Homework 04

Brown University

DATA 1010

Fall 2020

Problem 1

For the function $f(x) = x^2$, apply gradient descent with the starting point $x_0 = 4$.

(a) For which learning rate would the second step having length zero? For which learning rate would the *third* step be the first length-zero step?

(b) For which learning rates does the sequence of iterates never visit the ray $\{x \in \mathbb{R} : x < 0\}$?

(c) For which learning rates does the sequence of iterates diverge to $\pm\infty$?

In [2]:
```
using ForwardDiff: gradient
derivative(f, x) = gradient(x->f(x[1]), [x])[1]

function graddescent(f, x₀, ϵ, threshold)
    x = x₀
    step = 0
    df(x) = derivative(f, x)
    while abs(df(x)) > threshold
        step += 1
        x = x - ϵ*df(x)
    end
    return x, step
end
f(x) = x^2
x = 4
println(graddescent(f, x, 0.4999999, 1e-12))
println(graddescent(f, x, 0.49999, 1e-12))
```

```
(1.600000000510034e-13, 2)
(3.200000000006772e-14, 3)
```

(a) Using threshold $1e-12$, the learning rate 0.4999999 will have second step length 0 and 0. 49999 will have third step first length-zero step. If we do not threshold, the second step length is $2*(4-8*r)*r$ and r = 0.5 will have the second step length 0. The third step length is $2*(4-8*r-2*(4-8*r)*r)*r$, we also have r = 0.5 but in this case, the second step will not be the first length-zero step so we do not have the third step as first length-zero step in this problem.

In [54]:
```julia
using ForwardDiff: gradient
derivative(f, x) = gradient(x->f(x[1]), [x])[1]

function graddescent(f, x₀, ϵ, threshold)
    x = x₀
    step = 0
    df(x) = derivative(f, x)
    while abs(df(x)) > threshold
        step += 1
        x = x - ϵ*df(x)
        if x < 0
            return false
        end
    end
    return true
end
f(x) = x^2
x = 4
[(le, graddescent(f, x, le, 1e-12)) for le = 1:-0.05:0.1]
```

Out[54]:
```
19-element Array{Tuple{Float64,Bool},1}:
 (1.0, 0)
 (0.95, 0)
 (0.9, 0)
 (0.85, 0)
 (0.8, 0)
 (0.75, 0)
 (0.7, 0)
 (0.65, 0)
 (0.6, 0)
 (0.55, 0)
 (0.5, 1)
 (0.45, 1)
 (0.4, 1)
 (0.35, 1)
 (0.3, 1)
 (0.25, 1)
 (0.2, 1)
 (0.15, 1)
 (0.1, 1)
```

(b) For learning rates that are larger than 0.5, we will visit the ray $\{x \in \mathbb{R} \, : \, x < 0\}$. For r > 0.5, we will reach x < 0 after the first step. For x < 0.5, the step length * learning rate is always smaller than the current x value because $x - r * 2 * x > 0$ for x <= 0.5.

In [74]:
```julia
using ForwardDiff: gradient
derivative(f, x) = gradient(x->f(x[1]), [x])[1]

function graddescent(f, x₀, ϵ, threshold)
    x = x₀
    step = 0
    df(x) = derivative(f, x)
    last_value = abs(df(x))
    while abs(df(x)) > threshold
        x = x - ϵ*df(x)
        if abs(df(x)) > last_value
            return false
        else
            return true
        end
        last_value = abs(df(x))
    end
end
f(x) = x^2
x = 4
[(le, graddescent(f, x, le, 1e-12)) for le = 2:-0.1:0]
```

Out[74]:
```
21-element Array{Tuple{Float64,Bool},1}:
 (2.0, 0)
 (1.9, 0)
 (1.8, 0)
 (1.7, 0)
 (1.6, 0)
 (1.5, 0)
 (1.4, 0)
 (1.3, 0)
 (1.2, 0)
 (1.1, 0)
 (1.0, 1)
 (0.9, 1)
 (0.8, 1)
 (0.7, 1)
 (0.6, 1)
 (0.5, 1)
 (0.4, 1)
 (0.3, 1)
 (0.2, 1)
 (0.1, 1)
 (0.0, 1)
```

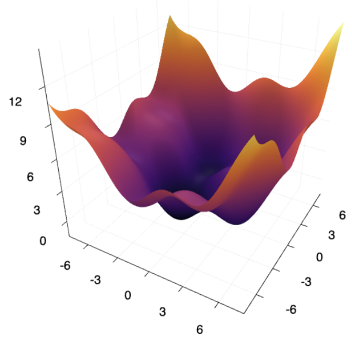
(c) If we have learning rate larger than or equal to 1, the sequence of iterates will diverge. For learning rate larger than 1, $|x - 2 * r * x| > |x|$, as a result, $f(x) = x^2$ will be larger since absolute value of x gets larger and the sequence diverges. For the learning rate equal to 1, $|x - 2 * r * x| = |x|$, as a result, x will jump between 4 and -4, which is also diverge. For learning rate smaller than 1, $|x - 2 * r * x| < |x|$, as a result, $f(x) = x^2$ will be smaller since absolute value of x gets smaller and the sequence converges.

Problem 2

Define function $f, \mathbb{R}^3 \rightarrow \mathbb{R}^1$ as:

$$f(x, y) = \sin(x) + \sin(y) + \frac{x^2 + y^2}{10}$$

(a) Use the code below to graph f. How many local minima do you see? Just from looking at the plot, where is the global minima (just a very rough estimate)?



In [39]:
```
using Plots
plotlyjs()
f(x,y) = sin(x) + sin(y) + (x^2 + y^2)/10
surface(-8:0.05:8, -8:0.05:8, f)
```

Out[39]:

(solution a) From the graph, I see 4 local minimum. The global minima is roughly at point (-1.5,-1.5)

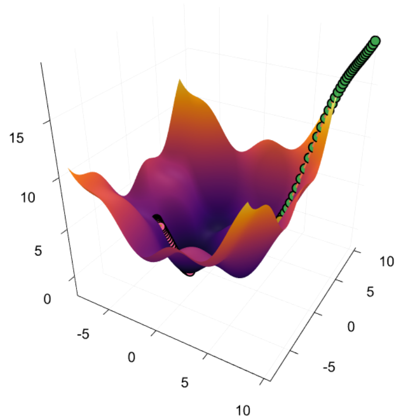
(b) Implement a vanilla gradient descent method with a tunable learning rate ϵ. Run that method on this surface with multiple starting points as well as learning rates. Plot the trajectories of your gradient descent on the surface. Do all of your trajectories end up in the global minima?

Here's some code showing how to add paths and points to the plot (the actual points having nothing to do with the problem).

In [2]:

```
using Plots
using ForwardDiff: gradient
plotlyjs(legend = false)
f(x,y) = sin(x) + sin(y) + (x^2 + y^2)/10
derivative(f, x) = gradient(x->f(x[1], x[2]), x)
surface(-8:0.05:8, -8:0.05:8, f)
function graddescent(f, x₀, ϵ, threshold)
    x = x₀
    df(x) = derivative(f, x)
    points = Tuple{Float64, Float64, Float64}[]
    while abs(df(x)[1]) > threshold || abs(df(x)[2]) > threshold
        point = (x[1],x[2], f(x[1],x[2]))
        push!(points, point)
        x = x - ϵ*df(x)
    end
    points
end
start = [[10.0,10.0], [6.0,2.0], [-5.0,-1.0]]
points = []
for s in start
    points = graddescent(f, s, 0.1,1e-12)
    # points = [(0, 0, 5), (0, 0, 10), (1, 1, 12)]
    path3d!(points, linewidth = 5)
    scatter3d!(points, markersize = 2)
end
path3d!(points, linewidth = 5)
```

Out[2]:



No. Because some points located near local minimas which are not global minimum.

(solution b) No, some will end at the local minima near them and points near global minima will end ad global minima.

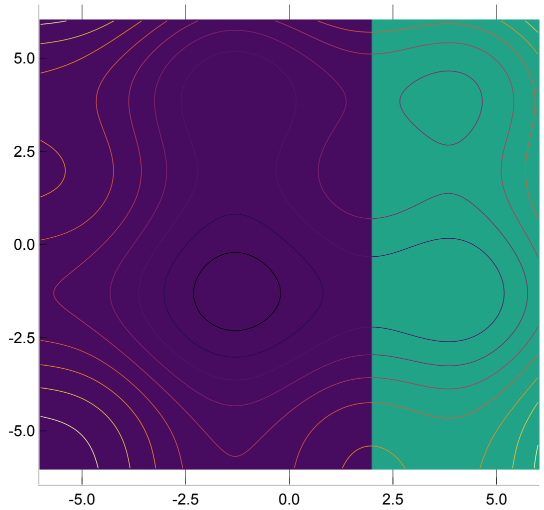
(c) Make a two-dimensional plot which colors each point according to which local minimum the gradient descent algorithm started at that point settles in. Try doing this for a few different learning rates, and comment on any interesting observations you make.

Hint: You don't have to do this entirely from scratch; it's done for a different function (though just for a single learning rate) in the Day-8 pre-class video.

In [3]:

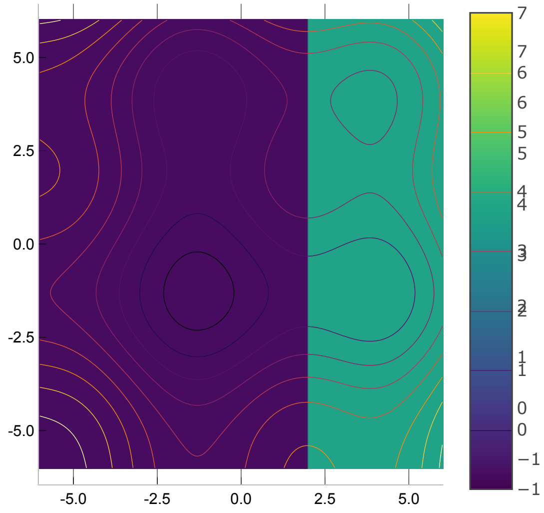
```
using Plots, SymPy
using LinearAlgebra
f(x,y) = sin(x) + sin(y) + (x^2 + y^2)/10
function graddescent_2(f, x₀, ϵ, threshold)
    df(x) = gradient(f,x)
    x = x₀
    while norm(df(x)) > threshold
        x = x - ϵ*df(x)
    end
    x
end
heatmap(-6:0.05:6, -6:0.05:6, (a,b) -> graddescent_2(v->f(v[1],v[2]), [a,b], 0.005, 1e-12)[1], ratio = 1, size = (400,400), fillcolor = :viridis);
# scatter!(Tuple.(graddescent_2(f, [1.2,-0.1], 0.25, 1e-12)), label = "trajectory", color="green", msw = 0, ms = 2)
contour!(-6:0.05:6, -6:0.05:6, f, colorbar = false)
```

Out[3]:



In [4]:
```
heatmap(-6:0.05:6, -6:0.05:6, (a,b) -> graddescent_2(v->f(v[1],v[2]), [a,b], 0.1, 1e-12)[1], ratio = 1, size = (400,400), fillcolor = :viridis);
# scatter!(Tuple.(graddescent_2(f, [1.2,-0.1], 0.25, 1e-12)), label = "trajectory", color="green", msw = 0, ms = 2)
contour!(-6:0.05:6, -6:0.05:6, f, colorbar = true)
```

Out[4]:



Different learning rates will cause gradient descent algorithm to go to different local minimum. If the learning rate is big, a point near a local minimum will have the change to jump over that local minimum and fall into another local minimum.

(d) Starting from a point pretty close to the global minimum, apply Newton's method (which we discussed in class), replacing $-\epsilon I$ in the gradient descent formula with $-H^{-1}$. Investigate whether this method converges faster.

You can compute the Hessian either using automatic differentiation or symbolically, as you prefer.

In [59]:
```julia
using ForwardDiff: hessian
f(x,y) = sin(x) + sin(y) + (x^2 + y^2)/10
derivative(f, x) = gradient(x->f(x[1], x[2]), x)
hes(f,x) = hessian(x->f(x[1],x[2]), x)
function graddescent(f, x₀, ϵ, threshold)
    x = x₀
    df(x) = derivative(f, x)
    step = 0
    points = Tuple{Float64, Float64, Float64}[]
    while norm(df(x)) > threshold
        step += 1
        point = (x[1],x[2], f(x[1],x[2]))
        push!(points, point)
        x = x - ϵ*df(x)
    end
    return step
end
function newton(f, x, lr, threshold)
    df(x) = derivative(f,x)
    H(x) = hes(f,x)
    step = 0
    while norm(df(x)) > threshold
        x = x - lr* H(x) \ df(x)
        step += 1
    end
    println(x)
    return step
end
x = [-1,-1]
@time graddescent(f, x, 0.01, 1e-12)
@time newton(f, x, 1, 1e-12)
```

```
  0.234253 seconds (611.68 k allocations: 32.785 MiB, 3.22% gc time)
[-1.306440008369511, -1.306440008369511]
  0.743219 seconds (1.75 M allocations: 91.177 MiB, 2.05% gc time)
```

Out[59]: 4

I find out that gradient descent is faster than the Newton's Method. Because Newton's method requires the computaion of Hessian mastrix which requires lots of time.

Problem 3

Consider a 100-person queue at a theatre, with each person in the queue having their own designated seat in a theatre which has 5 rows and 25 seats per row (so there will be 25 empty seats).

(a) If all of them were to be seated randomly (each choice being made uniformly at random from the empty seats, independently from previous choices), what is the expected number of people who end up in their designated seats?

(b) Initially, there are $17, 19, 21, 23, 20$ seats assigned in each row, respectively. If every person were to be seated randomly, what is the expected number of people who end up in their originally designated row?

(c) The first person to enter lost his seat number and decides to choose their seat randomly. Every other person who enters follows the rules below:

- If their designated seat is not already taken, they will take that seat.
- If their designated seat is already taken, they will choose a seat randomly.

Use Julia to sample 100,000 instances of this experiment and thereby approximate (i) the probability that the last person in line will end up in his/her own seat, and (ii) the number of people who end up in their own seat.

It might also be interesting to look at the some sample plots of the number of people who get to take their own seat by the time the nth person sits, as a function of n (this part is optional).

(solution a) Since each choice is made independently from previous choices, every person has $\frac{1}{125}$ chance to randomly choose their designated seat. There are 100 people, the expected number of correct choices should be $100 * \frac{1}{125}$, which is $\frac{4}{5}$.

(solution b) There are 17 people who have $\frac{17}{100}$ chance to choose the right row randomly and 19 people with probability $\frac{19}{100}$ and so on. The expectation will be $(0.17 * 17 + 0.19 * 19 + 0.2 * 20 + 0.21 * 21 + 0.23 * 23) \div 1.25$, which is 16.16.

(solution c) We find out that the probability of the last person picks the right seat is 0.96236 and the expected number who end up in their own seat is 97.445

In [47]:

```
function choose_seat(seat_record, n)
    if n == 1
        seat_index = rand(1:125)
        return seat_index
    end
    if seat_record[n] == 0
        return n
    else
        existing_seat = [i for i = 1:length(seat_record) if seat_record[i] == 0]
        seat_index = rand(existing_seat)
        return seat_index
    end
end
function assign_all(seat_record, n=100)
    for i = 1:n
        seat_record[choose_seat(seat_record, i)] = i
    end
    return seat_record
end
function proportion(n = 100000)
    p1 = []
    p2 = []
    for i = 1:n
        seat_record = [0 for i = 1:125]
        record = assign_all(seat_record)
        push!(p1, record[100] == 100)
        push!(p2, sum([record[j] == j for j=1:100]))
    end
    res1 = sum(p1) / n
    res2 = sum(p2) / n
    return res1, res2
end
r1, r2 = proportion()
r1, r2
```

Out[47]:

```
(0.96304, 97.45838)
```

Problem 4

Suppose we sample two independent random variables, X_1 and X_2, from the uniform distribution $\mathrm{Uniform}(0, 1)$.

(a) Explain briefly why this method is equivalent to uniformly sampling a point from the unit square $\{(a, b) \mid 0 \le a \le 1, 0 \le b \le 1\}$, and assigning a to x_1 and b to x_2. You will need to use mathematical expressions where necessary.

(b) What is the expected value and variance of $|X_1 - X_2|$?

(c) Let $Y_1 = \min(X_1, X_2)$ and $Y_1 = \max(X_1, X_2)$. What is the expected value of Y_1. What's the expected value of Y_2 What is the difference between these two expected values?

(d) Suppose we are now sampling **three** independent numbers from $\text{Uniform}(0, 1)$. What is the expected value of the smallest number of the three? As an optional addition, generalize your reasoning from 3 to an arbitrary positive integer n.

(solution a) The probability of getting x assigned to X_1 and y assigned to X_2 from unit square is $P(X_1 <= x, X_2 <= y) = (x-0) * (y-0)$. $P(X_1 <= x)$ on uniform distribution is $(x-0)/(1-0)$ and $P(X_2 <= y)$ on uniform distribution is $(y-0)/(1-0)$. Since X_1, X_2 are independent, so their joint distribution is the product of $P(X_1 <= x)$ and $P(X_2 <= y)$, which is same as $(x-0)(y-0)$. Therefore, they are equivalent.

(solution b) Expected value is $\frac{1}{3}$ and variance is $\frac{1}{18}$ (β) $\quad E(x_1 - x_2 \;|) = \frac{\int_0^1 \int_0^1 |x_1 - x_2| dx_2 dx_1}{1}$

$$\begin{aligned} E(|x_1 - x_2|) &= \int_0^1 \int_0^{x_1} x_1 - x_2 dx dx_1 + \int_0^1 \int_{x_1}^1 x_2 - x_1 dx_2 dx_1 \\ &= \int_0^1 \frac{1}{2} x_1^2 dx_1 + \int_0^1 \frac{1}{2} - x_1 + \frac{1}{2} x_1^2 dx_1 \\ &= \frac{1}{3} \end{aligned}$$

$$\begin{aligned} |x_1 - x_2| &= x_1 - x_2 \quad \text{if} \quad x_1 > x_2 \\ &= x_2 - x_1 \quad \text{if} \quad x_1 < x_2 \end{aligned}$$

$$\begin{aligned} \text{Var}(|x_1 - x_2|) &= \int_0^1 \int_0^1 \left(|x_1 - x_2| - \frac{1}{3}\right)^2 dx_1 dx_1 \\ &= \int_0^1 \int_0^1 (x_1 - x_1)^2 - \frac{2}{3} \mid x_1 - \frac{x_2}{81} + \frac{1}{9} dx_2 dx_1 \\ &= \int_0^1 \int_0^1 (x_1 - x_2)^2 + \frac{1}{9} dx_2 dx_1 \\ &= \int_0^1 \int_0^1 x_1^2 - 2x_1 x_2 + x_2^2 dx_2 dx_1 \quad - \frac{1}{9} \\ &= \int_0^1 x^2 x_2 - x_1 x_2^2 + \frac{1}{3} x_2^3 \Big|_0^1 dx_1 - \frac{1}{9} \\ &= \frac{1}{3} - \frac{1}{2} + \frac{1}{3} - \frac{1}{9} \\ &= \frac{1}{18} \end{aligned}$$

(solution c) The expected value of Y_1 is $\frac{1}{3}$ and Y_2 is $\frac{2}{3}$. The difference is expected value of $|X_1 - X_2|$, which is $\frac{1}{3}$.

$$(c) \min(x, y) = x \; if \; x < y$$

$$\begin{aligned} &\int_0^1 \int_0^y x dx dy \\ &= \int_0^1 \frac{1}{2} y^2 dy \\ &= \frac{1}{6} \end{aligned}$$

$\min(x, y) = y$ if $x > y$ $\int_0^1 \int_y^1 y dx dy = \frac{1}{6}$

$\frac{1}{6} + \frac{1}{6} = \frac{1}{3} \quad E(\tau_1)$

$\max(x, y) = x$ if $x > y$ $\int_0^1 \int_y^1 x dx dy = \int_0^1 \frac{1}{2} - \frac{1}{2} y^2 dy = \frac{1}{2} - \frac{1}{6} = \frac{1}{3}$

$\max(x, y) = y$ if $y > x$ $\int_0^1 \int_x^1 y dy dx = \frac{1}{3}$

$\frac{1}{3} + \frac{1}{3} = \frac{2}{3}$

(solution d) The expected value of $\frac{1}{4}$.

$(d) \quad \min(x, y, z) = x$ if $x < y < 2 \int_0^1 \int_0^2 \int_0^y x dx dy dz = \int_0^1 \int_0^2 \frac{1}{2} y^2 dy dz = \int_0^1 \frac{1}{6} z^3 dz = \frac{1}{24}$

$E = \frac{1}{24} \times 2 \times 3 = \frac{1}{4}$

times 2 means the other case $x < y < 2$ and times 3 means case min = y and case min = z The generalized expected value is $\frac{1}{n+1}$

Problem 5

Consider a random independent sequence of letters, with each letter uniformly distributed in {a, b, . . . , z}. Use simulation to estimate the expected number of letters that appear in the sequence up to the first appearance of aa. Repeat with ab in place of aa. Based on your findings, is the expected time to the first aa different from the expected time to the first ab?

In [69]:

```julia
using Statistics
letters = ['a'+i for i=0:25]
function simulation(letters, signal)
    history = []
    push!(history, rand(letters))
    push!(history, rand(letters))
    cnt = 2
    while history[end-1:end] != signal
        letter = rand(letters)
        push!(history, letter)
        cnt += 1
    end
    return cnt
end
function proportion(n=10000)
    aa = []
    ab = []
    for i = 1:n
        push!(aa, simulation(letters, ['a','a']))
        push!(ab, simulation(letters, ['a','b']))
    end
    return aa, ab
end
aa, ab = proportion()
println("aa:", mean(aa))
println("ab:", mean(ab))
```

```
aa:716.5728
ab:669.6362
```

The first aa appear around the 695th letter and first ab appears around 674th letter. Based on my simulations, expected time to find first aa is more than the expected time from first ab

In [ ]:

```

```