量力期末報告　b08202046 張昱騏

我們這次的問題基本上對應到的就是 moment problem，而 moment problem 常見的有兩種狀況，第一種是 Stieltjes moment problem 而這種情況其實是對應到我們這次的狀況，因為我們的r ∈ $(0, +\infty)$，而第二種則是 Hamburger moment problem 這種狀況對應到的是我們的範圍上下界為$(-\infty, +\infty)$。

而這類問題的意思是，在我們給定一個數列時(m_0, m_1, m_2, ...)時，在 Hamburger moment problem 的 case，我們能不能找到 positive 的 Borel measure 其中每一項滿足以下關係式(我們給定空間的拓樸中所有開集所形成的σ – algebra，而 Borel measure 就是定義在這個σ – algebra上面的 measure):

$$m_n = \int_{-\infty}^{\infty} x^n d\mu(x)$$

而在 Stieltjes moment problem 的 case 下則換成，

$$m_n = \int_{0}^{\infty} x^n d\mu(x)$$

而在我們的問題基本上就是找出何種情況下可以真的存在這種測度，而這個的結果是:

存在性的條件是判斷以下兩式子:

$$\Delta_n = \begin{bmatrix} m_0 & m_1 & m_2 & \cdots & m_n \\ m_1 & m_2 & m_3 & \cdots & m_{n+1} \\ m_2 & m_3 & m_4 & \cdots & m_{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_n & m_{n+1} & m_{n+2} & \cdots & m_{2n} \end{bmatrix} \quad \Delta_n^{(1)} = \begin{bmatrix} m_1 & m_2 & m_3 & \cdots & m_{n+1} \\ m_2 & m_3 & m_4 & \cdots & m_{n+2} \\ m_3 & m_4 & m_5 & \cdots & m_{n+3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n+1} & m_{n+2} & m_{n+3} & \cdots & m_{2n+1} \end{bmatrix}$$

而這兩個矩陣都應該是 positive semidefinite，這時我們的μ就存在，對於所有的 n。

而這比教授給我們條件再多了一個條件，所以接下來在程式模擬時會比較有多第二個條件時的限制以及只有教授給的限制(第一個限制)時的差別。

而唯一性的條件則是以下的式子成立時:

$$\sum_{n>1} m_n^{-1/(2n)} = \infty$$

(這個判別式是來自 Carleman's theorem for the Stieltjes moment problem)

而這些的證明可以參考以下參考資料:

https://arxiv.org/pdf/2008.12698.pdf，這邊有清楚說明每個 moment problem 的條件以及一些基本的分析課的結果。

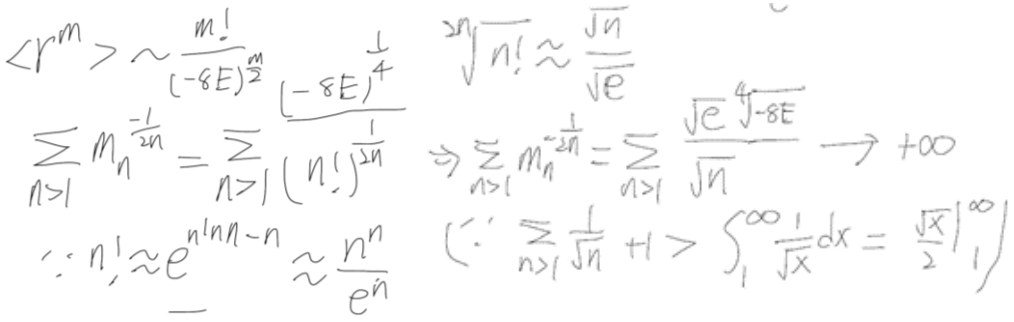
接著回到我們原本的問題我們是要找出當 Hamiltonian 是以下形式時

$$\mathrm{H} = \frac{P^2}{2} + \frac{l(l+1)}{2r^2} - \frac{1}{r}$$

而我們又有對於所有的 energy eigenstate，都有以下遞迴式，

$$0 = 8mE\langle r^{m-1}\rangle + (m-1)[m(m-2) - 4\ell(\ell+1)]\langle r^{m-3}\rangle + 4(2m-1)\langle r^{m-2}\rangle$$

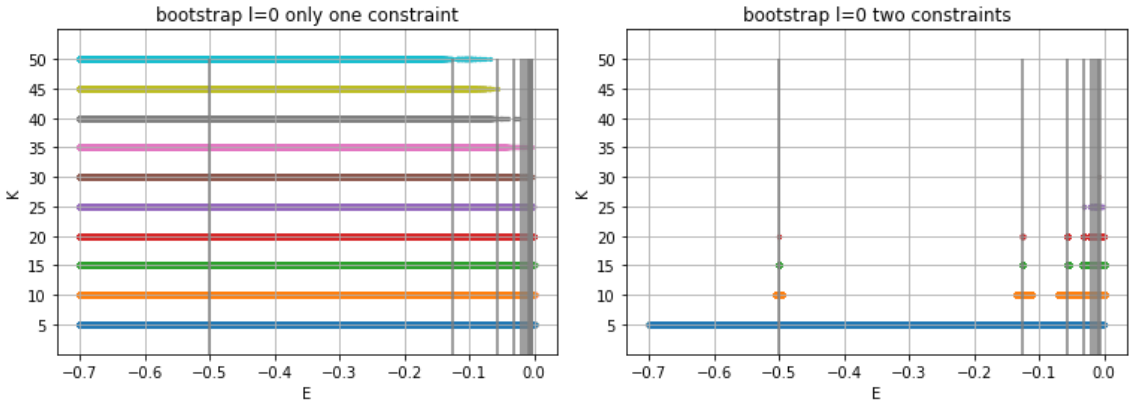
而 R 這個遞迴式我們只需要兩項就可以決定整個遞迴式的關係式，而我們知道 $\langle r^0\rangle = \langle 1\rangle$=1，when m=1，帶入上方遞迴式可以得到$8\mathrm{E}\langle r^0\rangle + 4(2-1)\langle r^{-1}\rangle = 0$，所以得到 $\langle r^{-1}\rangle = -2\mathrm{E}\langle r^0\rangle = -2\mathrm{E}$ ，所以我們可以得到每一項的值。
然後因為我們的取 expectation value 時就是找到一個對應的 measure，所以由此 measure 的存在性我們可以得到兩個矩陣都要是半正定的條件。
然而一個矩陣是半正定有許多等價的條件，第一個就是教授所給的每一個 diagonal 的 k× k block 的 determinant 都是大於等於 0，而第二個作法就是因為我們的 Hankel matrix 是對稱的，所以可以對角化，所以直接寫程式找出他的所有特徵值，藉由判斷全部特徵值是否大於等於 0 來判斷。
而在判斷我們矩陣的半正定性前，我們先分析一下我們的 hankel matrix 的 elements 的數值大小的關係，當我們遞迴 m 很大時，因為我們 l 遠小於 m 所以可以忽略，所以可以得到$\langle r^m\rangle \sim \frac{-m(m-1)}{8E}\langle r^{m-2}\rangle$，而由此我們可以帶入我們的唯一性判斷的式子中，可以發現$\langle r^m\rangle \sim \frac{m!}{(-8E)^{\frac{m}{2}}}$，然後藉由以下計算可以發現這符合我們的判別式的條件:



所以可以發現若對應的 Borel measure 存在，那一定是唯一的。這保證了如果存在的話那這種 measure 被完全定下來。
而我們 E 本來落在的位置就是在大於-0.5 到 0 之間因為我們的理論上的能量是 $\frac{-1}{2n^2}$對於 n 屬於正整數，所以可以發現我們的遞迴式成長速度是階層乘上 1/E 的 m 次方，所以當我們考慮 50*50 的 hankel matrix 時當 E=-0.125 時，我們的 $\langle r^{100}\rangle$趨近於 10 的 60 次方，所以可見我們的矩陣的 determinant 在計算上十分的困難，而這也導致我們比較不能使用方法二直接計算特徵值的方法，而且使用這種方式在電腦數值計算時因為我們的位數非常多，所以浮點誤差會導致嚴

重的影響，因為在 c++中最好的 long double 的有效數字最多也只有 19 位，所以這導致了我們計算精確度的上限，而這也導致我們在數值逼近上有著無法突破的上限。

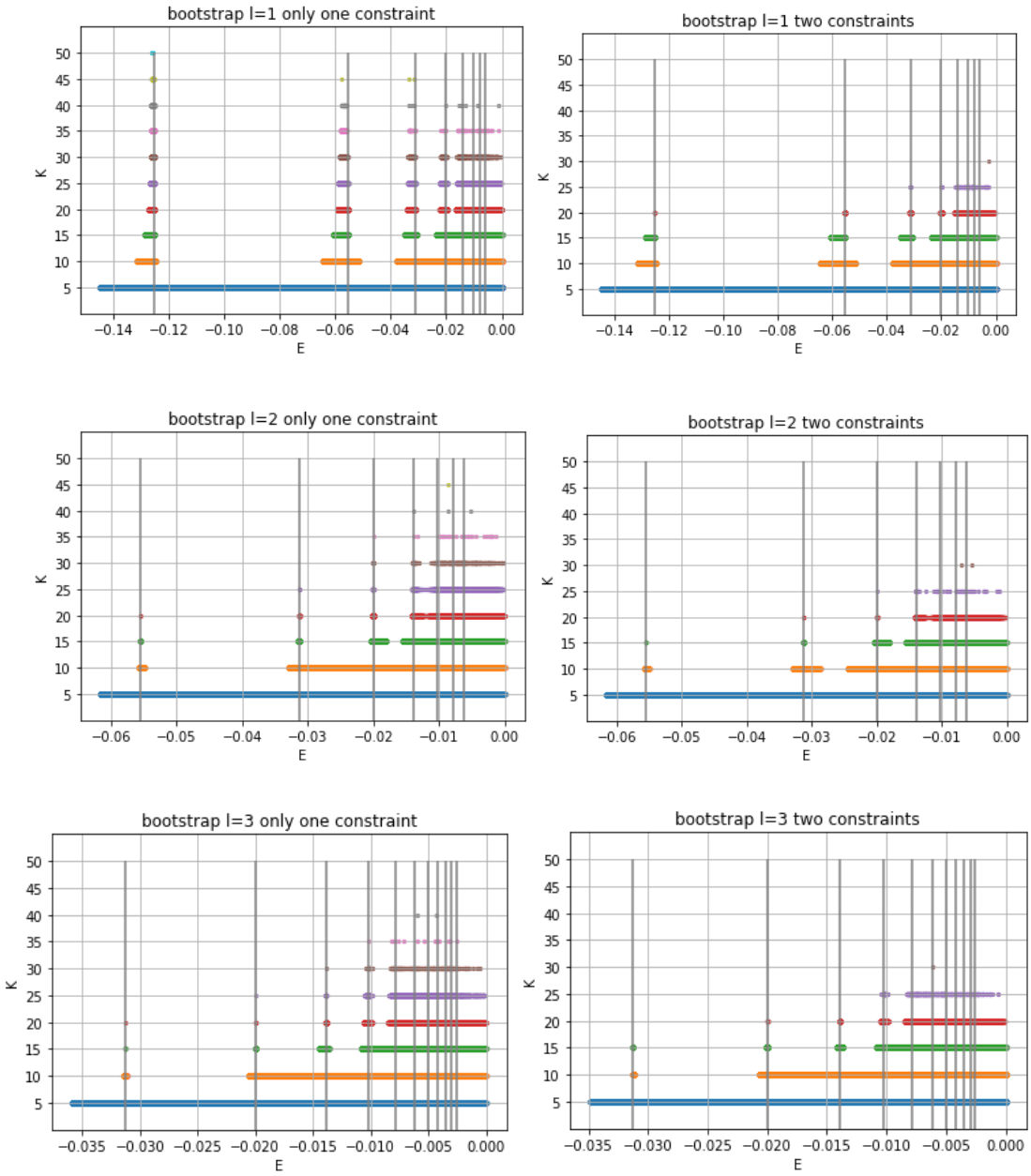
所以最後採用的方式就是使用 c++來判別我們的矩陣是否為半正定，而我們程式的流程大致如下，我們採用我們將 E 設為我們的變數，然後跑遍我們給定的 E 的範圍，一階矩陣的 determiant 是否大於 0，然後將留下的能量繼續跑 2 階矩陣，一直跑到 50 階，然而在判定矩陣是半正定的方法是看矩陣的斜對角的 block 的 determinant 都要大於等於 0，而計算龐大矩陣的 determinant 的方法是用 LU 分解的方式來幫助我們加速計算，如果我們使用伴隨矩陣的方法的話，我們所需要的複雜度將會是 O(n!)而用 LU 分解的複雜度則是 O(n^3)，這讓我們在程式上可以實作，但是這可能面臨到的問題就是我們 LU 分解會有用到除法這將會放大我們數值估計的誤差，但是我們仍猜測這不是主要誤差的來源，主要還是因為我們的矩陣元素的數值分布太寬，所以我們矩陣計算會有極大誤差(因為只有 19 位有效數字)。而以下就是我們的結果，我們先做我們對應給定角動量 l 後，我們矩陣要求的的大小為 K 時，我們允許的能量範圍的圖，然後在藉由 python 來將我們的結果畫出以下的圖:



若只有第一個 constraint 的話我們可以發現這給不了任何 condition，而在能量接近於 0 時，因為我們的矩陣中的 element 會正比於 1/E 的 n 次方，所以當 E 很小時會爆，所以這顯示的是我們程式數值計算的極限。

而在兩個 constraints 都加上後，我們可以得到如同我們對於氫原子模型的結論只有在能量為$\frac{-1}{2n^2}$才會有值，但是在多加上這個矩陣的條件後，我們可以發現我們能接受的能量一下就被消光了，所以這證明了我們數值分析上的誤差本來就很大，當矩陣階數超過 20 我們的電腦計算可能就不太可靠了。

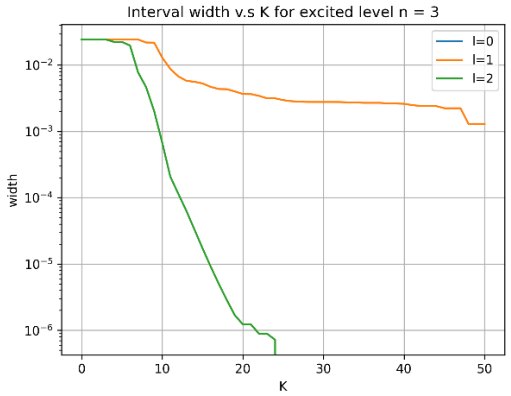
接著是 l>0 的情況:

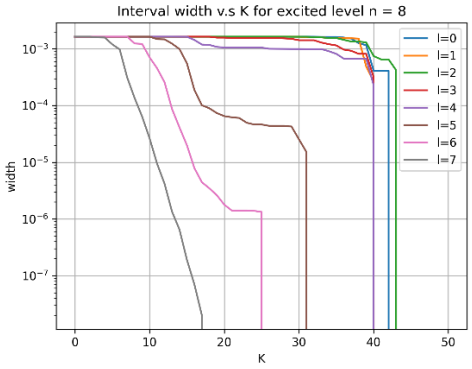


由以上的幾個圖可以得到我們的雙限制可以讓我們的收斂速度增快許多，但是因為是雙限制所以當我們的程式的數值方法出問題時(即 K>20)時，因為 float 誤差影響很大，所以當取兩個結果的交集時，常常會是 0，因為這結果會被兩邊誤差影響所以交集很有可能會歸零。
而以上的模擬我們也能簡單的發現當我們 l 越大時，我們有的能量級數至少是 n=l+1。

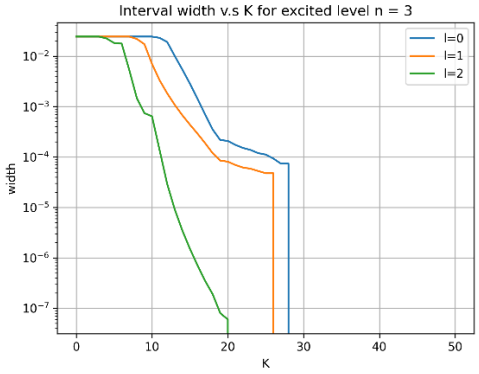
接下來我們分析對於不同 l 的收斂速度的估計，先單純從我們前一步的討論，可以稍微看到當 l 越大在每個特徵值附近的區間收縮的速度越快，然而為了能更好比較這個收斂的速度，我們考慮不同 l 但同樣特徵能量附近的允許區間，看區間對我們的矩陣大小(K)的關係，然後再將各種不同的 l 話到同一張圖來做

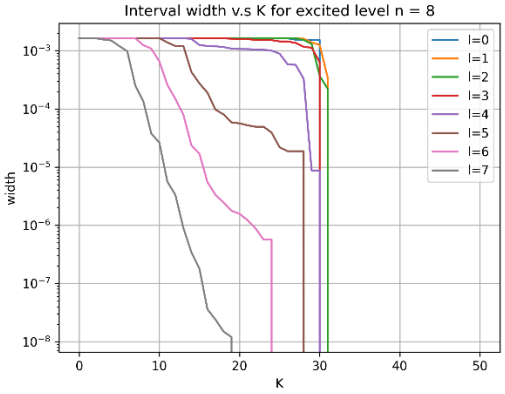
比較，經過程式模擬可以得到以下結果:





以上是我們只考慮只有第一個矩陣要試辦正定矩陣時的條件，然而接下來則是考慮兩個矩陣都要是半正定矩陣時所給出的條件:





然而上面程式我們取寬度的方式一開始我們將能量的範圍設定為我們要看的特徵能量相鄰的特徵能量當我們的上下界，然後再經過一層層矩陣的淘汰掉我們可能的 E 後，我們取包含我們所要觀察的特徵能量的連續能量區間，這就是我們的寬度，然後我們對此寬度取 log 可以做出上面四張圖。

然而由上面四圖，應該可以很輕易發現我們的 l 角動量將會嚴重影響我們的收斂速度，當 l 越大我們的區間變得越小，而且他們區間寬度下降的速率大概都是指數下降，然而當 K>20 時，這時我們數值計算上的因為符點數的誤差或是 LU 分解中除法帶來的誤差越來越大，所以會發現我們的斜線會突然驟降，這是因為我們數值計算技巧的極限所導致的。然而在上兩張圖沒有 l=0 的狀況，原因是因為當 l=0 時我們的第一個矩陣給不出任何 condition，所以只有在兩個條件都有時才會有 l=0 的線，而當有兩個矩陣的半正定條件時，在 K>20 以後很容易發生驟降的情況，這代表著我們在當數字 K>20 以後我們的那些看似允許的區域其實可能很多的是算錯的(或是存在誤差)，所以當我們兩個矩陣的條件各自都有自己不同的數值分析上的誤差時，很容易使理論上正確的點倍淘汰掉，因此下兩張圖顯示的不只是因為我們有兩個條件所以使我們的收斂變快，第二個其實也反映出我們數值模擬可以相信的區間可能只落在 K<20 以內的範圍。

然而這個結果也不是讓人意外的，因為當我們角動量越大時，我們對應的等效

位能越大，所以我們自然收斂的速度就會比較快，或者是因為我們等效位能上升所以位能井深下降所以能容許的能量分布變少，所以隨 K 上升允許的 E 區間下降的越快，因此此結論也相當符合直覺。

然而最後我在試圖找出新的遞迴式時，我嘗試過了需多種不同 operator 的可能性，例如在一開始考慮過$p_r = -\mathrm{i}\frac{d}{dr}$也考慮過徑向動量$-\mathrm{i}(\frac{d}{dr}+\frac{1}{r})$，但這兩個 operator 在處理上都不是很好處理，因為我們的位能不是 r 的正冪次，所以$\left(\frac{d}{dr}\right)^n$作用在$\frac{1}{r}$上時會出現 n 項，所以在處理遞迴上十分的麻煩所以我也沒有好方法來處理，而也考慮過像是升降算符的形式，例如

$$a^+ = ip_r + \frac{l}{r} - \frac{1}{l} \quad \mathrm{a} = -ip_r + \frac{l}{r} - \frac{1}{l}$$

而這樣會有$a^+a = H + \frac{1}{2l^2} \quad aa^+ = H - \frac{l}{r^2} + \frac{1}{2l^2}$ 所以可以看出升降算符的形式，但我也無法從這個關係式找出新的遞迴式。
所以我猜整個新的 constraint 應該就是來自我們今天的問題不是 Hamburger moment problem，因為此種問題對應到存在性的條件剛好就是只有第一個矩陣為半正定矩陣的條件而已，所以在我們今天球座標下 r 的定義域只有(0,+ ∞)，所以是 Stieltjes moment problem，所以條件比教授的多。

而以上數值分析有少部分參考以下論文:
https://arxiv.org/pdf/2108.08757.pdf?fbclid=IwAR142M6fwtXkdiT2-JAmRPN0JxqM7W6fDRDiME6iVrvnLBEDBdqvuRnqYYY

最後總結有以下幾點:

1. 可以發現我們當 l=0 時，若只有第一個條件時無法給出任何條件，但若有一和二條件時就可以給出正確的限制
2. 當 l 越大時，我們收斂的速度會越快，而且收斂的速度大概是以指數收斂
3. 當我們考慮矩陣到 20 階以上時，我們的程式上誤差可能非常的大，所以可能導致我們顯示的可以的範圍實際上是不行的，而且這誤差的來源是因為我們的 long double 只有 19 位有效數字，雖然他可以存很大的數字。
4. 而在計算我們矩陣 determinant 時，我們基本上沒有太多選擇，因為伴隨矩陣法演算法太慢但是不需要用到除法，然而用高斯消去法的話雖然不用除法但數字可能會變太大到超過我們的數值可允許範圍，所以只能折衷使用 LU 分解來計算
5. 而對於跑程式時，其實可以考慮我們的 operator 是為r^2或是r^k對於 k 是一個正整數，這樣跑可以幫助我們減少跑的時間而且收斂的速度可以快上很多，

但是因為我們程式的主要問題是最後算不準而不是在加速上出現問題，所以即使我們可以加速計算，但我們仍無法解決誤差的問題，所以我們就沒有使用這種方式來進行分析。

接下來是我們的程式的部分，但是程式主要 B08202054 的李杰銘寫的然後我提供一些數值方法的建議

這一頁是我們在計算那些 E 跟矩陣的階數 K 的相關性所使用的 c++

```
#include <bits/stdc++.h>

bool READOUT = true;
int constraint = 2;
int Kmax = 50;
int l = 1;
long double step = 1e-3;
long double Emin = -0.13;
long double Emax = -0.0;

long double r(long double E, int k, std::vector<long double>& record)
{
	if(k == -1) return -2*E;
	else if(record[k] == 0) record[k] = -(k*((k+1)*(k-1)-4*l*(l+1))*r(E,k-
2,record)+4*(2*k+1)*r(E,k-1,record))/(8*(k+1)*E);
	return record[k];
}

std::vector<std::vector<long double>> Hankel_matrix(long double E, int K)
{
	std::vector<long double> record(2*K,0);
	record[0] = 1.0;
	std::vector<std::vector<long double>> H(K, std::vector<long double>(K,0));
	for(int i = 0; i < K; i++)
	{
		for(int j = 0; j < K; j++) H[i][j] = r(E,i+j,record);
	}
	return H;
}
```

```
std::vector<std::vector<long double>> Hankel_matrix_2(long double E, int K)
{
	std::vector<long double> record(2*K,0);
	record[0] = 1.0;
	std::vector<std::vector<long double>> H(K, std::vector<long double>(K,0));
	for(int i = 0; i < K; i++)
	{
		for(int j = 0; j < K; j++) H[i][j] = r(E,i+j+1,record);
	}
	return H;
}

std::vector<std::vector<long double>> Hankel_matrix_3(long double E, int K)
{
	std::vector<long double> record(2*K,0);
	record[0] = 1.0;
	std::vector<std::vector<long double>> H(K, std::vector<long double>(K,0));
	for(int i = 0; i < K; i++)
	{
		for(int j = 0; j < K; j++) H[i][j] = pow(-1,i+j)*r(E,i+j,record);
	}
	return H;
}

long double determinant(std::vector<std::vector<long double>> A, int n)
{
	std::vector<std::vector<long double>> L(n, std::vector<long double>(n,0));
	std::vector<std::vector<long double>> U(n, std::vector<long double>(n,0));
	long double det = 1.0;
	for(int i = 0; i < n; i++)
	{
		U[i][i] = 1.0;
		L[i][0] = A[i][0];
		U[0][i] = A[0][i] / A[0][0];
	}
	for(int j = 0; j < n; j++)
	{
```

```
            for(int i = j; i < n; i++)
            {
                long double sum = 0.0;
                for(int k = 0; k < j; k++) sum += L[i][k] * U[k][j];
                L[i][j] = A[i][j] - sum;
            }
            for(int i = j; i < n; i++)
            {
                long double sum = 0.0;
                for(int k = 0; k < j; k++) sum += L[j][k] * U[k][i];
                U[j][i] = (A[j][i] - sum) / L[j][j];
            }
        }
        for(int i = 0; i < n; i++) det *= L[i][i] * U[i][i];
        return det;
}

int main()
{
        std::vector<long double> search_space;
        for(long double E = Emin; E <= Emax; E += step) search_space.push_back(E);
        for(int K = 1; K <= Kmax; K++)
        {
            std::vector<long double> search_space_new;
            for(auto E:search_space)
            {
                if(constraint == 1)
                {
                    if(determinant(Hankel_matrix(E,K),K) >= 0.0)
search_space_new.push_back(E);
                }
                else if(constraint == 2)
                {
                    if(determinant(Hankel_matrix(E,K),K) >= 0.0 &&
determinant(Hankel_matrix_3(E,K),K) >= 0.0)
                        search_space_new.push_back(E);
                }
            }
```

```
            search_space = search_space_new;
            printf("%d %ld\n", K, search_space.size());
            if(READOUT && K%5 == 0)
            {
                char filename[50];
                sprintf(filename, "K%d.txt", K);
                FILE* fout = fopen(filename,"w+");
                for(auto E:search_space) fprintf(fout, "%.10e\n", E);
                fclose(fout);
            }

        }
        return 0;
}
```

而這時用來畫圖的 python 程式如下:

```
import matplotlib.pyplot as plt
import numpy as np

#Kstep = np.array([20,25,30])
Kstep = np.linspace(5,50,10,dtype=np.int64)
for K in Kstep:
    E = []
    file = open(f'K{K}.txt','r')
    lines = file.readlines()
    for line in lines:
        E.append(np.float64(line))
    E = np.array(E)
    plt.scatter(E,np.array([K]*len(E)),s=5)

for n in range(2,10): plt.vlines(x=-1/2/n**2, ymin=0, ymax=50, colors='gray')

plt.yticks(Kstep)
plt.title('bootstrap l=3')
plt.xlabel('E')
plt.ylabel('K')
plt.ylim((np.min(Kstep)-5,np.max(Kstep)+5))
plt.grid()
```

plt.savefig('bootstrap_l3_2.png',dpi=300)

而下一頁是我在計算上用在劃出 l 的收斂速度所使用的程式 c++:

```
#include <bits/stdc++.h>
int constraint = 1;
int Kmax = 50;
int n = 3;
long double step = 1e-9;
long double width = 0.5/n/n-0.5/(n+1)/(n+1);
long double Emin = -0.5/n/n-width/2;
long double Emax = -0.5/n/n+width/2;

long double r(int l, long double E, int k, std::vector<long double>& record)
{
	if(k == -1) return -2*E;
	else if(record[k] == 0) record[k] = -(k*((k+1)*(k-1)-4*l*(l+1))*r(l,E,k-
2,record)+4*(2*k+1)*r(l,E,k-1,record))/(8*(k+1)*E);
	return record[k];
}

std::vector<std::vector<long double>> Hankel_matrix(int l, long double E, int K)
{
	std::vector<long double> record(2*K,0);
	record[0] = 1.0;
	std::vector<std::vector<long double>> H(K, std::vector<long double>(K,0));
	for(int i = 0; i < K; i++)
	{
		for(int j = 0; j < K; j++) H[i][j] = r(l,E,i+j,record);
	}
	return H;
}
```

```
std::vector<std::vector<long double>> Hankel_matrix_2(int l, long double E, int K)
{
	std::vector<long double> record(2*K,0);
	record[0] = 1.0;
	std::vector<std::vector<long double>> H(K, std::vector<long double>(K,0));
	for(int i = 0; i < K; i++)
	{
		for(int j = 0; j < K; j++) H[i][j] = r(l,E,i+j+1,record);
	}
	return H;
}

std::vector<std::vector<long double>> Hankel_matrix_3(int l, long double E, int K)
{
	std::vector<long double> record(2*K,0);
	record[0] = 1.0;
	std::vector<std::vector<long double>> H(K, std::vector<long double>(K,0));
	for(int i = 0; i < K; i++)
	{
		for(int j = 0; j < K; j++) H[i][j] = pow(-1,i+j)*r(l,E,i+j,record);
	}
	return H;
}

long double determinant(std::vector<std::vector<long double>> A, int n)
{
	std::vector<std::vector<long double>> L(n, std::vector<long double>(n,0));
	std::vector<std::vector<long double>> U(n, std::vector<long double>(n,0));
	long double det = 1.0;
	for(int i = 0; i < n; i++)
	{
		U[i][i] = 1.0;
		L[i][0] = A[i][0];
		U[0][i] = A[0][i] / A[0][0];
	}
	for(int j = 0; j < n; j++)
	{
		for(int i = j; i < n; i++)
```

```
            {
                long double sum = 0.0;
                for(int k = 0; k < j; k++) sum += L[i][k] * U[k][j];
                L[i][j] = A[i][j] - sum;
            }
            for(int i = j; i < n; i++)
            {
                long double sum = 0.0;
                for(int k = 0; k < j; k++) sum += L[j][k] * U[k][i];
                U[j][i] = (A[j][i] - sum) / L[j][j];
            }
        }
        for(int i = 0; i < n; i++) det *= L[i][i] * U[i][i];
        return det;
}

int main()
{
        for(int l = 0; l < n; l++)
        {
            std::vector<long double> search_space;
            for(long double E = Emin; E <= Emax; E += step)
search_space.push_back(E);
            char filename[50];
            sprintf(filename, "n%d_l%d.txt", n, l);
            FILE* fout = fopen(filename,"w+");
            fprintf(fout, "0 %.15Le\n", width);
            for(int K = 1; K <= Kmax; K++)
            {
                std::vector<long double> search_space_new;
                for(auto E:search_space)
                {
                    if(constraint == 1)
                    {
                        if(determinant(Hankel_matrix(l,E,K),K) >= 0.0)
                            search_space_new.push_back(E);
                    }
                    else if(constraint == 2)
```

```
                        {
                                if(determinant(Hankel_matrix(l,E,K),K) >= 0.0 &&
determinant(Hankel_matrix_2(l,E,K),K) >= 0.0)
                                        search_space_new.push_back(E);
                        }
                        else if(constraint == 3)
                        {
                                if(determinant(Hankel_matrix(l,E,K),K) >= 0.0 &&
determinant(Hankel_matrix_3(l,E,K),K) >= 0.0)
                                        search_space_new.push_back(E);
                        }
                }
                search_space = search_space_new;
                printf("%d %ld\n", K, search_space.size());
                if(search_space.empty()) fprintf(fout, "%d 0.0\n", K);
                else fprintf(fout, "%d %.15Le\n", K, search_space.back()-
search_space.front());
        }
        fclose(fout);
    }
    return 0;
}
```

而接下來是劃出 l 收斂性的圖的 python 程式:

```
from cProfile import label
import string
import matplotlib.pyplot as plt
import numpy as np

n = 3
c = 1
for l in range(n):
    K = []
    width = []
    file = open(f'n{n}_l{l}.txt','r')
    for line in file.readlines():
        line = line.split(' ')
        K.append(int(line[0]))
```

```
            width.append(float(line[1]))
        plt.plot(K,width,label=f'l={l}')
        file.close()

plt.xticks([0,10,20,30,40,50])
plt.xlabel('K')
plt.ylabel('width')
plt.yscale('log')
plt.legend()
plt.grid()
plt.title(f'Interval width v.s K for excited level n = {n}')
plt.savefig(f'width_K_n{n}_{c}.png',dpi=300)
```