E4732 CaseStudy 3 name: Yunfei Yan UNI: yy2516

(1) Description of coding environment.

I use Mobaxterm to code this case study 3 @clic-lab.cs.columbia.edu in linux environment, my compiler is git and my editor is vim.

For constructing surface, I use the matlab.

I use the following libraries to do the calculation:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<iostream>
#include<complex>
#include<gsl/gsl_multimin.h>
#include<algorithm>
```

(2) My answers to the question:

A & B .Calibrate the parameters and Construct the surface:

Before present the answers, I want to clarify some points first:

1. Since we only calibrate the out of money options, which means we have only 50 data points to calibrate models, when I evaluate with different set of parameters caused from different start points, I use the sum of square error of all data points including the training data (out of the money option) and test data (in the money option), which means the following equation:

$$\sum_{i=i}^{100} \frac{1}{100} (S_i^{\text{market price}} - S_{\text{i}}^{\text{model price}})^2 .$$

2. Since we pursue the minimum test error, a good prediction for out of the money option doesn't mean it has good volatility surface especially for Heston model, at least in this special case. (I have tried hundred times to get this conclusion).

3. I also have tried a lot of penalty function:

For heston:

Set 1:

Since we require that volatility could not be negative which means that:

$$2\kappa\theta > \sigma^2$$

To improve the stability of calibration result, we can add the sum of square error from starting point to our objective function, which means:

$$\sum_{i=i}^{50} (S_i^{\text{starting point model price}} - S_{\text{i}}^{\text{model price}})^2$$

Also, the long term mean for volatility and rate of mean reversion could also be positive. According to these two conditions, we have:

$$\kappa > 0;$$

$$\theta > 0;$$

So we can add the penalty function like this to our returned objective function:

$$[\max(0, -\kappa) + \max(0, -\theta)] \times 10^8$$

Set 2:

Instead of the method above, we can also add this penalty function to our objective function:

$$[\max(0,-\kappa)+\max(0,-\theta)+\max(0,-(2\kappa\theta>\sigma^2))]\times 10^8$$

For VGSA:

The model requires negative skewness which means we have following requirement:

$\theta < 0;$

To realize this condition, we could have the following penalty function:

$\max(0,\theta)\times 10^8$

Comments about the penalty functions above:

After my hundreds of experiments, I found that when we add the penalty function, it sometimes lead our parameters to some boundary, and the test error is far large compared with the ones without penalty function. So after comparing with different penalty functions, I decide to use no penalty function. The compared result is listed in the next section.

4. According to the calibration, I found that the step length for simplex method sometimes have big impact on our result. After my research, I choose the step length vary in three values: 1.0, 2.0, 3.0.

5. some definitions for the result:

(1) let me name the dataset I used as the starting point for Heston and VGSA:

Heston	A	B	C	D
sigma	1.014300	1.420500	2.000000	0.583266
kappa	4.954900	4.540800	4.500000	3.224887
theta	0.056200	0.045400	0.015000	0.038943
rho	-0.655200	-0.736300	-1.250000	-0.869719
v0	0.057000	0.023500	0.015000	0.017816
VGSA	E	F	G	
sigma	0.066513	0.102200	0.082400	
theta	-0.054227	-0.076100	-0.044700	
kappa	6.273433	8.114300	8.784300	
eta	3.014308	2.806000	2.642900	
lambda	10.575303	10.364600	17.383100	
nu	0.237939	0.181900	0.248000	

(2) for heston, the condition is $2\kappa\theta > \sigma^2$*, for VGSA, the condition is* $\theta < 0;$

(3) penalty has the consistent meaning with the ones I mentioned above.

6. calibration result: (I choose the red line parameter set for each model)

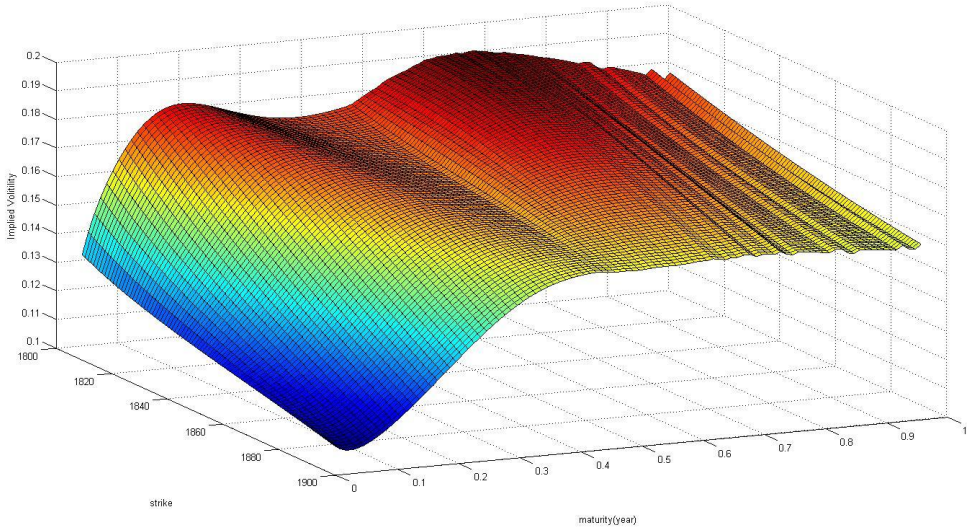
DAY 1

heston	spread									
converge	step	condition	test error	begin	penalty	sigma	kappa	theta	rho	v0
259	1.0	-0.24	1.7835	A	n	0.82399	7.0634	0.0311	-0.7071	0.0128
332	2.0	-0.456	1.8731	A	n	0.96505	7.6049	0.0312	-0.6515	0.0131
190	1.0	0.0472	1.6069	B	n	0.5426	5.4257	0.0315	-0.8745	0.0126
223	2.0	0.0313	1.6927	B	n	0.62563	7.0254	0.0301	-0.9107	0.0121
218	1.0	0.1397	1.448	D	n	0.32781	3.738	0.0331	-1.1324	0.0127

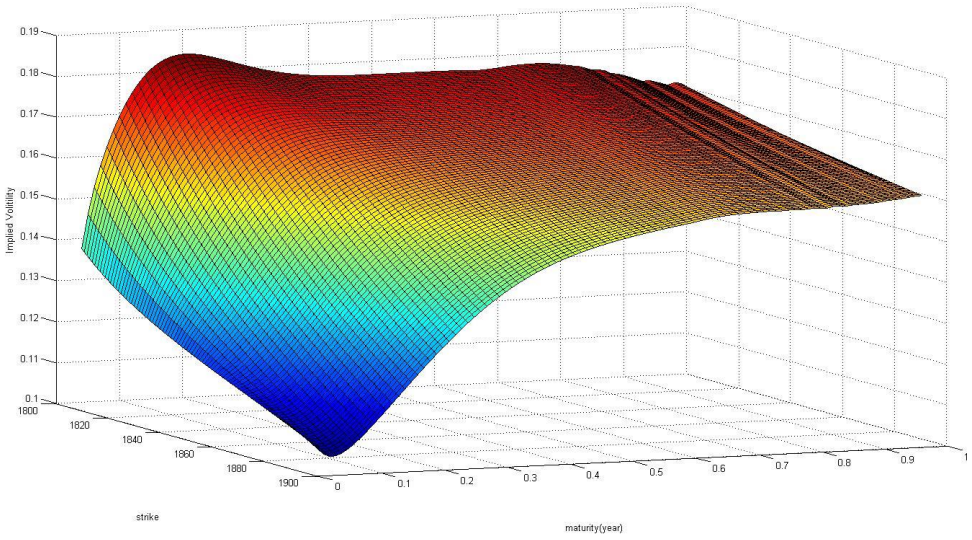
156	*2.0*	*0.1401*	*1.4548*	*D*	*n*	*0.32812*	*3.7516*	*0.033*	*-1.1315*	*0.0127*
254	*2.0*	*0.1242*	*1.4791*	*C*	*n*	*0.42164*	*4.7732*	*0.0316*	*-1.0166*	*0.0125*
419	*1.0*	*0.3461*	*11.34*	*A*	*y*	*-0.0367*	*6.7224*	*0.0258*	*-0.9245*	*0.0105*
213	*1.0*	*-0.526*	*2.2178*	*B*	*y*	*0.94759*	*5.0561*	*0.0368*	*-0.7507*	*0.0173*
270	*1.0*	*0.1384*	*1.4855*	*D*	*y*	*0.3523*	*4.0191*	*0.0327*	*-1.1071*	*0.0126*

From the result above, we found that performance of adding penalty function to objective function is bad, especially when we don't have a good starting dataset. If we want to use the distance from origin point, we must first locate a good starting point. Also, v0 seems stable no matter how we calibrate the model, which means we can set v0 at this quantity when we begin to calibrate the model. Rho and Sigma vary a lot during our calibration, we can see the surface of two of them later. The surfaces I plot are listed below:

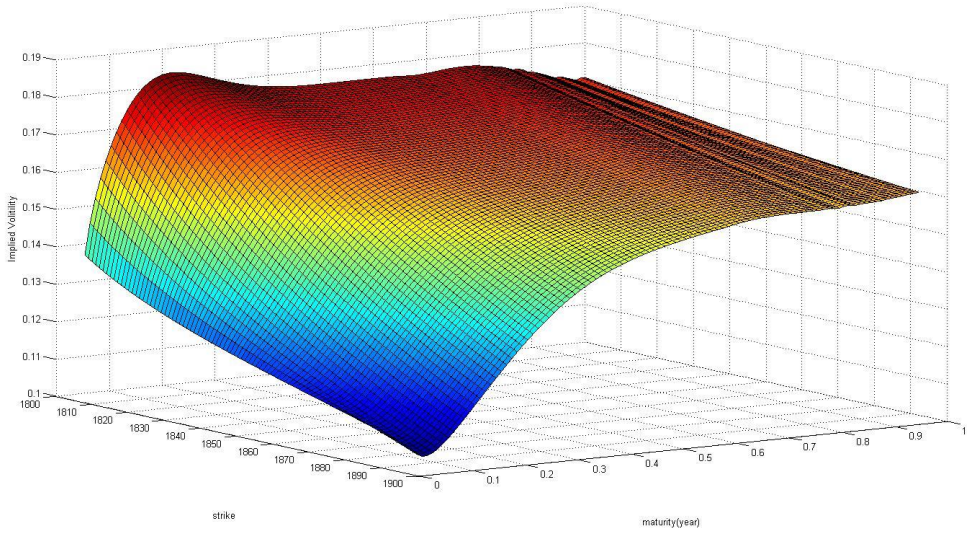
1. test error: 1.488



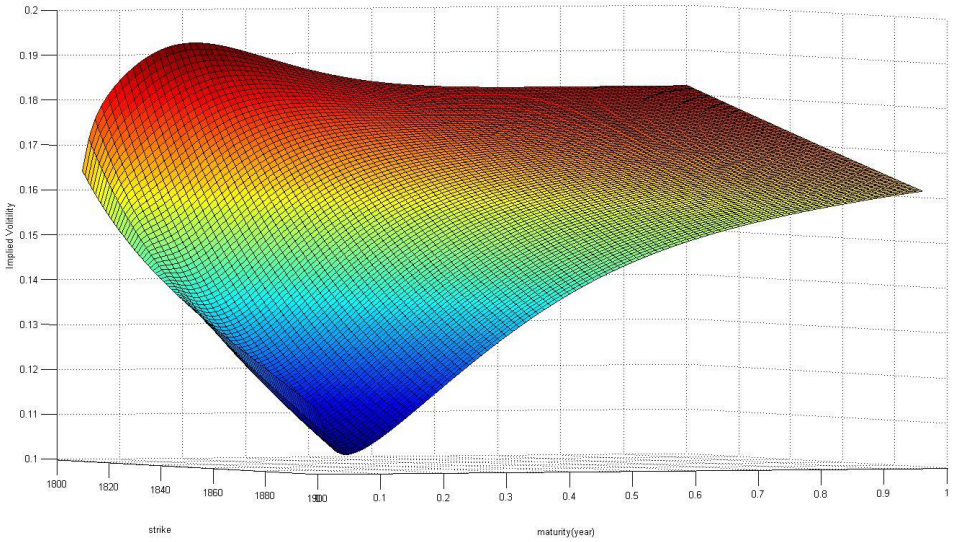
2. test error: 1.6096



3. test error: 1.8731



4. test error: 2.2178



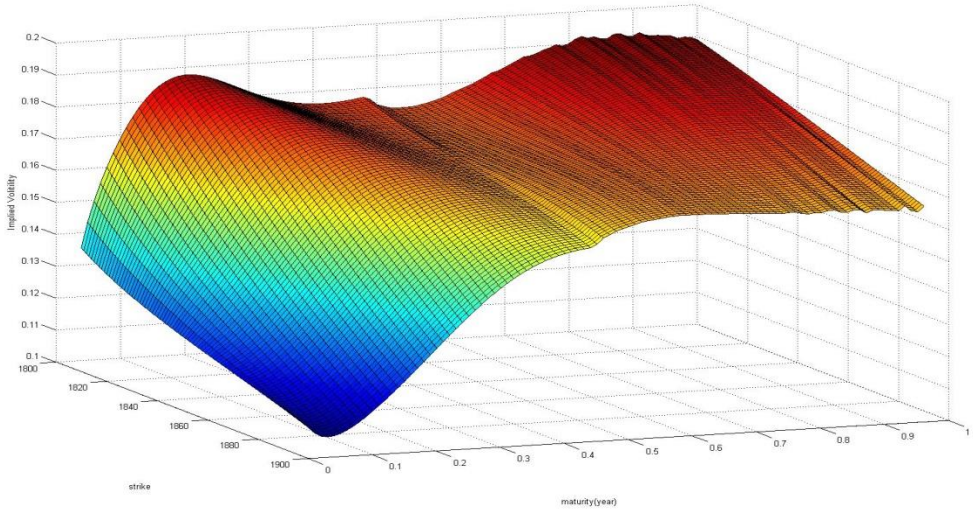
From the graphs above, we could discover that when T approaches the maximum, there exists some unstable phenomenon. The graph of test error 2.2178 has some implied smile, but the lowest test error one doesn't have this implied smile. However, besides the one with lowest test error, other model parameters make the vol decrease with the increase of t, which is not practical. All in all, I choose the lowest test error one as my calibrated model parameter.

heston	*equal*									
converge	*step*	*condition*	*test error*	*begin*	*penalty*	*sigma*	*kappa*	*theta*	*rho*	*v0*
198	*1.0*	*-0.123*	*1.5212*	*A*	*n*	*0.71568*	*6.131*	*0.0318*	*-0.7773*	*0.013*
290	*1.0*	*-1.856*	*2.263*	*B*	*n*	*1.57013*	*9.4038*	*0.0324*	*-0.5427*	*0.015*

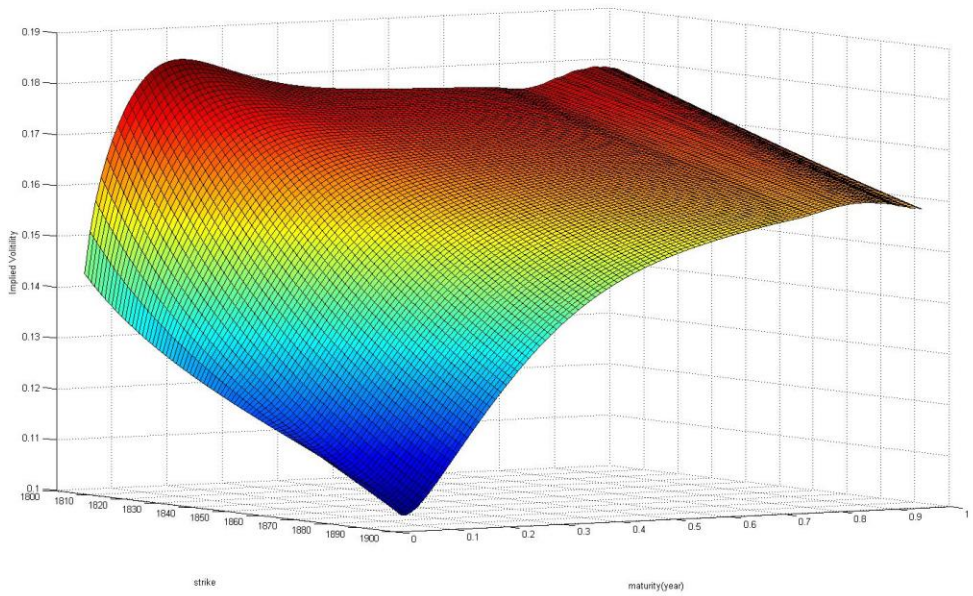
255	*1.0*	*0.1344*	*1.4184*	*D*	*n*	*0.32987*	*3.6451*	*0.0334*	*-1.129*	*0.0128*
589	*1.0*	*-0.089*	*225.67*	*A*	*y*	*0.60655*	*2.8971*	*0.0481*	*-0.7225*	*0.0308*
143	*1.0*	*-0.942*	*2.3759*	*B*	*y*	*1.16584*	*5.5419*	*0.0376*	*-0.7005*	*0.0184*
248	*1.0*	*0.1136*	*1.3318*	*D*	*y*	*0.37349*	*3.7824*	*0.0335*	*-1.0812*	*0.0129*
503	*2.0*	*0.1329*	*1.3904*	*D*	*y*	*0.32818*	*3.5898*	*0.0335*	*-1.1349*	*0.0128*

Again, theta and v0 are robust to our starting point, which means they play less role that the other parameters in this special situation. From the result above, we can see the best test error occurs when we add the penalty function. Both sigma, kappa and rho vary a lot. I plot some surfaces:

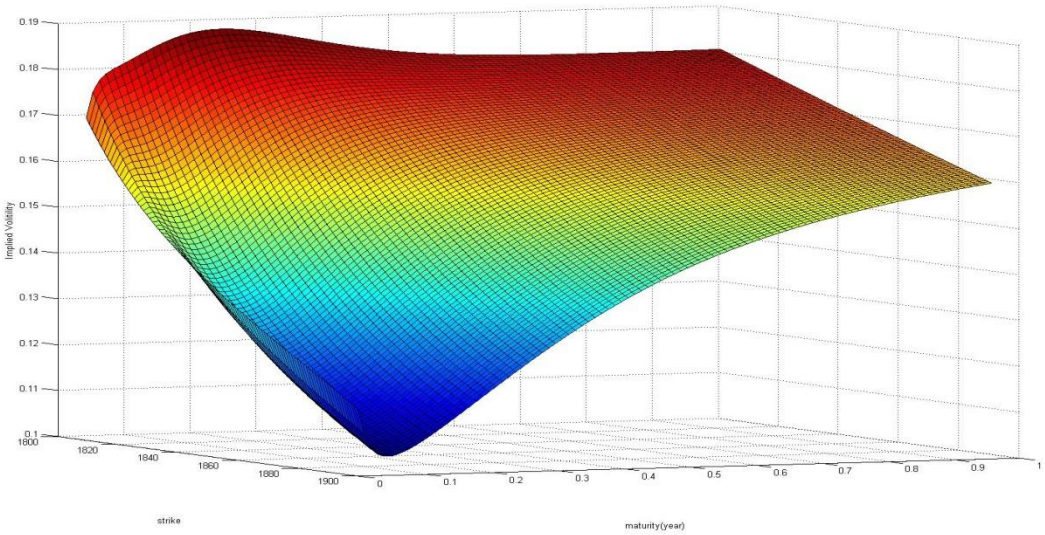
1. test error: 1.3318



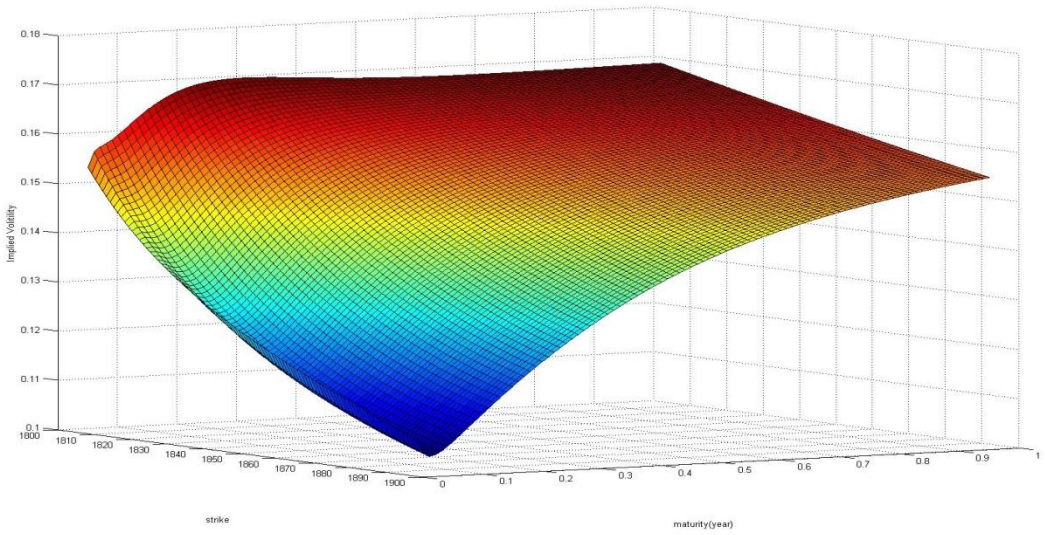
2. test error: 1.5212



3. test error: 2.3759



4. test error: 2.26



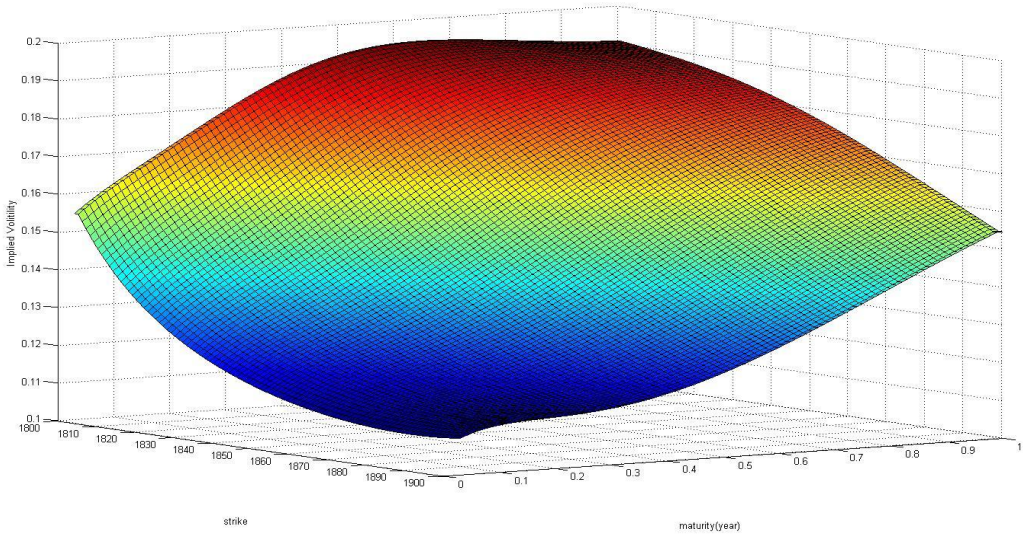
From the surfaces above, we could discover that all has some week volatility smile phenomenon. The one with lowest test error is not stable when the time approaches the maximum. But it is consistent with the market that the vol increases when the time to maturity increases. I still chose the lowest test error one as the solution to this situation.

vgsa	*spread*										
steps	*step*	*condition*	*test error*	*begin*	*penalty*	*sigma*	*theta*	*kappa*	*eta*	*lambda*	*nu*
932	*2.0*	*-0.057*	*1.6137*	*F*	*n*	*0.15302*	*-0.057*	*6.1935*	*1.1915*	*4.8376*	*0.73019*
545	*3.0*	*-0.041*	*1.5521*	*F*	*n*	*0.04742*	*-0.041*	*7.3005*	*5.21456*	*11.398*	*0.162241*
1434	*1.0*	*-0.044*	*1.3491*	*E*	*n*	*0.11217*	*-0.044*	*1.2629*	*2.45294*	*3.5605*	*0.54625*

1961	*3.0*	*-0.045*	*1.3271*	*E*	*n*	*0.1109*	*-0.045*	*1.0373*	*2.52337*	*3.2835*	*0.541652*
832	*1.0*	*-0.038*	*1.5523*	*G*	*n*	*0.04475*	*-0.038*	*7.3434*	*5.74525*	*12.073*	*0.144564*
921	*3.0*	*-0.036*	*1.8411*	*G*	*n*	*0.06568*	*-0.036*	*13.17*	*4.29826*	*19.363*	*0.264198*

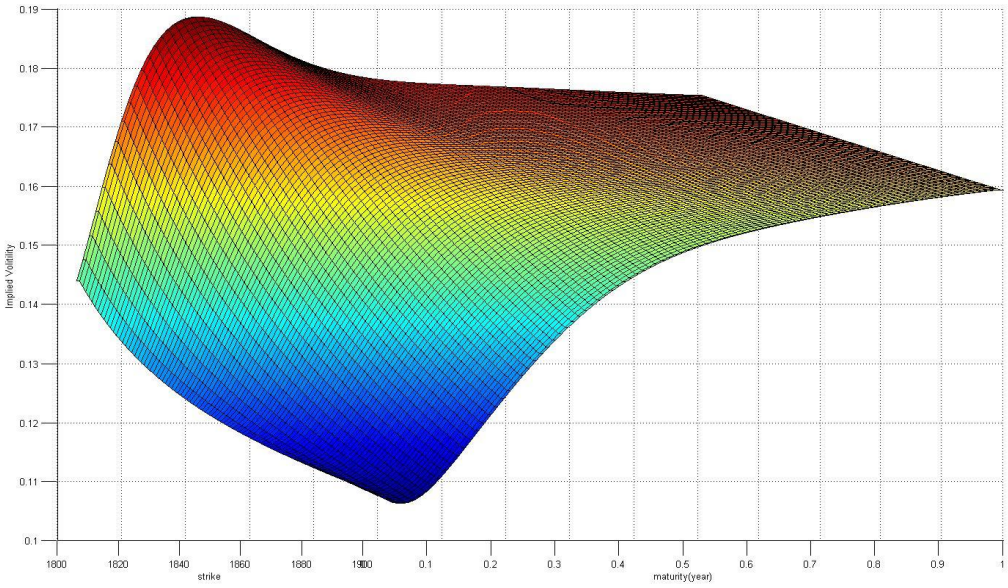
From the result above, besides theta, all the parameters vary a lot, which means the curve is very unsmooth, and the best parameter we find here takes 1961 steps to converge, the parameters in this situation may be unstable. The surfaces are here:

1. test error: 1.32

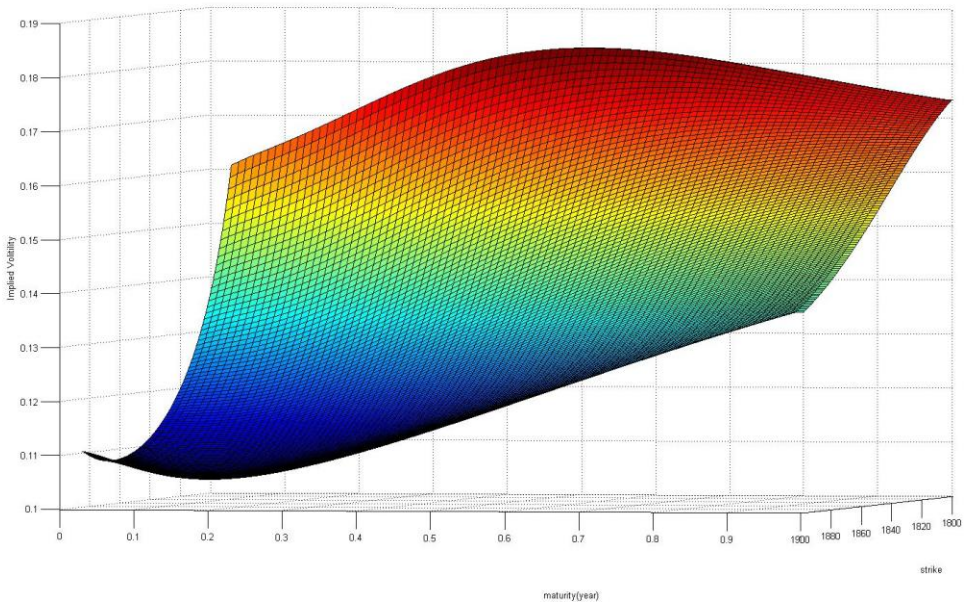


The surface is beautiful.

2. test error: 1.5521

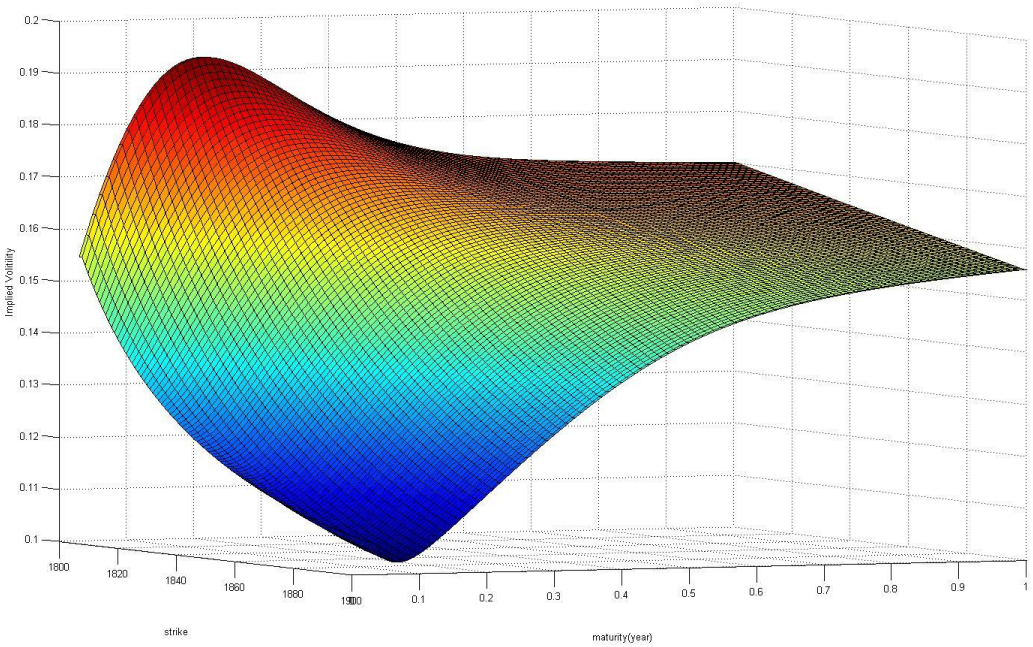


3. test error: 1.6137



At this time, this surface is perfect to me.

4. test error: 1.84

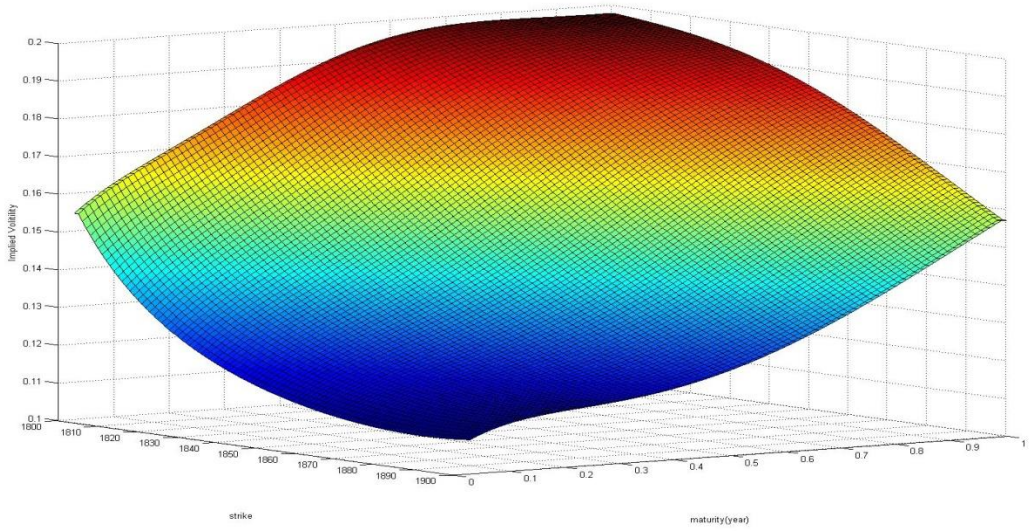


From the graphs above, we can see that surface from third is perfect, meanwhile it doesn't have the lowest test error point and the simplex method does not converge to this point when we use different start point set. This means that a model with good prediction of option price doesn't mean that it has beautiful surface. From these 4 graphs, we can see the surface can vary a lot when these parameters are very close.

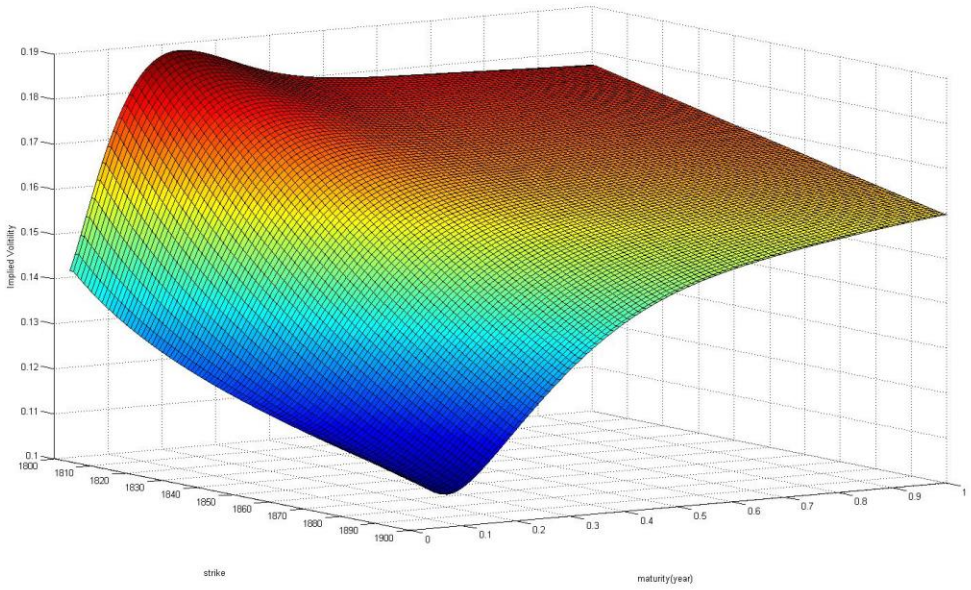
vgsa	equal										
steps	step	condition	test error	begin	penalty	sigma	theta	kappa	eta	lambda	nu
1134	2.0	-0.044	1.3899	F	n	0.11194	-0.044	0.3305	3.82536	2.7616	0.549932
804	3.0	-0.036	1.4542	F	n	0.04514	-0.036	6.7991	5.77862	12.006	0.143345
841	1.0	-0.041	1.368	E	n	0.1168	-0.041	1.3089	2.41441	3.7969	0.568694
948	2.0	-0.046	1.3491	E	n	0.11079	-0.046	0.4371	3.17916	2.7312	0.54477
640	2.0	-0.045	1.4273	G	n	0.05073	-0.045	5.7463	4.4795	9.3139	0.180433
940	3.0	-0.042	1.3692	G	n	0.11432	-0.042	0.941	2.66625	3.4323	0.557744

Sigma is comparatively stable in this equal method.

1. test error: 1.3491



2. test error: 1.4542



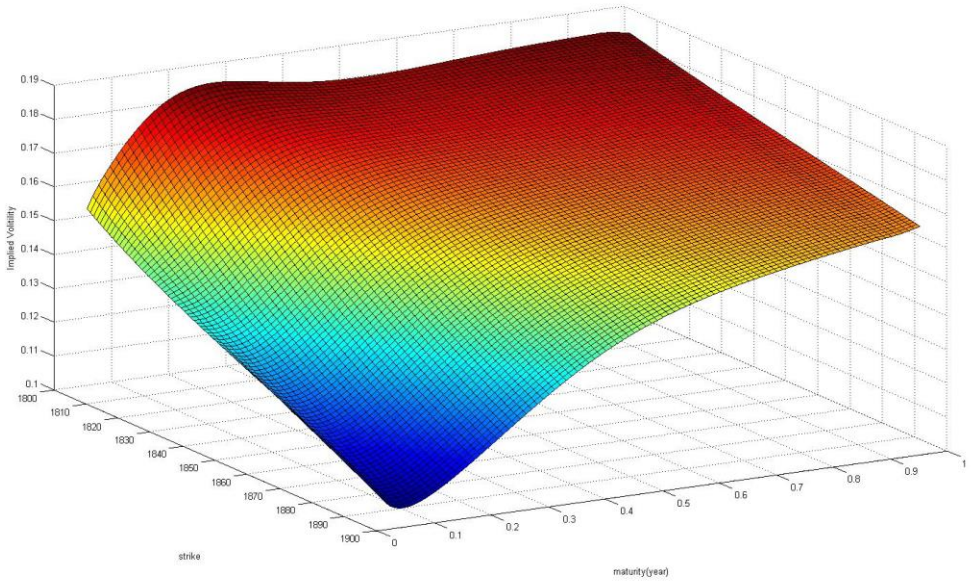
Although these two method are very close at their test error, we could see that their surface are very different. I choose the first one as the solution to this situation.

DAY 2

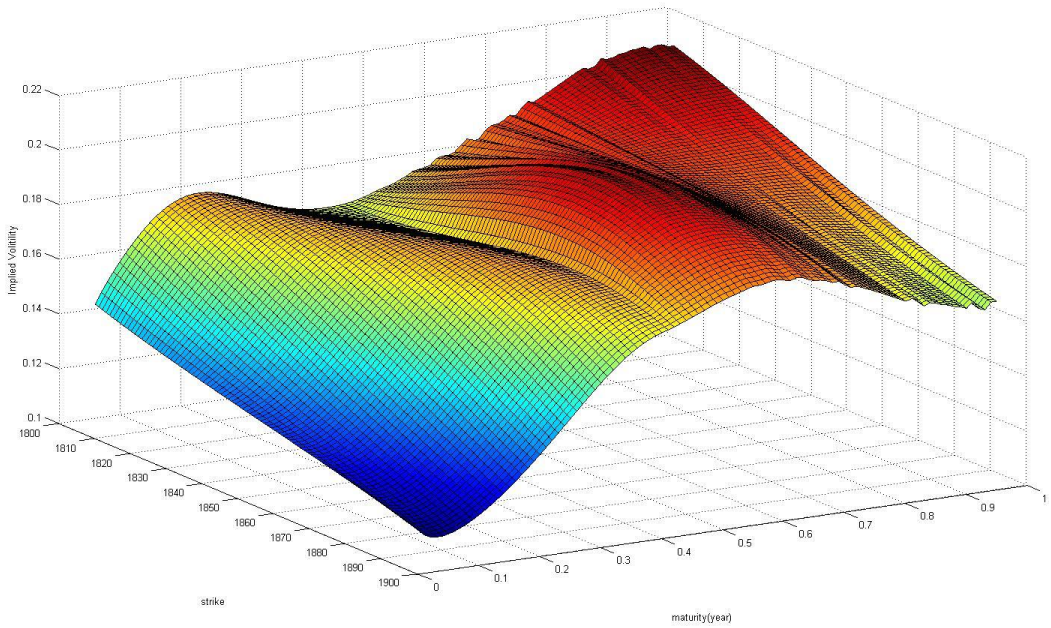
heston	*spread*									
converge	*step*	*condition*	*test error*	*begin*	*penalty*	*sigma*	*kappa*	*theta*	*rho*	*v0*
163	*1.0*	*0.0653*	*1.9635*	*A*	*n*	*0.5588*	*5.9879*	*0.0315*	*-0.9239*	*0.0164*
306	*2.0*	*0.1356*	*1.6157*	*A*	*n*	*0.27343*	*3.0338*	*0.0347*	*-1.284*	*0.0161*
385	*1.0*	*0.1438*	*1.6892*	*B*	*n*	*0.39052*	*4.5899*	*0.0323*	*-1.1052*	*0.0161*
323	*2.0*	*0.1211*	*1.5932*	*B*	*n*	*0.26827*	*2.6971*	*0.0358*	*-1.259*	*0.0163*
252	*1.0*	*0.0626*	*2.1003*	*C*	*n*	*0.59054*	*6.6327*	*0.031*	*-0.937*	*0.0165*
238	*2.0*	*-0.574*	*4.1674*	*C*	*n*	*1.10583*	*10.962*	*0.0296*	*-0.6688*	*0.0171*
279	*1.0*	*0.0766*	*28.021*	*A*	*y*	*-0.4969*	*9.8581*	*0.0164*	*-1.6256*	*0.0247*
314	*1.0*	*-0.12*	*1.2904*	*B*	*y*	*0.63894*	*3.8933*	*0.037*	*-0.8723*	*0.018*
302	*2.0*	*-0.089*	*1.2837*	*B*	*y*	*0.58327*	*3.2249*	*0.0389*	*-0.8697*	*0.0178*

Although our local minimum test error is 1.2904, but the condition is negative, which means the local volatility may be negative in this set of parameters. We will see the surface of these two and make some comparison.

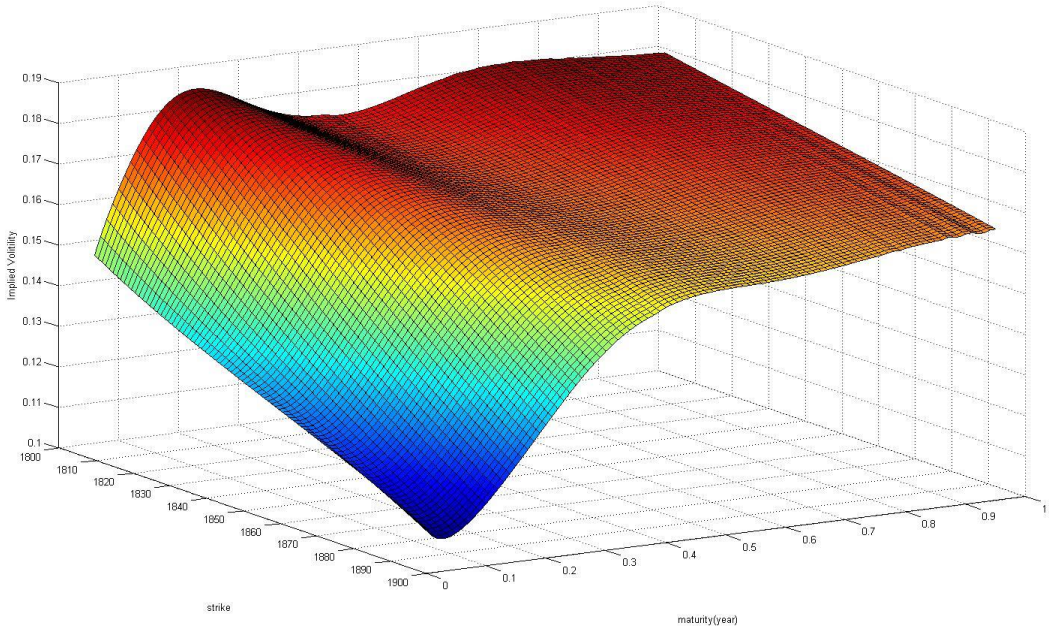
1. test error: 1.28



2. test error: 1.59



3. test error: 4.1



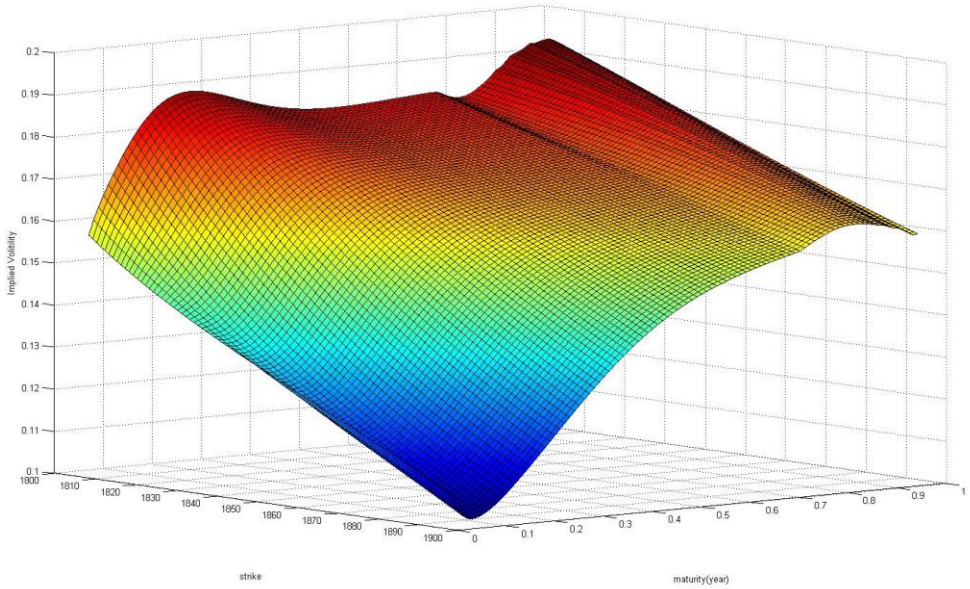
All the tree surfaces are not satisfied, I choose the lowest test error one to contribute to the prediction.

heston	*equal*									
converge	*step*	*condition*	*test error*	*begin*	*penalty*	*sigma*	*kappa*	*theta*	*rho*	*v0*
210	*1.0*	*0.0569*	*1.8349*	*A*	*n*	*0.53748*	*5.3645*	*0.0322*	*-0.9084*	*0.0165*
225	*2.0*	*0.0457*	*1.8233*	*A*	*n*	*0.57018*	*5.5699*	*0.0322*	*-0.893*	*0.0167*
293	*1.0*	*0.1239*	*1.6509*	*B*	*n*	*0.40616*	*4.4231*	*0.0327*	*-1.0513*	*0.0163*

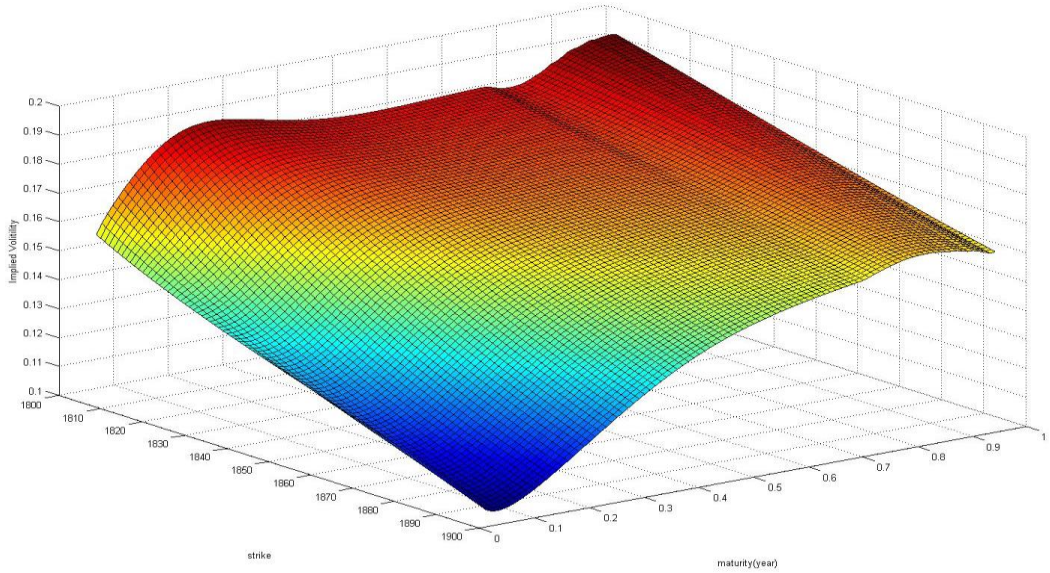
172	*2.0*	*0.0244*	*1.8182*	*B*	*n*	*0.5824*	*5.6598*	*0.0321*	*-0.8871*	*0.0168*
243	*1.0*	*0.0253*	*1.8148*	*C*	*n*	*0.58227*	*5.6742*	*0.0321*	*-0.8909*	*0.0168*
628	*2.0*	*-10.32*	*4.3322*	*C*	*n*	*3.32758*	*8.2778*	*0.0454*	*-0.4067*	*0.0301*
192	*1.0*	*0.2478*	*24.183*	*A*	*y*	*0.54776*	*5.8643*	*0.0467*	*-1.7443*	*0.0226*
363	*1.0*	*-0.12*	*1.2665*	*B*	*y*	*0.64059*	*3.9183*	*0.0371*	*-0.8774*	*0.018*
160	*2.0*	*-0.743*	*1.8368*	*B*	*y*	*1.08786*	*6.1317*	*0.0359*	*-0.753*	*0.0201*
338	*1.0*	*-1.935*	*364.8*	*C*	*y*	*1.48397*	*6.481*	*0.0206*	*-0.6502*	*0.016*
107	*2.0*	*-3.374*	*365.63*	*C*	*y*	*1.91031*	*4.9104*	*0.028*	*-0.6648*	*0.0185*

Local minimum has the unsatisfied condition -0.12.

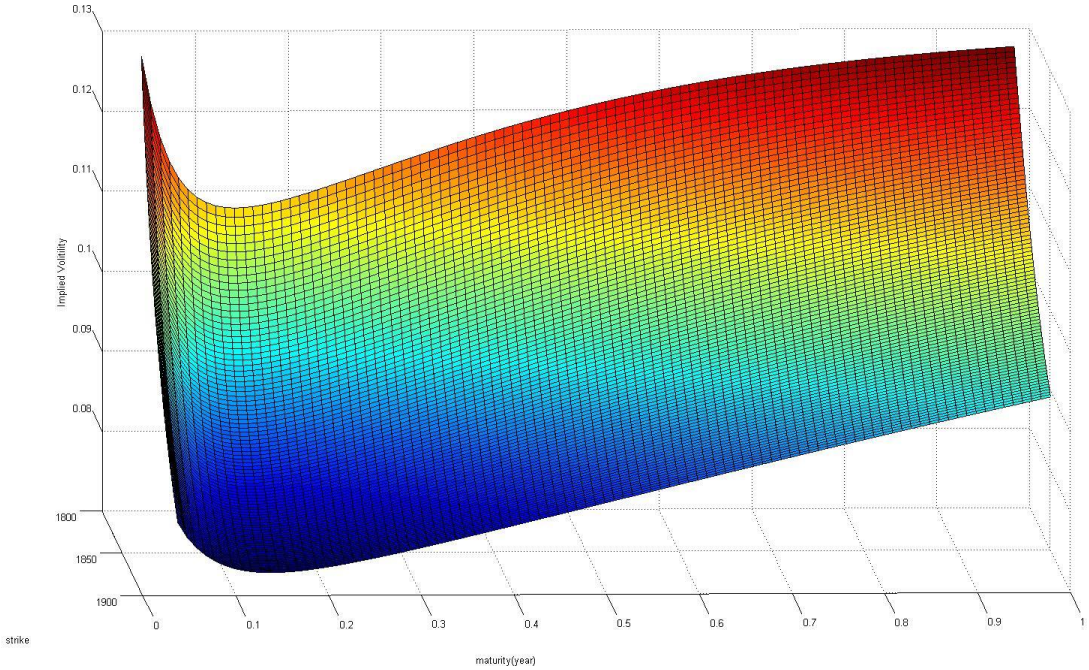
1. test error: 1.26



2. test error: 1.6



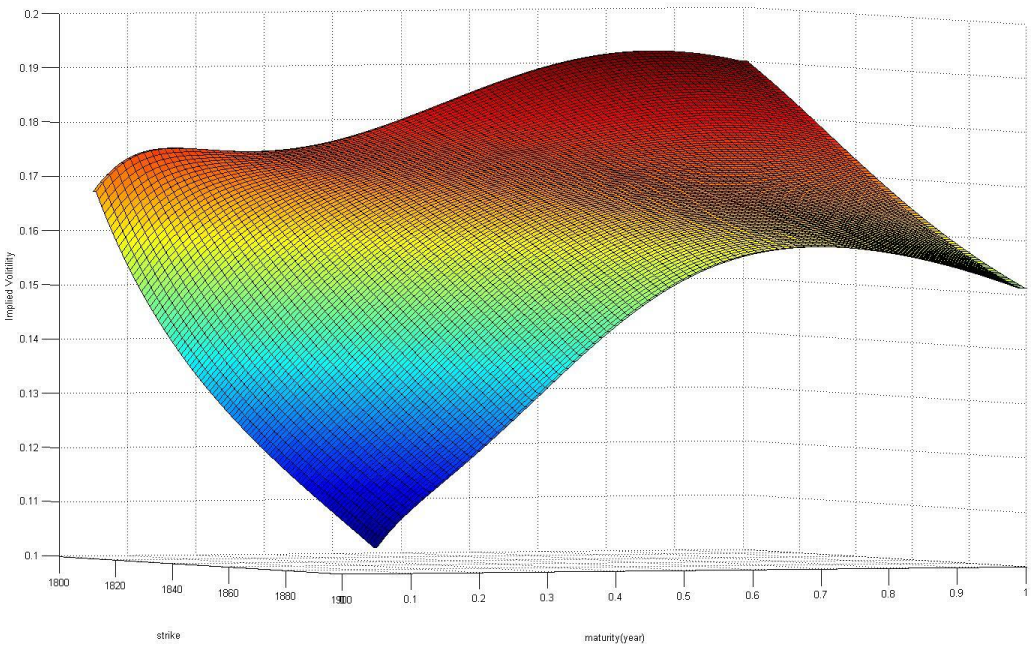
3. test error: 364.8



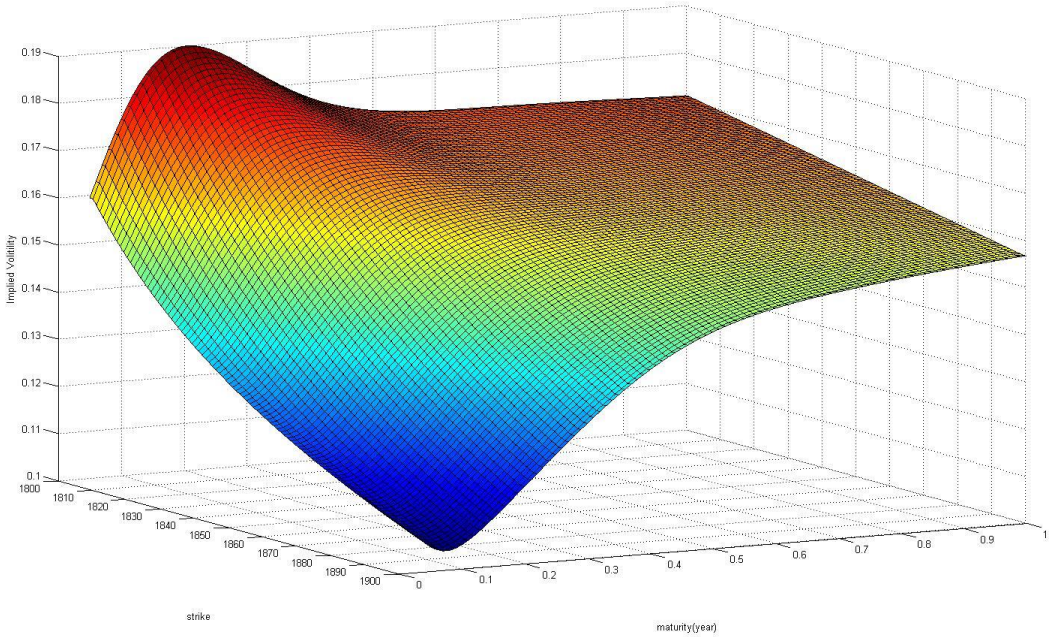
Although the third one has the biggest test error, it's surface seems good enough. This shows again that a good surface and a good prediction don't have much relation, I think it's depends on different markets and different products.

vgsa		*spread*									
steps	*step*	*condition*	*test error*	*begin*	*penalty*	*sigma*	*theta*	*kappa*	*eta*	*lambda*	*nu*
553	*1.0*	*-0.058*	*1.7547*	*E*	*n*	*0.06891*	*-0.058*	*3.4011*	*2.81231*	*5.3892*	*0.255847*
925	*2.0*	*-0.091*	*1.6993*	*E*	*n*	*0.07081*	*-0.091*	*1.437*	*0.70373*	*2.6244*	*0.312082*
747	*3.0*	*-0.098*	*1.676*	*E*	*n*	*0.06722*	*-0.098*	*1.1391*	*-0.2923*	*2.3773*	*0.304181*
663	*1.0*	*-0.058*	*1.9068*	*F*	*n*	*0.06417*	*-0.058*	*8.9604*	*3.38397*	*10.933*	*0.226316*
707	*2.0*	*-0.058*	*1.8974*	*F*	*n*	*0.06355*	*-0.058*	*11.01*	*3.42248*	*13.188*	*0.219702*
655	*1.0*	*-0.057*	*2.0053*	*G*	*n*	*0.06048*	*-0.057*	*10.391*	*3.52775*	*13.055*	*0.204707*
656	*2.0*	*-0.057*	*2.0053*	*G*	*n*	*0.06048*	*-0.057*	*10.391*	*3.52775*	*13.055*	*0.204707*

1: test error: 1.676



2. test error: 1.8974

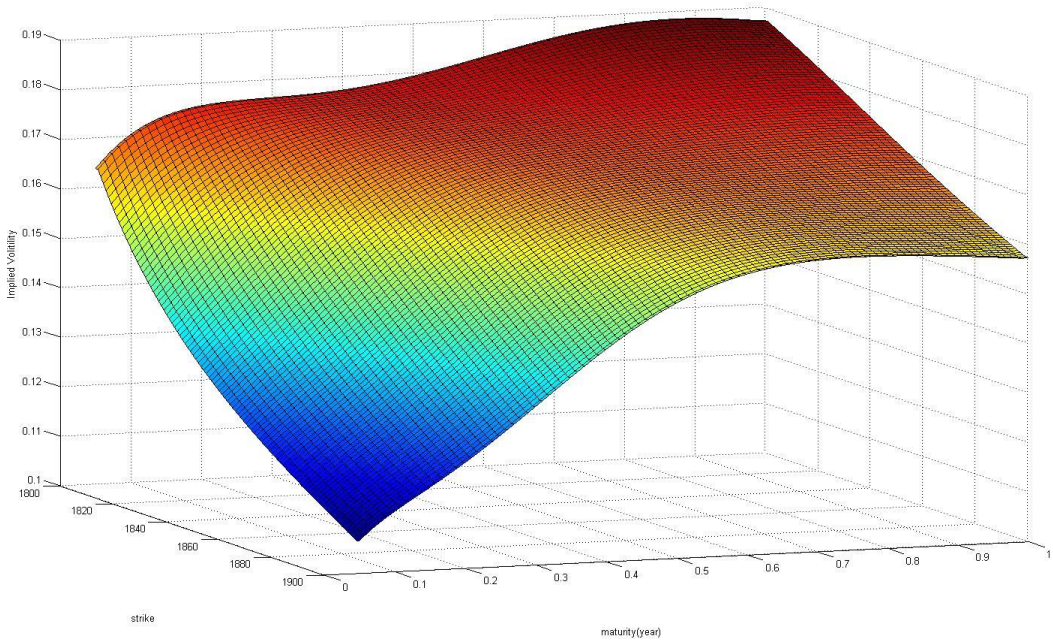


Obviously, the surface from lowest test error is the best.

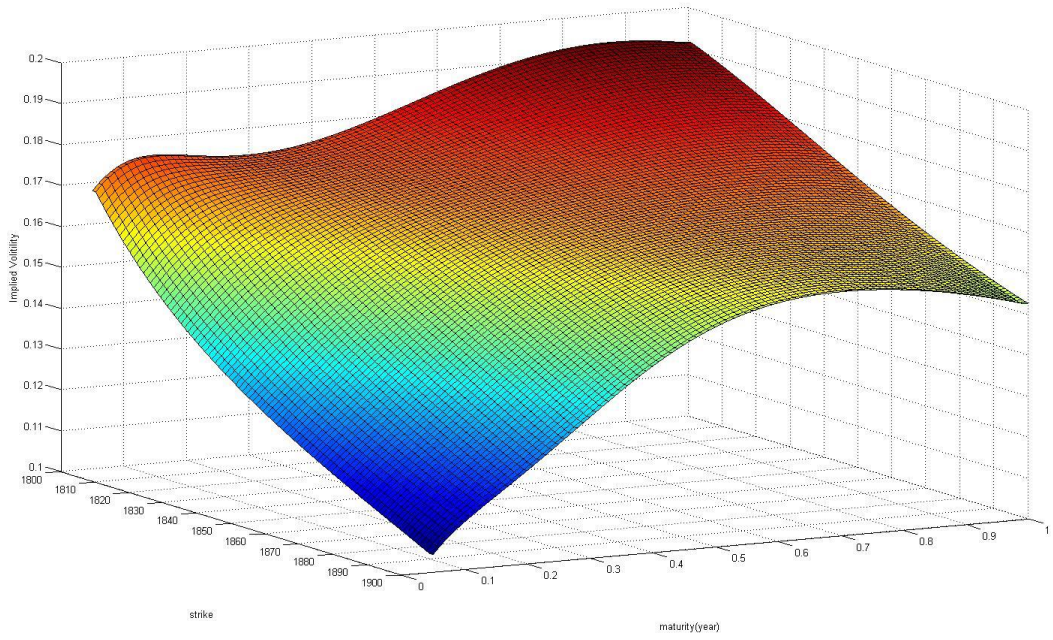
vgsa	*equal*										
steps	*step*	*theta*	*test error*	*begin*	*penalty*	*sigma*	*theta*	*kappa*	*eta*	*lambda*	*nu*
636	*1.0*	*-0.048*	*1.7789*	*F*	*n*	*0.05903*	*-0.048*	*7.5591*	*4.00089*	*11.003*	*0.192693*
714	*3.0*	*-0.039*	*1.8154*	*F*	*n*	*0.04987*	*-0.039*	*7.6478*	*5.1671*	*12.479*	*0.142796*
796	*1.0*	*-0.089*	*1.719*	*E*	*n*	*0.07245*	*-0.089*	*1.4193*	*0.8539*	*2.63*	*0.317642*

798	*2.0*	*-0.089*	*1.719*	*E*	*n*	*0.07245*	*-0.089*	*1.4193*	*0.8539*	*2.63*	*0.317642*
875	*3.0*	*-0.1*	*1.7339*	*E*	*n*	*0.06647*	*-0.1*	*1.0874*	*-0.4247*	*2.2785*	*0.305017*
607	*1.0*	*-0.043*	*1.8771*	*G*	*n*	*0.06108*	*-0.043*	*10.456*	*4.24329*	*15.143*	*0.194048*
609	*2.0*	*-0.043*	*1.8771*	*G*	*n*	*0.06108*	*-0.043*	*10.456*	*4.24329*	*15.143*	*0.194048*

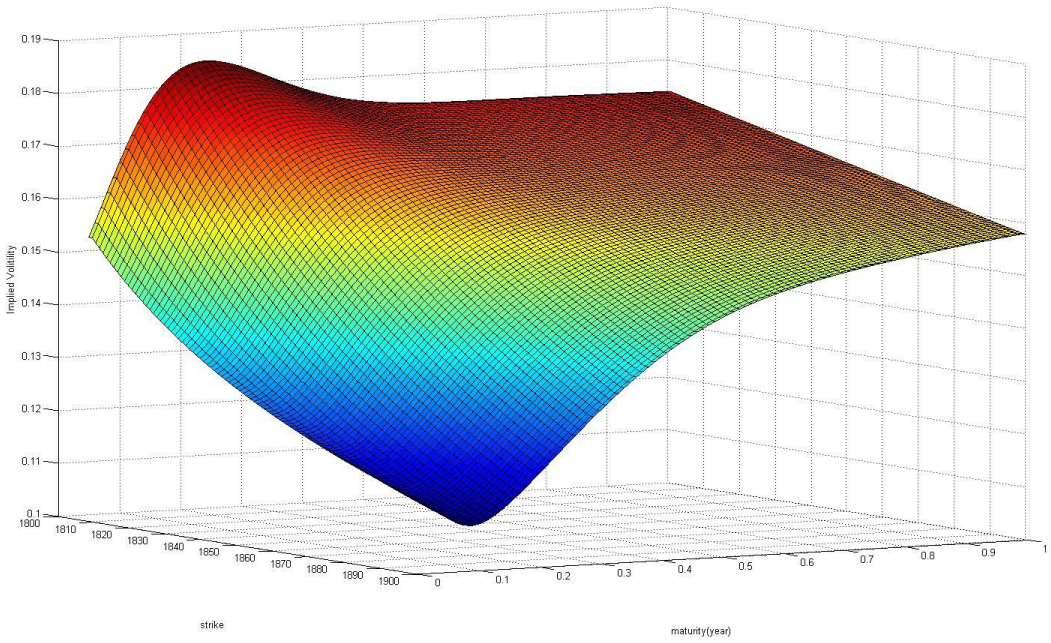
1. test error : 1.71



2. test error : 1.73



3. test error : 1.77



These three close parameters set give us three different surfaces, according to my opinion, I think the second one is the best.

C .Observations and findings on these two different models:

1. Heston models has the property of volatility clustering, which makes the volatility varies around some balance or some mean level.

2. Heston models in this practical problem haven't show much volatility smile. On the contrary, VGSA has shown smile a lot in different parameters.

3. VGSA sometimes have smile both for time to maturity and strike price, which means it can grasp more information in the market compared with Heston. Maybe because it permits the time jump and give the model itself more freedom to describe the market.

4. For VGSA, it has 6 parameters which means it has more degree of freedom, and it should be catch the market information more efficiently than the heston one. But the disadvantage about VGSA is that it takes more time to calibrate the parameters.

5. After I tried many times, I think we can first treat the whole dataset as tranning data and find a local minimum as our starting point for calibration. Then I found that heston converges faster than the VGSA. This may because VGSA's dimension is 6^{th} more than the heston, but I think it is because that VGSA's degree of freedom makes it more active moving in the object space. Once we have found a stable point, the model will always converge to this point. On the contrary, it may change heavily with the different starting point because its dimension is smaller than the VGSA, and it is easy to find a local minimum when the curve are not convex!

D. Pricing the UOP by two models:

I am a new coder, and at this time I can't play with the random generator efficiently in C++, professor Ali told us that we can use matlab to finish this part, so I convert to matlab without any hesitation.

The price of UOP:

Heston: 0.5481

VGSA: 4.0227

Implied volatility from Heston: 0.0291

Implied volatility from VGSA: 0.0337

(4) Description of the logic present in the written source code

In Heston.cc, I have written the ***main function*** *and* ***sereval functions listed below:***
struct Option: for storing the data for each option
die: check the error and exit the function
datareading: read the data from .txt into struct Option
hestoncf: characteristic function of Heston model
fftheston: compute the fft price for calibration
ffthestoncall: for construct the local volatility surface
hestonsurface: call the ffthestoncall to construct the local volatility
my_heston: to compute the objection function and penalty function for calibration.

In vgsa.cc, I have write the main function and serveral functions listed below:
struct Option: for storing the data for each option
die: check the error and exit the function
datareading: read the data from .txt into struct Option
vgcf: characteristic function for variance gamma process
circf: characteristic function for CIR process
vgascf: characteristic function for vgsa process and call the vgcf and circf.
fftvgsa: compute the fft price of option
myvgsa: compute the objective function for calibration
fftvgsacall: compute the vgsa call price to construct the local volatility.
vgsasurface: call fftvgsacall to compute the vgsa call price to construct the local volatility.

In simulation.txt, I put the several function together.
Hestonpath: path for Heston simulation
Vgsapath: path for vgsa simulation
Localpath: path for vgsa simulation
Knockbarrier: judge whether stock had cross the barrier

I download my code from Mobaxterm directly with .cc type. You can open it using notepad (I have already tried, and it works.) If you cannot open it, please contact me.
Thank you!