1cm_vector_matrix_isometricaltransforms

November 17, 2026

1 LESSON 1 NOTEBOOK

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

1.1 Vectors and matrices

1.1.1 Using Numpy

[2]:
```
x=np.array([0, 1]) #Create a vector
print(x)
```

```
[0 1]
```

[3]:
```
print("First element of the vector:", x[0]) # 0 is the index of the first␣
 ↪element
print("Second element of the vector:", x[1]) # 1 is the index of the second␣
 ↪element
```

```
First element of the vector: 0
Second element of the vector: 1
```

[4]:
```
A=np.array([[1, 2],[3,4]]) #Create a matrix
print(A)
```

```
[[1 2]
 [3 4]]
```

[5]:
```
print("First row, second column :",A[0,1]) #A[i,j] to access the i-th element␣
 ↪of the j-th column
```

```
First row, second column : 2
```

[6]:
```
y=A@x
print(y) #Even if it is printed as a row vector, y is still a column vector!
```

```
[2 4]
```

Carefull ! The matrix product is `@` not `*` :

[7]:
```
A*x
```

[7]:
```
array([[0, 2],
       [0, 4]])
```

There is a lot of way to do the same operation, e.g. `@` can be done with the method `np.dot`. The `Numpy` documentation will be your best friend to learn all the whereabout. The exercices on Moodle are here for you to train. Some exercices must be done prior to the first practical work.

[8]:
```
u = np.array([3,-5])
v = np.array([2,3])
w= -3*u+5*v
print(w)
A = np.array([u,v]).T #transpose is mandatory as we want u and v as the column␣
 ↪of the matrix and Numpy default write them as rows.
x=np.array([-3,5])
print(A)
print(A@x)
```

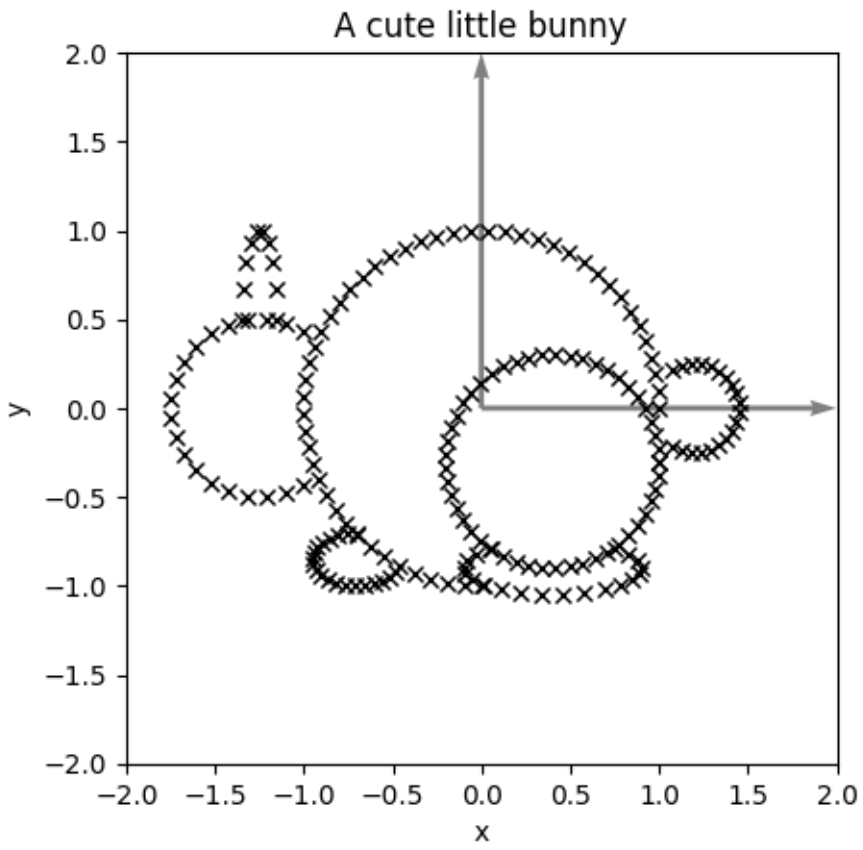
```
[ 1 30]
[[ 3  2]
 [-5  3]]
[ 1 30]
```

1.2 Isometries

Isometries are geometrical transformations that change neither the lengths nor the angles of an object. In the following we consider that the center of our object is at the coordinate $(0, 0)$.

[9]:
```
cloud2d= np.loadtxt("cloud2D.txt", dtype=float)
f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx")
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim([-2,2])
plt.ylim([-2,2])
```

[9]:
```
(-2.0, 2.0)
```



We want to apply some isometrical transformations on this bunny. To do so, we have to apply the transformation matrix to each point in the cloud. Hence, we have to be carefull of the dimension of the array.

[10]:
```
cloud2d.shape
```

[10]:
```
(190, 2)
```

The values returned are (number rows, number of columns). Here we have 190 points in 2 dimensions.

[11]:
```
cloud2d
```

[11]:
```
array([[ 1.00000000e+00,  0.00000000e+00],
       [ 9.95379113e-01,  9.60230259e-02],
       [ 9.81559157e-01,  1.91158629e-01],
       [ 9.58667853e-01,  2.84527587e-01],
       [ 9.26916757e-01,  3.75267005e-01],
       [ 8.86599306e-01,  4.62538290e-01],
       [ 8.38088105e-01,  5.45534901e-01],
       [ 7.81831482e-01,  6.23489802e-01],
```

```
[ 7.18349350e-01,  6.95682551e-01],
[ 6.48228395e-01,  7.61445958e-01],
[ 5.72116660e-01,  8.20172255e-01],
[ 4.90717552e-01,  8.71318704e-01],
[ 4.04783343e-01,  9.14412623e-01],
[ 3.15108218e-01,  9.49055747e-01],
[ 2.22520934e-01,  9.74927912e-01],
[ 1.27877162e-01,  9.91790014e-01],
[ 3.20515776e-02,  9.99486216e-01],
[-6.40702200e-02,  9.97945393e-01],
[-1.59599895e-01,  9.87181783e-01],
[-2.53654584e-01,  9.67294863e-01],
[-3.45365054e-01,  9.38468422e-01],
[-4.33883739e-01,  9.00968868e-01],
[-5.18392568e-01,  8.55142763e-01],
[-5.98110530e-01,  8.01413622e-01],
[-6.72300890e-01,  7.40277997e-01],
[-7.40277997e-01,  6.72300890e-01],
[-8.01413622e-01,  5.98110530e-01],
[-8.55142763e-01,  5.18392568e-01],
[-9.00968868e-01,  4.33883739e-01],
[-9.38468422e-01,  3.45365054e-01],
[-9.67294863e-01,  2.53654584e-01],
[-9.87181783e-01,  1.59599895e-01],
[-9.97945393e-01,  6.40702200e-02],
[-9.99486216e-01, -3.20515776e-02],
[-9.91790014e-01, -1.27877162e-01],
[-9.74927912e-01, -2.22520934e-01],
[-9.49055747e-01, -3.15108218e-01],
[-9.14412623e-01, -4.04783343e-01],
[-8.71318704e-01, -4.90717552e-01],
[-8.20172255e-01, -5.72116660e-01],
[-7.61445958e-01, -6.48228395e-01],
[-6.95682551e-01, -7.18349350e-01],
[-6.23489802e-01, -7.81831482e-01],
[-5.45534901e-01, -8.38088105e-01],
[-4.62538290e-01, -8.86599306e-01],
[-3.75267005e-01, -9.26916757e-01],
[-2.84527587e-01, -9.58667853e-01],
[-1.91158629e-01, -9.81559157e-01],
[-9.60230259e-02, -9.95379113e-01],
[-1.83697020e-16, -1.00000000e+00],
[-1.00000000e+00,  4.33012702e-01],
[-1.10074259e+00,  4.77202501e-01],
[-1.20871033e+00,  4.98292247e-01],
[-1.31867678e+00,  4.95261042e-01],
[-1.42531878e+00,  4.68255621e-01],
```

```
[-1.52347408e+00,  4.18583239e-01],
[-1.60839126e+00,  3.48648401e-01],
[-1.67595970e+00,  2.61836457e-01],
[-1.72290862e+00,  1.62349735e-01],
[-1.74696534e+00,  5.50041105e-02],
[-1.74696534e+00, -5.50041105e-02],
[-1.72290862e+00, -1.62349735e-01],
[-1.67595970e+00, -2.61836457e-01],
[-1.60839126e+00, -3.48648401e-01],
[-1.52347408e+00, -4.18583239e-01],
[-1.42531878e+00, -4.68255621e-01],
[-1.31867678e+00, -4.95261042e-01],
[-1.20871033e+00, -4.98292247e-01],
[-1.10074259e+00, -4.77202501e-01],
[-1.00000000e+00, -4.33012702e-01],
[-1.15000000e+00,  5.00000000e-01],
[-1.15603074e+00,  6.71010072e-01],
[-1.17339556e+00,  8.21393805e-01],
[-1.20000000e+00,  9.33012702e-01],
[-1.23263518e+00,  9.92403877e-01],
[-1.26736482e+00,  9.92403877e-01],
[-1.30000000e+00,  9.33012702e-01],
[-1.32660444e+00,  8.21393805e-01],
[-1.34396926e+00,  6.71010072e-01],
[-1.35000000e+00,  5.00000000e-01],
[ 1.07500000e+00, -2.16506351e-01],
[ 1.12537130e+00, -2.38601250e-01],
[ 1.17935516e+00, -2.49146123e-01],
[ 1.23433839e+00, -2.47630521e-01],
[ 1.28765939e+00, -2.34127810e-01],
[ 1.33673704e+00, -2.09291620e-01],
[ 1.37919563e+00, -1.74324200e-01],
[ 1.41297985e+00, -1.30918228e-01],
[ 1.43645431e+00, -8.11748673e-02],
[ 1.44848267e+00, -2.75020552e-02],
[ 1.44848267e+00,  2.75020552e-02],
[ 1.43645431e+00,  8.11748673e-02],
[ 1.41297985e+00,  1.30918228e-01],
[ 1.37919563e+00,  1.74324200e-01],
[ 1.33673704e+00,  2.09291620e-01],
[ 1.28765939e+00,  2.34127810e-01],
[ 1.23433839e+00,  2.47630521e-01],
[ 1.17935516e+00,  2.49146123e-01],
[ 1.12537130e+00,  2.38601250e-01],
[ 1.07500000e+00,  2.16506351e-01],
[-7.00000000e-01, -7.00000000e-01],
[-7.54670273e-01, -7.03630543e-01],
```

```
[-8.06694109e-01, -7.14346426e-01],
[-8.53553178e-01, -7.31628924e-01],
[-8.92979163e-01, -7.54641438e-01],
[-9.23063560e-01, -7.82269996e-01],
[-9.42350066e-01, -8.13177177e-01],
[-9.49905077e-01, -8.45866849e-01],
[-9.45362873e-01, -8.78756595e-01],
[-9.28943332e-01, -9.10254314e-01],
[-9.01441276e-01, -9.38835288e-01],
[-8.64188006e-01, -9.63115991e-01],
[-8.18986848e-01, -9.81921063e-01],
[-7.68025866e-01, -9.94340200e-01],
[-7.13771940e-01, -9.99772227e-01],
[-6.58851352e-01, -9.97954196e-01],
[-6.05922657e-01, -9.88974110e-01],
[-5.57547986e-01, -9.73266672e-01],
[-5.16069022e-01, -9.51592236e-01],
[-4.83493649e-01, -9.25000000e-01],
[ 1.00000000e+00, -3.00000000e-01],
[ 9.95074008e-01, -2.23273703e-01],
[ 9.80376918e-01, -1.47807250e-01],
[ 9.56150054e-01, -7.48397971e-02],
[ 9.22791222e-01, -5.56946880e-03],
[ 8.80848173e-01,  5.88663183e-02],
[ 8.31009610e-01,  1.17409530e-01],
[ 7.74093881e-01,  1.69098889e-01],
[ 7.11035541e-01,  2.13085658e-01],
[ 6.42870006e-01,  2.48647574e-01],
[ 5.70716552e-01,  2.75200712e-01],
[ 4.95759937e-01,  2.92309070e-01],
[ 4.19230947e-01,  2.99691730e-01],
[ 3.42386184e-01,  2.97227468e-01],
[ 2.66487440e-01,  2.84956747e-01],
[ 1.92780967e-01,  2.63081053e-01],
[ 1.22477026e-01,  2.31959584e-01],
[ 5.67300039e-02,  1.92103353e-01],
[-3.38053416e-03,  1.44166798e-01],
[-5.68675750e-02,  8.89370372e-02],
[-1.02852863e-01,  2.73209407e-02],
[-1.40581321e-01, -3.96697565e-02],
[-1.69433448e-01, -1.10935069e-01],
[-1.88935494e-01, -1.85304823e-01],
[-1.98767236e-01, -2.61557868e-01],
[-1.98767236e-01, -3.38442132e-01],
[-1.88935494e-01, -4.14695177e-01],
[-1.69433448e-01, -4.89064931e-01],
[-1.40581321e-01, -5.60330243e-01],
```

```
[-1.02852863e-01, -6.27320941e-01],
[-5.68675750e-02, -6.88937037e-01],
[-3.38053416e-03, -7.44166798e-01],
[ 5.67300039e-02, -7.92103353e-01],
[ 1.22477026e-01, -8.31959584e-01],
[ 1.92780967e-01, -8.63081053e-01],
[ 2.66487440e-01, -8.84956747e-01],
[ 3.42386184e-01, -8.97227468e-01],
[ 4.19230947e-01, -8.99691730e-01],
[ 4.95759937e-01, -8.92309070e-01],
[ 5.70716552e-01, -8.75200712e-01],
[ 6.42870006e-01, -8.48647574e-01],
[ 7.11035541e-01, -8.13085658e-01],
[ 7.74093881e-01, -7.69098889e-01],
[ 8.31009610e-01, -7.17409530e-01],
[ 8.80848173e-01, -6.58866318e-01],
[ 9.22791222e-01, -5.94430531e-01],
[ 9.56150054e-01, -5.25160203e-01],
[ 9.80376918e-01, -4.52192750e-01],
[ 9.95074008e-01, -3.76726297e-01],
[ 1.00000000e+00, -3.00000000e-01],
[ 4.64466094e-02, -7.93933983e-01],
[-2.95269772e-02, -8.23217243e-01],
[-7.92137412e-02, -8.57199566e-01],
[-9.95728792e-02, -8.93801254e-01],
[-8.93584227e-02, -9.30782301e-01],
[-4.91954909e-02, -9.65879488e-01],
[ 1.84579659e-02, -9.96944886e-01],
[ 1.09461592e-01, -1.02207731e+00],
[ 2.18246015e-01, -1.03973866e+00],
[ 3.38153684e-01, -1.04884809e+00],
[ 4.61846316e-01, -1.04884809e+00],
[ 5.81753985e-01, -1.03973866e+00],
[ 6.90538408e-01, -1.02207731e+00],
[ 7.81542034e-01, -9.96944886e-01],
[ 8.49195491e-01, -9.65879488e-01],
[ 8.89358423e-01, -9.30782301e-01],
[ 8.99572879e-01, -8.93801254e-01],
[ 8.79213741e-01, -8.57199566e-01],
[ 8.29526977e-01, -8.23217243e-01],
[ 7.53553391e-01, -7.93933983e-01]])
```

1.2.1 Orthogonal group

Rotation The principle of the rotation is to rotate the basis $\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$ in wich the object coordinates are written. The new basis after a rotation of an angle θ will be $\left\{ \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}, \begin{pmatrix} -\sin(\theta) \\ \cos(\theta) \end{pmatrix} \right\}$.

The matrix of rotation will be given by the following matrix

$$R_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

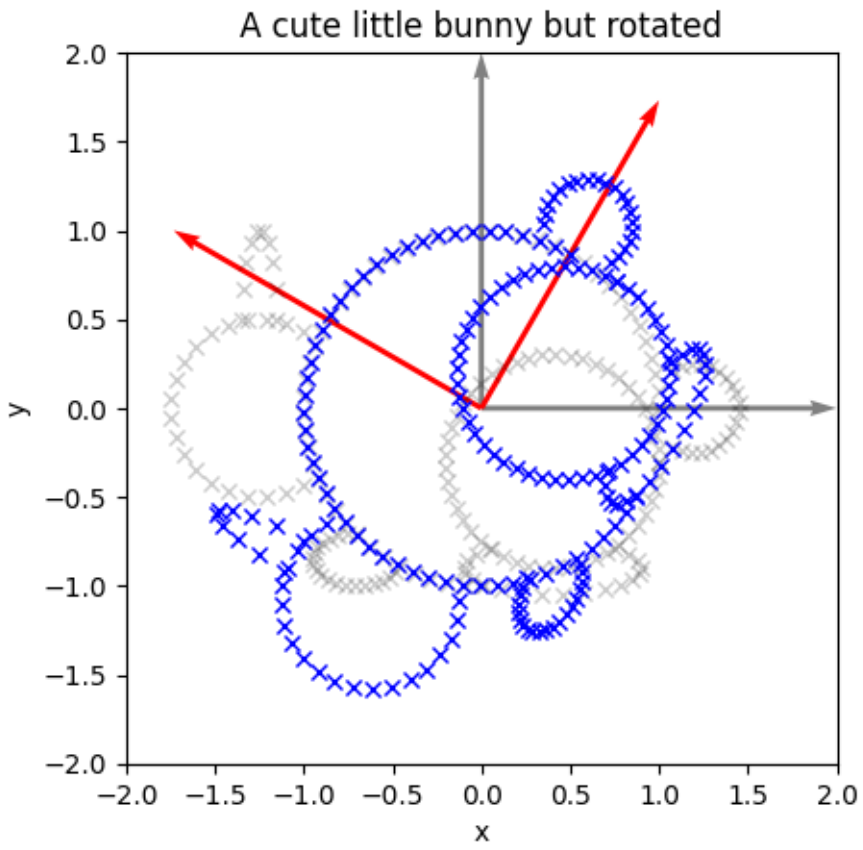
[12]:
```
theta=np.pi/3

R = np.array([[np.cos(theta),-np.sin(theta)],[np.sin(theta),np.cos(theta)]])

cloud2d_rot = R @ cloud2d.T #Instead of doing a loop for each point to apply␣
 ↪the transformation, we transpose the table.

f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny but rotated")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx", alpha=0.2)
plt.plot(cloud2d_rot[0,:],cloud2d_rot[1,:],"bx") #Carefull !! We changed the␣
 ↪order of indices because now the vectors are stored as columns
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
plt.quiver([0.,0.],[0.,0.],R[0,:],R[1,:],color="red",scale=0.5,scale_units="xy")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim([-2,2])
plt.ylim([-2,2])
```

[12]:
```
(-2.0, 2.0)
```



Reflections

Horizontal reflection

$$S_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

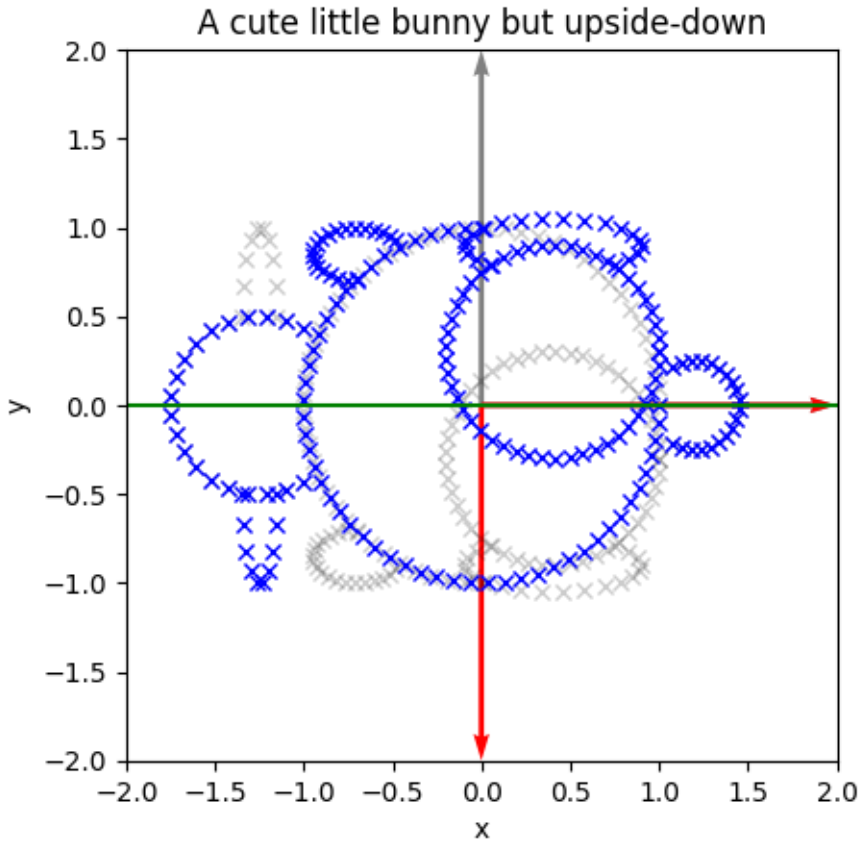
[13]:
```
Sx = np.array([[1,0],[0,-1]])

cloud2d_Sx = Sx @ cloud2d.T #Instead of doing a loop for each point to apply␣
 ↪the transformation, we transpose the table.

f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny but upside-down")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx", alpha=0.2)
plt.plot(cloud2d_Sx[0,:],cloud2d_Sx[1,:],"bx") #Carefull !! We changed the␣
 ↪order of indices because now the vectors are stored as columns
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
```

```
plt.quiver([0.,0.],[0.,0.],Sx[0,:],Sx[1,:],color="red",scale=0.
 ↪5,scale_units="xy")
plt.plot([-2,2],[0,0],"g")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim([-2,2])
plt.ylim([-2,2])
```

[13]: (-2.0, 2.0)



Vertical reflection

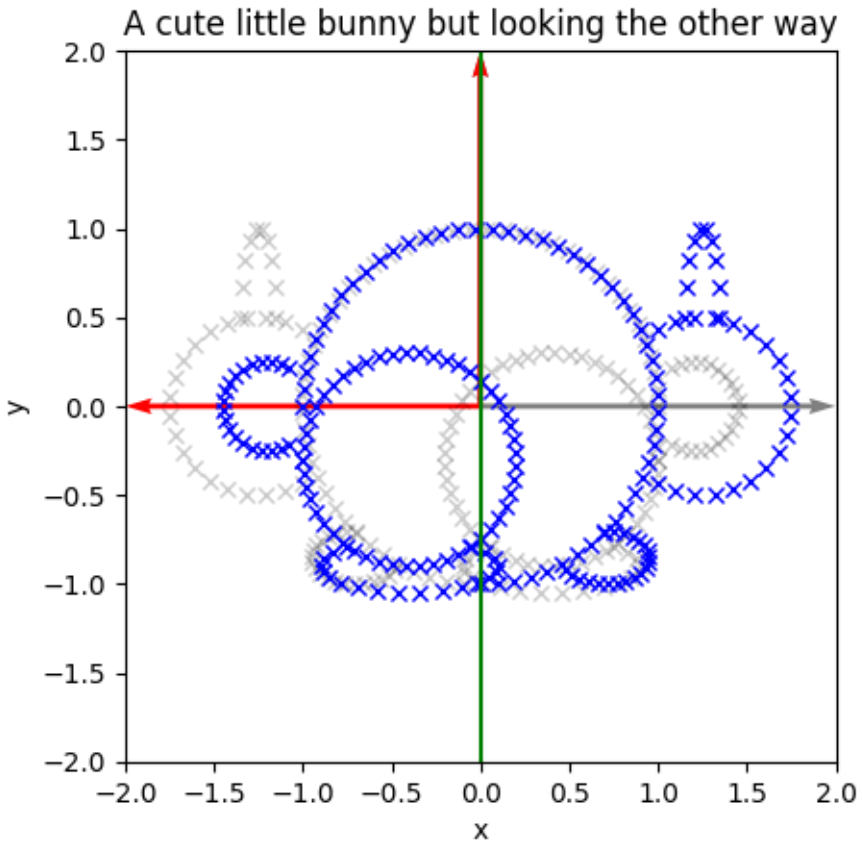
$$S_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

[14]:
```
Sy = np.array([[-1,0],[0,1]])

cloud2d_Sy = Sy @ cloud2d.T #Instead of doing a loop for each point to apply␣
 ↪the transformation, we transpose the table.
```

```
f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny but looking the other way")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx", alpha=0.2)
plt.plot(cloud2d_Sy[0,:],cloud2d_Sy[1,:],"bx") #Carefull !! We changed the␣
 ↪order of indices because now the vectors are stored as columns
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
plt.quiver([0.,0.],[0.,0.],Sy[0,:],Sy[1,:],color="red",scale=0.
 ↪5,scale_units="xy")
plt.plot([0,0],[-2,2],"g")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim([-2,2])
plt.ylim([-2,2])
```

[14]: (-2.0, 2.0)



Reflections For now, we dont know the matrix expression of a reflection through any other axis. However, we know how to do an horizontal reflection. Let us set an axis such that the angle between

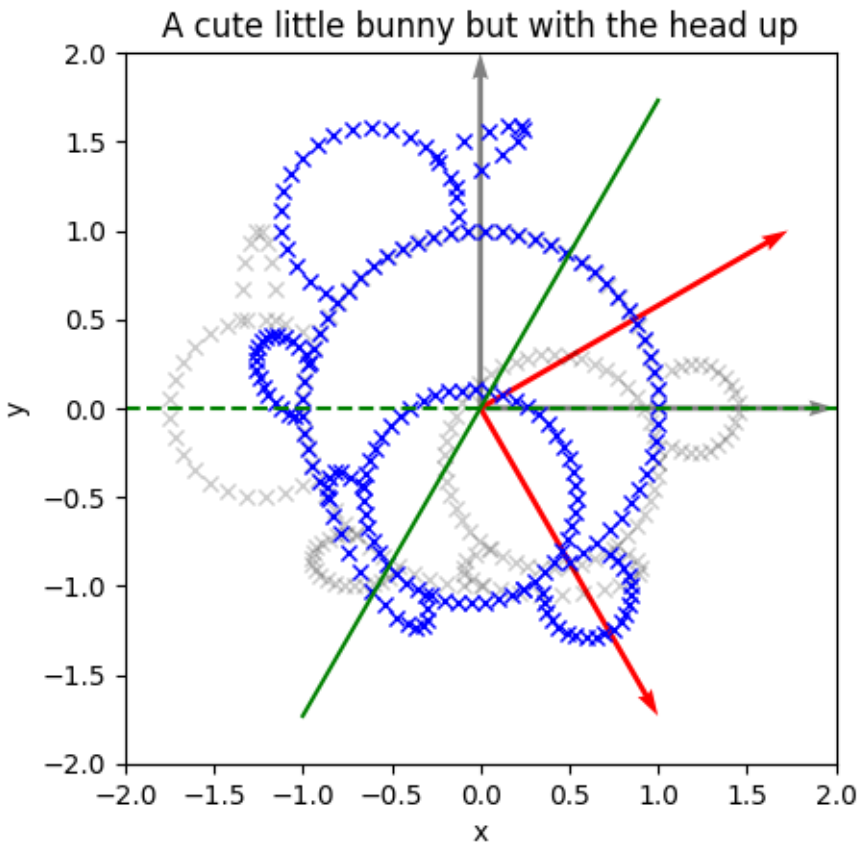
the axis and the absisca axis is θ. First, we will rotate the axis and the object of an angle $-\theta$ to align the axis with the absisca axis.

Remark: The rotation matrix of angle $-\theta$ is the inverse of the rotation matrix of angle θ (retrieve the calculus manualy), i.e. $R_{-\theta} = R_\theta^{-1}$.

[15]:
```
cloud2d_Ri = np.linalg.inv(R) @ cloud2d.T #Instead of doing a loop for each␣
 ↪point to apply the transformation, we transpose the table.

f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny but with the head up")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx", alpha=0.2)
plt.plot(cloud2d_Ri[0,:],cloud2d_Ri[1,:],"bx") #Carefull !! We changed the␣
 ↪order of indices because now the vectors are stored as columns
plt.plot([-2*np.cos(theta),2*np.cos(theta)],[-2*np.sin(theta),2*np.
 ↪sin(theta)],"g")
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
plt.quiver([0.,0.],[0.,0.],np.linalg.inv(R)[0,:],np.linalg.inv(R)[1,:
 ↪],color="red",scale=0.5,scale_units="xy")
plt.plot([-2,2],[0,0],"g--")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim([-2,2])
plt.ylim([-2,2])
```

[15]:
```
(-2.0, 2.0)
```



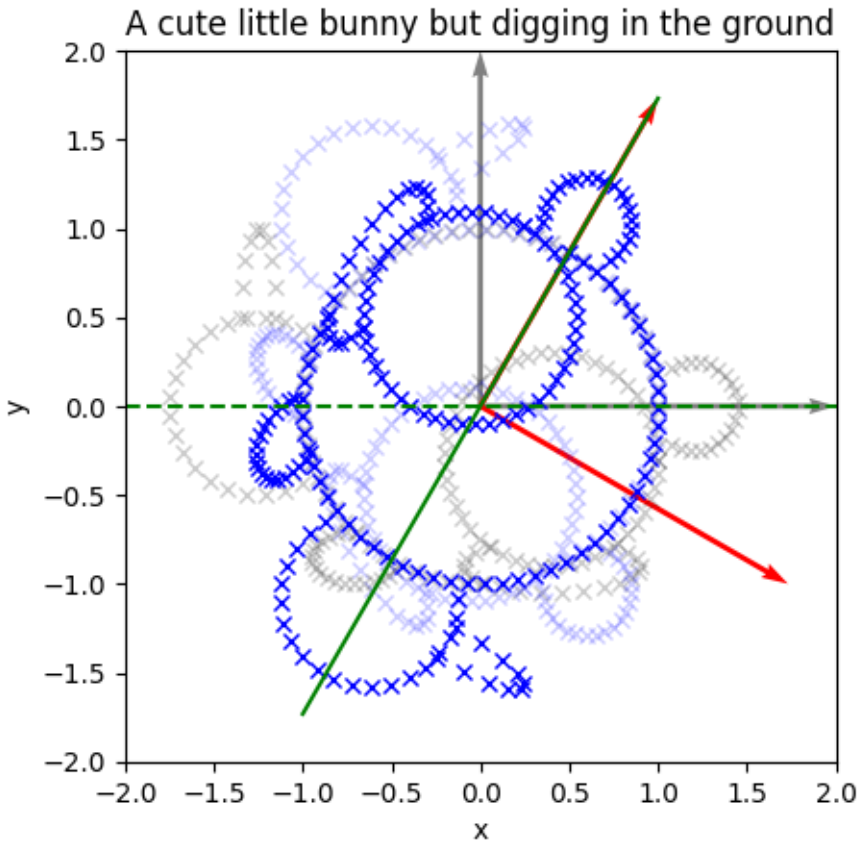
Then, we proceed to the horizontal reflection in the new basis.

[16]:
```
cloud2d_SRi = Sx@np.linalg.inv(R) @ cloud2d.T #Instead of doing a loop for each␣
 ↪point to apply the transformation, we transpose the table.

f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny but digging in the ground")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx", alpha=0.2)
plt.plot(cloud2d_Ri[0,:],cloud2d_Ri[1,:],"bx", alpha=0.2) #Carefull !! We␣
 ↪changed the order of indices because now the vectors are stored as columns
plt.plot(cloud2d_SRi[0,:],cloud2d_SRi[1,:],"bx") #Carefull !! We changed the␣
 ↪order of indices because now the vectors are stored as columns
plt.plot([-2*np.cos(theta),2*np.cos(theta)],[-2*np.sin(theta),2*np.
 ↪sin(theta)],"g")
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
plt.quiver([0.,0.],[0.,0.],(Sx@np.linalg.inv(R))[0,:],(Sx@np.linalg.inv(R))[1,:
 ↪],color="red",scale=0.5,scale_units="xy")
plt.plot([-2,2],[0,0],"g--")
plt.xlabel("x")
```

```
plt.ylabel("y")
plt.xlim([-2,2])
plt.ylim([-2,2])
```

[16]: (-2.0, 2.0)



And then we rotate back the reflection axis and the object, which give overall the following matrix:

$$S_\theta = R_\theta S_x R_{-\theta} = \begin{pmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{pmatrix}.$$

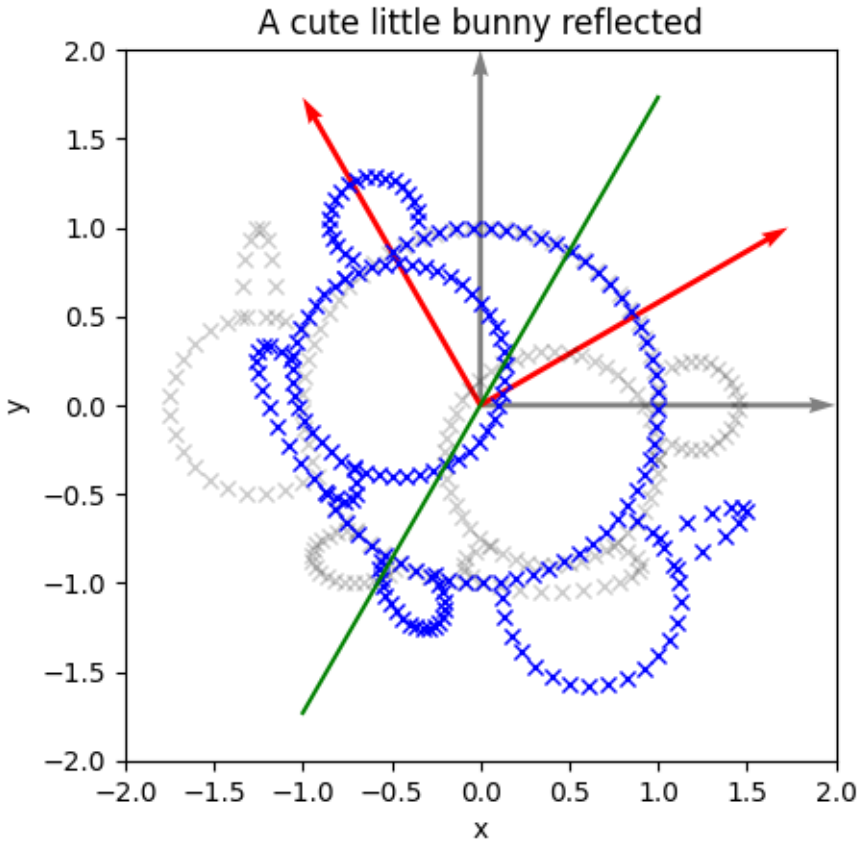
You should try to retrive the matrix by doing the calculus manually.

[17]:
```
Sa =R @ Sx @ np.linalg.inv(R)
cloud2d_Sa = Sa @ cloud2d.T #Instead of doing a loop for each point to apply␣
 ↪the transformation, we transpose the table.

f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny reflected")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx", alpha=0.2)
```

```
plt.plot(cloud2d_Sa[0,:],cloud2d_Sa[1,:],"bx") #Carefull !! We changed the␣
 ↪order of indices because now the vectors are stored as columns
plt.plot([-2*np.cos(theta),2*np.cos(theta)],[-2*np.sin(theta),2*np.
 ↪sin(theta)],"g")
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
plt.quiver([0.,0.],[0.,0.],Sa[0,:],Sa[1,:],color="red",scale=0.
 ↪5,scale_units="xy")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim([-2,2])
plt.ylim([-2,2])
```

[17]: (-2.0, 2.0)



1.2.2 Properties

All of these isometries compose the **orthogonal** group. An orthogonal matrix M is such that $M^tM = I$. Moreover reflections matrices are symmetric, and these transformations are involutions, thus in such a case $MM = Id$.

[18]:
```
print("Rotation :", R.T@R)
print("Horizontal reflection :", Sx.T@Sx)
print("Vertical reflection :", Sy.T@Sy)
print("General reflection :", Sa.T@Sa)
```

```
Rotation : [[ 1.00000000e+00 -1.48741681e-17]
 [-1.48741681e-17  1.00000000e+00]]
Horizontal reflection : [[1 0]
 [0 1]]
Vertical reflection : [[1 0]
 [0 1]]
General reflection : [[1.00000000e+00 8.52594875e-17]
 [8.52594875e-17 1.00000000e+00]]
```

Remark: we are supposed to have identity matrix, then why do we have these small values instead of 0? Because there is a lot of numerical approximation. Here we can consider that these values are actually 0.

1.2.3 Translation group

Translations are not orthogonal transformations and cannot be written as a matrix operation of the same dimension of the object.

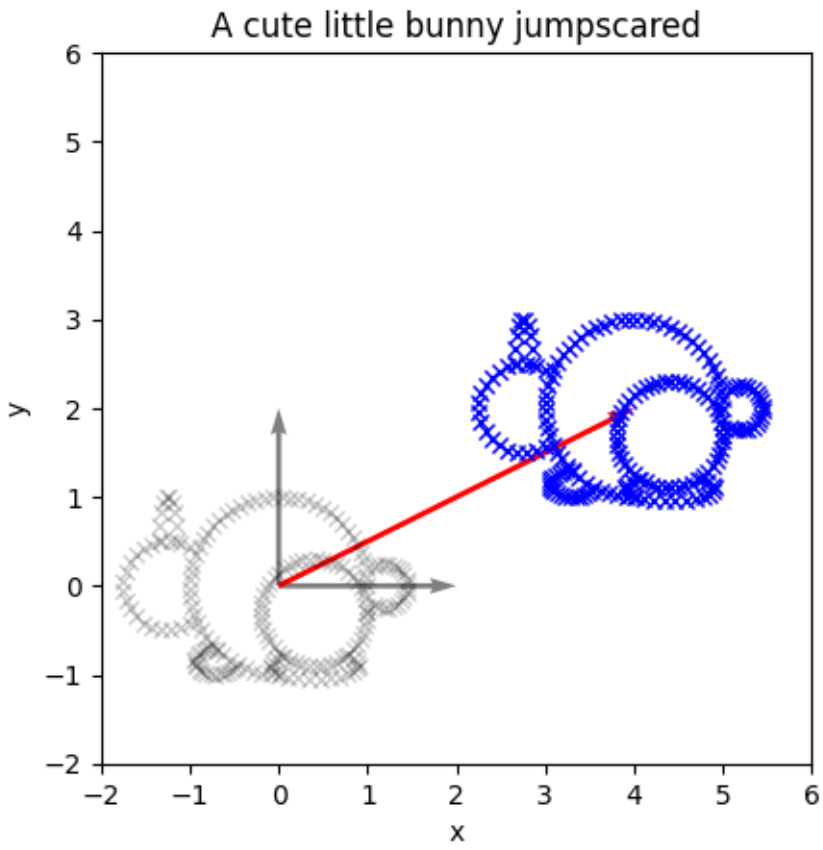
The result of the translation of a point x by a vector v is given by:

$$y = x + v.$$

[19]:
```
v=np.array([4,2])
cloud2d_t=cloud2d + v

f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny jumpscared")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx", alpha=0.2)
plt.plot(cloud2d_t[:,0],cloud2d_t[:,1],"bx")
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
plt.quiver([0.],[0.],[4],[2],color="red",scale=1,scale_units="xy")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim([-2,6])
plt.ylim([-2,6])
```

[19]:
```
(-2.0, 6.0)
```



For some application, it can be convenient to have only linear operations and not affine operations. It is possible to transform the translation as a linear operator by adding 1 to the dimension :

$$\begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 + v_1 \\ x_2 + v_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & v_1 \\ 0 & 1 & v_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}.$$

[38]:
```
M=np.array([[1, 0, v[0]],[0, 1, v[1]],[0, 0, 1]])
cloud3d=np.concatenate((cloud2d.T, np.ones((1,cloud2d.shape[0]))), axis=0)␣
 ↪#adding a third column full of ones in the array.
# Note that the dimension is given as (1,n) where n is the number of points, to␣
 ↪indicate that this is a column vector.

cloud2d_t=M@cloud3d

f=plt.figure()
ax=f.add_subplot(1,1,1, aspect="equal")
plt.title("A cute little bunny jumpscared")
plt.plot(cloud2d[:,0],cloud2d[:,1],"kx", alpha=0.2)
```

```
plt.plot(cloud2d_t[0,:],cloud2d_t[1,:],"bx")  #Carefull !! We changed the order␣
↪of indices because now the vectors are stored as columns
plt.quiver([0.,0.],[0.,0.],[1,0],[0,1],color="gray",scale=0.5,scale_units="xy")
plt.quiver([0.],[0.],[4],[2],color="red",scale=1,scale_units="xy")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim([-2,6])
plt.ylim([-2,6])
```

[38]: (-2.0, 6.0)

