AA203 - Spring 2019
Problem Set Solutions #1

Problem 1.

Part (a)

The gradient of $f_1(x)$ is

$$\nabla f_1(x,y) = \begin{pmatrix} -\frac{4x}{2x^2+y^2-10} \\ -\frac{2y}{2x^2+y^2-10} \end{pmatrix},$$

which is equal to $(0,0)$ for $(x,y) = (0,0)$, so the first-order necessary conditions are satisfied. The Hessian is

$$\begin{pmatrix} \frac{4(2x^2-y^2+10)}{(2x^2+y^2-10)^2} & \frac{8xy}{(2x^2+y^2-10)^2} \\ \frac{8xy}{(2x^2+y^2-10)^2} & \frac{2(-2x^2+y^2+10)}{(2x^2+y^2-10)^2} \end{pmatrix}$$

At $(x,y) = (0,0)$, the Hessian is

$$\begin{pmatrix} 2/5 & 0 \\ 0 & 1/5 \end{pmatrix} \succ 0.$$

Therefore, $(0,0)$ satisfies both the necessary and sufficient conditions for a local minimum of f_1.

The gradient of f_2 is

$$\nabla f_2(x,y) = \begin{pmatrix} 2x(-2x^2+2y+1) \\ 2x^2 \end{pmatrix},$$

which is equal to $(0,0)$ for $(x,y) = (0,0)$, so the first-order necessary conditions are satisfied. The Hessian is

$$\begin{pmatrix} -12x^2+4y+2 & 4x \\ 4x & 0 \end{pmatrix}$$

At the origin, the Hessian is

$$\begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} \succeq 0.$$

which is positive semi-definite. Hence, the second-order necessary condition is satisfied. However, since the sufficient condition is not satisfied, this computation alone does not tell us whether $(0,0)$ is indeed a local minimum for f_2.

Part (b)

We established in part (a) that $(0,0)$ is a local minimum for f_1, as it satisfies the sufficient condition. Indeed, $(0,0)$ is a global minimum for f_1. To see this, note that if $x^2 \geq 0$ and $y^2 \geq 0$, then $10 - 2x^2 - y^2 \leq 10$ for all $x, y \in \text{dom}(f)$. Then, since $-\log$ is a decreasing function, we have

$$f_1(x,y) \geq -\log(10) = f_1(0,0)$$

for all $(x,y) \in \text{dom}(f)$, i.e., $(0,0)$ is a global minimum. Other acceptable arguments include that f_1 is a convex function over a convex domain, which implies that any local minima are global minima.

$(0,0)$ is not a global minimum for f_2. In fact, this function is unbounded below (fix $y = 0$ and take $x \to +\infty$). However, despite the inconclusiveness of the test from part (a), $(0,0)$ is indeed a local minimum for f_2. To see this, note that for all x, y such that $|x|, |y| \leq 1/4$, we have

$$x^2(1 + 2y - x^2) \geq 0 = f_2(0,0),$$

since both terms in the product (x^2 and $(1 + 2y - x^2)$) are non-negative.

Problem 2.

Note that to be a regular point, we simply need $\nabla h(x) \neq 0$, which is achieved for all feasible points. For any minimizer satisfying the regularity conditions, there exists a scalar λ such that

$$\nabla L(x) = \nabla f(x) + \lambda \nabla h(x) = 0$$
$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \lambda \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} = 0$$

So the KKT conditions are

$$1 + 2\lambda x_1 = 0$$
$$1 + 2\lambda x_2 = 0$$
$$x_1^2 + x_2^2 = 2$$

The two stationary points are

$$(x_1, x_2) = (1,1) \qquad (x_1, x_2) = (-1,-1).$$

The first is the highest, so it is our (global) maximum, the second is the lowest, so it is our (global) minimum.

We are able to identify these points as global extrema since the constraint is closed, bounded, and has no boundary points. Thus the global extrema must appear among the candidate local extrema. It should be noted that this problem is not a convex optimization problem, as the domain (a circle) is not convex.

Problem 3.

Note that to be a regular point, we simply need $\nabla g(x) \neq 0$, which is achieved for all feasible points. At the optimal point, we know that $\mu g(x) = 0$ (this is complementary slackness in the KKT conditions). There are two cases.

Case 1: $\mu = 0$. Then, the KKT conditions are

$$\nabla f(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 0, \qquad x_1 + x_2 + x_3 \leq -3.$$

Clearly, the last two equations contradict each other, so there are no local minima in the region where the constraints is inactive.

Case 2: $g(x) = 0$. Then, the KKT conditions are

$$\nabla f(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \mu \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 0, \qquad \mu \geq 0, \qquad x_1 + x_2 + x_3 = -3.$$

The only solution to the KKT conditions is

$$(x_1, x_2, x_3, \mu) = (-1, -1, -1, 1).$$

The condition $\mu \geq 0$ holds in this case, so this is our minimum (to prove that it is a minimum, note that the Hessian is the identity matrix).

Problem 4.

Part (a)

Suppose $x^* \in \mathbb{R}^n$ is a local minimum. By first-order optimality conditions,

$$\nabla f(x^*) = 0$$

Since $\nabla f(x^*) = Qx^* - b$, we have

$$Qx^* = b.$$

The matrix Q is positive definite and, therefore, invertible. It follows that $x^* = Q^{-1}b$.

The Hessian of f at any point is $\nabla^2 f(x) = Q \succ 0$. The Hessian being positive definite at any point x, it follows that f is convex. Therefore, x^* is a global minima.

Part (b)

Starting from a point x_0, the Newton method with step size $\eta = 1$ performs the following update

$$x_1 = x_0 - \nabla^2 f(x_0)^{-1} \nabla f(x_0).$$

We have

$$\nabla f(x_0) = Qx_0 - b, \qquad \nabla^2 f(x_0) = Q.$$

Therefore,

$$\begin{aligned} x_1 &= x_0 - Q^{-1}(Qx_0 - b) \\ &= x_0 - x_0 + Q^{-1}b \\ &= Q^{-1}b \\ &= x^* \end{aligned}$$

Then, $x_1 = x^*$.

In performing the Newton's step, we need to invert the matrix Q. Inverting a dense and unstructured matrix $Q \in \mathbb{R}^n$ requires $\mathcal{O}(n^3)$ floating point operations (flops), which can be prohibitive when n is large. When Q is positive definite, this time complexity can be improved, but is still of order n^3 (for instance, check out how the "Cholesky decomposition" can be used to speed up the inversion of a positive definite matrix).

Part (c)

Let $x \in \mathbb{R}^n$. We have

$$\begin{aligned} \|Sx\|_2 &= \|U(\Sigma U^\top x)\|_2 \\ &= \|\Sigma U^\top x\|_2 \end{aligned}$$

where the last inequality is justified by the fact $\|Uz\|_2 = \|z\|_2$ for any z, since U is an orthogonal matrix.

For the second inequality, let $z \in \mathbb{R}^n$. Then,

$$\begin{aligned}
\|\Sigma z\|_2 &= \sqrt{\sum_{i=1}^{n} \mu_i^2 z_i^2} \\
&\leq \sqrt{\sum_{i=1}^{n} \left(\max_k |\mu_k|^2\right) z_i^2} \\
&= \left(\max_{i=1,\ldots,n} |\mu_i|\right) \sqrt{\sum_{i=1}^{n} z_i^2} \\
&= \left(\max_{i=1,\ldots,n} |\mu_i|\right) \|z\|_2.
\end{aligned}$$

For the third inequality, let $x \in \mathbb{R}^n$. Then,

$$\begin{aligned}
\|Sx\|_2 &= \|\Sigma U^\top x\|_2 \\
&\leq \left(\max_{i=1,\ldots,n} |\mu_i|\right) \|U^\top x\|_2 \\
&= \left(\max_{i=1,\ldots,n} |\mu_i|\right) \|x\|_2.
\end{aligned}$$

where the last inequality holds since U preserves the norm.

Part (d)

Let $\{v_1, \ldots, v_n\}$ be an orthonormal family of eigenvectors of Q, associated with the eigenvalues $\lambda_1, \ldots, \lambda_n$ (such a family exists by the spectral decomposition theorem). By definition, $Qv_i = \lambda_i v_i$ for all $i = 1, \ldots, n$.

It holds that

$$(I - \eta Q)v_i = v_i - \eta Q v_i = v_i - \eta \lambda_i v_i = (1 - \eta\lambda_i)v_i.$$

Therefore, $\{v_1, \ldots, v_n\}$ is also an orthonormal family of eigenvectors of $I - \eta Q$, associated with the eigenvalues $1 - \eta\lambda_1, \ldots, 1 - \eta\lambda_n$.

Part (e)

At time step $k \geq 0$, we have

$$\begin{aligned}
x^{(k+1)} - x^* &= x^{(k)} - \eta \nabla f(x^{(k)}) - x^* \\
&= x^{(k)} - \eta(Qx^{(k)} - b) - x^* \\
&= x^{(k)} - \eta Q x^{(k)} + \eta b - x^*
\end{aligned}$$

Using $b = Qx^*$, we obtain

$$x^{(k+1)} - x^* = (I - \eta Q)\,(x^{(k)} - x^*).$$

Taking the norm,

$$\|x^{(k+1)} - x^*\|_2 = \|\,(I - \eta Q)\,(x^{(k)} - x^*)\|_2.$$

Using question (c), we know that the eigenvalues of $I - \eta Q$ are $1 - \eta\lambda_1, \ldots, 1 - \eta\lambda_n$. Using question (b), we have

$$\|\,(I - \eta Q)\,(x^{(k)} - x^*)\|_2 \leq \left(\max_{i=1,\ldots,n} |1 - \eta\lambda_i|\right) \|x^{(k)} - x^*\|_2.$$

Hence,

$$\|x^{(k+1)} - x^*\|_2 \leq \left(\max_{i=1,\ldots,n} |1 - \eta\lambda_i|\right) \|x^{(k)} - x^*\|_2.$$

which is exactly $\delta_{k+1} \leq (\max_{i=1,\ldots,n} |1 - \eta\lambda_i|)\,\delta_k$.

The induction proof is immediate. For $k = 0$, we have $\delta_1 \leq \gamma(\eta)\delta_0$. Then, we assume that the induction hypothesis is true for some $k \geq 0$, that is, we assume that $\delta_k \leq \gamma(\eta)^k\delta_0$. Using the fact that $\delta_{k+1} \leq \gamma(\eta)\delta_k$, it follows that $\delta_{k+1} \leq \gamma(\eta)\gamma(\eta)^k\delta_0 = \gamma(\eta)^{k+1}\delta_0$.

A sufficient condition is $\max_{i=1,\ldots,n} |1 - 2\lambda_i| < 1$, so that the term $\gamma(\eta)^k$ goes to 0 as $k \to \infty$. After some algebra, we obtain that if $\eta < 1/(2\max_{i=1,\ldots,n} \lambda_i)$, then $\gamma(\eta) < 1$.

Part (f)

Let $d_k = -(Qx^{(k)} - b)$. The optimal step size satisfies

$$\eta_k = \arg\min_{\eta \geq 0} f(x^{(k)} + \eta d_k).$$

Set $\varphi(\eta) = f(x^{(k)} + \eta d_k)$. The function φ is convex in η. Therefore, any η_k that satisfies

$$\varphi'(\eta_k) = 0.$$

is a global minimizer. The last equation is

$$d_k^\top (Q(x^{(k)} + \eta_k d_k) - b) = 0,$$

Rearranging the latter expression, we obtain

$$\eta_k = \frac{\|d_k\|_2^2}{d_k^\top Q d_k}.$$

Part (g)

The optimal solution is $(0, 0)$.

For $\gamma = 10$, the problem is ill-conditioned: the largest eigenvalue of Q is much greater (by a factor 10) than its smallest eigenvalue.

When starting from $x^{(0)} = (5, 1)$, the local curvature is large. The gradient points far from the point $(0, 0)$, and so do the gradients at the next iterates. Hence, gradient descent zigs zags with exact line search. On the other hand, with a constant step size, the sufficient condition we have derived above for convergence is very conservative: the gradient method takes very small steps. Although it does not zig zag compared to exact line search, it progresses very slowly.

When starting from $x^{(0)} = (1, 5)$, the curvature is more homogeneous, so that gradient descent does not zig zag much, and converges in a few steps to the minimum.

For $\gamma = 2$, the problem is much better conditioned. The initialization has less impact on the number of time steps required to converge to the minimum.

Problem 5.

At any time step $t \geq 0$,

$$x_{t+1} = Ax_t + Bu_t.$$

By induction, it can be shown that

$$x_t = A^t x_0 + \sum_{i=0}^{t-1} A^{t-1-i} B u_i.$$

We can write the relationship between $x_0, \ldots, x_T$ and $u_0, \ldots, u_{T-1}$ in matrix form as

$$\begin{aligned}
\begin{pmatrix} x_0 \\ \vdots \\ x_T \end{pmatrix} &= \begin{bmatrix} 0 & & & \ldots & 0 \\ B & 0 & & \ldots & 0 \\ AB & B & 0 & \ldots & 0 \\ \vdots & & & \ddots & \\ A^{T-1}B & A^{T-2}B & & \ldots & B \end{bmatrix} \begin{pmatrix} u_0 \\ \vdots \\ u_{T-1} \end{pmatrix} + \begin{bmatrix} I_n \\ A \\ \vdots \\ A^T \end{bmatrix} x_0 \\
&= Cu + Dx_0.
\end{aligned}$$

Next, we introduce two block diagonal matrices $\widehat{Q} \in \mathbb{R}^{n(T+1) \times n(T+1)}$ and $\widehat{R} \in \mathbb{R}^{mT \times mT}$,

$$\widehat{Q} = \begin{bmatrix} Q & & & 0 \\ & \ddots & & \\ & & Q & \\ 0 & & & Q_T \end{bmatrix}$$

and

$$\widehat{R} = \begin{bmatrix} R & & 0 \\ & \ddots & \\ 0 & & R \end{bmatrix}$$

We can then vectorize the cost function as

$$J(u) = (Cu+Dx_0)^\top \widehat{Q}(Cu+Dx_0)+u^\top \widehat{R}u = u^\top (C^\top \widehat{Q}C+\widehat{R})u+2(C^\top \widehat{Q}Dx_0)^\top u+x_0^\top D^\top \widehat{Q}Dx_0.$$

Finally, we can set $\widetilde{Q} = 2(C^\top \widehat{Q}C + \widehat{R})$, and $\widehat{b} = -2C^\top \widehat{Q}Dx_0$.

By solving the linear system $\widetilde{Q}u = \widetilde{b}$, we obtain the optimal vector of decision variables u^*, and we find that the optimal cost is $J(u^*) \approx 2.94$.

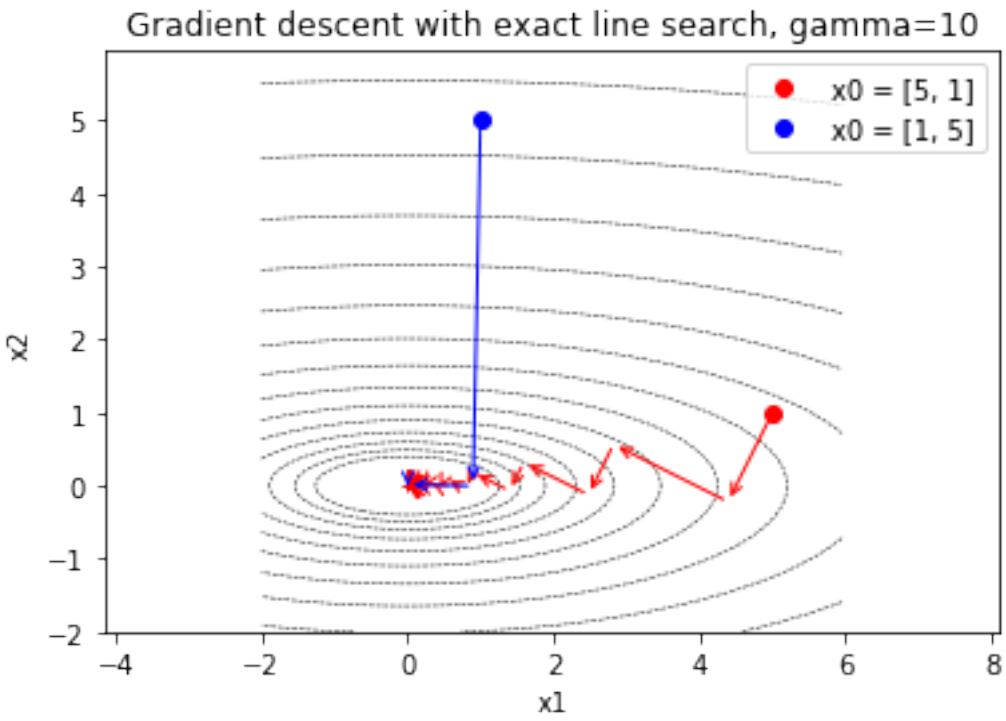
AA203 - Code for Problem 4, HW1

April 11, 2019

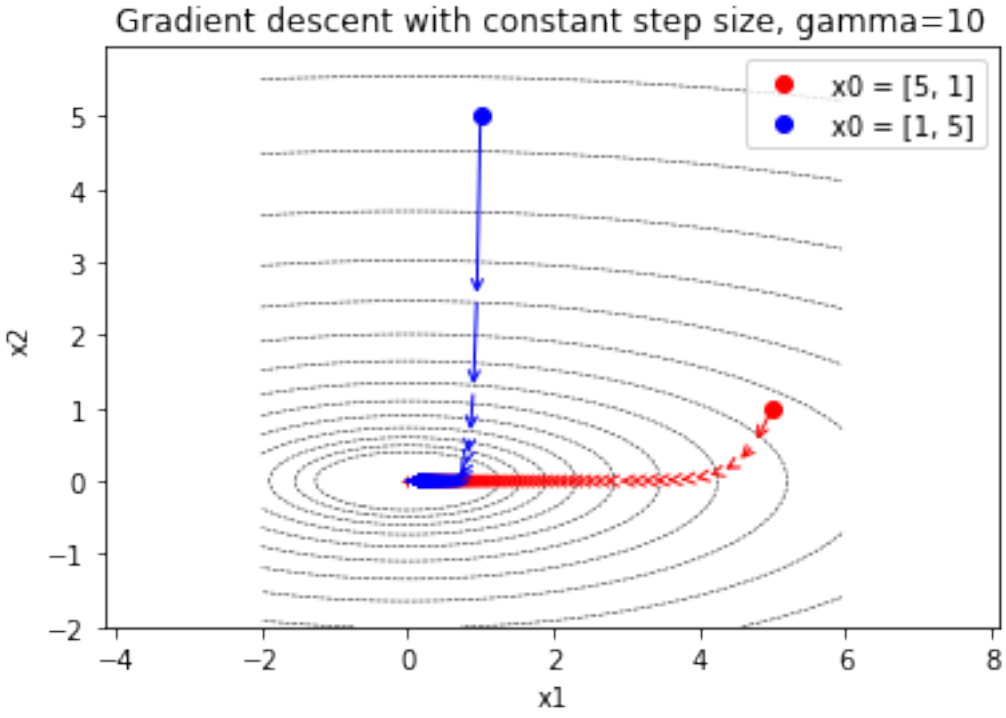
In [26]:
```
import numpy as np
import matplotlib.pyplot as plt
import time

from utils import *
```

In [27]:
```
gamma = 10.
Q = np.diag([1, gamma])

def gradientdescent(x, tol=1e-3, stepsize=None):
    """
    x: initial state
    tol: tolerance threshold, the algorithm stops
         when the norm of the gradient is less than the threshold.
    stepsize: if None, runs exact line search.
              Otherwise, the value provided is the value of the constant step size.
    """
    iterates = []
    while np.linalg.norm(np.dot(Q,x)) > tol:
        iterates.append(x)
        d = -np.dot(Q, x)
        if not stepsize:
            eta = np.inner(d,d) / np.inner(d, np.dot(Q, d))
        else:
            eta = stepsize
        x = x + eta * d
    return iterates

exactls_goodinit = gradientdescent(np.array([1,5]))
exactls_badinit = gradientdescent(np.array([5, 1]))
constantss_goodinit = gradientdescent(np.array([1, 5]),
                                      stepsize=0.999/(2*gamma))
constantss_badinit = gradientdescent(np.array([5, 1]),
                                     stepsize=.999/(2*gamma))
```

In [28]:
```
plot_res_badc(exactls_goodinit,
              exactls_badinit)
```

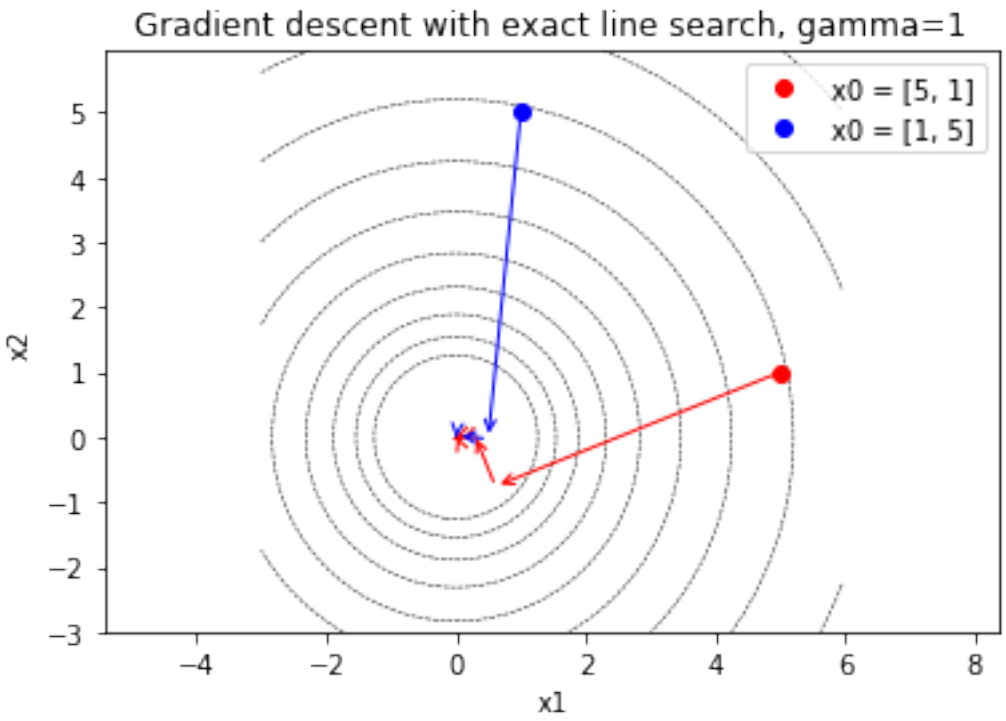


In [29]:
```
plot_res_badc(constantss_goodinit,
              constantss_badinit,
              stepsize=.999/(2*gamma))
```

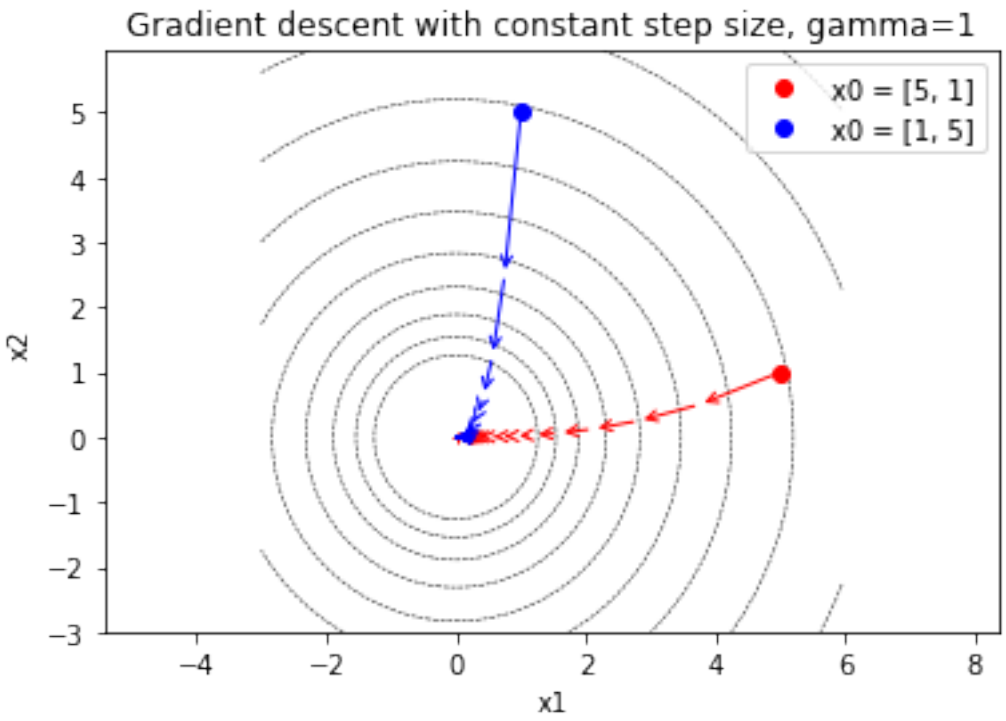


In [30]:
```
gamma=2

Q = np.diag([1, gamma])

def gradientdescent(x, tol=1e-3, stepsize=None):
    """
    x: initial state
    tol: tolerance threshold, the algorithm stops
         when the norm of the gradient is less than the threshold.
    stepsize: if None, runs exact line search.
         Otherwise, the value provided is the value of the constant step size.
    """
    iterates = []
    while np.linalg.norm(np.dot(Q,x)) > tol:
        iterates.append(x)
        d = -np.dot(Q, x)
        if not stepsize:
            eta = np.inner(d,d) / np.inner(d, np.dot(Q, d))
        else:
            eta = stepsize
        x = x + eta * d
    return iterates

exactls_goodinit = gradientdescent(np.array([1,5]))
exactls_badinit = gradientdescent(np.array([5, 1]))
constantss_goodinit = gradientdescent(np.array([1, 5]),
                                      stepsize=0.999/(2*gamma))
constantss_badinit = gradientdescent(np.array([5, 1]),
                                     stepsize=0.999/(2*gamma))
```

In [31]:
```
plot_res_wellc(exactls_goodinit,
               exactls_badinit,
               gamma=1)
```




```
In [32]: plot_res_wellc(constantss_goodinit,
                        constantss_badinit,
                        gamma=1,
                        stepsize=.999/(2*gamma))
```


Gradient descent with constant step size, gamma=1
x0 = [5, 1]
x0 = [1, 5]
x1
x2

AA203 - Code for Problem 5, HW1.

April 11, 2019

In [1]:
```
import numpy as np
```

In [2]:
```
# define problem parameters

A = np.array([[1,1.], [0, 1.]])
B = np.array([[0.], [1.]])

Q = np.eye(2)
R = np.eye(1)
Qf = 10.*np.eye(2)
xinit = np.array([1,0])

T = 20
```

Build matrices C, D, $\hat{Q}$, $\hat{R}$, $\tilde{Q}$ and $\tilde{R}$.

In [3]:
```
# build C and D
D = np.vstack([np.linalg.matrix_power(A, ii) for ii in range(T+1)])
C_ = np.hstack([np.dot(np.linalg.matrix_power(A, T-1-ii), B) for ii in range(T)])
C = np.vstack( [np.hstack([C_[:,T-1-ii:], np.zeros((2,T-1-ii))]) for ii in range(T)] )
C = np.vstack( [np.zeros((2,T)), C] )

from scipy.linalg import block_diag

# build \hat{Q}
Qhat = np.copy(Q)
for _ in range(T-1):
    Qhat = block_diag(Qhat, Q)
Qhat = block_diag(Qhat, Qf)

# build \hat{R}
Rhat = np.copy(R)
for _ in range(T-1):
    Rhat = block_diag(Rhat,R)

# \tilde{Q} and \tilde{B}
Qtilde = 2*(np.dot(np.dot(C.T, Qhat), C) + Rhat)
btilde = - 2 * np.dot(np.dot(C.T, Qhat), np.dot(D,xinit))
```

Compute optimal policy by solving $\tilde{Q}u = b$

```
In [4]: uopt = np.dot(np.linalg.pinv(Qtilde), btilde)
```

```
In [5]: # simulate system
        cost = 0.
        x = np.copy(xinit)
        for ii in range(T):
            cost += np.dot( np.dot(Q,x), x ) + np.dot(np.dot(R, uopt[ii]), uopt[ii])
            x = A.dot(x) + B.dot(uopt[ii]).reshape(-1,)
        cost += np.dot(np.dot(Qf,x), x)
        cost = np.float(cost)
```

```
In [6]: print('The optimal is equal to', cost)
```

```
The optimal is equal to 2.9471229667070147
```