Vector calculus (MM3)

Contents

Vector functions

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.

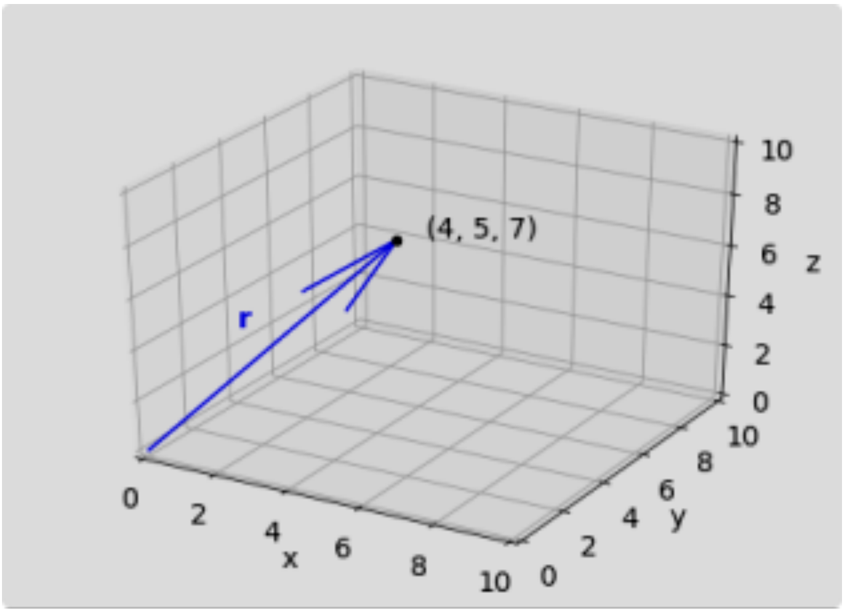
For example, if $f(t)$ represents a real function of t, then vector function of t would be:

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

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:

▸ **Show code cell source**

```
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:

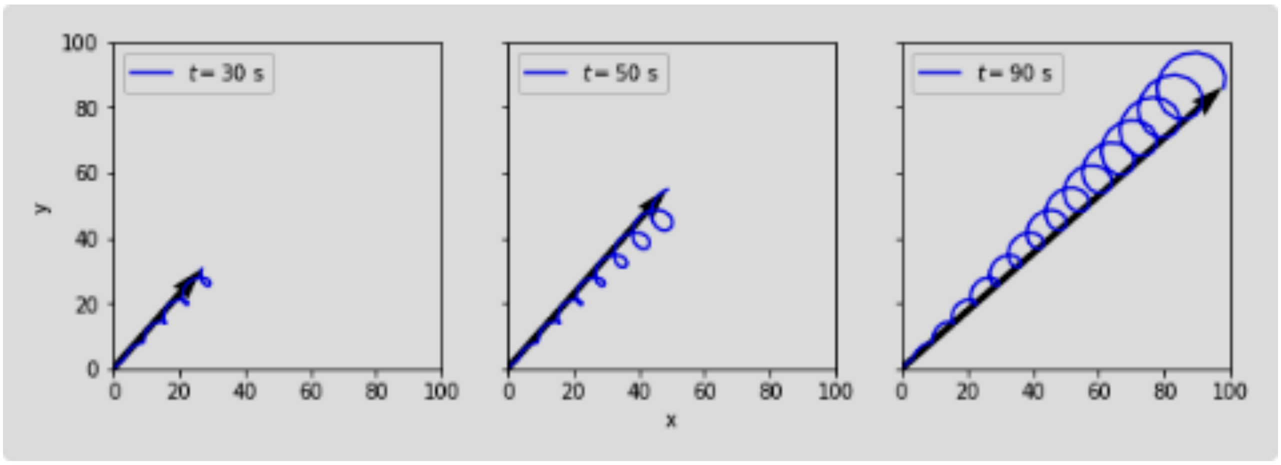
Describe the shape of the geometric representation of the vector function given by

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

```python
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)
```

▶ **Show code cell source**

```python
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

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:

```python
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.

```
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 =
```

$$-5\sin(t)$$

```
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)])
```

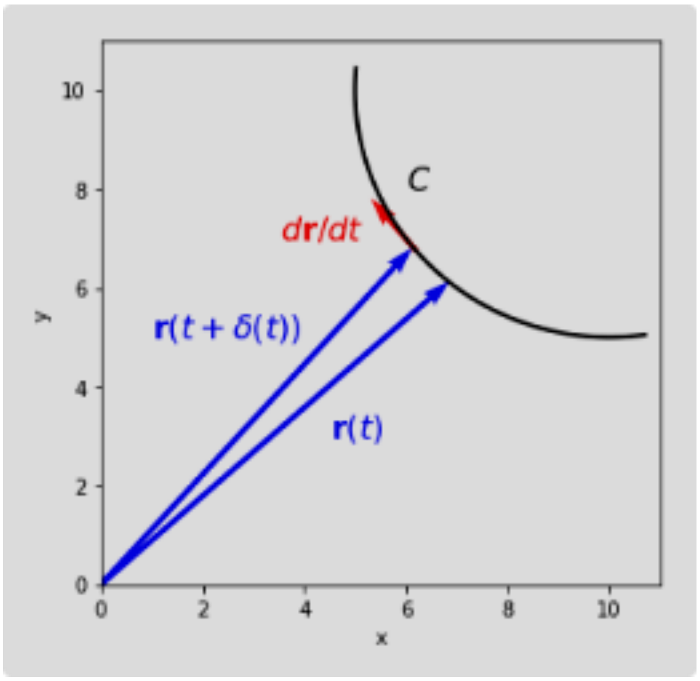
Unit tangent $\hat{\mathbf{t}}$ to the curve C can be defined as:

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

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

▸ **Show code cell source**

```
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()
```



Differentiation rules

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

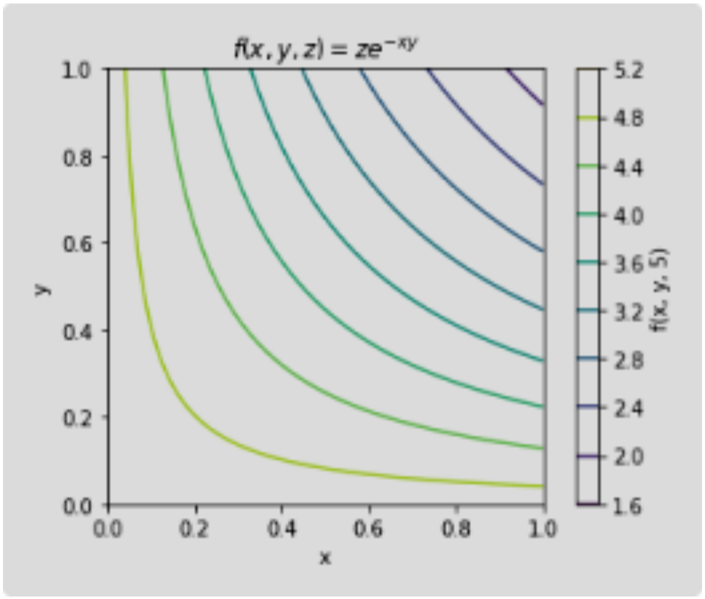
Scalar field is a scalar quantity that varies with position and can be written as $\Omega(x,y,z) = \Omega(\mathbf{r})$. Scalar fields are for example temperature and density. 3D scalar fields can be represented as iso-surfaces in 3D or contour plots in 2D for specific z value.

For example, let's plot a scalar field:

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

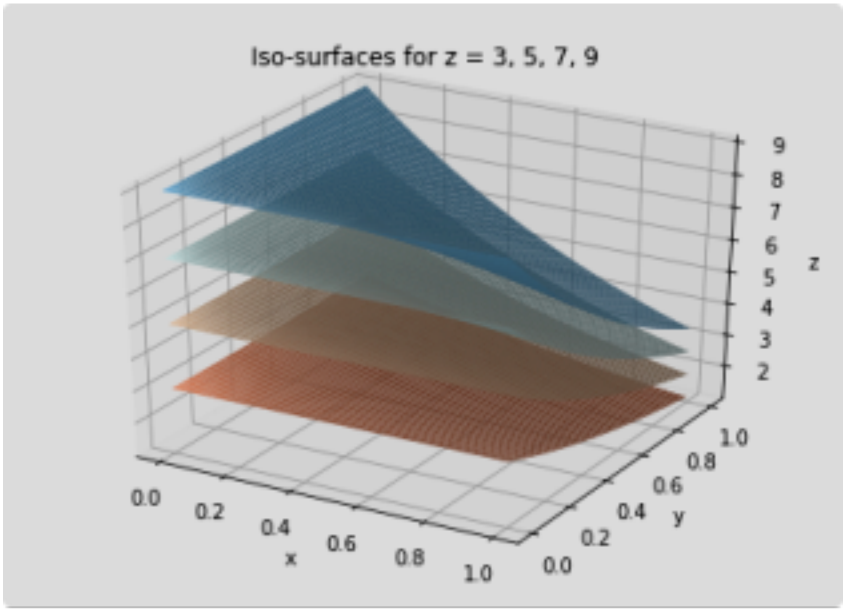
▶ **Show code cell source**

```
# 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()
```



▸ **Show code cell source**

```python
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()
```



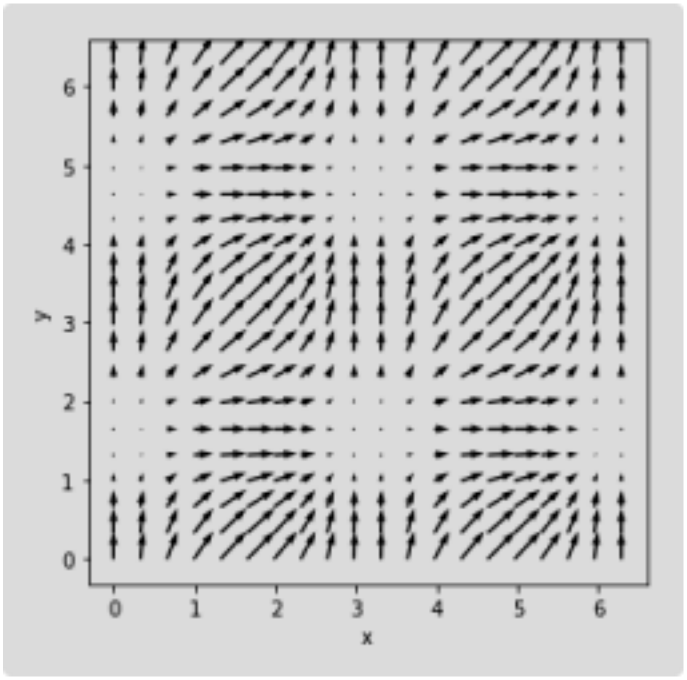
Vector field is a vector quantity that varies as a function of position and can be written as $\mathbf{F}(x, y, z) = \mathbf{F}(\mathbf{r})$. A vector field is for example ocean currents or wind field.

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:

```
# 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
```

▸ **Show code cell source**

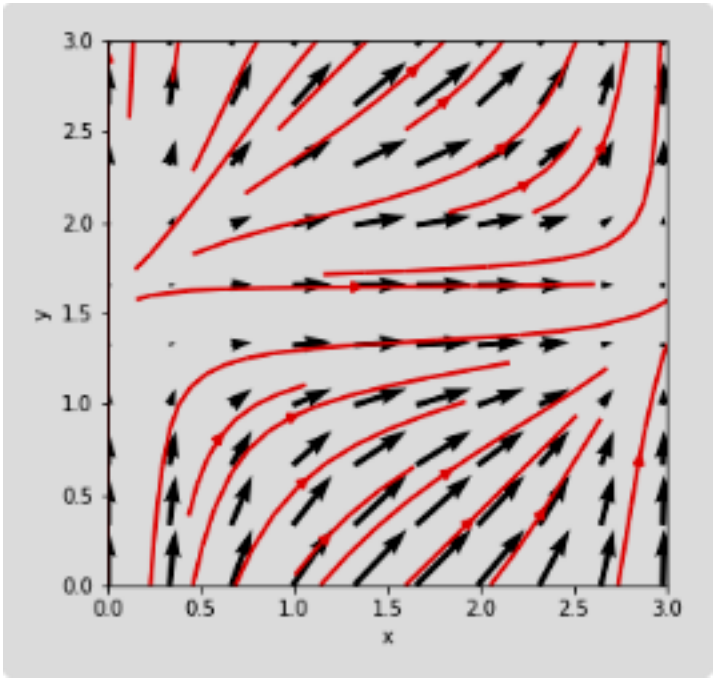
```
plt.figure(figsize=(5,5))
plt.quiver(X, Y, U, V, scale=20,
          width=0.005)

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

plt.show()
```



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

▸ **Show code cell source**

```
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

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

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

```
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))
```

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

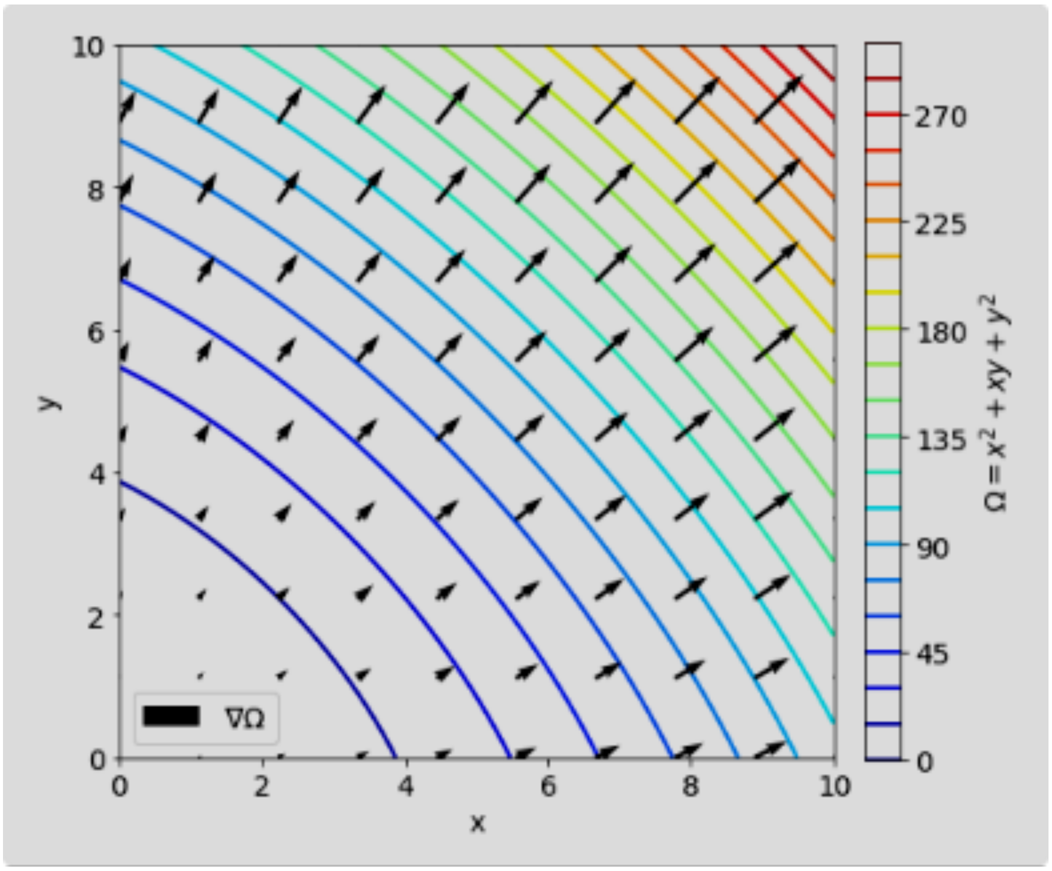
$$(2\mathbf{x} + \mathbf{y})\hat{\mathbf{i}} + (\mathbf{x} + 2\mathbf{y})\hat{\mathbf{j}}$$

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 $\nabla\Omega$.

▸ **Show code cell source**

```
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 $\hat{\mathbf{a}}$:

```
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:

```
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))
```

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

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

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:

```
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
```

```
-2.8577380332470415
```

The divergence of a vector field

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 $\mathbf{F} = (x^2, 3y, x^3)$:

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

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

```
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:

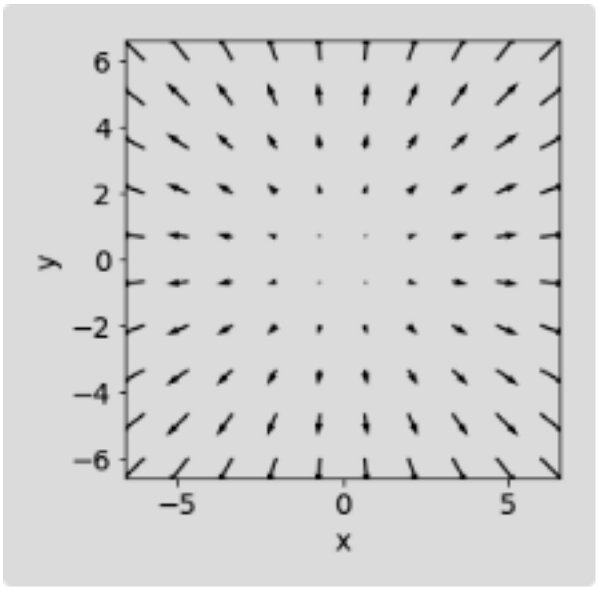
```
v1 = R.x*R.i+ R.y*R.j
divergence(v1)
```

$$2$$

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

▸ **Show code cell source**

```
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

Curl of a vector field is defined as:

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

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:

```
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:

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

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

Laplacian

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:

```
divergence(gradient(Omega))
```

$$8\mathbf{x} + 2\mathbf{y}\mathbf{z}$$

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 $\nabla^2 f_x, \nabla^2 f_y, \nabla^2 f_z$:

```
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
```

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

The $\nabla^2 f_x$ is:

```
divergence(gradient(R.x**2*R.y*R.z))
```

$$2\mathbf{y}\mathbf{z}$$

The $\nabla^2 f_y$:

```
divergence(gradient(R.x*R.y*R.z**2))
```

$$2\mathbf{x}\mathbf{y}$$

The $\nabla^2 f_z$:

```
divergence(gradient((R.x*R.y-2*R.y**2+2*R.x**2*R.z)))
```

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