Homework 5 math analysis

Tiberia-Giulia Farkas

November 2024

I want to apologize for the formatting of the attached document—it's my first time using LaTeX, and I'm still learning how to properly structure everything. I appreciate your understanding and any feedback you might have!

Let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable function for which we want to find a local minimum using the following gradient descent method:

$$x_{n+1} = x_n - \eta f'(x_n),$$

where $x_1 \in \mathbb{R}$ is a starting value and $\eta > 0$ is the step size (also called learning rate). Notice that:

- if $f'(x_n) > 0$, then $x_{n+1} < x_n$,
- if $f'(x_n) < 0$, then $x_{n+1} > x_n$,
- if $f'(x_n) = 0$, then $x_{n+1} = x_n$ and the algorithm would end.

If the sequence (x_n) has a limit x, then $f'(x) = 0$.

(a) Take a particular convex function f and show that for a small value of η, the method converges to the minimum of f.

(b) Show that by using a larger η, the method can converge faster (in fewer steps).

(c) Show that taking η too large might lead to the divergence of the method.

(d) Take a particular nonconvex function f and show that, depending on the starting value, the method can end up in a local minimum, not in the global minimum.

Example Functions

- Convex function: $f(x) = x^2$, $f'(x) = 2x$, $x_{n+1} = x_n - \eta \cdot 2x_n = x_n(1 - 2\eta)$.
- Nonconvex function: $f(x) = -2x^2$, $f'(x) = -4x$.

Gradient Descent Algorithm

```
#----------------------------------------------------
#Take a particular convex function f and show that for a
   small value of   the method
#converges to the minimum of f.
#----------------------------------------------------
def convex_function(x: int):
    return x ** 2

def derivative_function(x: int):
    return 2 * x

def nonconvex_function(x: int):
    return -2 * x ** 2

def derivate_nonconvex_function(x: int):
    return -4 * x


def gradient_descent(function, derivative_function, x0: int,
   iterations: int, learning_rate: float):
    x = x0
    x_points = []
    iter = 0
    while iter < iterations:
        x_new = x - learning_rate * derivative_function(x)
        if abs(function(x_new)) > 1e10:
            #print("Divergence!")
            break
        x = x_new
        x_points.append(x)
        iter += 1

    return x_points, iter

G = np.linspace(-2, 2, 200)
conv_G = convex_function(G)
plt.plot(G, conv_G, label=r'$f(x) = x^2$')

```

```python
x, iter = gradient_descent(convex_function, derivative_function, 2, 30, 0.1)
y = []
for i in x:
    y.append(convex_function(i))
print(y)

plt.title("For small n, the convex function f converges to the minimum of f \n n = 0.1, iterations = 30")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x, y, "*-", label = 'path of the gradient descent method')
plt.legend()
plt.show()

#-----------------------------------------------------
#Show that by using a larger eta, the method
#can converge faster (in fewer steps).
#-----------------------------------------------------

G = np.linspace(-10, 10, 200)
conv_G = convex_function(G)
plt.plot(G, conv_G, label=r'$f(x) = x^2$')

x, nr_iter = gradient_descent(convex_function, derivative_function, 10, 10, 0.8)
y = []
for i in x:
    y.append(convex_function(i))
print(y)

plt.title(f"Show that by using a larger   the method can converge faster (in fewer steps).\n n = 0.8, iterations = {nr_iter}")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x, y, "*-", label = 'path of the gradient descent method')
plt.plot(x, y, 'r-', 2)
plt.ylim([0, 60])
plt.legend()
plt.show()

#-----------------------------------------------------
#Show that taking   too large might lead to
#the divergence of the method.
#-----------------------------------------------------

```

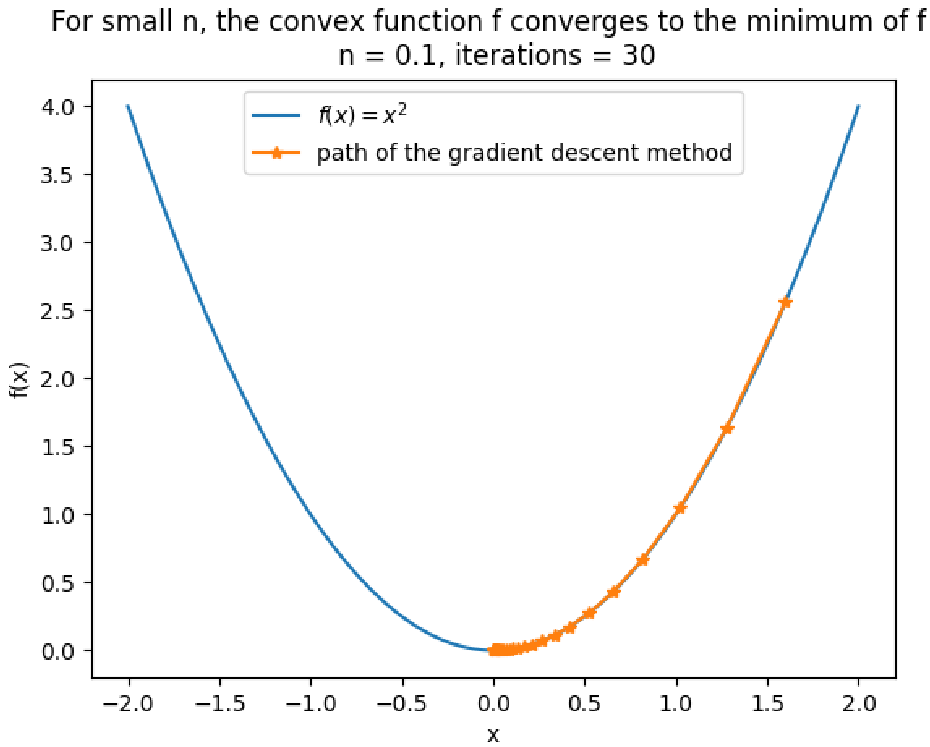
```
G = np.linspace(-20, 20, 200)
conv_G = convex_function(G)
plt.plot(G, conv_G, label=r'$f(x) = x^2$')  # Plot the function

# Gradient descent path with divergence
ns = [1.5, 2.0, 2.5]
for n in ns:
    x, nr_iter= gradient_descent(convex_function, derivative_function, x0=1, iterations=1000, learning_rate=n)
    if x[-1] > 1e10:
        print("The method diverged!")
    else:
        print(f"For learning rate = {n}, the method converged to the minimum of f: {x[-1]:.6f} in {nr_iter} iterations")
        plt.plot([convex_function(i) for i in x if convex_function(i) < 1e10], "*-", label=f"eta={n}")

plt.title("Show that taking   too large might lead to the divergence of the method")
plt.xlabel("Iterations")
plt.ylabel("f(x)")
plt.xlim([-10, 20])
plt.ylim([0, 20])
plt.legend()
plt.show()

#------------------------------------------------------------
#Take a particular nonconvex function f and show that,
#depending on the starting value,
#the method can end up in a local minimum,
#not in the global minimum.
#------------------------------------------------------------

x, iter = gradient_descent(nonconvex_function, derivate_nonconvex_function, 1, 50, 0.01)
print(f"For learning rate = 0.1, minimum of the nonconvex function is {x[-1]:.6f}")

X = np.linspace(-2, 2, 400)
Y = nonconvex_function(X)

plt.plot(X, Y, label = "$f(x) = -2*x^2$")
plt.plot(x, [nonconvex_function(i) for i in x], "*-", label = "path of gradient descent")
```

```
plt.title("The method can end up in a local minimum for a 
    nonconvex function")
plt.xlabel("x")
plt.ylabel("$f(x) = -2x^2$")
plt.xlim([-3, 8])
plt.legend()
plt.show()
```

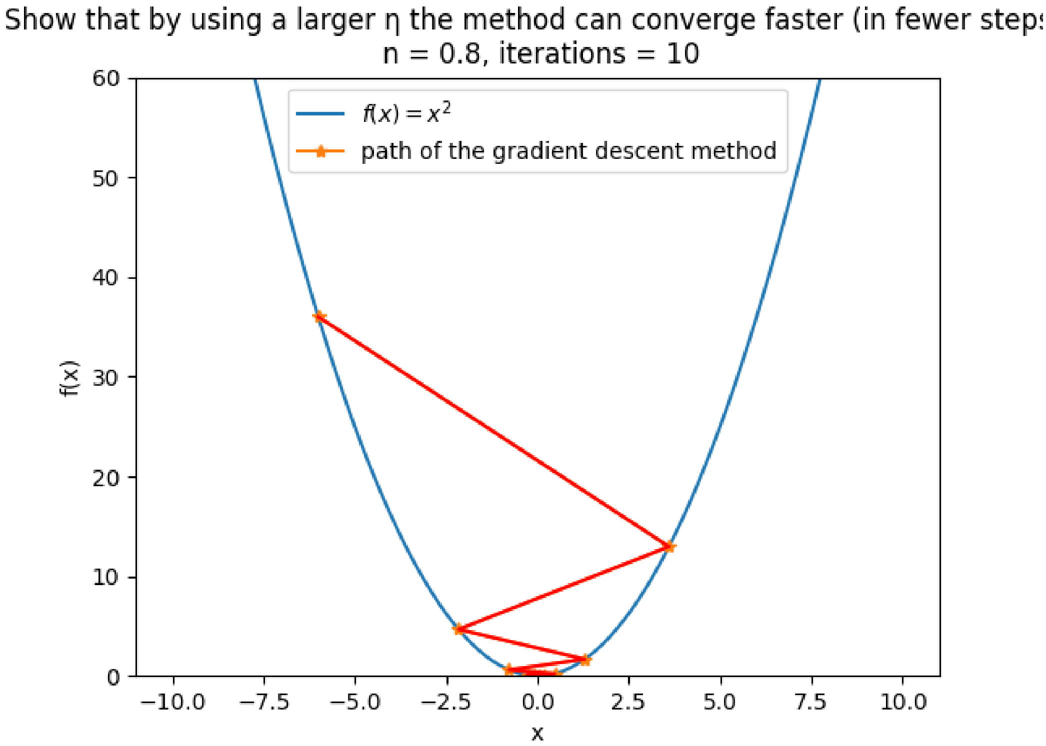
Listing 1: Gradient Descent Algorithm

Plots and Results

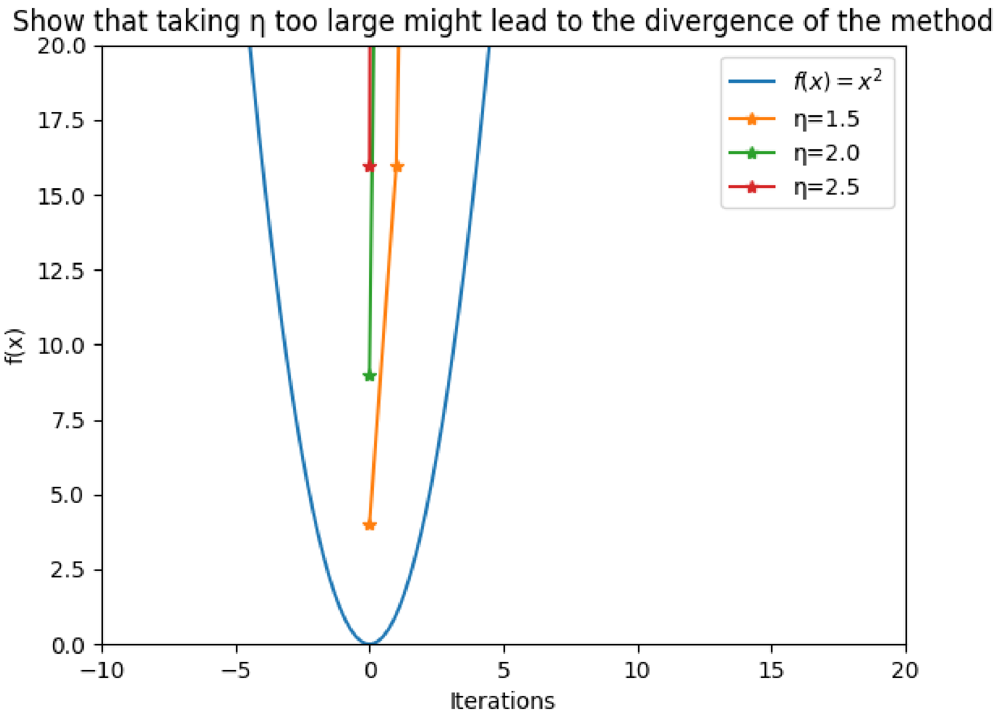
(a) For a small η, the convex function $f(x) = x^2$ converges to its minimum:



(b) Using a larger η, the convergence is faster:



(c) For a very large η, the method diverges:



(d) For a nonconvex function $f(x) = -2x^2$, depending on the starting value, the algorithm may end up in a local minimum: