Solutions to Part B of Problem Sheet 1

Solution (1.5)

(a) The unit circle with respect to the ∞-norm is the square with corners $(\pm 1, \pm 1)^\top$.

(b) The trick in transforming the unconstrained problem

$$\text{minimize} \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty \tag{1}$$

into a constrained linear programming problem is to characterise the ∞-norm as the solution of a minimization problem. In fact, for any set of numbers $x_1, \ldots, x_n$,

$$\max_{1 \le i \le n} |x_i| = \min_{\forall i \colon |x_i| \le t} t.$$

Put simply, the *maximum* of a set of non-negative numbers is the *smallest* upper bound on these numbers. We can further replace the condition $|x_i| \le t$ by $-t \le x_i \le t$, so that the problem (5) becomes

$$\begin{aligned} \underset{(\boldsymbol{x},t)}{\text{minimize}} \quad & t \\ \text{subject to} \quad & -t \le \boldsymbol{a}_1^\top \boldsymbol{x} - b_1 \le t \\ & \cdots \\ & -t \le \boldsymbol{a}_m^\top \boldsymbol{x} - b_m \le t, \end{aligned} \tag{2}$$

where $\boldsymbol{a}_i^\top$ are the rows of the matrix $\boldsymbol{A}$. This problem can be brought into *standard form* by replacing each condition with the pair of conditions

$$\begin{aligned} \boldsymbol{a}_i^\top \boldsymbol{x} - t &\le b_i \\ -\boldsymbol{a}_i^\top \boldsymbol{x} - t &\le -b_i. \end{aligned}$$

The solution $\boldsymbol{x}$ of Problem (5) can be read off the solution $(\boldsymbol{x}, t)$ of Problem (6).

Solution (1.6) We want to apply gradient descent to the function

$$f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$$

as described in Lecture 3.

The Python implementation, using numpy, looks as follows.

In [1]:
```
import numpy as np
import numpy.linalg as la

def graddesc(A, b, x, tol):
    # Compute the negative gradient r = A^T(b-Ax)
    r = np.dot(A.transpose(),b-np.dot(A,x))
    # Start with an empty array
    xout = []
    while la.norm(r,2) > tol:
        # If the gradient is bigger than the tolerance
        Ar = np.dot(A,r)
        alpha = np.dot(r,r)/np.dot(Ar,Ar)
        x = x + alpha*r
        xout.append(x)
        r = r-alpha*np.dot(A.transpose(),Ar)
    return np.array(xout).transpose()

A = np.array([[1,2], [2,1], [-1,0]])
b = np.array([10, -1, 0])
tol = 1e-4
x = np.zeros(2)

traj = graddesc(A, b, x, tol)
```

We can plot the trajectory on top of a contour plot.

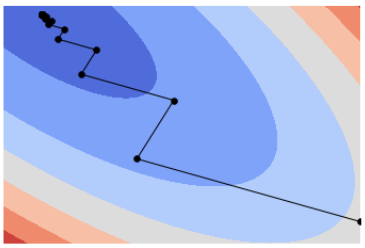
In [2]:
```
import matplotlib.pyplot as plt
% matplotlib inline

# Define the function we aim to minimize
def f(x):
    return np.dot(np.dot(A,x)-b,np.dot(A,x)-b)

# Create a mesh grid
xx = np.linspace(-3,1,100)
yy = np.linspace(2,6,100)
X, Y = np.meshgrid(xx, yy)
Z = np.zeros(X.shape)
for i in range(Z.shape[0]):
    for j in range(Z.shape[1]):
        Z[i,j] = f(np.array([X[i,j], Y[i,j]]))

# Get a nice monotone colormap
cmap = plt.cm.get_cmap("coolwarm")

# Plot the contours and the trajectory
plt.contourf(X, Y, Z, cmap = cmap)
plt.plot(traj[0,:], traj[1,:], 'o-k')
plt.show()
```



Solution (1.7) The gradient and the Hessian of the Rosenbrock function are

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{pmatrix}, \quad \nabla^2 f(\boldsymbol{x}) = \begin{pmatrix} -400x_2 + 1200x_1^2 + 2 & -400x_1 \\ -400x_1 & -400x_1 \end{pmatrix}$$

The point $(1, 1)^\top$ is a stationary point, as the gradient at this point vanishes. Moreover, with a computer program (Python or Matlab) one verifies that the eigenvalues of the Hessian matrix are positive, from which it follows that the matrix is positive definite. By the second order optimality conditions, it follows that $(1, 1)^\top$ is a local minimum. Moreover, it is the only local minimum, as there is no other point for which the gradient vanishes. The contour plot looks as follows.

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt

def rosenbrock(X, Y):
    return ((1-X)**2)+100*(Y-X**2)**2

xx = np.linspace(-2,2,100)
yy = np.linspace(-2,2,100)
X, Y = np.meshgrid(xx,yy)
Z = rosenbrock(X, Y)

%matplotlib inline
plt.figure()
levels = [0.1,0.5,1.0,2.0,5.0,10.0,50.0,100.0,200.0,400.0,600.0]
cmap = plt.cm.get_cmap("coolwarm")

cp = plt.contour(X,Y,Z,levels,cmap = cmap)
#plt.clabel(cp, inline=1, fontsize=10)
plt.title('Level sets of Rosenbrock function')
plt.xlabel('x')
plt.ylabel('y')
plt.plot([1],[1],'ro')
plt.show()
```