HOMEWORK 2

Elena Gómez

January 23, 2023

PROBLEM 0: Homework checklist

I worked alone

PROBLEM 1: Optimization software

We'll be frequently using software to optimize functions, this question will help familiarize you with a piece of optimization software relevant for our study.
The function we'll study is the exponentiated Rosenbrock function:

$$f(x) = exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2)$$

1. Show a contour plot of this function

In [1]:

```
using Plots; pythonplot()
using LaTeXStrings
```

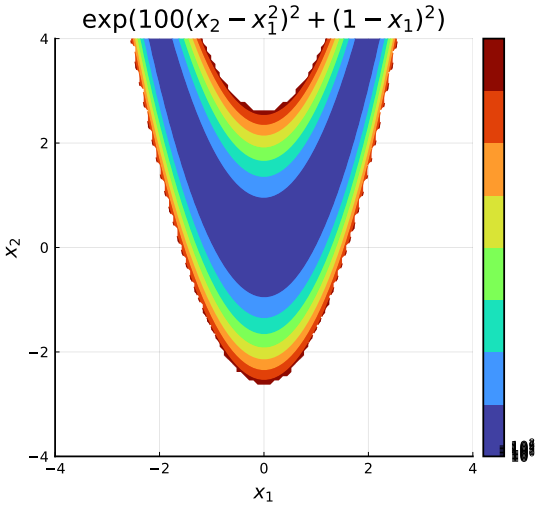
In [2]:

```
h(x, y) = exp(100*(y-x^2)^2 + (1-x)^2)

x = range(-4,4 ,length=100)
y = range(-4, 4, length=100)
z = @. h(x', y)

tv = 0:8
tl = [L"10^{%$i}" for i in tv]
contourf(x, y, log10.(z), color=:turbo, levels=8,
    colorbar_ticks=(tv, tl), aspect_ratio=:equal,
    title=L"\exp(100(x_{2} - x_{1}^{2})^{2} + (1-x_{1})^2 )", xlabel=L"x_{1}", ylabel=L"x_{2}")
```

Out[2]:




```
sys:1: UserWarning: The following kwargs were not used by contour: 'label'
```

2. Write the gradient and Hessian of this function.

Firstly, let's compute the gradient

$$g(x) = \nabla f(x_1, x_2) = \frac{\partial f}{\partial x_1}\imath + \frac{\partial f}{\partial x_2}\jmath =$$
$$= \frac{\partial f}{\partial x_1} exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2)\imath + \frac{\partial f}{\partial x_2} exp(100(x_2 - x_1^2)^2\imath + (1 - x_1)^2)\jmath =$$
$$= exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2) \cdot [-400x_1(x_2 - x_1^2) - 2(1 - x_1)]\imath + exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2) \cdot 200(x_2 - x_1^2)\jmath$$

Now, let's compute the hessian

$$\mathbb{Hess}(\mathbb{f}) = \left[\frac{\partial^2 \mathbf{f(x)}}{\partial x_1 \partial x_2}\right] = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_2} \\ \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} \end{bmatrix}$$

Where
applying $z(x) = f(x) \cdot g(x) = f(x)' \cdot g(x) + f(x) \cdot g(x)'$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} = exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2) \cdot [-400x_1(x_2 - x_1^2) - 2(1 - x_1)]^2 + exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2) \cdot [-400x_2 + 1200x_2^2 + 2]$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} = exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2)) \cdot [200x_1(x_2 - x_1^2) - 2(1 - x_1)]^2 + 200 \cdot exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2))$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_2} = \frac{\partial^2 f(\mathbf{x})}{\partial x_2 x_1} = exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2) \cdot [-400x_1(x_2 - x_1^2) - 2(1 - x_1)] \cdot [200(x_2 - x_1^2)] - 400x_1 exp(100(x_2 - x_1^2)^2 + (1 - x_1)^2)$$

3. By inspection, what is the minimizer of this function? (Feel free to find the answer by other means, e.g. looking it up, but make sure you explain why you know that answer must be a global minimizer.)

In a multivariate function, the sufficient conditions to prove that a function has a minimizer are:
$g(x) = 0$
$H(x) > 0$

We have to find the point $x = (x_1, x_2)$ that fulfills the conditions mentioned above

I saw on the internet that there is a global minimum at $x = (1, 1)$. But if this point is a global minimum it has to fulfill the sufficient conditions. So, let's see if the sufficient conditions are fulfilled at this point.

Firstly, let's see if $g(x) = 0$

$$g(1, 1) = exp(100(1 - 1^2)^2 + (1 - 1)^2) \cdot [-400(1 - 1^2) - 2(1 - 1)]\imath + exp(100(1 - 1^2)^2 + (1 - 1)^2) \cdot 200(1 - 1^2)\jmath = 0\imath + 0\jmath$$

so $g(1, 1) = 0$

No, let's see if $H(x) > 0$

$$H(1, 1) = \begin{bmatrix} 802 & (-400) \\ (-400) & 200 \end{bmatrix}$$

Where:

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} = exp(100(1 - 1^2)^2 + (1 - 1)^2) \cdot [-400(1 - 1^2) - 2(1 - 1)]^2 + exp(100(1 - 1^2)^2 + (1 - 1)^2) \cdot [-400 + 1200 + 2] = 802$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} = exp(100(1 - 1^2)^2 + (1 - 1)^2)) \cdot [200(1 - 1^2) - 2(1 - 1)]^2 + 200 \cdot exp(100(1 - 1^2)^2 + (1 - 1)^2)) = 200$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_2} = \frac{\partial^2 f(\mathbf{x})}{\partial x_2 x_1} = exp(100(1 - 1^2)^2 + (1 - 1)^2) \cdot [-400(1 - 1^2) - 2(1 - 1)] \cdot [200(1 - 1^2)] - 400exp(100(1 - 1^2)^2 + (1 - 1)^2) = -400$$

$H(x) > 0$ if its minors are greater than 0

$M_{11} = 802 > 0$

$M_{22} = 800 \cdot 200 - (-400) \cdot (-400) = 400 > 0$

So, $H(1, 1) > 0$

The 2 sufficient conditions are fulfilled, so the function has a global minimum at $x = (1, 1)$

And the solution I saw is the correct one

---

4. Explain how any optimization package could tell that your solution is a local minimizer

An optimization package could tell that my solution is a local minimizer because at this point the gradient is equal to zero(saddle point) and when comparing $f(1, 1)$ with f of other points around $(1, 1)$, $f(1, 1)$ is smaller that f of any point around $(1, 1)$

Optimization packages work iteratively. They start at the point you input and they try different numbers until an optimal solution is found.

The number of iterations also depends on the point you input. If you input a point that is close to a minimizer the algorithm will found the solution faster than if you input a solution that is not close to a minimizer.

They try to find a point in which the gradient is equal to 0 (a critical point). Maybe they do not find a point where the gradient is equal to 0, but it is close to 0. They also see if at this point x, f(x) is smaller than f at any other point around x. In order to proof that this point is a MINIMIZER

5. Use Optim.jl (or Poblano for Matlab, or Scikit.learn for python) to optimize this function starting from a few different points. Be adversarial if you wish. Does it always get the answer correct? Show your code. You must use a call where you provide the gradient.

In [3]:

```
#packages
using Plots, LinearAlgebra, Random, SparseArrays
using Optim
```

In [4]:

```
#function to find (in this case) the minimum
function f(x)
    return exp(100 * (x[2]-x[1]^2)^2 + (1-x[1])^2)
end
function g1!(storage::Vector, x::Vector)
storage[1] = exp(100 * (x[2]-x[1]^2)^2 + (1-x[1])^2) * (-400*x[1]*(x[2]-x[1]^2)-2*(1-x[1]))
storage[2] = exp(100 * (x[2]-x[1]^2)^2 + (1-x[1])^2)*200*(x[2] - x[1]^2)
end
```

Out[4]:

```
g1! (generic function with 1 method)
```

Let's try with GradientDescent()

In [5]:

```
soln0 = optimize(f, g1!, [0.0, 0.0], GradientDescent())
#when we input (0,0) it does not find an accurate solution in less than 1000 iterations
```

Out[5]:

```
 * Status: failure (reached maximum number of iterations)

 * Candidate solution
    Final objective value:     1.007254e+00

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 2.67e-04 ≤ 0.0e+00
    |x - x'|/|x'|          = 2.92e-04 ≤ 0.0e+00
    |f(x) - f(x')|         = 1.54e-05 ≤ 0.0e+00
    |f(x) - f(x')|/|f(x')| = 1.53e-05 ≤ 0.0e+00
    |g(x)|                 = 1.17e-01 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    1000
    f(x) calls:    2534
    ∇f(x) calls:   2534
```

In [6]:

```
soln0.minimizer
```

Out[6]:

```
2-element Vector{Float64}:
 0.9149949423686361
 0.837068172825764
```

In [7]:

```
soln1 = optimize(f, g1!, [1.0, 1.0], GradientDescent())
#when we input (1,1), it does not find an accurate solution in less than 1000 iterations
```

Out[7]:

```
 * Status: success

 * Candidate solution
    Final objective value:     1.000000e+00

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 0.00e+00 ≤ 0.0e+00
    |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|         = NaN ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = NaN ≤/0.0e+00
    |g(x)|                 = 0.00e+00 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    0
    f(x) calls:    1
    ∇f(x) calls:   1
```

In [8]:

```
soln1.minimizer
```

Out[8]:

```
2-element Vector{Float64}:
 1.0
 1.0
```

In [9]:

```

soln05 = optimize(f, g1!, [0.5, 0.5], GradientDescent())
#when we input (0.5,0.5) it does not find an accurate solution in less than 1000 iterations
```

Out[9]:

```
 * Status: failure (reached maximum number of iterations)

 * Candidate solution
    Final objective value:     1.003523e+00

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 1.66e-04 ≤/0.0e+00
    |x - x'|/|x'|          = 1.77e-04 ≤/0.0e+00
    |f(x) - f(x')|         = 6.83e-06 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 6.81e-06 ≤/0.0e+00
    |g(x)|                 = 7.59e-02 ≤/1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    1000
    f(x) calls:    2542
    ∇f(x) calls:   2542
```

In [10]:

```
soln05.minimizer
```

Out[10]:

```
2-element Vector{Float64}:
 0.9408213915033533
 0.8847668622446094
```

As I said before the solution is more or less accurate depending of the point we input. When we input a point closer to the $x = (1, 1)$ the solution is more accurate than when we input numbers that are not close to $x = (1, 1)$

In [11]:

```
soln10 = optimize(f, g1!, [10.0, 10.0], GradientDescent())

```

Out[11]:

```
 * Status: failure

 * Candidate solution
    Final objective value:     Inf

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 0.00e+00 ≤ 0.0e+00
    |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|         = NaN ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = NaN ≤/0.0e+00
    |g(x)|                 = Inf ≤/1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    0
    f(x) calls:    1
    ∇f(x) calls:   1
```

In [12]:

```
soln11 = optimize(f, g1!, [1.0000001, 1.0000000001], GradientDescent())
#if we input (1.0000001, 1.0000000001) it finds an accurate solution in less than 1000 iterations
```

Out[12]:

```
 * Status: success

 * Candidate solution
    Final objective value:     1.000000e+00

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 5.15e-11 ≤/0.0e+00
    |x - x'|/|x'|          = 5.15e-11 ≤/0.0e+00
    |f(x) - f(x')|         = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|/|f(x')| = 0.00e+00 ≤ 0.0e+00
    |g(x)|                 = 1.96e-08 ≤/1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    3
    f(x) calls:    9
    ∇f(x) calls:   9
```

In [13]:

```
soln11.minimizer
```

Out[13]:

```
2-element Vector{Float64}:
 1.0000000199058734
 1.0000000399097855
```

Let's use BFGS()

In [14]:

```
1 solnB0 = optimize(f, g1!, [0.0, 0.0], BFGS())
2 #when we input (0,0) it finds an accurate solution in less than 1000 iterations
```

Out[14]:

```
 * Status: success

 * Candidate solution
    Final objective value:     1.000000e+00

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 1.82e-07 ≤/0.0e+00
    |x - x'|/|x'|          = 1.82e-07 ≤/0.0e+00
    |f(x) - f(x')|         = 8.88e-15 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 8.88e-15 ≤/0.0e+00
    |g(x)|                 = 2.17e-10 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    21
    f(x) calls:    64
    ∇f(x) calls:   64
```

In [15]:

```
1 solnB0.minimizer
```

Out[15]:

```
2-element Vector{Float64}:
 1.000000000001222
 1.000000000002993
```

In [17]:

```
1 solnB1 = optimize(f, g1!, [1.0, 1.0], BFGS())
2 #when we input (1,1), it  finds an accurate solution in less than 1000 iterations
```

Out[17]:

```
 * Status: success

 * Candidate solution
    Final objective value:     1.000000e+00

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 0.00e+00 ≤ 0.0e+00
    |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|         = NaN ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = NaN ≤/0.0e+00
    |g(x)|                 = 0.00e+00 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    0
    f(x) calls:    1
    ∇f(x) calls:   1
```

In [18]:

```
1 solnB1.minimizer
2
```

Out[18]:

```
2-element Vector{Float64}:
 1.0
 1.0
```

In [19]:

```
solnB05 = optimize(f, g1!, [0.5, 0.5], BFGS())
#when we input (0.5,0.5) it  finds an accurate solution in less than 1000 iterations
```

Out[19]:

```
 * Status: success

 * Candidate solution
    Final objective value:     1.000000e+00

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 3.42e-09 ≤/0.0e+00
    |x - x'|/|x'|          = 3.42e-09 ≤/0.0e+00
    |f(x) - f(x')|         = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|/|f(x')| = 0.00e+00 ≤ 0.0e+00
    |g(x)|                 = 1.31e-13 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    10
    f(x) calls:    47
    ∇f(x) calls:   47
```

In [20]:

```
solnB05.minimizer
```

Out[20]:

```
2-element Vector{Float64}:
 0.9999999999999987
 0.999999999999997
```

In [21]:

```
solnB10 = optimize(f, g1!, [10.0, 10.0], BFGS())
#when we input (10,10) does not find an accurate solution in less than 1000 iterations
```

Out[21]:

```
 * Status: failure

 * Candidate solution
    Final objective value:     Inf

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 0.00e+00 ≤ 0.0e+00
    |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|         = NaN ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = NaN ≤/0.0e+00
    |g(x)|                 = Inf ≤/1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    0
    f(x) calls:    1
    ∇f(x) calls:   1
```

In [22]:

```
soln10.minimizer
```

Out[22]:

```
2-element Vector{Float64}:
 10.0
 10.0
```

As we can see, when we use BFGS() instead of Gradientdescendent(), the function find better solutions.

In [90]:

```
soln = optimize(f, g1!, [3.0, 3.0], GradientDescent())

```

Out[90]:

```
 * Status: failure

 * Candidate solution
    Final objective value:     Inf

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 0.00e+00 ≤ 0.0e+00
    |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|         = NaN ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = NaN ≤/0.0e+00
    |g(x)|                 = Inf ≤/1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    0
    f(x) calls:    1
    ∇f(x) calls:   1
```

When we use the gradient descendent and input (3,3) the maximum number of iterations is reached

PROBLEM 2: Optimization software

Repeat all the steps for problem 1 for the Booth function

$$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

Using the same algorithm as in problem 1, which function takes the most iterations when starting from the point (3,3).

1. Show a contour plot of this function

In [23]:

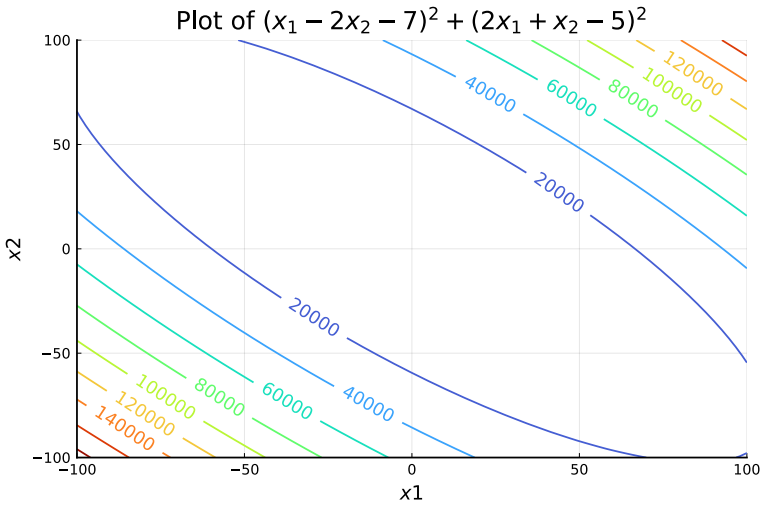
```
using Plots; pythonplot()

f(x, y) = (x+(2*y) - 7)^2 + ((2*x) + y -5)^2

x = range(-100, 100, length=100)
y = range(-100, 100, length=50)
z = @. f(x', y)

contour(x, y, z, levels=10, color=:turbo, clabels=true, cbar=false, lw=1)
title!(L"Plot of $(x_{1}- 2x_{2} - 7)^2 + (2x_{1} + x_{2} -5)^2$")
xlabel!(L"x1")
ylabel!(L"x2")
```

Out[23]:



2. Write the gradient and Hessian of this function.

Firstly, let's compute the gradient

$$g(x) = \nabla f(x_1, x_2) = \frac{\partial f}{\partial x_1}\imath + \frac{\partial f}{\partial x_2}\jmath = \quad = [2 \cdot (x_1 + 2x_2 - 7) + 4 \cdot (2x_1 + x_2 - 5)]\imath + [4 \cdot (x_1 + 2x_2 - 7) + 2 \cdot (2x_1 + x_2 - 5)]\jmath =$$

Now, let's compute the hessian matrix

$$\mathbb{Hess}(\mathbb{f}) = \left[\frac{\partial^2 \mathbf{f}(\mathbf{x})}{\partial x_1 \partial x_2} \right] = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_2} \\ \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} \end{bmatrix}$$

Where
$\frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} = 2 + 8 = 10$
$\frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} = 8 + 2 = 10$
$\frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_2} = \frac{\partial^2 f(\mathbf{x})}{\partial x_2 x_1} = 4 + 4 = 8$

3. By inspection, what is the minimizer of this function? (Feel free to find the answer by other means, e.g. looking it up, but make sure you explain why you know that answer must be a global minimizer.)

In a multivariate function, the sufficient conditions to prove that a function has a minimizer are:
$g(x) = 0$
$H(x) > 0$

Let's find the point x that makes $g(x) = 0$ and $H(x) = 0$

$g(x) = \nabla f(x_1, x_2) = \frac{\partial f}{\partial x_1} \imath + \frac{\partial f}{\partial x_2} \jmath = \ = [2 \cdot (x_1 + 2x_2 - 7) + 4 \cdot (2x_1 + x_2 - 5)] \imath + [4 \cdot (x_1 + 2x_2 - 7) + 2 \cdot (2x_1 + x_2 - 5)] \jmath = 0$

we have to find x1,x2 that make both terms be 0. We have to solve the following system of equations:

$2 \cdot (x_1 + 2x_2 - 7) + 4 \cdot (2x_1 + x_2 - 5) = 0$
$4 \cdot (x_1 + 2x_2 - 7) + 2 \cdot (2x_1 + x_2 - 5) = 0$

$10x_1 + 8x_2 - 34 = 0$
$8x_1 + 10x_2 - 38 = 0$

$g(x) = 0$ at $x = (1, 3)$, if $H(1, 3) > 0$, $x = (1, 3)$ is a global minimum

Let's compute $H(1, 3)$

$$H(1, 3) = \begin{bmatrix} 10 & 8 \\ \\ 8 & 10 \end{bmatrix}$$

Let's compute the minors of the Hessian to see if $H(x) > 0$

$M_{11} = 10 > 0$

$M_{22} = 10 \cdot 10 - 8 \cdot 8 = 36 > 0$

The 2 sufficient conditions are fulfilled, so the function has a global minimum at $x = (1, 3)$

4. Explain how any optimization package could tell that your solution is a local minimizer

An optimization package could tell that my solution is a local minimizer because at this point the gradient is equal to zero(saddle point) and when comparing $f(1, 3)$ with f of other points around $(1, 3)$, $f(1, 3)$ is smaller that f of any point around $(1, 3)$

Optimization packages work iteratively. They start at the point you input and they try different numbers until an optimal solution is found.

The number of iterations also depends on the point you input. If you input a point that is close to a minimizer the algorithm will found the solution faster than if you input a solution that is not close to a minimizer.

They try to find a point in which the gradient is equal to 0 (a critical point). Maybe they do not find a point where the gradient is equal to 0, but it is close to 0. They also see if at this point x, f(x) is smaller than f at any other point around x. In order to proof that this point is a MINIMIZER

5. Use Optim.jl (or Poblano for Matlab, or Scikit.learn for python) to optimize this function starting from a few different points. Be adversarial if you wish. Does it always get the answer correct? Show your code. You must use a call where you provide the gradient.

In [24]:

```
using Plots, LinearAlgebra, Random, SparseArrays
using Optim
```

In [25]:

```
function f(x)
    return (x[1] + 2*x[2] - 7)^2 + (2*x[1] + x[2] -5)^2

end
function g2!(storage::Vector, x::Vector)
storage[1] = 2*(x[1] + 2*x[2] -7) +4*(2x[1] + x[2] -5)
storage[2] = 4*(x[1] +2*x[2] -7) + 2*(2*x[1] +x[2] - 5)
end
```

Out[25]:

```
g2! (generic function with 1 method)
```

In [26]:

```
soln2 = optimize(f, g2!, [0.0, 0.0], GradientDescent())
#when we input (0,0), it finds an accurate solution in less than 1000 iterations
```

Out[26]:

```
 * Status: success

 * Candidate solution
    Final objective value:     3.104698e-17

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 9.38e-09 ≤/0.0e+00
    |x - x'|/|x'|          = 3.13e-09 ≤/0.0e+00
    |f(x) - f(x')|         = 1.42e-15 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 4.58e+01 ≤/0.0e+00
    |g(x)|                 = 8.32e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    11
    f(x) calls:    33
    ∇f(x) calls:   33
```

In [27]:

```
soln2.minimizer
```

Out[27]:

```
2-element Vector{Float64}:
 1.0000000039636276
 2.999999996085006
```

In [28]:

```
soln21 = optimize(f, g2!, [1.0, 1.0], GradientDescent())
#when we input (1,1), it finds an accurate solution in less than 1000 iterations

```

Out[28]:

```
 * Status: success

 * Candidate solution
    Final objective value:     3.562248e-18

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 1.69e-09 ≤/0.0e+00
    |x - x'|/|x'|          = 5.63e-10 ≤/0.0e+00
    |f(x) - f(x')|         = 4.16e-17 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 1.17e+01 ≤/0.0e+00
    |g(x)|                 = 2.96e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    17
    f(x) calls:    51
    ∇f(x) calls:   51
```

In [29]:

```
soln21.minimizer
```

Out[29]:

```
2-element Vector{Float64}:
 1.000000001350138
 2.999999998682792
```

In [30]:

```

soln25 = optimize(f, g2!, [0.5, 0.5], GradientDescent())
#when we input (0.5,0.5), it finds an accurate solution in less than 1000 iterations

```

Out[30]:

```
 * Status: success

 * Candidate solution
    Final objective value:     1.094753e-17

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 4.26e-09 ≤/0.0e+00
    |x - x'|/|x'|          = 1.42e-09 ≤/0.0e+00
    |f(x) - f(x')|         = 2.84e-16 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 2.59e+01 ≤/0.0e+00
    |g(x)|                 = 5.02e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    13
    f(x) calls:    39
    ∇f(x) calls:   39
```

In [31]:

```
soln25.minimizer
```

Out[31]:

```
2-element Vector{Float64}:
 1.0000000023581452
 2.999999997680355
```

In [32]:

```
soln215 = optimize(f, g2!, [1.5, 1.5], GradientDescent())
#when we input (1.5,1.5), it finds an accurate solution in less than 1000 iterations


```

Out[32]:

```
 * Status: success

 * Candidate solution
    Final objective value:     8.561159e-18

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 1.49e-09 ≤/0.0e+00
    |x - x'|/|x'|          = 4.96e-10 ≤/0.0e+00
    |f(x) - f(x')|         = 2.68e-17 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 3.14e+00 ≤/0.0e+00
    |g(x)|                 = 5.04e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    29
    f(x) calls:    87
    ∇f(x) calls:   87
```

In [33]:

```
soln215.minimizer
```

Out[33]:

```
2-element Vector{Float64}:
 1.000000002114249
 2.999999997987642
```

In [34]:

```
soln2105 = optimize(f, g2!, [10.5, 10.5], GradientDescent())
```

Out[34]:

```
 * Status: success

 * Candidate solution
    Final objective value:     5.039074e-18

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 1.54e-08 ≤/0.0e+00
    |x - x'|/|x'|          = 5.13e-09 ≤/0.0e+00
    |f(x) - f(x')|         = 4.15e-15 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 8.23e+02 ≤/0.0e+00
    |g(x)|                 = 3.22e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    7
    f(x) calls:    21
    ∇f(x) calls:   21
```

In [35]:

```
soln2105
```

Out[35]:

```
 * Status: success

 * Candidate solution
    Final objective value:     5.039074e-18

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 1.54e-08 ≤/0.0e+00
    |x - x'|/|x'|          = 5.13e-09 ≤/0.0e+00
    |f(x) - f(x')|         = 4.15e-15 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 8.23e+02 ≤/0.0e+00
    |g(x)|                 = 3.22e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    7
    f(x) calls:    21
    ∇f(x) calls:   21
```

As we can see we find an accurate solution in all the cases we tried.

Let's try with BFGS()

In [36]:

```
solnB2 = optimize(f, g2!, [0.0, 0.0], BFGS())
#when we input (0,0), it finds an accurate solution in less than 1000 iterations
```

Out[36]:

```
 * Status: success

 * Candidate solution
    Final objective value:     7.888609e-31

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 8.94e-01 ≤/0.0e+00
    |x - x'|/|x'|          = 2.98e-01 ≤/0.0e+00
    |f(x) - f(x')|         = 1.58e+00 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 2.00e+30 ≤/0.0e+00
    |g(x)|                 = 3.55e-15 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    2
    f(x) calls:    5
    ∇f(x) calls:   5
```

In [37]:

```
1 solnB2.minimizer
```

Out[37]:

```
2-element Vector{Float64}:
 0.9999999999999998
 3.0000000000000004
```

In [38]:

```
1 solnB21 = optimize(f, g2!, [1.0, 1.0], BFGS())
2 #when we input (1,1), it finds an accurate solution in less than 1000 iterations
```

Out[38]:

```
 * Status: success

 * Candidate solution
    Final objective value:     7.888609e-31

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 8.99e-01 ≤/0.0e+00
    |x - x'|/|x'|          = 3.00e-01 ≤/0.0e+00
    |f(x) - f(x')|         = 1.58e+00 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 2.00e+30 ≤/0.0e+00
    |g(x)|                 = 3.55e-15 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    2
    f(x) calls:    5
    ∇f(x) calls:   5
```

In [39]:

```
1 solnB21.minimizer
```

Out[39]:

```
2-element Vector{Float64}:
 0.9999999999999998
 3.0000000000000004
```

In [40]:

```
1
2 solnB25 = optimize(f, g2!, [0.5, 0.5], BFGS())
3 #when we input (0.5,0.5), it finds an accurate solution in less than 1000 iterations
4
```

Out[40]:

```
 * Status: success

 * Candidate solution
    Final objective value:     0.000000e+00

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 8.96e-01 ≤/0.0e+00
    |x - x'|/|x'|          = 2.99e-01 ≤/0.0e+00
    |f(x) - f(x')|         = 1.58e+00 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = Inf ≤/0.0e+00
    |g(x)|                 = 0.00e+00 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    2
    f(x) calls:    5
    ∇f(x) calls:   5
```

In [41]:

```
1 solnB25.minimizer
```

Out[41]:

```
2-element Vector{Float64}:
 0.9999999999999998
 3.0
```

In [42]:

```
solnB215 = optimize(f, g2!, [1.5, 1.5], BFGS())
#when we input (1.5,1.5), it finds an accurate solution in less than 1000 iterations
```

Out[42]:

```
 * Status: success

 * Candidate solution
    Final objective value:     7.888609e-31

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 9.06e-01 ≤/0.0e+00
    |x - x'|/|x'|          = 3.02e-01 ≤/0.0e+00
    |f(x) - f(x')|         = 1.57e+00 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 1.99e+30 ≤/0.0e+00
    |g(x)|                 = 3.55e-15 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    2
    f(x) calls:    5
    ∇f(x) calls:   5
```

In [43]:

```
solnB215.minimizer
```

Out[43]:

```
2-element Vector{Float64}:
 0.9999999999999997
 3.0
```

In [44]:

```
solnB2105 = optimize(f, g2!, [10.5, 10.5], BFGS())
```

Out[44]:

```
 * Status: success

 * Candidate solution
    Final objective value:     6.310887e-30

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 8.90e-01 ≤/0.0e+00
    |x - x'|/|x'|          = 2.97e-01 ≤/0.0e+00
    |f(x) - f(x')|         = 1.58e+00 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 2.50e+29 ≤/0.0e+00
    |g(x)|                 = 1.07e-14 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    2
    f(x) calls:    5
    ∇f(x) calls:   5
```

In [45]:

```
solnB2105.minimizer
```

Out[45]:

```
2-element Vector{Float64}:
 0.9999999999999994
 2.9999999999999996
```

because I input point that are close to the global minimizer

Using the same algorithm as in problem 1, which function takes the most iterations when starting from the point $(3, 3)$.

Let's compute the algorithm for the first function

In [46]:

```
soln = optimize(f, g1!, [3.0, 3.0], GradientDescent())
```

Out[46]:

```
 * Status: failure

 * Candidate solution
    Final objective value:     2.000000e+01

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 0.00e+00 ≤ 0.0e+00
    |x - x'|/|x'|          = 0.00e+00 ≤ 0.0e+00
    |f(x) - f(x')|         = NaN ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = NaN ≤/0.0e+00
    |g(x)|                 = Inf ≤/1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    0
    f(x) calls:    1
    ∇f(x) calls:   1
```

Let's compute the algorithm for the second function

In [47]:

```
soln = optimize(f, g2!, [3.0, 3.0], GradientDescent())

```

Out[47]:

```
 * Status: success

 * Candidate solution
    Final objective value:     3.562246e-18

 * Found with
    Algorithm:     Gradient Descent

 * Convergence measures
    |x - x'|               = 1.69e-09 ≤/0.0e+00
    |x - x'|/|x'|          = 5.63e-10 ≤/0.0e+00
    |f(x) - f(x')|         = 4.16e-17 ≤/0.0e+00
    |f(x) - f(x')|/|f(x')| = 1.17e+01 ≤/0.0e+00
    |g(x)|                 = 2.96e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    17
    f(x) calls:    51
    ∇f(x) calls:   51
```

The function of problem 1 reach total number of iterations (1000) while the function of problem 2 find the minimizer in 17 iterations. Because $(3, 3)$ is closer to the global minimizer of the second function.

Problem 3: Optimization theory

Suppose that f : R → R (i.e. is univariate) and is four times continuously differentiable. Show that the following conditions imply that x^* is a local minimizer.

i. $f'(x^*) = 0$
ii. $f''(x^*) = 0$
iii. $f'''(x^*) = 0$
iv. $f''''(x^*) > 0$

ANSWER

$f''''(x)$ is positive in a small neighborhood because f is four times differentiable.

$\exists r > 0$ such that $|x - x^* < r| \longrightarrow f''''(x) > 0$

Let's use the Taylor Theorem to show that the conditions above imply that x_* is a local minimizer. (For f in x^* with h in the small neighborhood)

$\forall |h| < r, \ \ \exists \alpha \in [0, 1],$
$f(x^* + h) = f(x^*) + \frac{h}{1!} f'(x^*) + \frac{h^2}{2!} f''(x^*) + \frac{h^3}{3!} f'''(x^*) + \frac{h^4}{4!} f''''(x^* + \alpha h)$

The first 3 conditions are :
i. $f'(x^*) = 0$
ii. $f''(x^*) = 0$

iii.$f'''(x^*) = 0$

If we substitute $f'(x^*) = 0$, $f''(x^*) = 0$ and $f'''(x^*) = 0$ in the Taylor's theorem, we get:

$\forall |h| < r, \ \ \exists \alpha \in [0,1],$
$f(x^* + h) = f(x^*) + \frac{h^4}{4!} f''''(x^* + \alpha h)$

Since x^* is in the small neighborhood in which f'''' is still positive: $f''''(x^* + \alpha) > 0$

$|(x^* + \alpha h) - x^*| = |\alpha h| < r \longrightarrow f''''(x^* + \alpha h) > 0$

So, x^* is a strict local minimizer

Therefore, x^* is a strict local minimizer: $\forall |h| < 0, f(x^* + h) > f(x^*)$

Problem 4: Convexity
Convex functions are all the rage these days, and one of the interests of students in this class. Let's do some matrix analysis to show that a function is convex. Solve problem 2.7 in the textbook, which is:
Suppose that $f(x) = x^T Q x$, where Q is an $n \times n$ symmetric positive semi-definite matrix. Show that this function is convex using the definition of convexity, which can be equivalently reformulated:
$f(y + \alpha(x - y)) - \alpha f(x) - (1 - \alpha) f(y) \leq 0$
for all $0 \leq \alpha \leq 1$ and all $x, y \in R^n$. This type of function will frequently arise in our subsequent studies, so it's an important one to understand.

ANSWER

if $f(x) = x^T Q x$, let's compute $f(y + \alpha(x - y)) - \alpha f(x) - (1 - \alpha) f(y)$:

$f(y + \alpha(x - y)) - \alpha f(x) - (1 - \alpha) f(y) = (y + \alpha(x - y))^T \cdot Q \cdot (y + \alpha(x - y)) \cdot y^T \cdot Q \cdot y =$

$= y^T Q y + \alpha y^T Q x - \alpha y^T Q y + \alpha x^T Q y - \alpha y^T Q y + \alpha^2 x^T Q x - \alpha^2 x^T Q y - \alpha^2 x^T Q x + \alpha^2 x^T Q y - \alpha x^T Q x - (1 - \alpha) y^T Q y =$

$= \alpha(\alpha - 1)(y^T Q y - y^T Q x - x^T Q y + x^T Q y) =$

$= \alpha(\alpha - 1)(x - y)^T Q (x - y) = \alpha\,(\alpha - 1)(x - y)^T Q (x - y)$

As $0 \leq \alpha \leq 1$, $\alpha\,(\alpha - 1) \leq 0$

As Q is an nxn symmetric positive semi-definite matrix,
$(x - y)^T Q (x - y) \geq 0$

So,for all $0 \leq \alpha \leq 1$ and all $x, y \in R^n$

$f(y + \alpha(x - y)) - \alpha f(x) - (1 - \alpha) f(y) \leq 0$

In [ ]:

```
1
```