

# Create a symbol t variable will go by
t = sym.Symbol("t")

# Differentiate Fx with respect to t
print('dfx/dt =')
display(sym.diff(t+(t/10)*sym.sin(t), t))

# Differentiate Fy with respect to t
print('dfy/dt =')
display(sym.diff(t+(t/10)*sym.cos(t), t))
```

```
dfx/dt =
```

$$\frac{t\cos(t)}{10} + \frac{\sin(t)}{10} + 1$$

```
dfy/dt =
```

$$-\frac{t\sin(t)}{10} + \frac{\cos(t)}{10} + 1$$

How can we represent vector differentiation geometrically?

For that, we will use the position vector $\mathbf{r}$. Letting $\mathbf{F}(t) = \mathbf{r}(t)$:

$$\frac{d\mathbf{F}}{dt} = \frac{d\mathbf{r}}{dt} = \lim_{\delta t \to 0} \frac{\mathbf{r}(t+\delta t) - \mathbf{r}(t)}{\delta t} = \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt}\right).$$

Let's find tangent $\mathbf{t}(t)$ to the curve created by $\mathbf{r}(t) = [10 + 5\sin t, 10 + 5\cos t]$ as t goes from 0 to 2π.

In [ ]:
```
t = sym.Symbol("t")

print('dx/dt =')
display(sym.diff(10+5*sym.sin(t)))

print('dy/dt =')
display(sym.diff(10+5*sym.cos(t)))
```

```
dx/dt =
```

$$5\cos(t)$$

```
dy/dt =
```

$$-5\sin(t)$$

In [ ]:
```
def r(t):
    return np.array([10+5*np.sin(t), 10+5*np.cos(t)])

def dr(t):
    return np.array([5*np.cos(t), -5*np.sin(t)])
```

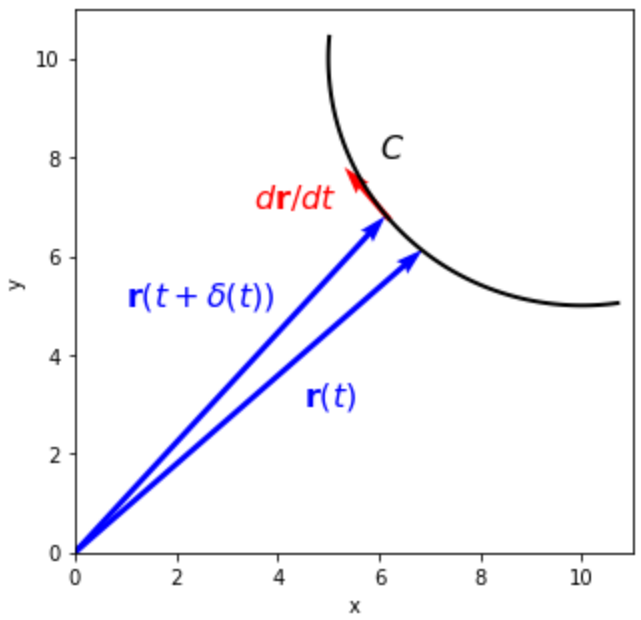
Unit tangent $\hat{\mathbf{t}}$ to the curve C can be defined as:

$$\hat{\mathbf{t}} = \frac{d\mathbf{r}/dt}{|d\mathbf{r}/dt|}.$$

Unit tangent is a vector in the same direction as vector derivative but its length is one.

In [ ]:
```
t = np.linspace(3,4.8,100)

plt.figure(figsize=(5,5))

plt.plot(r(t)[0], r(t)[1], color="black",
        lw=2)


plt.quiver(0, 0, r(3.8)[0], r(3.8)[1],
           color="blue", scale=1,
           scale_units="xy", width=0.01)

plt.quiver(0, 0, r(4)[0], r(4)[1],
           color="blue", scale=1,
           scale_units="xy", width=0.01)

# Calculate dr(4) unit vector
```

```
# linalg.norm gets the length of a vector
x = (dr(4)/np.linalg.norm(dr(4)))[0]
y = (dr(4)/np.linalg.norm(dr(4)))[1]

# Plot unit vector exaggerated
plt.quiver(r(4)[0], r(4)[1],
           x, y,
           scale=8, color="red", width=0.01)

plt.text(4.5, 3, r"$\mathbf{r}(t)$", color="blue",
         fontsize=16)

plt.text(1, 5, r"$\mathbf{r}(t+\delta(t))$", color="blue",
         fontsize=16)

plt.text(6, 8, r"$C$", color="black", fontsize=16)
plt.text(3.5, 7, r"$d\mathbf{r}/dt$", color="red", fontsize=16)

plt.xlim(0,11)
plt.ylim(0,11)

plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



If vectors $\mathbf{a}$, $\mathbf{b}$ and scalar λ are functions of t, we can then differentiate:

$$\frac{d}{dt}(\mathbf{a}+\mathbf{b}) = \frac{d\mathbf{a}}{dt} + \frac{d\mathbf{b}}{dt},$$

$$\frac{d}{dt}(\lambda\mathbf{a}) = \frac{d\lambda}{dt}\mathbf{a} + \frac{d\mathbf{a}}{dt}\lambda,$$

$$\frac{d}{dt}(\mathbf{a}\cdot\mathbf{b}) = \frac{d\mathbf{a}}{dt}\cdot\mathbf{b} + \frac{d\mathbf{b}}{dt}\cdot\mathbf{a} = \frac{d}{dt}(\mathbf{b}\cdot\mathbf{a}),$$

$$\frac{d}{dt}(\mathbf{a}\times\mathbf{b}) = \frac{d\mathbf{a}}{dt}\times\mathbf{b} + \frac{d\mathbf{b}}{dt}\times\mathbf{a} \neq \frac{d}{dt}(\mathbf{b}\times\mathbf{a}).$$

Scalar and vector fields

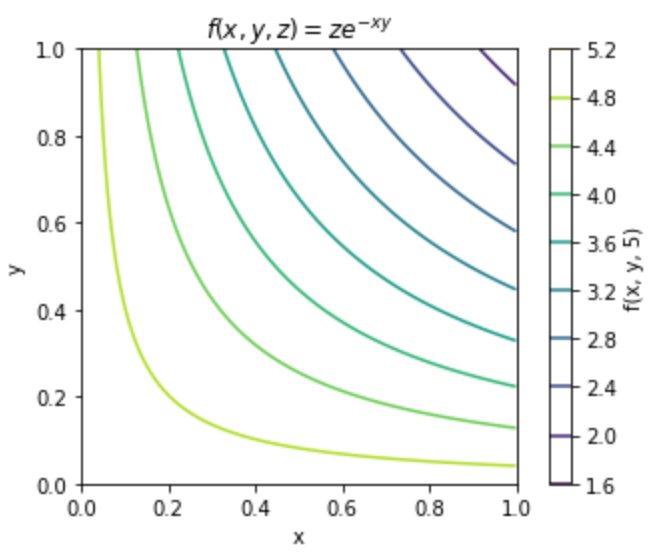
{index}

Scalar field is a scalar quantity that varies with position and can be written as $\Omega(x,y,z) = \Omega(\mathbf{r})$. Scalar fields are for example temperature and density. 3D scalar fields can be represented as iso-surfaces in 3D or contour plots in 2D for specific z value.

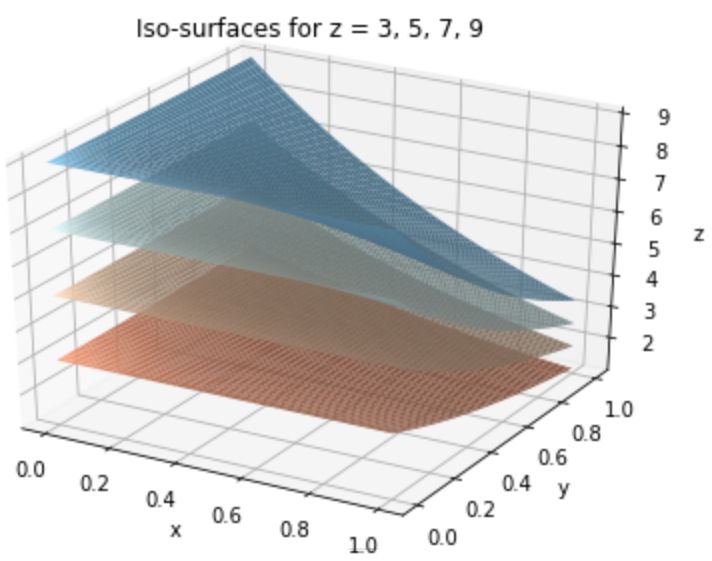
For example, let's plot a scalar field:

$$f(x,y,z) = ze^{-xy}.$$

In [ ]:
```
# Create x and y values between 0 and 1
x = np.linspace(0, 1, 100)
y = np.linspace(0, 1, 100)

# Create 2D grid ourt of x and y
X, Y = np.meshgrid(x, y)
Z = 5

# Create a contour plot
plt.contour(X, Y, Z*np.exp(-X*Y))

plt.colorbar(label="f(x, y, %g)" % Z)

plt.title(r"$f(x, y, z) = ze^{-xy}$")
plt.xlabel("x")
plt.ylabel("y")

plt.gca().set_aspect("equal")

plt.show()
```



In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D

# Create plot
fig = plt.figure(figsize=(7,5))

# Add subplot with 3D axes projection
ax = fig.add_subplot(111, projection='3d')

colours = ["lightsalmon", "peachpuff", "lightcyan", "lightskyblue"]

# Plot surface
for idx, z in enumerate(range(3,10,2)):
    ax.plot_surface(X, Y, z*np.exp(-X*Y), color=colours[idx])

# Add extras
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")

ax.set_title("Iso-surfaces for z = 3, 5, 7, 9")

# Display
plt.show()
```



{index}

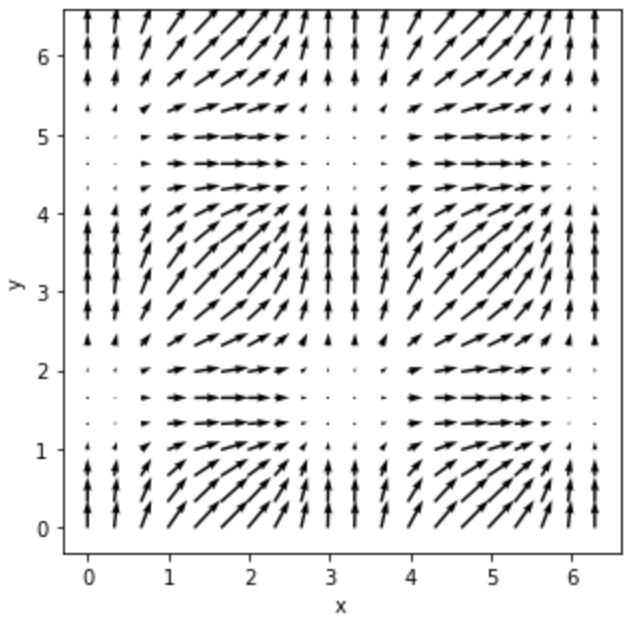
Vector field is a vector quantity that varies as a function of position and can be written as $\mathbf{F}(x, y, z) = \mathbf{F}(\mathbf{r})$. A vector field is for example ocean currents or wind field.

One way to visualise vector field is to use a *quiver plot*, where at each location we can draw an arrow with length and direction based on magnitude and direction of the field at that location. Quiver plots in Python can be plotted using 2D array of locations and x and y components of vector field at each location:

{index}

In [ ]:
```
# Locations
x = np.linspace(0, 2*np.pi, 20)
y = np.linspace(0, 2*np.pi, 20)

X, Y = np.meshgrid(x,y)

# x and y components of the field
U = np.sin(X)**2
V = np.cos(Y)**2
```

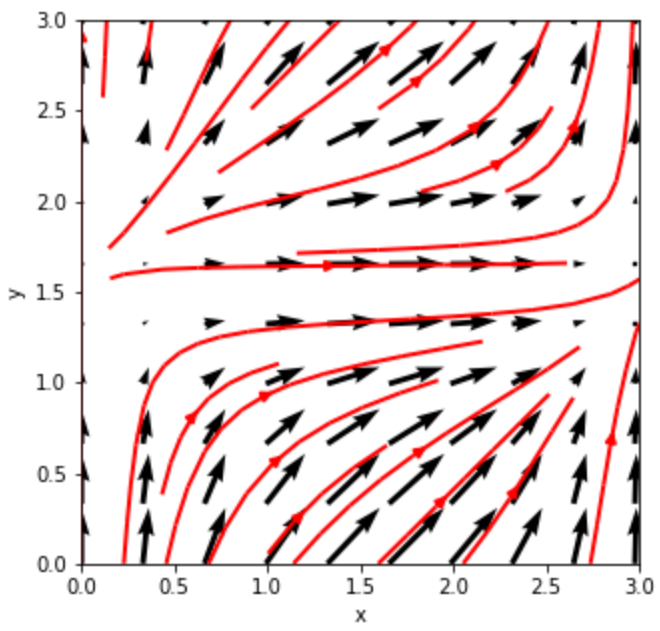
In [ ]:
```
plt.figure(figsize=(5,5))
plt.quiver(X, Y, U, V, scale=20,
           width=0.005)

plt.xlabel("x")
plt.ylabel("y")

plt.show()
```



Vector fields can also be expressed with *streamline plots*. Vector field is tangent to the streamlines:

{index}

In [ ]:
```
plt.figure(figsize=(5,5))

plt.quiver(X, Y, U, V, scale=10, width=0.01)

plt.streamplot(X, Y, U, V, color="red", linewidth=2)

plt.xlabel("x")
plt.ylabel("y")

plt.xlim(0,3)
plt.ylim(0,3)

plt.show()
```



Derivatives

Gradient of a scalar field

{index}

We can take derivatives of a scalar field in three directions x, y and z. Three derivatives then create a vector field known as *gradient*:

$$\nabla\Omega = \left(\frac{\partial\Omega}{\partial x}, \frac{\partial\Omega}{\partial y}, \frac{\partial\Omega}{\partial z}\right).$$

The ∇ symbol is known as "nabla" or "del" and it is defined as:

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right).$$

The vector $\nabla\Omega$ represents the rate of change of Ω in space.

Let's use SymPy to find gradient of a scalar field

$$\Omega = x^2 + xy + y^2.$$

In [ ]:
```
from sympy.vector import gradient, CoordSys3D

R = CoordSys3D(' ')

Omega = R.x**2+R.x*R.y+R.y**2

# Print omega
display(Omega)

# Print grad omega
display(gradient(Omega))
```

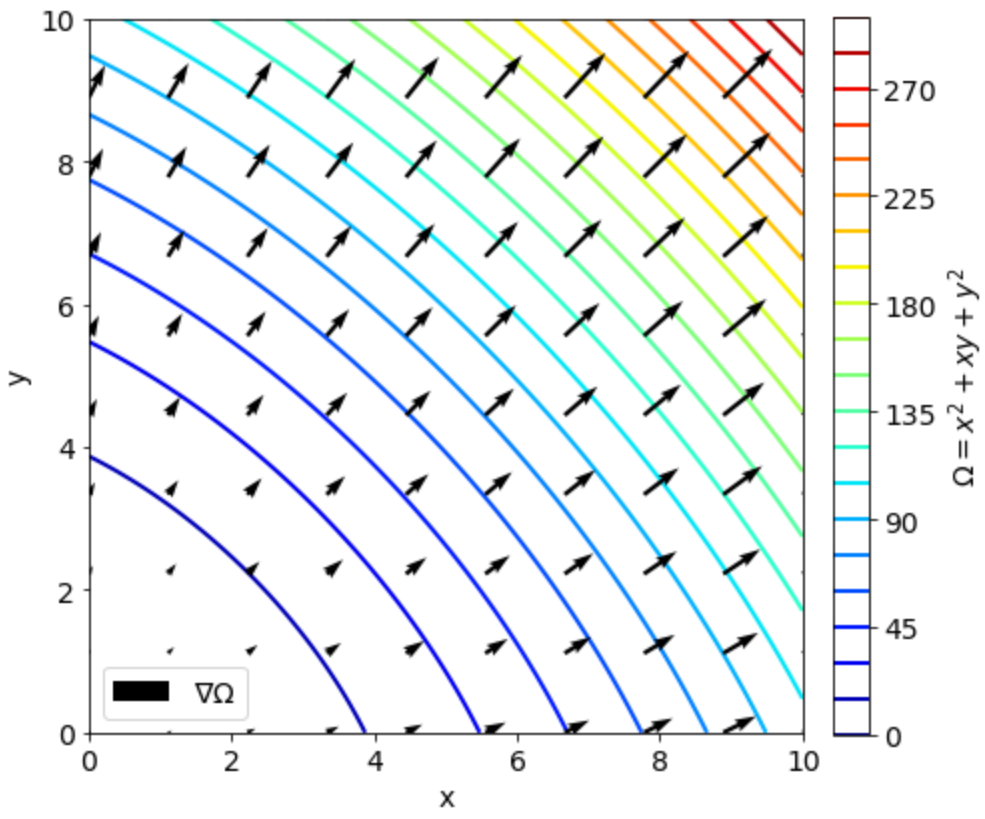
$$\mathbf{x}^2 + \mathbf{xy} + \mathbf{y}^2$$

$$(2\mathbf{x} + \mathbf{y})\,\hat{\mathbf{i}} + (\mathbf{x} + 2\mathbf{y})\hat{\mathbf{j}}$$

The gradient becomes a vector in $\hat{\mathbf{i}}$ (x) and $\hat{\mathbf{j}}$ (y) directions.

Let's create a contour plot of scalar field Ω and quiver plot of $\nabla\Omega$.

In [ ]:
```
plt.rcParams.update({'font.size': 14})
plt.figure(figsize=(7,7))

x = np.linspace(0,10,100)
y = np.linspace(0,10,100)

X, Y = np.meshgrid(x, y)

Omega = X**2+X*Y+Y**2

# Create contour plot of scalar field
plt.contour(X, Y, Omega, 20, cmap="jet",
            zorder=1, linewidths=2)

plt.colorbar(label=r"$\Omega=x^2+xy+y^2$",
             fraction=0.046, pad=0.04)

x = np.linspace(0,10,10)
y = np.linspace(0,10,10)

X, Y = np.meshgrid(x, y)

# Create quiver plot of the gradient
plt.quiver(X, Y, 2*X+Y, X+2*Y,
           label=r"$\nabla\Omega$", zorder=2)

plt.legend(loc="lower left")

plt.xlabel("x")
plt.ylabel("y")

plt.gca().set_aspect("equal")

plt.show()
```



Scalar field Ω will vary at different rate in different directions. The *directional derivative* of a unit vector $\hat{\mathbf{a}}$ is:

$$\frac{\partial\Omega}{\partial a} = \hat{\mathbf{a}} \cdot \nabla\Omega.$$

For example, we are given scalar field $\Omega = x^2yz + 4xz^2$ and we would like to know its derivative in the direction of $\mathbf{a} = (2, -1, -1)$ at point $P = (1, -2, -1)$.

First, we need to find $\hat{\mathbf{a}}$:

In [ ]:
```
a = np.array([2,-1,-1])

a_len = np.linalg.norm(a)

a_hat = a/a_len

print(a)
print(a_hat)
```

```
[ 2 -1 -1]
[ 0.81649658 -0.40824829 -0.40824829]
```

Through SymPy we can find the gradient of Ω:

{index}

In [ ]:
```
R = CoordSys3D(' ')

Omega = R.x**2*R.y*R.z+4*R.x*R.z**2

# Print omega
display(Omega)

# Print grad omega
display(gradient(Omega))
```

$$\mathbf{x}^2\mathbf{yz} + 4\mathbf{xz}^2$$

$$(2\mathbf{xyz} + 4\mathbf{z}^2)\hat{\mathbf{i}} + (\mathbf{x}^2\mathbf{z})\hat{\mathbf{j}} + (\mathbf{x}^2\mathbf{y} + 8\mathbf{xz})\hat{\mathbf{k}}$$

The directional derivative is a dot product between unit vector $\hat{\mathbf{a}}$ and $\nabla\Omega$:

$$\hat{\mathbf{a}} \cdot \nabla\Omega = 0.816 \times (2xyz + 4z^2) - 0.408 \times (x^2z) - 0.408 \times (x^2y + 8xz).$$

Now, we can evaluate it at point P:

In [ ]:
```
P = np.array([1,-2,1])

print(a_hat[0]*(2*P[0]*P[1]*P[2]+4*P[2]**2)+a_hat[1]*P[0]**2*P[2]+a_hat[2]*(P[0]**2*P[1]+8*P[0]*P[2]))
```

```
-2.8577380332470415
```

The divergence of a vector field

{index}

The *divergence* of a vector field $\mathbf{F} = (f_x, f_y, fz)$ is:

$$\text{div}\mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial f_x}{\partial x} + \frac{\partial f_y}{\partial y} + \frac{\partial z}{\partial z}.$$

For example, take divergence of a vector field $\mathbf{F} = (x^2, 3y, x^3)$:

{index}

In [ ]:
```
from sympy.vector import divergence
R = CoordSys3D(' ')
v1 = R.x**2*R.i+ 3*R.y*R.j+R.x**3*R.k
v1
```

$$(\mathbf{x}^2)\hat{\mathbf{i}} + (3\mathbf{y})\hat{\mathbf{j}} + (\mathbf{x}^3)\hat{\mathbf{k}}$$

In [ ]:
```
divergence(v1)
```

$$2\mathbf{x} + 3$$

If divergence is applied to the flow velocity of some fluid, the divergence represents the net amount of fluid entering or leaving a particular point. The divergence is zero for incompressible fluids. Positive divergence implies that density of the fluid decreases at that location, while negative divergence suggests it is increasing.

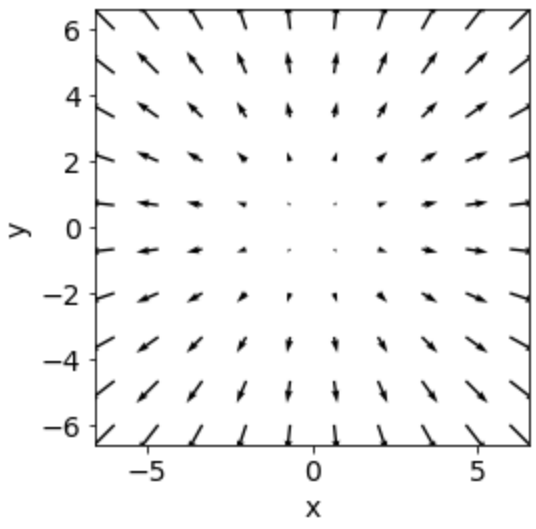
If we look at the field $\mathbf{F}(x, y) = (x, y)$, its divergence is:

In [ ]:
```
v1 = R.x*R.i+ R.y*R.j
divergence(v1)
```

$$2$$

If that field indicated flow velocity, the positive divergence would mean flow increasing outward, which can indeed be seen in a quiver plot:

In [ ]:
```
x = np.linspace(-6, 6, 10)
y = np.linspace(-6, 6, 10)

X, Y = np.meshgrid(x, y)

plt.quiver(X, Y, X, Y)
plt.gca().set_aspect("equal")

plt.xlabel("x")
plt.ylabel("y")

plt.show()
```



Curl of a vector field

```
{index}
```

Curl of a vector field is defined as:

$$\text{curl}\mathbf{F} = \nabla \times \mathbf{F} = \Big(\frac{\partial f_z}{\partial y} - \frac{\partial f_y}{\partial z}\Big)\hat{\mathbf{i}} + \Big(\frac{\partial f_x}{\partial z} - \frac{\partial f_z}{\partial x}\Big)\hat{\mathbf{j}} + \Big(\frac{\partial f_y}{\partial x} - \frac{\partial f_x}{\partial y}\Big)\hat{\mathbf{k}}.$$

The result is another vector field. For example, let's take divergence of a vector field $\mathbf{F} = (z, x, y)$:

$$\nabla \times \mathbf{F} = (1-0)\hat{\mathbf{i}} + (1-0)\hat{\mathbf{j}} + (1-0)\hat{\mathbf{k}} = \hat{\mathbf{i}} + \hat{\mathbf{j}} + \hat{\mathbf{k}}.$$

To check our answer we can use SymPy:

```
{index}
```

In [ ]:
```
from sympy.vector import curl
R = CoordSys3D(' ')
v1 = R.z*R.i+ R.x*R.j+R.y*R.k
curl(v1)
```

$$\hat{\mathbf{i}} + \hat{\mathbf{j}} + \hat{\mathbf{k}}$$

Curl of a vector field that is a gradient of a scalar field

If we take a gradient of a scalar field Ω and take a curl of that field, the identity states:

$$\nabla \times (\nabla \Omega) = 0.$$

Curl of a gradient field is always zero. We can check what SymPy returns:

In [ ]:
```
Omega = R.x**2*R.y*R.z+4*R.x*R.z**2
curl(gradient(Omega))
```

$$\hat{\mathbf{0}}$$

Laplacian

```
{index}
```

Laplacian is defined as:

$$\nabla^2 = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2}.$$

Laplacian can operate on both scalar and vector fields. For a vector field, it is defined as:

$$\nabla^2 \mathbf{F} = (\nabla^2 f_x, \nabla^2 f_y, \nabla^2 f_z).$$

If we take $\Omega = x^2yz + 4 * xz^2$, then Laplacian would be:

In [ ]:
```
divergence(gradient(Omega))
```

$$8\mathbf{x} + 2\mathbf{yz}$$

If we take $\mathbf{F} = x^2yz\hat{\mathbf{i}} + xyz^2\hat{\mathbf{j}} + (xy - 2y^2 + 2x^2z)\hat{\mathbf{k}}$, then we can calculate $\nabla^2 f_x, \nabla^2 f_y, \nabla^2 f_z$:

In [ ]:
```
F = R.x**2*R.y*R.z*R.i+R.x*R.y*R.z**2*R.j +(R.x*R.y-2*R.y**2+2*R.x**2*R.z)*R.k
F
```

$$(\mathbf{x^2yz})\hat{\mathbf{i}} + (\mathbf{xyz^2})\hat{\mathbf{j}} + (\mathbf{2x^2z + xy - 2y^2})\hat{\mathbf{k}}$$

The $\nabla^2 f_x$ is:

In [ ]:
```
divergence(gradient(R.x**2*R.y*R.z))
```

$$2\mathbf{yz}$$

The $\nabla^2 f_y$:

In [ ]:
```
divergence(gradient(R.x*R.y*R.z**2))
```

$$2\mathbf{xy}$$

The $\nabla^2 f_z$:

In [ ]:
```
divergence(gradient((R.x*R.y-2*R.y**2+2*R.x**2*R.z)))
```

$$4\mathbf{z} - 4$$