Simple Constrained Optimization in 2D

Peter Amerkhanian

2023-07-04

Contents

import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D  
import numpy as np

The following are my notes on a basic calculus 1 homework question. I liked the question a lot, so decided to write out it all out for my future use.

# 1. The optimization problem

Question: **If 2000 square centimeters of material is available to make a box with a square base and an open top, what is the largest box that you can possibly create?**

The following are the core pieces of the question:  
1. *2000 square centimeters of material is available to create a box*

This means that the surface area of whatever we create will be 2000 cm^2

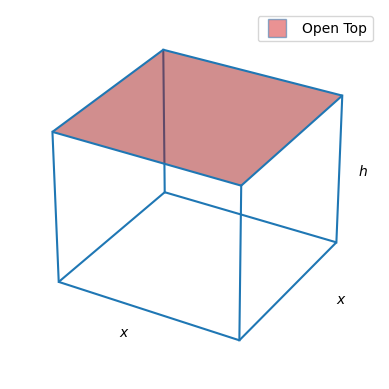
1. *The box has a square base*

This means that the we can equate base and width and just use one variable, , for all sides of the box except height,

1. *The box has an open top*

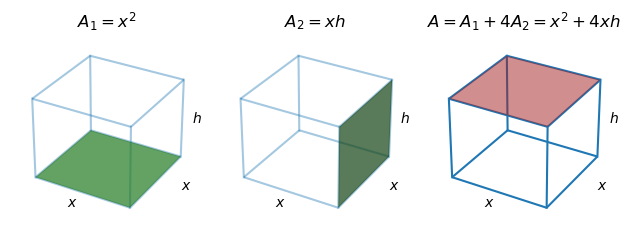
The resulting box is pictured below, with the red plane denoting the open box.

fig, ax = plt.subplots(subplot\_kw={"projection": "3d"})  
r = [-1,1]  
X, Y = np.meshgrid(r, r)  
one = np.ones(4).reshape(2, 2)  
  
h = 100  
  
ax.plot\_surface(X,Y, one\*h, alpha=.5, color='tab:red')  
ax.plot\_wireframe(X,Y,one\*h, alpha=1)  
ax.plot\_wireframe(X,Y,-one\*h, alpha=1)  
  
ax.plot(-np.array([1, 1]), np.array([1, 1]), np.array([-h, h]), c="tab:blue")  
ax.plot(np.array([1, 1]), -np.array([1, 1]), np.array([-h, h]), c="tab:blue")  
ax.plot(-np.array([1, 1]), -np.array([1, 1]), np.array([-h, h]), c="tab:blue")  
ax.plot(np.array([1, 1]), np.array([1, 1]), np.array([-h, h]), c="tab:blue")  
  
ax.plot([], [], "s", alpha=.5, markeredgecolor="tab:blue", markersize=13, c="tab:red", label="Open Top")  
ax.legend()  
  
ax.axis("off")  
  
ax.text(0, -1.5, -h, r"$x$")  
ax.text(1.5, 0, -h, r"$x$")  
ax.text(1.2, 1, 0, r"$h$");



This is a fairly straightforward constrained optimization function. We want to maximize the volume of an open box – (the fact that the box is open makes no difference for the volume) – but within the constraint that the surface area of the open box – – not exceed 2,000 . We establish what looks like in the case of an open box below.

fig, ax = plt.subplots(1, 3, subplot\_kw={"projection": "3d"})  
r = [-1,1]  
X, Y = np.meshgrid(r, r)  
one = np.ones(4).reshape(2, 2)  
  
  
def plot\_ax(ax, alpha=1):  
 ax.plot\_wireframe(X,Y,one\*h, alpha=alpha)  
 ax.plot\_wireframe(X,Y,-one\*h, alpha=alpha)  
  
 ax.plot(-np.array([1, 1]), np.array([1, 1]), np.array([-h, h]), c="tab:blue", alpha=alpha)  
 ax.plot(np.array([1, 1]), -np.array([1, 1]), np.array([-h, h]), c="tab:blue", alpha=alpha)  
 ax.plot(-np.array([1, 1]), -np.array([1, 1]), np.array([-h, h]), c="tab:blue", alpha=alpha)  
 ax.plot(np.array([1, 1]), np.array([1, 1]), np.array([-h, h]), c="tab:blue", alpha=alpha)  
 ax.axis("off")  
 ax.text(0, -1.5, -h, r"$x$")  
 ax.text(1.5, 0, -h, r"$x$")  
 ax.text(1.2, 1, 0, r"$h$")  
  
plot\_ax(ax[0], .4)  
ax[0].plot\_surface(X,Y, -one\*h, alpha=.7, color='tab:green')  
ax[0].set\_title(r"$A\_1=x^2$")  
  
plot\_ax(ax[1], .4)  
ax[1].plot\_surface(np.array([[1, 1],[1, 1]]),  
 np.array([[-1, -1], [1, 1]]),  
 np.array([[h, -h], [h, -h]]),  
 alpha=.7,  
 color='tab:green')  
ax[1].set\_title(r"$A\_2=xh$")  
  
  
plot\_ax(ax[2])  
ax[2].plot\_surface(X,Y, one\*h, alpha=.5, color='tab:red')  
ax[2].set\_title(r"$A=A\_1 + 4A\_2=x^2 + 4xh$")  
  
   
fig.tight\_layout()



We have two multivariate functions: , and , but to optimize, we will ideally just work with one, single variable function. We obtain that function via a two step process:  
1. Substitute our surface area constraint into and find :

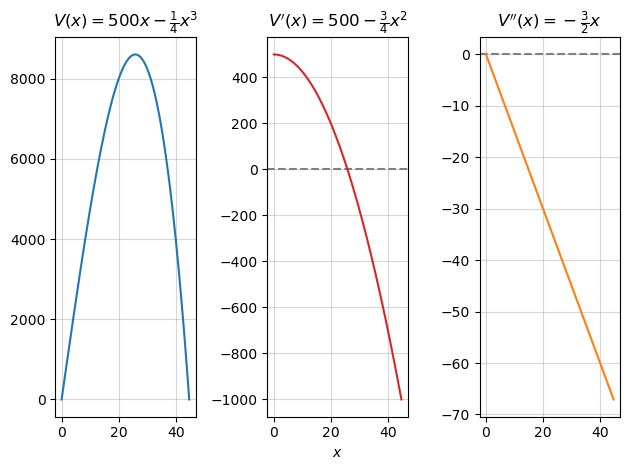
1. Substitute into to produce :

With this function in hand, we can find the value of that maximizes the volume – we’ll call this . Before we optimize, we’ll establish the domain of our search.  
We know that the range of is , and the domain of is obtained as follows:

Note that x must be positive, so

In other words, we are looking for the side length that will maximize the volume of our open box. We compute this objective function’s first and second derivatives, and , to find and analyze the optimal value. All resulting functions are plotted over the original function’s domain below.

V = lambda x: 500\*x - (x\*\*3)\*.25  
V\_1 = lambda x: 500 - (x\*\*2)\*.75  
V\_2 = lambda x: -1.5 \* x  
x = np.linspace(0, np.sqrt(2000), 1000)  
  
fig, ax = plt.subplots(1, 3, sharex=True)  
ax[0].plot(x, V(x))  
ax[0].grid(alpha=.5)  
ax[0].set\_title(r"$V(x) = 500x - \frac{1}{4}x^3$")  
  
ax[1].axhline(0, color="grey", linestyle="--")  
ax[1].plot(x, V\_1(x), c="tab:red")  
ax[1].grid(alpha=.5)  
ax[1].set\_xlabel(r"$x$")  
ax[1].set\_title(r"$V'(x) = 500 - \frac{3}{4}x^2$")  
  
ax[2].axhline(0, color="grey", linestyle="--")  
ax[2].plot(x, V\_2(x), c="tab:orange")  
ax[2].grid(alpha=.5)  
ax[2].set\_title(r"$V''(x) = -\frac{3}{2}x$")  
  
  
fig.tight\_layout();



To optimize the function and find the maximizing side length, we set to determine the at which the function has a slope of zero:

x\_critical = 20\*np.sqrt(15)/3  
x\_critical

25.81988897471611

V\_1(x\_critical)

0.0

We proceed to substitute this value into the second derivative to analyze this critical point and determine whether is a maximum volume or a minimum. Thus is:

V\_2(x\_critical)

-38.72983346207417

Given that , we can confirm that the second derivative is “concave down,” and thus represents a maximum value. **We conclude that the maximum possible volume of an open box with a surface area of 2,000 c^2 is**:

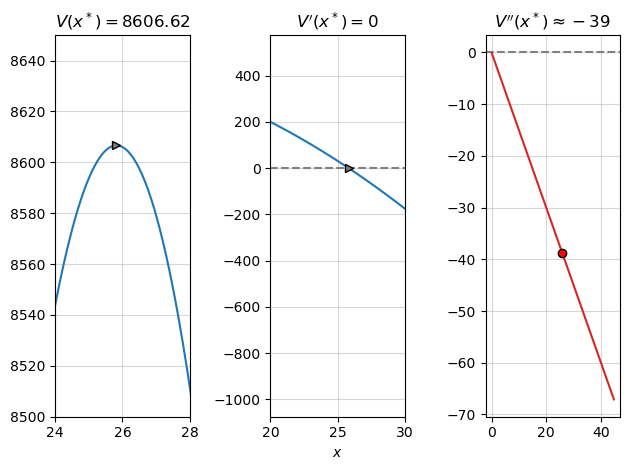
print(V(x\_critical), "cm^3")

8606.629658238704 cm^3

## 1.1 A note on the second derivative test

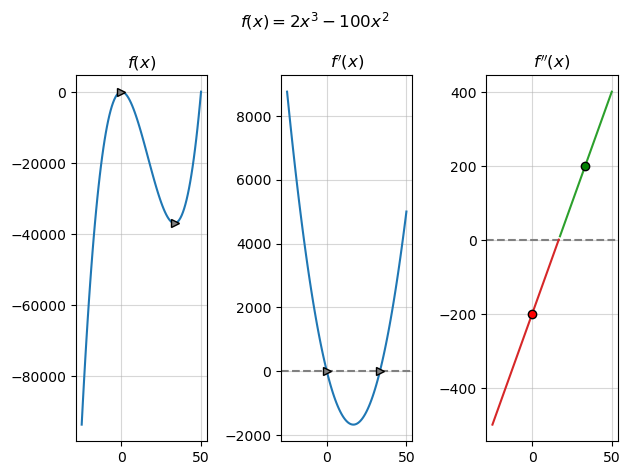
When we determined , we determined that coming off of this critical value, the slope of the function is decreasing – we are *coming down* from a maximum. In the figure below, the critical value is plotted as a black point. It’s clear that if we were to move slightly to the right of the critical value, the slope of the function would decrease, and we see this directly in the plots of the first and second derivatives.

fig, ax = plt.subplots(1, 3)  
ax[0].plot(x, V(x))  
ax[0].grid(alpha=.5)  
ax[0].set\_title(r"$V(x^\*) = 8606.62$")  
ax[0].set\_ylim(8500,8650)  
ax[0].set\_xlim(24, 28)  
ax[0].plot(x\_critical, V(x\_critical), ">", color="grey", markeredgecolor="black")  
  
ax[1].axhline(0, color="grey", linestyle="--")  
ax[1].plot(x, V\_1(x), c="tab:blue")  
ax[1].grid(alpha=.5)  
ax[1].set\_xlabel(r"$x$")  
ax[1].set\_title(r"$V'(x^\*) = 0$")  
ax[1].set\_xlim(20, 30)  
ax[1].plot(x\_critical, V\_1(x\_critical), ">", color="grey", markeredgecolor="black")  
  
ax[2].axhline(0, color="grey", linestyle="--")  
ax[2].plot(x[V\_2(x) > 0], V\_2(x)[V\_2(x) > 0], c="tab:green")  
ax[2].plot(x[V\_2(x) < 0], V\_2(x)[V\_2(x) < 0], c="tab:red")  
ax[2].grid(alpha=.5)  
ax[2].set\_title(r"$V''(x^\*) \approx -39$")  
ax[2].plot(x\_critical, V\_2(x\_critical), "o", alpha=1, color="red", markeredgecolor="black")  
  
fig.tight\_layout();



Expanding on this point, consider another arbitrary function that has more than one critical point, . In this case the function has one maximum and one minimum, so the second derivative test will be more important for analyzing each point. The function is plotted below, along with its first and second derivative.

f = lambda x: 2\*x\*\*3 - 100\*x\*\*2  
f\_1 = lambda x: 6\*x\*\*2 - 200\*x  
f\_2 = lambda x: 12\*x - 200  
  
x = np.linspace(-25, 50, 100)  
  
fig, ax = plt.subplots(1, 3)  
ax[0].plot(x, f(x))  
ax[0].grid(alpha=.5)  
ax[0].plot(100/3, f(100/3), ">", color="grey", markeredgecolor="black")  
ax[0].plot(0, f(0), ">", color="grey", markeredgecolor="black")  
  
ax[0].set\_title(r"$f(x)$")  
  
ax[1].axhline(0, color="grey", linestyle="--")  
ax[1].plot(x, f\_1(x), c="tab:blue")  
ax[1].grid(alpha=.5)  
ax[1].plot(100/3, f\_1(100/3), ">", color="grey", markeredgecolor="black")  
ax[1].plot(0, f\_1(0), ">", color="grey", markeredgecolor="black")  
  
ax[1].set\_title(r"$f'(x)$")  
  
ax[2].axhline(0, color="grey", linestyle="--")  
ax[2].plot(x[f\_2(x) > 0], f\_2(x)[f\_2(x) > 0], c="tab:green")  
ax[2].plot(x[f\_2(x) < 0], f\_2(x)[f\_2(x) < 0], c="tab:red")  
  
ax[2].grid(alpha=.5)  
ax[2].plot(100/3, f\_2(100/3), "o", color="green", markeredgecolor="black")  
ax[2].plot(0, f\_2(0), "o", color="red", markeredgecolor="black")  
  
ax[2].set\_title(r"$f''(x)$")  
  
fig.suptitle(r"$f(x) = 2x^3 - 100x^2$")  
  
fig.tight\_layout()



If we were to want to maximize this function by taking the first derivative and solving for 0, we would find two critical values – a maximum and a minimum. In this case, the second derivative test would be used to conclude which of these points is the maximum and which is the minimum. We substitute each critical value into the second derivative- denotes the maximum (the slope of the function is decreasing off of this point), while denotes the maximum (the slope of the function is increasing off of this point).