2cm_blinear_algebra

November 17, 2026

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1 Bilinear Algebra and analytic geometry

1.1 Norm examples

1.1.1 Euclidian norm

The Euclidian norm you know since highschool, you used it to calculate the length of segment and vectors. For any vector $\mathbf{x} \in \mathbb{R}^N$ it is given by :

$$\|\mathbf{x}\|_2 := \sqrt{\sum_{i=1}^{N} x_i^2} = \sqrt{\mathbf{x}^t \mathbf{x}}$$

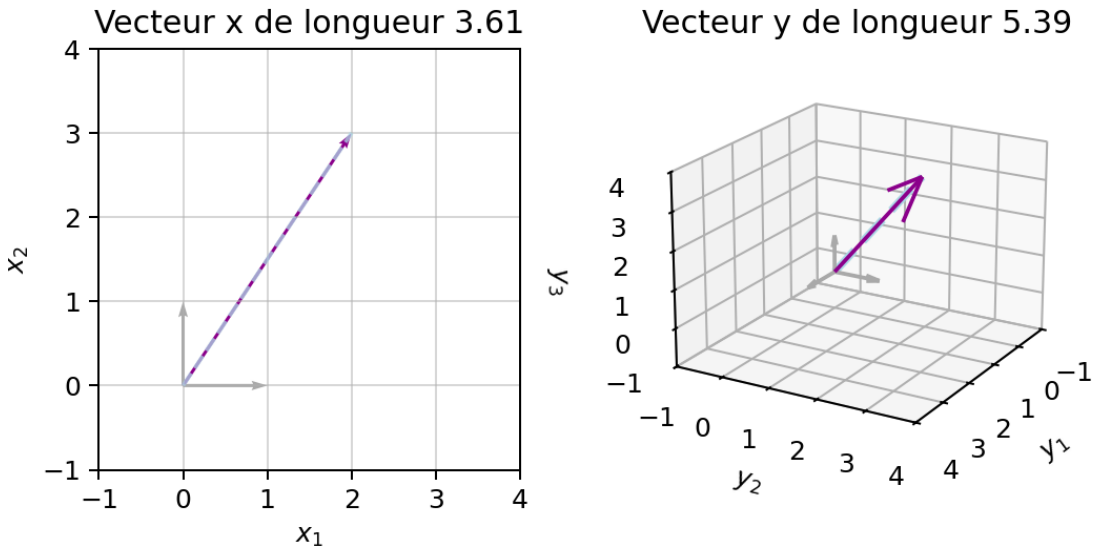
It is also calle the ℓ_2 norm.

[2]:
```
x = np.array([2,3])
y = np.array([2,3,4])
```

[3]:
```
# Plot vectors and their norm
f=plt.figure(dpi=180)
f.subplots_adjust(0.1,0.1,0.9,0.9,0.3,0.5)
ax2d=f.add_subplot(1,2,1,aspect="equal")
ax2d.grid("on", alpha=0.5)
ax2d.quiver([0.],[0.],[1.],[0.],color="darkgray",scale=1., scale_units="x")
ax2d.quiver([0.],[0.],[0.],[1.],color="darkgray",scale=1., scale_units="x")
ax2d.quiver([0.],[0.],[x[0]],[x[1]],color="darkmagenta",scale=1.,␣
 ↪scale_units="x")
ax2d.plot([0,x[0]],[0,x[1]],color="lightblue",ls="--",alpha=0.75)
ax2d.set_xlim([-1,4])
ax2d.set_ylim([-1,4])
ax2d.set_xlabel(r"${x_1}$")
ax2d.set_ylabel(r"${x_2}$")
ax2d.set_title("Vecteur x de longueur {}".format(np.round(np.sqrt(x.T@x),␣
 ↪decimals=2)))
ax3d=f.add_subplot(1,2,2, projection="3d")
```

```
ax3d.view_init(20, 30,0)
ax3d.quiver([0.],[0.],[0.],[1.],[0.],[0.],color="darkgray")
ax3d.quiver([0.],[0.],[0.],[0.],[1.],[0.],color="darkgray")
ax3d.quiver([0.],[0.],[0.],[0.],[0.],[1.],color="darkgray")
ax3d.quiver([0.],[0.],[0.],[y[0]],[y[1]],[y[2]],color="darkmagenta")
ax3d.plot([0,y[0]],[0,y[1]],[0,y[2]],color="lightblue",ls="--",lw=3)
ax3d.set_xlim([-1,4])
ax3d.set_ylim([-1,4])
ax3d.set_zlim([-1,4])
ax3d.set_xlabel(r"${y_1}$")
ax3d.set_ylabel(r"${y_2}$")
ax3d.set_zlabel(r"${y_3}$")
ax3d.set_title("Vecteur y de longueur {}".format(np.round(np.sqrt(y.T@y),␣
 ↪decimals=2)))
```

[3]: `Text(0.5, 0.92, 'Vecteur y de longueur 5.39')`



1.1.2 Manhattan Norm

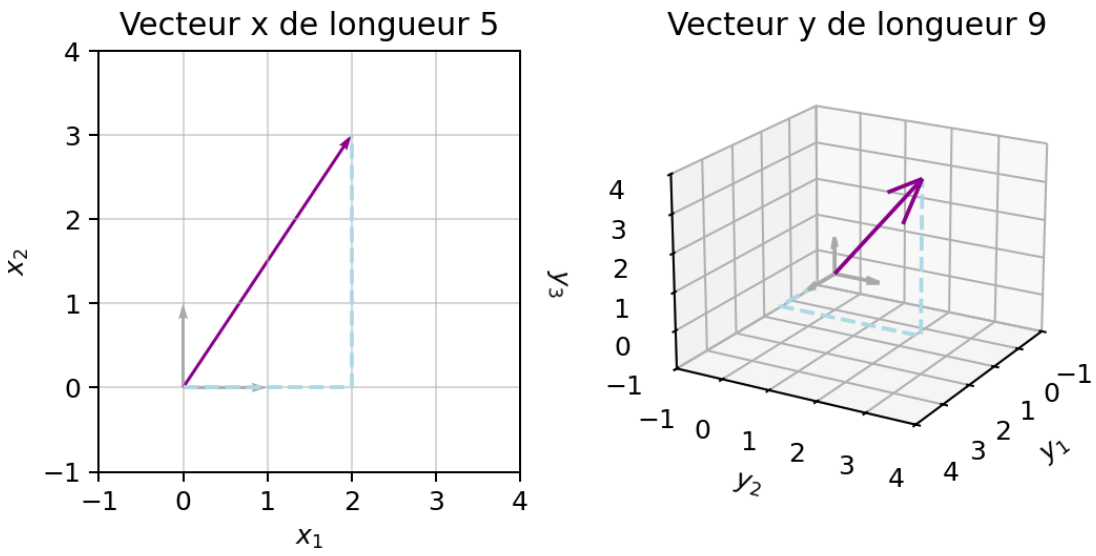
Also called 'taxicab' norm, it correspond to the length a of the path done following a grid of 1 unit, from the begining to the end. For any vector $\mathbf{x} \in \mathbb{R}^N$ it is given by :

$$\|\mathbf{x}\|_1 := \sum_{i=1}^{N} |x_i|$$

It is also calle the ℓ_1 norm.

[4]:
```
# Plot vectors and their norm
f=plt.figure(dpi=180)
f.subplots_adjust(0.1,0.1,0.9,0.9,0.3,0.5)
ax2d=f.add_subplot(1,2,1,aspect="equal")
ax2d.grid("on", alpha=0.5)
ax2d.quiver([0.],[0.],[1.],[0.],color="darkgray",scale=1., scale_units="x")
ax2d.quiver([0.],[0.],[0.],[1.],color="darkgray",scale=1., scale_units="x")
ax2d.quiver([0.],[0.],[x[0]],[x[1]],color="darkmagenta",scale=1.,␣
 ↪scale_units="x")
ax2d.plot([0,x[0],x[0]],[0,0,x[1]],color="lightblue",ls="--")
ax2d.set_xlim([-1,4])
ax2d.set_ylim([-1,4])
ax2d.set_xlabel(r"${x_1}$")
ax2d.set_ylabel(r"${x_2}$")
ax2d.set_title("Vecteur x de longueur {}".format(np.sum(np.abs(x))))
ax3d=f.add_subplot(1,2,2, projection="3d")
ax3d.view_init(20, 30,0)
ax3d.quiver([0.],[0.],[0.],[1.],[0.],[0.],color="darkgray")
ax3d.quiver([0.],[0.],[0.],[0.],[1.],[0.],color="darkgray")
ax3d.quiver([0.],[0.],[0.],[0.],[0.],[1.],color="darkgray")
ax3d.quiver([0.],[0.],[0.],[y[0]],[y[1]],[y[2]],color="darkmagenta")
ax3d.set_xlim([-1,4])
ax3d.set_ylim([-1,4])
ax3d.set_zlim([-1,4])
ax3d.set_xlabel(r"${y_1}$")
ax3d.set_ylabel(r"${y_2}$")
ax3d.set_zlabel(r"${y_3}$")
ax3d.set_title("Vecteur y de longueur {}".format(np.sum(np.abs(y))))
ax3d.
 ↪plot([0,y[0],y[0],y[0]],[0,0,y[1],y[1]],[0,0,0,y[2]],color="lightblue",ls="--")
```

[4]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x7bbce24ab340>]
```



1.1.3 Chebyshev norm

Also called 'chessboard' norm, it correspond to the maximal distance to the origin on a square grid. For any vector $\mathbf{x} \in \mathbb{R}^N$ it is given by :

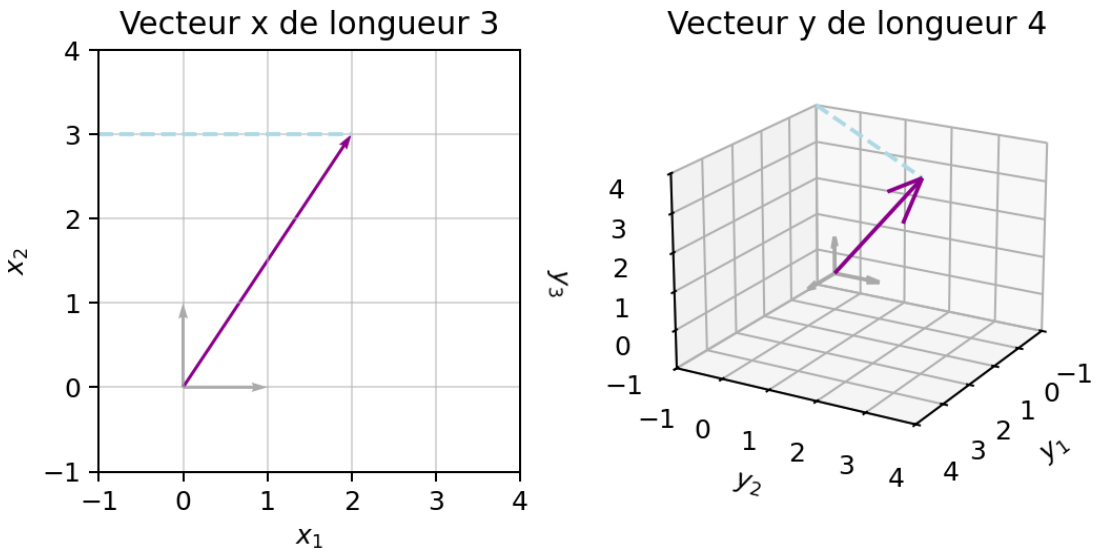
$$\|\mathbf{x}\|_\infty := \max(|x_1|, |x_2|, \dots, |x_N|)$$

It is also calle the ℓ_∞ norm. It is also the limit of the ℓ_p norm.

[6]:
```
# Plot vectors and their norm
f=plt.figure(dpi=180)
f.subplots_adjust(0.1,0.1,0.9,0.9,0.3,0.5)
ax2d=f.add_subplot(1,2,1,aspect="equal")
ax2d.grid("on", alpha=0.5)
ax2d.quiver([0.],[0.],[1.],[0.],color="darkgray",scale=1., scale_units="x")
ax2d.quiver([0.],[0.],[0.],[1.],color="darkgray",scale=1., scale_units="x")
ax2d.quiver([0.],[0.],[x[0]],[x[1]],color="darkmagenta",scale=1.,␣
 ↪scale_units="x")
ax2d.plot([-1,x[0]],[x[1],x[1]],color="lightblue",ls="--")
ax2d.set_xlim([-1,4])
ax2d.set_ylim([-1,4])
ax2d.set_xlabel(r"${x_1}$")
ax2d.set_ylabel(r"${x_2}$")
ax2d.set_title("Vecteur x de longueur {}".format(np.max(np.abs(x))))
ax3d=f.add_subplot(1,2,2, projection="3d")
ax3d.view_init(20, 30,0)
ax3d.quiver([0.],[0.],[0.],[1.],[0.],[0.],color="darkgray")
ax3d.quiver([0.],[0.],[0.],[0.],[1.],[0.],color="darkgray")
```

```
ax3d.quiver([0.],[0.],[0.],[0.],[0.],[1.],color="darkgray")
ax3d.quiver([0.],[0.],[0.],[y[0]],[y[1]],[y[2]],color="darkmagenta")
ax3d.set_xlim([-1,4])
ax3d.set_ylim([-1,4])
ax3d.set_zlim([-1,4])
ax3d.set_xlabel(r"${y_1}$")
ax3d.set_ylabel(r"${y_2}$")
ax3d.set_zlabel(r"${y_3}$")
ax3d.set_title("Vecteur y de longueur {}".format(np.max(np.abs(y))))
ax3d.plot([-1,y[0]],[-1,y[1]],[y[2],y[2]],color="lightblue",ls="--")
```

[6]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x7bbce29fe4c0>]
```



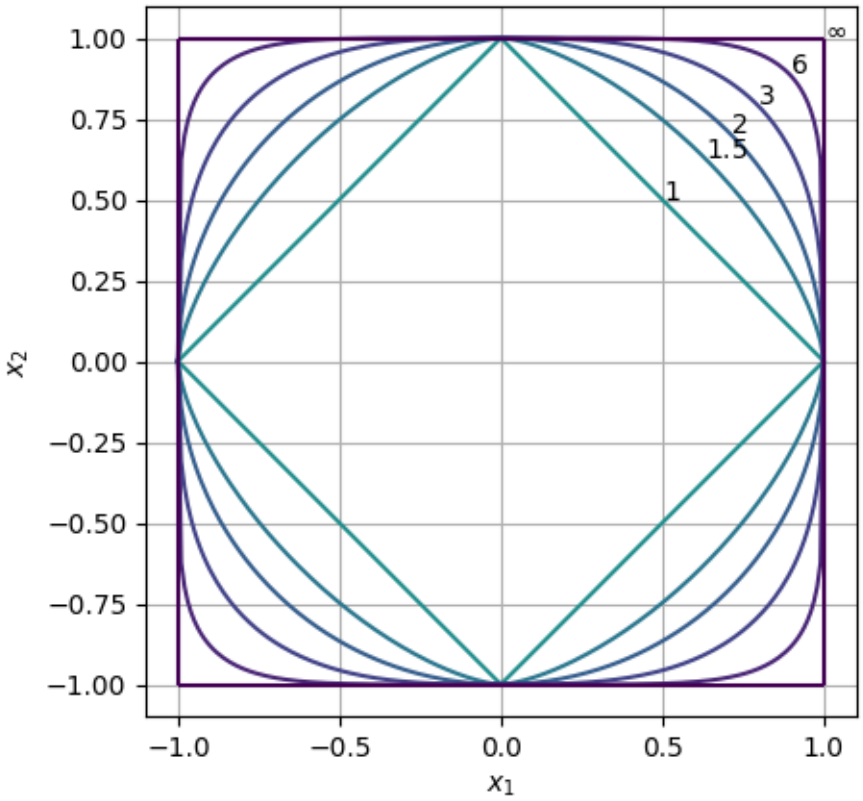
1.1.4 ℓ_p norms (wooclap)

For $p \geq 1$ and any vector $\mathbf{x} \in \mathbb{R}^N$, is given by :

$$\|\mathbf{x}\|_p := \sqrt[p]{\sum_{i=1}^{N} |x_i|^p}$$

[6]:
```
# Plot the different p-norms
colmap=plt.get_cmap("viridis")
xx=np.linspace(-1,1,200)
f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
ax.set_xlabel("$x_1$")
ax.set_ylabel("$x_2$")
```

```
#l1 norm
ax.plot(xx,1-np.abs(xx), color=colmap(125))
ax.plot(xx,np.abs(xx)-1, color=colmap(125))
ax.text(0.5, 0.5, "$1$")

#l1.5 norm
ax.plot(xx,(1-np.abs(xx)**1.5)**(1/1.5), color=colmap(100))
ax.plot(xx,-(1-np.abs(xx)**1.5)**(1/1.5), color=colmap(100))
ax.text(0.5**(1/1.5), 0.5**(1/1.5), "$1.5$")

#l2 norm
ax.plot(xx,np.sqrt(1-xx**2), color=colmap(75))
ax.plot(xx,-np.sqrt(1-xx**2), color=colmap(75))
ax.text(0.5**0.5, 0.5**0.5, "$2$")

#l3 norm
ax.plot(xx,(1-np.abs(xx)**3)**(1/3), color=colmap(50))
ax.plot(xx,-(1-np.abs(xx)**3)**(1/3), color=colmap(50))
ax.text(0.5**(1/3), 0.5**(1/3), "$3$")

#l6 norm
ax.plot(xx,(1-np.abs(xx)**6)**(1/6), color=colmap(25))
ax.plot(xx,-(1-np.abs(xx)**6)**(1/6), color=colmap(25))
ax.text(0.5**(1/6), 0.5**(1/6), "$6$")

#infinit norm
ax.vlines([-1,1], ymin=-1, ymax=1, colors=colmap(0))
ax.hlines([-1,1], xmin=-1, xmax=1, colors=colmap(0))
ax.text(1, 1, "$\infty$")

ax.grid("on")
```



And what about $0 < p < 1$? These functions are does not respect the triangular inequality, thus they are not considered as norms but **quasi-norms**.

1.2 Inner prouct

1.2.1 Angles

Let $\mathbf{x}, \mathbf{y} \in V$ be two vectors, than the angle ϕ between both vectors is such that:

$$\cos(\phi) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}.$$

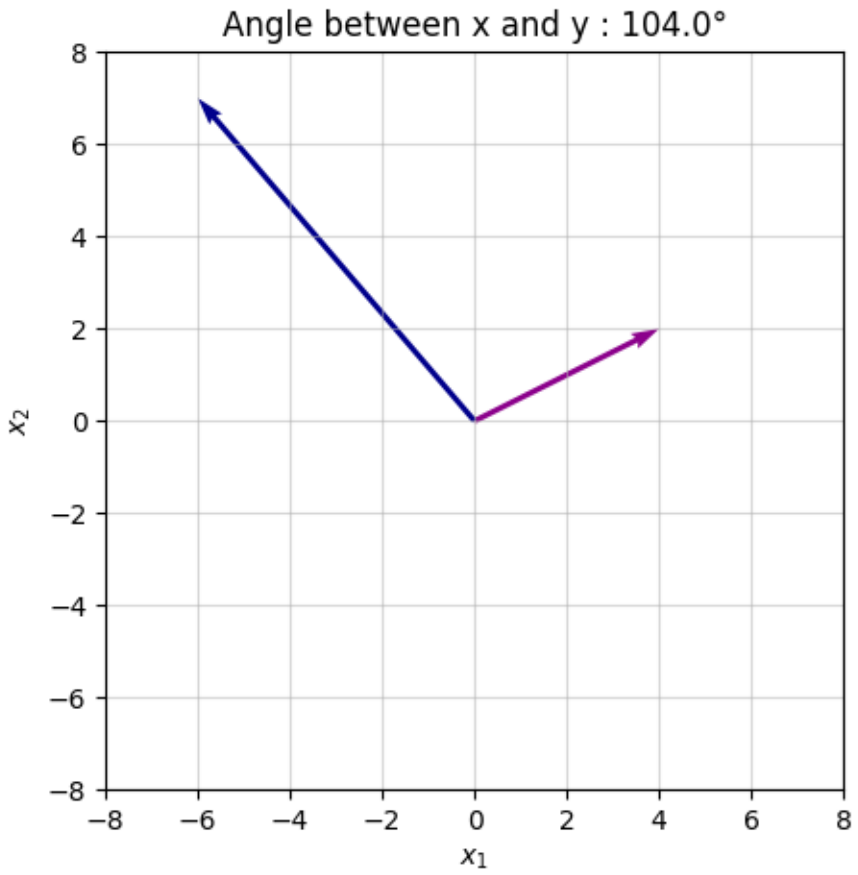
Let $\mathbf{x} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} -6 \\ 7 \end{pmatrix}$.

[7]:
```
x=np.array([4,2])
y=np.array([-6,7])
phi = np.arccos(x.T@y/(np.sqrt(x.T@x)*np.sqrt(y.T@y))) # angle in radian
```

[8]:
```
# Plot vectors and angle
f=plt.figure(dpi=100)
f.subplots_adjust(0.1,0.1,0.9,0.9,0.3,0.5)
```

```
ax2d=f.add_subplot(1,1,1,aspect="equal")
ax2d.grid("on", alpha=0.5)
ax2d.quiver([0.],[0.],[x[0]],[x[1]],color="darkmagenta",scale=1.,␣
 ↪scale_units="x")
ax2d.quiver([0.],[0.],[y[0]],[y[1]],color="darkblue",scale=1., scale_units="x")

ax2d.set_xlim([-8,8])
ax2d.set_ylim([-8,8])
ax2d.set_xlabel(r"${x_1}$")
ax2d.set_ylabel(r"${x_2}$")
ax2d.set_title("Angle between x and y : {}°".format(np.round(phi*180/np.
 ↪pi,decimals=1)))
```

[8]: `Text(0.5, 1.0, 'Angle between x and y : 104.0°')`



Previous angle was calculated with the dot product. Now, what happen if we change the inner product ? Let's set the matrix:

$$\mathbf{M} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

this matrix is symmetric and positive definit, thus we can defind the inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\mathbf{t}\mathbf{M}\mathbf{y}$.

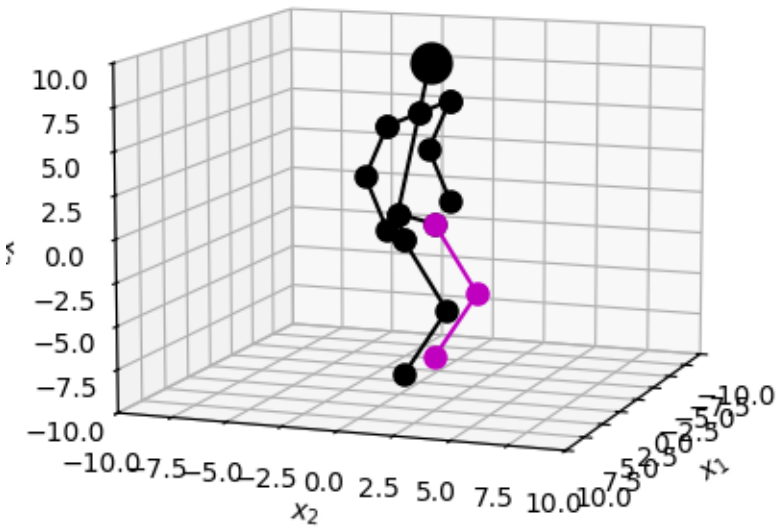
[9]:
```
M=np.array([[2,1],[1,1]])
phi = np.arccos(x.T@M@y/(np.sqrt(x.T@M@x)*np.sqrt(y.T@M@y)))
print("Angle using the new inner prouct: {}".format(phi*180/np.pi))
```

```
Angle using the new inner prouct: 114.22774531795419
```

This works also in 3D or any dimension. Let's do it

[10]:
```
#Just the teacher having fun drawing a stick person
f=plt.figure()
ax3d=f.add_subplot(1,1,1, projection="3d")
ax3d.view_init(10, 20,0)
ax3d.set_xlim([-10,10])
ax3d.set_ylim([-10,10])
ax3d.set_zlim([-10,10])
ax3d.set_xlabel("$x_1$")
ax3d.set_ylabel("$x_2$")
ax3d.set_zlabel("$x_3$")
ax3d.plot([-2,0,2.],[0,-1,0],[-1,0.,-1], "ko-", ms=8)
ax3d.plot([-2,-2,-2.],[0,2,0],[-1,-5.,-9], "mo-", ms=8)
ax3d.plot([2,2,2.],[0,2,0],[-1,-5.,-9], "ko-", ms=8)
ax3d.plot([0.,0.],[-1.,0],[0.,6], "ko-", ms=8)
ax3d.plot([-4.,4.],[0.,0],[6.,6], "ko-", ms=8)
ax3d.plot([-4,-4,-4.],[0,-1,0],[6,3.,0], "ko-", ms=8)
ax3d.plot([4,4,4.],[0,-1,0],[6,3.,0], "ko-", ms=8)
ax3d.plot([0.,0.],[0.,0.5],[6.,9], "ko-", ms=8)
ax3d.plot([0.],[0.5],[9], "ko-", ms=15)
```

[10]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x76ce5be427c0>]
```



The left hip is at the position $(-2, 0, -1)$, the left knee at the position $(-2, 2, -5)$ and the left foot at the position $(-2, 0, -9)$. Let $\mathbf{u}$ be the vector going from the knee to the hip and $\mathbf{v}$ the vector from the knee to the foot, then we have:

$$\mathbf{u} = \begin{pmatrix} -2-(-2) \\ 0-2 \\ -1-(-5) \end{pmatrix} = \begin{pmatrix} 0 \\ -2 \\ 4 \end{pmatrix} \quad \text{and} \quad \mathbf{v} = \begin{pmatrix} -2-(-2) \\ 0-2 \\ -9-(-5) \end{pmatrix} = \begin{pmatrix} 0 \\ -2 \\ -4 \end{pmatrix}.$$

[14]:
```
u = np.array([0, -2, 4])
v = np.array([0, -2, -4])
psi = np.arccos(u.T@v/(np.sqrt(u.T@u)*np.sqrt(v.T@v)))
print("The angle between the left thigh and left calf: {}".format(psi*180/np.
  ↪pi))
```

```
The angle between the left thigh and left calf: 126.86989764584402
```

It is used in many applications as 3d rendering, optimization, remonte controle of robots...

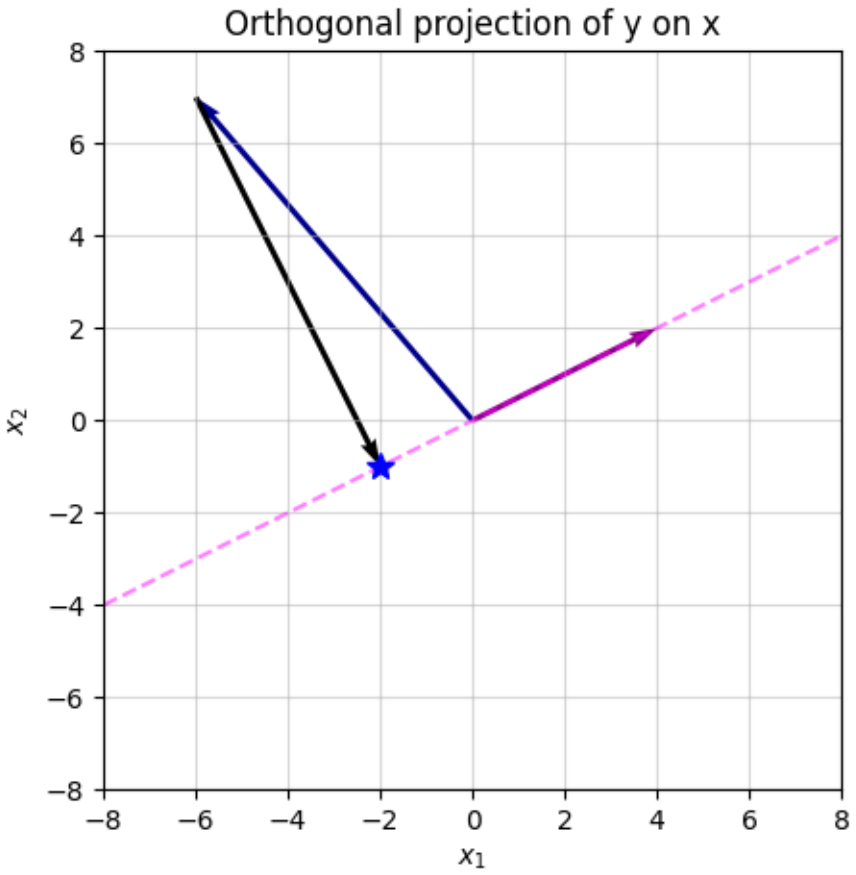
1.2.2 Orthogonal projection

The orthogonal projection of a point on a line or any subspace, is the point of the subspace the closest to the given point.

[12]:
```
# Orthogonal projection figure
p = x.reshape(2,1)@x.reshape(1,2)/(x.T@x)@y
```

```
f=plt.figure(dpi=100)
f.subplots_adjust(0.1,0.1,0.9,0.9,0.3,0.5)
ax2d=f.add_subplot(1,1,1,aspect="equal")
ax2d.grid("on", alpha=0.5)
ax2d.plot([-8,8],[-4,4],color="magenta",ls="dashed",alpha=0.5)
ax2d.quiver([0.],[0.],[x[0]],[x[1]],color="darkmagenta",scale=1.,␣
 ↪scale_units="x")
ax2d.quiver([0.],[0.],[y[0]],[y[1]],color="darkblue",scale=1., scale_units="x")
ax2d.
 ↪quiver([y[0]],[y[1]],[-(y-p)[0]],[-(y-p)[1]],color="black",ls="dashed",scale=1.
 ↪, scale_units="x")
ax2d.plot([p[0]],[p[1]],"b*",ms=10)
ax2d.set_xlim([-8,8])
ax2d.set_ylim([-8,8])
ax2d.set_xlabel(r"${x_1}$")
ax2d.set_ylabel(r"${x_2}$")
ax2d.set_title("Orthogonal projection of y on x")
```

[12]: Text(0.5, 1.0, 'Orthogonal projection of y on x')



Let $\mathbf{m_1}, \dots, \mathbf{m_N}$ describing the subspace $U \in V$ (a vectore is sufficient to describe a line, two vectors for a plane, ...) and $\mathbf{M} = \begin{pmatrix} \mathrm{m}_1 & \dots & \mathrm{m}_\mathrm{N} \end{pmatrix}$ where the vectors $\mathbf{m_i}$ are the column of the matrix $\mathbf{M}$, then the projection of a vector $\mathbf{x}$ on the subspace is given by:

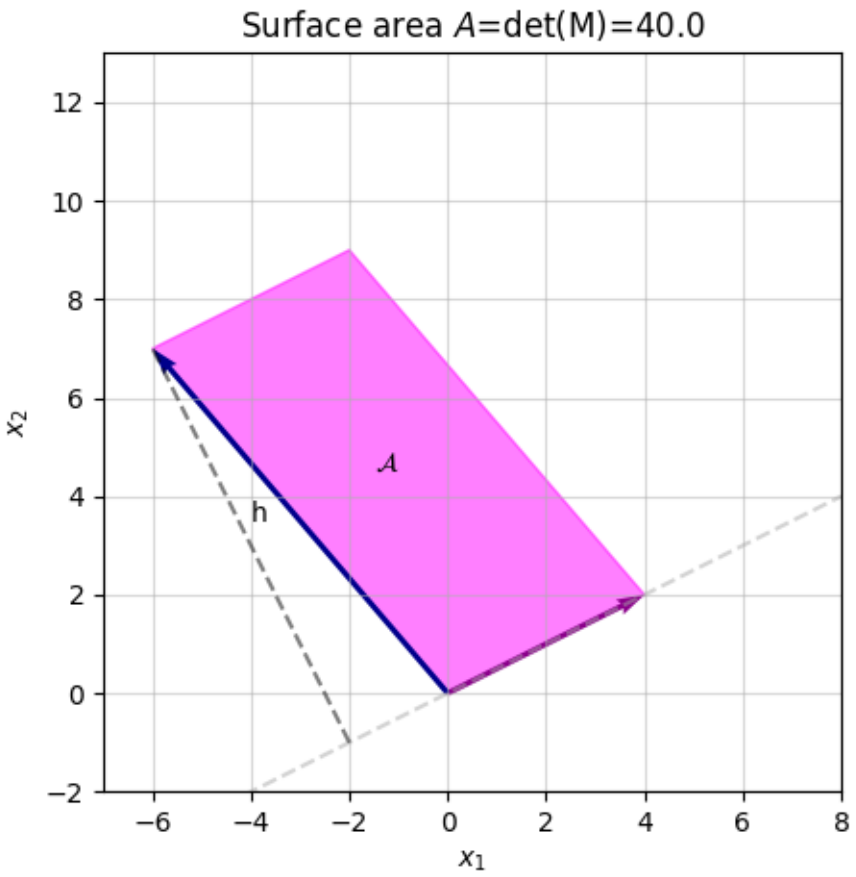
$$\Pi_U(\mathbf{x}) = \mathbf{M}(\mathbf{M^t M})^{-1}\mathbf{M^t x}.$$

1.2.3 Surfaces/Volumes

[ ]:
```
M=np.array([x,y]).T
A=np.linalg.det(M)
```

[88]:
```
#Plot of the area and vectors
f=plt.figure(dpi=100)
f.subplots_adjust(0.1,0.1,0.9,0.9,0.3,0.5)
ax2d=f.add_subplot(1,1,1,aspect="equal")
ax2d.grid("on", alpha=0.5)
ax2d.plot([-8,8],[-4,4],color="darkgray",ls="dashed",alpha=0.5)
ax2d.text(-4,3.5,"h")
ax2d.plot([p[0],y[0]],[p[1],y[1]],color="black",ls="dashed",alpha=0.5)
ax2d.fill([0.,x[0],(x+y)[0],y[0]],[0.,x[1],(x+y)[1],y[1]],color="magenta",␣
 ↪alpha=0.5)
ax2d.text(-1.5,4.5,"$\mathcal{A}$")
ax2d.quiver([0.],[0.],[x[0]],[x[1]],color="darkmagenta",scale=1.,␣
 ↪scale_units="x")
ax2d.quiver([0.],[0.],[y[0]],[y[1]],color="darkblue",scale=1., scale_units="x")
ax2d.set_xlim([-7,8])
ax2d.set_ylim([-2,13])
ax2d.set_xlabel(r"${x_1}$")
ax2d.set_ylabel(r"${x_2}$")
ax2d.set_title("Surface area $A$=det(M)={}".format(np.round(A)))
```

[88]:
```
Text(0.5, 1.0, 'Surface area $A$=det(M)=40.0')
```



[91]:
```
h=p-y
A2=np.sqrt(h.T@h)*np.sqrt(x.T@x)
print("Area computed using the length of the hight and basis of the␣
  ↪parallelogram A=||h||||x||={}".format(np.round(A2)))
```

```
Area computed using the length of the hight and basis of the parallelogram
A=||h||||x||=40.0
```

We have the same value. The same goes in 3 dimensions.