CS164 ASSIGNMENT 1: UNCONSTRAINED
& EQUALITY CONSTRAINED OPTIMIZATION
October 2020
Akmarzhan Abylay

Problem Description

Consider a roof drainage channel made with a single sheet of metal, bent in two places to create a trapezoidal cross-section. The channel is open on top and is required to carry the largest amount of water possible. As shown in the figure below, the left-hand edge of the channel is bent upwards at 90°, since it has to be bolted onto the wall just underneath the roof. We wish to compute the base length b, length of the right-hand side bent segment a, as well as the angle at which it is bent θ in order to maximize the cross-sectional area. Note that the height to which the left-hand end needs to be bent is determined from a and θ. We are given that the width of the metal sheet is $W = 3m$, which then allows b to be determined also.

Exercise 1

(a) **Question:** Compute an expression for the cross-sectional area of the channel A as a function of a and θ only. Produce both a surface plot and a contour plot of the function $A(a, \theta)$, over the domain of physically-realistic values for a and θ. You may find matplotlib useful to generate these plots (matplotlib.org/tutorials/),but you are welcome to use other plotting tools if you wish.

Answer: An expression for the cross-sectional area would include two pieces: one for the rectangle and one for the right-angled triangle. We can say that the area is equal to:

$$A = bh + \frac{acos(\theta)h}{2},$$

where h is the height of the rectangle. We also know that $h = asin(\theta)$, as well as that $a + b + h = 3$ from the instructions. We can recombine and get:

$$A = asin(\theta)(3 - a - asin(\theta) + \frac{acos(\theta)}{2})$$

You can see the plots below (see Appendix A for code). I took 0-3 to be a realistic value for a since the sum of a, b and h cannot be larger than 3, but also a length cannot be negative in our case. At the same time, I decided that 0 to π was a possible value for the θ since it cannot bend over 180° and it should be over 0° since otherwise it wouldn't represent roof drainage and will let the water spill. So, we have $0 < a < 3$ and $0 < \theta < \pi$.

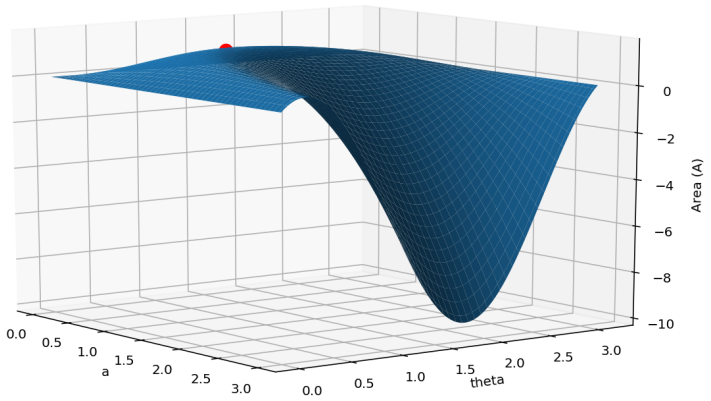

Figure 1: 3d Plot for the area function

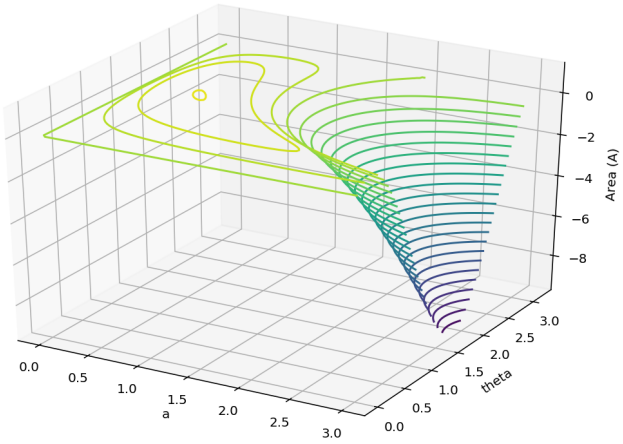

Figure 2: 3d Contour Plot for the area function

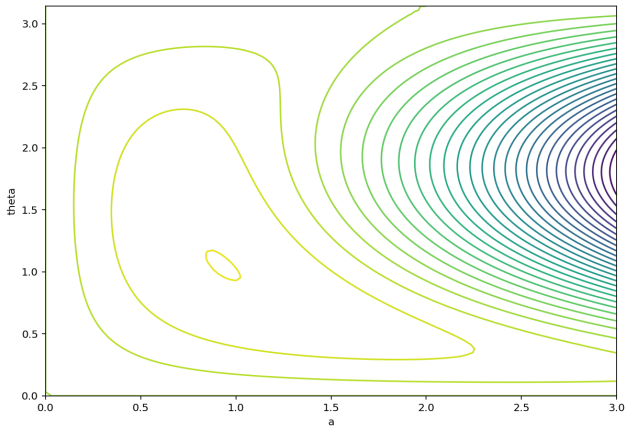

Figure 3: 2d Contour Plot for the area function

I made a simple 3d plot, a 3d contour plot, and an ordinary contour plot for this function. The red point in the first figure represents the maximum.

(b) **Question:** Estimate the values of a and θ that maximize the area from your plot. Does the maximum appear to be unique over the domain of physically-realistic values? Justify your answer.

Answer: To answer this question, we need to find the critical points of this function either analytically or graphically. By using a graph, we can say that there is a single maximum point on the domain, which is $(a, \theta) = (4\sqrt{3} - 6, \frac{\pi}{3})$, where the function evaluates to $9 - \frac{9\sqrt{3}}{2} \approx 1.21$. Analytically we can arrive at it through calculating the partial derivatives relative to a or θ and making them equal to 0. Below are the calculations. For the partial derivatives, we use the chain rule and get that:

$$\begin{aligned}
\frac{\partial A}{\partial a} &= sin(\theta)(3 - a - asin(\theta) + \frac{acos(\theta}{2}) + asin(\theta)(-1 - sin(\theta) + \frac{cos(\theta)}{2}) \\
&= sin(\theta)(3 - 2a - 2asin(\theta) + acos(\theta)) \\
\frac{\partial A}{\partial \theta} &= acos(\theta)(3 - a - asin(\theta) + \frac{acos(\theta)}{2}) + asin(\theta)(-acos(\theta) + \frac{asin(\theta)}{2}) \\
&= \frac{1}{2}(-asin^2(\theta) + acos^2(\theta) - 2cos(\theta)(2asin(\theta) + a - 3))
\end{aligned}$$

Now we equal both of these equations to 0 and solve since we have two unknowns and two equations. Fast-forward, we will get different critical points, where after plugging the values into the function itself, we find that the max is at $(a, \theta) = (4\sqrt{3} - 6, \frac{\pi}{3})$, where the function evaluates to $9 - \frac{9\sqrt{3}}{2} \approx 1.21$, the same thing as we see graphically.

This is a unique value over the domain of physically-realistic values after we calculate all the critical points and plug their values inside the original function. There was another point worth considering other than the actual maximum, which is $(2, \pi)$, since all the others either had an angle or the side of 0. However, it turned out to be 0, while the actual max was at around 1.21.

(c) **Question:** Determine whether or not the negated function $A(a, \theta)$ is coercive, justifying your answer.

Answer: A function is coercive if for any sequence $\{x_k\} \in int\ dom\ f$ tending to the boundary, it holds that the corresponding value sequence $\{f(x_k)\}$ tends to positive infinity. Simply saying, we should show that the limit of the module of $||\mathbf{x}||$ tends to positive infinity or:

$$\lim_{||\mathbf{x}|| \to +\infty} f(x) = +\infty$$

Intuitively, we know that reaching infinity for the negative area is equivalent to the actual area reaching negative infinity, which doesn't make sense because our area can only be non-negative. Since we only consider a domain of physically-realistic values, the negative area wouldn't tend to infinity, while the arguments tend to its bounds over the domain. Thus, the function is not coercive.

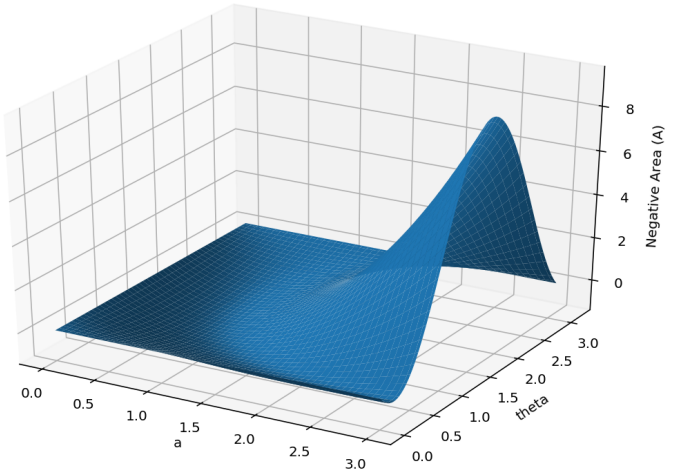

Figure 4: 3d Plot for the negative area (see Appendix B for code)

Similarly, we can look into the terms that make up the function. θ is used within the trigonometric terms, which are bounded by -1 and 1. They cannot contribute to the exploding terms. If we consider the direction of the term a, the function still is bounded by $0 < a < 3$, which means that the function isn't coercive as it doesn't tend to infinity as the magnitude of the arguments grows.

Exercise 2

(a) **Question:** Using multivariable calculus, find the exact values of a and θ that maximize the cross-sectional area of the channel. To accomplish this, consider b as a variable and use Lagrange multipliers on the area function $A(a, b, \theta)$, constraining the three variables by the width of the sheet - this will simplify calculations compared to maximizing $A(a, \theta)$ directly. Prove that the solution you found

maximizes the area by applying appropriate tests, assuming the domain is restricted to physically-realistic solutions.

Answer: I defined our main function and the constraint function, as well as the Lagrangian.

$$f(a,b,\theta) = a\sin\theta(b + \frac{acos\theta}{2})$$
$$g(a,b,\theta) = a + b + a\sin\theta = W = 3$$
$$\mathcal{L}(a,b,\theta,\lambda) = f(\mathbf{x}) - \lambda(g(\mathbf{x}) - W)$$

So we get:

$$\mathcal{L}(a,b,\theta,\lambda) = a\sin\theta(b + \frac{acos\theta}{2}) - \lambda(a + b + a\sin\theta - 3)$$

We set $\nabla\mathcal{L} = \mathbf{0}$, so that we get a system of linear equations like the following:

$$\nabla\mathcal{L} = \begin{bmatrix} \mathcal{L}_a(a,b,\theta,\lambda) \\ \mathcal{L}_b(a,b,\theta,\lambda) \\ \mathcal{L}_\theta(a,b,\theta,\lambda) \\ \mathcal{L}_\lambda(a,b,\theta,\lambda) \end{bmatrix} = \begin{bmatrix} a\sin\theta\cos\theta + bsin(\theta) - \lambda sin(\theta) - \lambda \\ a\sin\theta - \lambda \\ \frac{a}{2}(a\cos^2\theta - a\sin^2\theta - 2\lambda cos(\theta) + 2b\cos\theta) \\ 3 - a - b - a\sin\theta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Solving this will give us 5 solutions (all series), which are the same as in 1(c) but with an additional b and λ. From them, only one solution $(a,b,\theta,\lambda) = (4\sqrt{3} - 6, 3 - \sqrt{3}, \frac{\pi}{3}, 6 - 3\sqrt{3}) \approx (0.93, 1.27, 1.05, 0.80)$ fits our physically-realistic values. λ here also represents the sensitivity of the function value (A) at the optimal point to changes in the constraint function.

To confirm that this is the actual maximum point, we can use the bordered Hessian second-order derivative test. Our Hessian will look like the following:

$$H(\mathcal{L}) = \begin{bmatrix} 0 & g_a & g_b & g_\theta \\ g_a & f_{aa} & f_{ab} & f_{a\theta} \\ g_b & f_{ba} & f_{bb} & f_{b\theta} \\ g_\theta & f_{\theta a} & f_{\theta b} & f_{\theta\theta} \end{bmatrix}$$

From here we can calculate all of the values and find out the Hessian.

$$H(\mathcal{L}) = \begin{bmatrix} 0 & 1 + \sin\theta & 1 & a\cos\theta \\ 1 + \sin\theta & \sin\theta\cos\theta & \sin\theta & a(\cos^2\theta - \sin^2\theta) + bcos(\theta) \\ 1 & \sin\theta & 0 & a\cos\theta \\ a\cos\theta & a(\cos^2\theta - \sin^2\theta) + bcos(\theta) & a\cos\theta & -asin(\theta)(2acos(\theta) + b)) \end{bmatrix}$$

Now let's plug in the actual values.

$$H(\mathcal{L}^*) = \begin{bmatrix} 0 & 1 + \sin\frac{\pi}{3} & 1 & 2\sqrt{3} - 3 \\ 1 + \sin\frac{\pi}{3} & 0.5\sin\frac{\pi}{3} & \sin\frac{\pi}{3} & \frac{9 - 5\sqrt{3}}{2} \\ 1 & \sin\frac{\pi}{3} & 0 & 2\sqrt{3} - 3 \\ 2\sqrt{3} - 3 & \frac{9 - 5\sqrt{3}}{2} & 2\sqrt{3} - 3 & 45 - 27\sqrt{3} \end{bmatrix}$$

Now we will evaluate the signs of minors of the bordered Hessian, which are the determinants of the sub-matrices along the diagonal containing the top-left corner. We know that:

$$H_1^B = \begin{bmatrix} 0 & 1 + \sin\frac{\pi}{3} \\ 1 + \sin\frac{\pi}{3} & 0.5\sin\frac{\pi}{3} \end{bmatrix}$$
$$H_2^B = \begin{bmatrix} 0 & 1 + \sin\frac{\pi}{3} & 1 \\ 1 + \sin\frac{\pi}{3} & 0.5\sin\frac{\pi}{3} & \sin\frac{\pi}{3} \\ 1 & \sin\frac{\pi}{3} & 0 \end{bmatrix}$$

$$H_3^B = \begin{bmatrix} 0 & 1+\sin\frac{\pi}{3} & 1 & 2\sqrt{3}-3 \\ 1+\sin\frac{\pi}{3} & 0.5\sin\frac{\pi}{3} & \sin\frac{\pi}{3} & \frac{9-5\sqrt{3}}{2} \\ 1 & \sin\frac{\pi}{3} & 0 & 2\sqrt{3}-3 \\ 2\sqrt{3}-3 & \frac{9-5\sqrt{3}}{2} & 2\sqrt{3}-3 & 45-27\sqrt{3} \end{bmatrix}$$

Calculating the determinants, we will find that:

$$H_1^B = -3.481956,\ H_2^B = 2.798912,\ H_3^B = -4.939999$$

We can see that there are alternating signs, where $H_1^B < 0$, $H_2^B > 0$ and $H_3^B < 0$, which means that our point is a maximum on our specified domain.

Exercise 3

(a) **Question:** Considering once again the function $A(a, \theta)$, implement gradient ascent with a line search (exact or backtracking) in Python and from the initial state (1.5, 0), verify that you reach the same optimal solution. Produce a table of steps showing how the algorithm converges, and plot the convergence over the contour plot of $A(a, \theta)$.

Answer: I adapted my code from the pre-class work to find the maximum of this area function using gradient ascent. I used the exact line search (i.e., bisection method) from the initial point (1.5, 0) and arrived at the same solution as I obtained analytically, which is around (0.9282, 1.0472), resulting in an area of 1.206. See Appendix C for the code.

I also plotted the convergence of the algorithm over the contour plot (see Figure 5; see Appendix D for code). The bright red point is the maximum and the black lines represent the convergence.

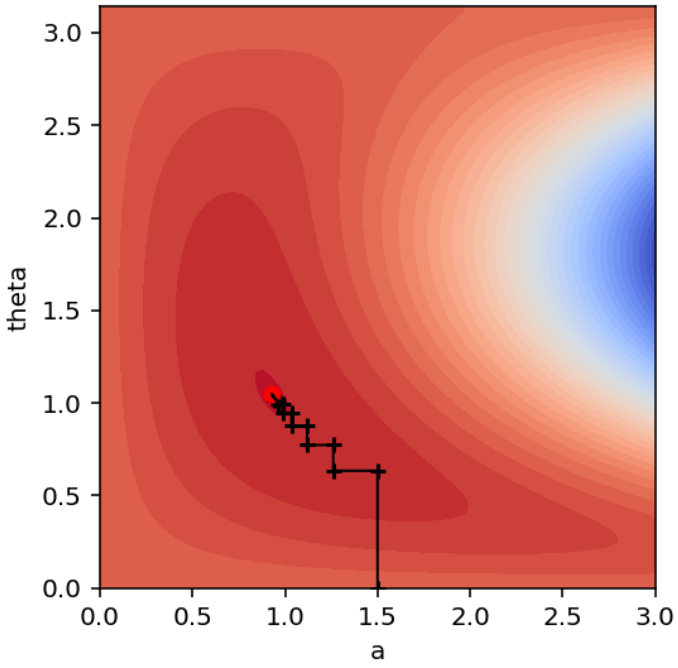

Figure 5: Convergence over the contour plot

I also isolated the important part to see the convergence better (see Appendix D for code).

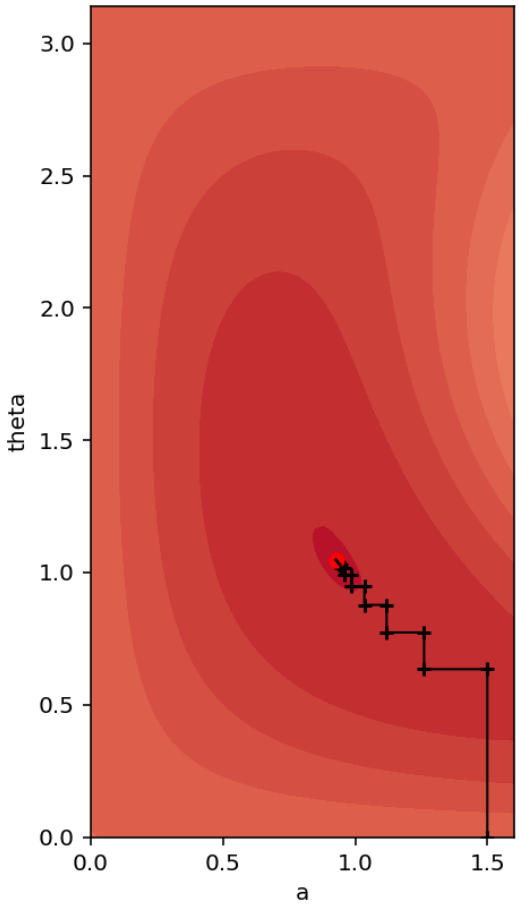

Figure 6: Convergence over the contour plot (smaller domain)

I am not going to include all the steps because there were over 80, but they looked like this:

Steps	
Step 0:	(1.5, 0.0)
Step 1:	(1.5, 0.63411827)
Step 2:	(1.26085324, 0.63411806)
Step 3:	(1.26085349, 0.77303092)
Step 4:	(1.11906063, 0.77303125)
Step 10:	(0.95929131, 0.99203181)
Step 20:	(0.92940148, 1.04514573)
Step 30:	(0.92821438, 1.04709038)
Step 40:	(0.92819085, 1.04718469)
Step 50:	(0.92819828, 1.04719406)
Step 60:	(0.92820148, 1.04719638)
Step 70:	(0.92820262, 1.04719715)
Step 80:	(0.92820302, 1.04719741)
Step 83*:	(0.92820338, 1.04719765)

Step 83 is where the algorithm stopped as the gradient became smaller than $1e-6$, which means that we converged to our maximum (optimal point).

Note: If you exclude the figures, then the report takes around 3.5 pages.

Appendix A

Plotting the 3d representation and the contour plot.

[1]:
```
#loading libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
%matplotlib inline
```

[2]:
```
#generating data in the realistic bound
a, theta = np.meshgrid(np.linspace(0,3,100), np.linspace(0, np.pi,100))

#function for the area
function = a*np.sin(theta)*(3-a-a*np.sin(theta)+(a*np.cos(theta)/2))

#plotting the figure in 3d
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(a, theta, function)
ax.set_ylabel("theta")
ax.set_xlabel("a")
ax.set_zlabel("Area (A)")

#max point
ax.scatter(4*np.sqrt(3)-6, np.pi/3, 9-(9*np.sqrt(3)/2), marker='o', color='red', s=100)
# ax.scatter(2, np.pi, 0, marker='o', color='red', s=100) #testing another critical␣
 ↪point

#plotting the 3d contour plot
fig1 = plt.figure()
ax1 = fig1.add_subplot(111, projection='3d')
ax1.contour3D(a, theta, function, 30)
ax1.set_ylabel("theta")
ax1.set_xlabel("a")
ax1.set_zlabel("Area (A)")

#plotting the contout plot
fig2 = plt.figure()
ax2 = fig2.add_subplot(111)
ax2.contour(a, theta, function, 30)
ax2.set_ylabel("theta")
ax2.set_xlabel("a")
```

Appendix B

Plotting the negative area.

[3]:
```
#plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
ax.plot_surface(a, theta, -function)
ax.set_ylabel("theta")
ax.set_xlabel("a")
ax.set_zlabel("Negative Area (A)")
```

Appendix C

Exact line search (i.e., bisection method) and contour plot.

[4]:
```
#loading libraries
import numpy as np
import matplotlib.pyplot as plt
from numdifftools import Hessian

#function and gradient
def function(x):
    return x[0]*np.sin(x[1])*(3-x[0]-x[0]*np.sin(x[1])+(x[0]*np.cos(x[1])/2))

def grad_f(x):
    return np.array([np.sin(x[1])*(3-2*x[0]*np.sin(x[1])+x[0]*np.cos(x[1])-2*x[0]),
                                (x[0]/2)*(-x[0]*np.sin(x[1])**2+x[0]*np.
 ↪cos(x[1])**2-2*np.cos(x[1])*(2*x[0]*np.sin(x[1])+x[0]-3))])

#gradient ascent for maximizing the output
def gradient_ascent(function, grad_f, x_0, n_steps, epsilon):
    x = x_0
    iterations = [x]

    for i in range(n_steps):
        gradient = grad_f(x)

        if np.abs(np.sum(gradient)) <= epsilon:
            break
        d = gradient #descent direction

        alpha = line_search(grad_f, x, d, n_steps, epsilon) #step size

        x = x+alpha*d
        iterations.append(x) #for the plot

    return iterations


#exact line search to identify the alpha
def line_search(grad_f, x, d,
                n_steps, epsilon):

    h_der = lambda alpha : -np.dot(grad_f(x+alpha*d).T,d)

    alpha_l = 0
    alpha_u = 1
```

```
    #pseudo-code from the reading
    for i in range(n_steps):
        if h_der(alpha_u) <= 0:
            alpha_u = 2*alpha_u

    for i in range(n_steps):
        alpha_t = (alpha_u+alpha_l)/2
        result = h_der(alpha_t)

        if np.abs(result) <= epsilon:
            break
        elif result > 0:
            alpha_u = alpha_t
        elif result < 0:
            alpha_l = alpha_t

    return alpha_t
```

[45]:
```
#starting point
x_0 = np.array([[1.5],[0]])
n_steps = 1000
epsilon = 1e-6

#finding the solution and printing all steps
iterations = gradient_ascent(function, grad_f, x_0,
                             n_steps, epsilon)
for i, step in enumerate(iterations): #I didn't print all of them because
    #there were too many steps
    if i%4==0:
        print("Step %d:" %i, (step[0][0], step[1][0]))

print("Optimal point after %d steps:" %(len(iterations)-1), (iterations[-1][0][0],␣
 ↪iterations[-1][1][0]))

print("Area at the optimal solution:", function([iterations[-1][0][0],␣
 ↪iterations[-1][1][0]]))
```

```
Step 0: (1.5, 0.0)
Step 4: (1.1190606313558384, 0.7730312545115311)
Step 8: (0.9863897932862294, 0.9480983300662003)
Step 12: (0.9445940420007354, 1.0174052410837708)
Step 16: (0.932710236068908, 1.038883520121397)
Step 20: (0.9294014833746413, 1.0451457342899888)
Step 24: (0.9284885213743078, 1.0465995785273647)
Step 28: (0.9282522630536445, 1.0470123046948294)
Step 32: (0.9281972551333068, 1.0471333031382613)
Step 36: (0.9281887834158559, 1.0471712536387225)
Step 40: (0.9281908536730932, 1.0471846900665451)
Step 44: (0.9281943168752361, 1.047190333129365)
Step 48: (0.9281971682676722, 1.0471931563890482)
Step 52: (0.9281991973614943, 1.047194764393905)
Step 56: (0.9282005712052589, 1.047195751033354)
```

```
Step 60: (0.9282014835553216, 1.0471963789308936)
Step 64: (0.9282020846605541, 1.0471967851724142)
Step 68: (0.9282024793989244, 1.0471970498975778)
Step 72: (0.928202738260759, 1.047197222934153)
Step 76: (0.9282029079186375, 1.0471973361861218)
Step 80: (0.9282030190850309, 1.0471974103500472)
Optimal point after 83 steps: (0.9282033840672292, 1.0471976537495975)
Area at the optimal solution: 1.2057713659399871
```

Appendix D

[28]:
```
#values of for the area
A = np.empty(a.shape)
for i in range(a.shape[0]):
    for j in range(a.shape[1]):
        A[i,j] = function(np.array([a[i,j], theta[i,j]]))

#plotting the contour plot
fig = plt.figure(figsize=(6,12))
ax1 = fig.add_subplot(111)

ax1.contourf(a,theta,A,40, cmap="coolwarm")
ax1.set_xlim(0,3)
ax1.set_ylim(0,np.pi)
ax1.set_aspect("equal")
ax1.set_ylabel("theta")
ax1.set_xlabel("a")

#plotting the movement of the gradient descent/ascent (for maximization)
a, theta = zip(*iterations)
ax1.scatter(a,theta,c="black",marker="+",linewidths=2)
ax1.plot(a,theta,c="black",linewidth=1)
ax1.scatter(a[-1],theta[-1],c="red")
```

[29]:
```
#plotting the contour plot
fig = plt.figure(figsize=(6,12))
ax1 = fig.add_subplot(111)

ax1.contourf(a,theta,A,40, cmap="coolwarm")
ax1.set_xlim(0,1.6)
ax1.set_ylim(0,np.pi)
ax1.set_aspect("equal")
ax1.set_ylabel("theta")
ax1.set_xlabel("a")

#plotting the movement of the gradient descent/ascent (for maximization)
a, theta = zip(*iterations)
ax1.scatter(a,theta,c="black",marker="+",linewidths=2)
ax1.plot(a,theta,c="black",linewidth=1)
ax1.scatter(a[-1],theta[-1],c="red")
```