Course Project Report: Advanced Math Analysis with Matlab

KeZheng Xiong
22920202204622

December 26, 2021

Abstract

The report is for the end-of-term project of Advanced Math Analysis with MATLAB fall 2021 course. All the source code is open-sourced on the Github repository `https://github.com/SmartPolarBear/matlab-math-analysis-csxmu-2021` under **GPLv3** license.

Contents

1 Problem 1

1.1 Problem Description

Given function $F(x, y) = 0.2x^2 + 0.1y^2 + sin(x + y)$, please work out its gradient. Based on the gradient, please find out the local extreme of function F(x,y) when both x and y are in the range of $[-2 * \pi, 2 * \pi]$. The 2D and 3D views of the function is given in Fig. 1.

1.2 Solution

1.2.1 The gradient of the function

I get the gradient of the function using the following code

```
syms x y;
f=0.2*x^2+0.1*y^2+sin(x+y);
diff(f,x)
diff(f,y)
```

Code 1: Gradient Calculation

Based on the result, the gradient is

$$\nabla \cdot f(x, y) = (\frac{2 * x}{5} + cos(x + y), \frac{y}{5} + cos(x + y)) \tag{1}$$

1.2.2 Find the extreme values

To find the extreme values of $F(x, y)$ with gradient decent method, we walk little steps towards the direction of the gradient. To formalize this idea, the algorithm is shown as follows.

Algorithm 1 Gradient Descent

Input: Initial point x_0, a constant α, $k = 0$

while termiation condition does not hold **do**

$k = k + 1$

$x_{k+1} = x_k - \alpha \nabla \cdot f(x_k)$

Various problems occurs if this brute-force algorithm is implemented directly. The speed of convergence is annoyingly slow if parameters are not chosen right. In fact, I never succeeded finding a set of parameters that works. A well-known optimization called Stochastic gradient descent, or SGD, improves the performance significantly.

In the brute-force algorithm, The given parameter α controls how long a step is from the point $\mathbf{x_0}$ towards the direction of the gradient. It is a fixed value. On the contrary, in SGD gradient descent algorithm, the length of a step , m , which is also referred to as "learning rate", will be changed each round according to the current position. To be more exact, SGD algorithm tries to find a learning rate m, so that it can minimize the function.

$$h(x, y, m) = \mathbf{x_0} + m \cdot \nabla \cdot f(x, y) \tag{2}$$

So, in the implementation of the algorithm, the following equation is solved each round in the while-loop to get the expected m value

$$\frac{\partial h}{\partial m} = 0 \tag{3}$$

The implementation is shown in Code 2. There are four results of the function. *endp* is the final point, or the result of gradient descent algorithm. *num* is the number of steps taken. *hist* is a vector which records all the coordinates of each step, and *list* contains the corresponding function values. The three parameters are the function, the initial point and the precision, respectively.

```
1
2     function [endp,num,hist,list]=gradient_descent(f,x0,eps)
3         syms x y m;
4         d=-[diff(f,x);diff(f,y)];
5
6         nd=subs(d,x,x0(1));
7         nd=subs(nd,y,x0(2));
8         nrm=double(norm(nd));
9
10        list=[];
11
12        k=0;
13        while(nrm>=eps)
14        nx0=x0+m*nd;
15        nf=subs(f,x,nx0(1));
16        nf=subs(nf,y,nx0(2));
17        h=diff(nf,m);
18        mm=solve(h);
19
20        x0=x0+mm*nd;
21        k=k+1;
22
23        hist(k,:)=[double(x0(1));double(x0(2))];
24        vf=subs(f,x,x0(1));
25        vf=subs(vf,y,x0(2));
26        list = [list double(eval(vf))];
27
28        nd=subs(d,x,x0(1));
29        nd=subs(nd,y,x0(2));
30        nrm=double(norm(nd));
31        end
32
33        num=k;
34        endp=x0;
35    end
```

Code 2: SGD Gradient Descent

To test the implementation, Each point of the steps taken in the procedure of gradient descent algorithm is plotted on Figure 1, in red scattered lines, and the results are in Table 1. In Table 1, the 4th column shows the value from the built-in $fminsearch$ function, which stresses that this implementation of SGD gradient descent algorithm gives the right results.

The initial points for gradient descent chosen are $(-3,-4)$, $(0,-1)$, and $(-2,-2)$. The choice of initial points are significant. It can **influence the speed of convergence dramatically**. Given a bad initial point, the algorithm may not reach the point of convergence at all, or take a unbearable long time to reach there. It worth mentioning that the second choice of $(0,-1)$ is especially difficult. It takes a great many attempts to find this point which contributes to a high speed of convergence.

Figure 1: The visualization of the steps of the SGD gradient descent algorithm

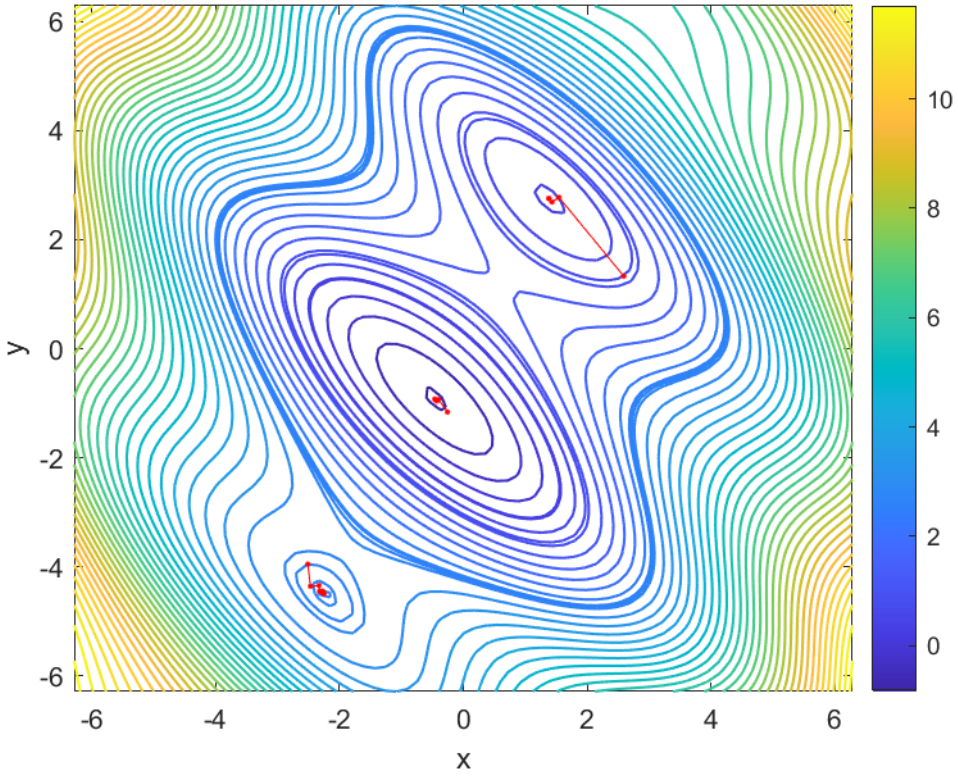

Table 1: The Extreme Values of $F(x, y)$

No.	X-Y Coordinates	$F(x, y)$	Result of $fminsearch()$
0	$(-2.2469, -4.4902)$	2.5874	2.5874
1	$(-0.4611, -0.9241)$	-0.8549	-0.8549
2	$(1.3768, 2.7529)$	0.3020	0.3020

1.3 Analysis and Conclusion

Gradient descent is a widely-used approach to finding the extreme values of any given function. There are many optimizations for this method, which will improve the speed of convergence. SGD is adopted in this project, which yields both right results and satisfying speed.

SGD gradient descent has a major flaw that it is known for "easy to be trapped in a local minimum", which may account for the difficulties in finding a good initial point for the second extreme value. More sophisticated optimization such as Momentum gradient descent and Ada-grad gradient descent can be applied if the speed of convergence is too slow for some more complex functions.

2 Problem 2

2.1 Problem Description

A factory supplies engine for its customer. According to the contract, the factory should deliver its product to the customer in the end of the 1st season: 40 engines; in the end of the 2nd season: 60 engines; in the end of the 3rd season: 80 engines. Due to limited productivity, to its most, the factory is only able to produce 100 machines in

a single season. The production cost is given as $f(x) = 50x + 0.2x^2$, where x is the number of machines produced in one season. It is possible to produce machines more than the required quota in each season. In this case, the storage cost is 4 dollars for one machine/season. Assuming there is no reservations at the beginning for the first season, please work out a production plan that minimizes the production cost

2.2 Solution

2.2.1 The Cost Function

Let x_1, x_2 and x_3 be the number of machines produced in season 1, 2 and 3, respectively.

Firstly, the production cost $P(\mathbf{x})$ is the sum of each season's production cost.

$$P(\mathbf{x}) = 0.2(x_1^2 + x_2^2 + x_3^2) + 50(x_1 + x_2 + x_3) \tag{4}$$

Secondly, there are costs for storage if more machines than the need is produced each season, so the storage cost $S(\mathbf{x})$ is the sum of each season's storage cost.

$$\begin{aligned} S(\mathbf{x}) = &\quad 4 * (x_1 + (x_1 + x_2 - 40) + (x_1 + x_2 + x_3 - (40 + 60))) \\ = &\quad 12x_1 + 8x_2 + 4x_3 - 560 \end{aligned} \tag{5}$$

The total cost, says $F(\mathbf{x})$, is the sum of $P(\mathbf{x})$ and $S(\mathbf{x})$

$$\begin{aligned} F(\mathbf{x}) = &\quad P(\mathbf{x}) + S(\mathbf{x}) \\ = &\quad 0.2(x_1^2 + x_2^2 + x_3^2) + 62x_1 + 58x_2 + 54x_3 - 560 \end{aligned} \tag{6}$$

According to the requirements for MATLAB to solve a quadratic programming problem, function $G(x)$ is defined as follows:

$$F(\mathbf{x}) = G(\mathbf{x}) - 560 \tag{7}$$

And it can be standardize:

$$\begin{aligned} G(\mathbf{x}) &= \frac{1}{2}\mathbf{x}\mathbf{H}\mathbf{x}^\mathbf{T} + \mathbf{f}^\mathbf{T}\mathbf{x} \\ H &= \begin{pmatrix} 0.4 & 0 & 0 \\ 0 & 0.4 & 0 \\ 0 & 0 & 0.4 \end{pmatrix} \\ f^T &= \begin{pmatrix} 62 & 58 & 54 \end{pmatrix} \end{aligned} \tag{8}$$

2.2.2 The Constraints

The most obvious constraints come from the max production limit.

$$\begin{aligned} 0 \le x_1 \le 100 \\ 0 \le x_2 \le 100 \\ 0 \le x_3 \le 100 \end{aligned} \tag{9}$$

Write it in the form for MATLAB to solve

$$\begin{aligned} \mathbf{lb} \le \mathbf{x} \le \mathbf{ub} \\ lb^T = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \\ ub^T = \begin{pmatrix} 100 & 100 & 100 \end{pmatrix} \end{aligned} \tag{10}$$

To meet the requirements of the contract, x_1 and x_2 should be enough to be delivered, and the total production $x_1 + x_2 + x_3$ should be just enough for the total demand $40 + 80 + 100$.

So we have

$$\begin{aligned} \mathbf{Ax} &\geq \mathbf{b} \\ \mathbf{A_{eq}x} &= \mathbf{b_{eq}} \end{aligned} \tag{11}$$

Where

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \\ b &= \begin{pmatrix} 60 \\ 100 \end{pmatrix} \end{aligned} \tag{12}$$

And

$$\begin{aligned} A_{eq} &= \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \\ b_{eq} &= \begin{pmatrix} 180 \end{pmatrix} \end{aligned} \tag{13}$$

2.2.3 Code

Now that we have translated the problem into equations and inequalities and have written them in the standard form for *quadprog*() in MATLAB to solve, the solution can be easily implemented in Code 3.

```
clear;

H=[0.4 0 0;0 0.4 0;0 0 0.4];
f=[62;58;54];

A=[-1 0 0;-1 -1 0];
b=[-60;-100];

Aeg=[1 1 1];
beg=[180];

ub=[100;100;100];
lb=[0;0;0];

[res,fval,exitflag,output,lambda] = quadprog(H,f,A,b,Aeg,beg,lb,ub);


x=sym('x',[1,3]);
f=0.2*(x(1)^2+x(2)^2+x(3)^2)+62*x(1)+58*x(2)+54*x(3)-560;
fv=subs(f,x,res');

res
double(fv)

```

Code 3: Solve using *quadprog*()

Which give the result that

$$F_{min}(\mathbf{x}) = 1.2030e + 04 = 12030 \tag{14}$$

When

$$\begin{aligned} x_1 &= 60 \\ x_2 &= 55 \\ x_3 &= 65 \end{aligned} \tag{15}$$

3 Problem 3

3.1 Problem Description

Prove: Among the top 32 World Cup finals, the information entropy of the event that a team winning is 5.

3.2 Proof

The definition of information entropy by Shannon is

$$H(X) = -\sum_{i} P(x_i) log_2(P(x_i)) \tag{16}$$

Let p_i be the possibility for team i to win, the Shannon entropy H is

$$H(X) = -\sum_{i=1}^{32} P(x_i) log_2(P(x_i)) \tag{17}$$

Subject to the nature of possibility, the following constraints should be satisfied.

$$P(x_i) \geq 0 \tag{18}$$

And

$$\sum_{i=0}^{32} P(x_i) = 1 \tag{19}$$

$H(X)$ is a multivariate function. Though it has too many variables to be plotted, it is possible to see how it scales by acquiring its maximums and minimums. To achieve this, Lagrangian multiplier method is used to implement Code 4.

```
clear;
% define 32 symbols
p=sym('p',[1,32]);

% construct f, the Shannon entropy. and sum, the sum of all posibilities, which
    should be 1, as a constraint
f=p(1)*log2(p(1));
sum=p(1);

for i=p(2:32)
f=f+i*log2(i);
sum=sum+i;
end

f=-f;

syms r

l=f+r*(sum-1);

% get the partial derivative for each symbol
eqns=[];
for i=p
eqns=[eqns diff(l,i)];
end

eqns=[eqns diff(l,r)];
vars=[p r];
```

```
28
29 solution=solve(eqns,vars);
30
31 ps=subs(p,solution);
32 fs=subs(f,solution);
33
34 % output the result
35 format rat;
36 ps
37 eval(fs)
```

Code 4: Get the maximum for $H(x)$

The result of Code 4 is that

$$H(x) = 5 \iff p_i = 0.03125, i = 1, 2, 3, ..., 32 \tag{20}$$

Therefore, the proposition is proved. Among the top 32 World Cup finals, the information entropy of the event that a team winning is 5 when the possibility for each team to win is equal. The entropy will be less than 5 otherwise.

4 Acknowledgment

Thanks to the open-source project **gitignore** (`https://github.com/github/gitignore`) to provide me with enough information to apply source code version control tools efficiently, especially *git*, to a project in MATLAB and LaTeX, which I never did before.