10.Lagrange 預配置一維權重統整

Chen Peng Chung

January 18, 2026

Contents

一 Lagrange 權重指標記憶體對照表

Table 1: 宣告權重指標記憶體：double*（有 BFL 者 f 改小寫）

變數符號		編號	索引 i						
Var	**F**	0	1	2	3	4	5	6	
X	-	0	X0_0	X0_1	X0_2	X0_3	X0_4	X0_5	X0_6
		2	X2_0	X2_1	X2_2	X2_3	X2_4	X2_5	X2_6
Y	-	0	Y0_0	Y0_1	Y0_2	Y0_3	Y0_4	Y0_5	Y0_6
		2	Y2_0	Y2_1	Y2_2	Y2_3	Y2_4	Y2_5	Y2_6
Xi	F	3	XiF3_0	XiF3_1	XiF3_2	XiF3_3	XiF3_4	XiF3_5	XiF3_6
		4	XiF4_0	XiF4_1	XiF4_2	XiF4_3	XiF4_4	XiF4_5	XiF4_6
		5	XiF5_0	XiF5_1	XiF5_2	XiF5_3	XiF5_4	XiF5_5	XiF5_6
		6	XiF6_0	XiF6_1	XiF6_2	XiF6_3	XiF6_4	XiF6_5	XiF6_6
		15	XiF15_0	XiF15_1	XiF15_2	XiF15_3	XiF15_4	XiF15_5	XiF15_6
		16	XiF16_0	XiF16_1	XiF16_2	XiF16_3	XiF16_4	XiF16_5	XiF16_6
		17	XiF17_0	XiF17_1	XiF17_2	XiF17_3	XiF17_4	XiF17_5	XiF17_6
		18	XiF18_0	XiF18_1	XiF18_2	XiF18_3	XiF18_4	XiF18_5	XiF18_6
XBFL	f	37	XBFLf37_0	XBFLf37_1	XBFLf37_2	XBFLf37_3	XBFLf37_4	XBFLf37_5	XBFLf37_6
		38	XBFLf38_0	XBFLf38_1	XBFLf38_2	XBFLf38_3	XBFLf38_4	XBFLf38_5	XBFLf38_6
		49	XBFLf49_0	XBFLf49_1	XBFLf49_2	XBFLf49_3	XBFLf49_4	XBFLf49_5	XBFLf49_6
		410	XBFLf410_0	XBFLf410_1	XBFLf410_2	XBFLf410_3	XBFLf410_4	XBFLf410_5	XBFLf410_6
YBFL	f	3	YBFLf3_0	YBFLf3_1	YBFLf3_2	YBFLf3_3	YBFLf3_4	YBFLf3_5	YBFLf3_6
		4	YBFLf4_0	YBFLf4_1	YBFLf4_2	YBFLf4_3	YBFLf4_4	YBFLf4_5	YBFLf4_6
		15	YBFLf15_0	YBFLf15_1	YBFLf15_2	YBFLf15_3	YBFLf15_4	YBFLf15_5	YBFLf15_6
		16	YBFLf16_0	YBFLf16_1	YBFLf16_2	YBFLf16_3	YBFLf16_4	YBFLf16_5	YBFLf16_6
XiBFL	f	3	XiBFLf3_0	XiBFLf3_1	XiBFLf3_2	XiBFLf3_3	XiBFLf3_4	XiBFLf3_5	XiBFLf3_6
		4	XiBFLf4_0	XiBFLf4_1	XiBFLf4_2	XiBFLf4_3	XiBFLf4_4	XiBFLf4_5	XiBFLf4_6
		15	XiBFLf15_0	XiBFLf15_1	XiBFLf15_2	XiBFLf15_3	XiBFLf15_4	XiBFLf15_5	XiBFLf15_6
		16	XiBFLf16_0	XiBFLf16_1	XiBFLf16_2	XiBFLf16_3	XiBFLf16_4	XiBFLf16_5	XiBFLf16_6

注意事項:

- 對於 Y 方向 BFL 條件下的權重指標型記憶體：為 YBFLf3_0 ↔ YBFLParaF378_d[0] ,YBFLf4_0 ↔ YBFLParaF4910_d[0]
- 對於 Lagrange 權重指標陣列的元素的命名規則:
  - 二維陣列：矩陣名稱 [A] [B]
  - 矩陣名稱：變數符號 + Para + F + (內插牽涉的分佈函數編號) + _d
  - A：內插成員編號 (0 base)
  - B：空間位置點編號 (idx = j*NX6*NZ6+ *NX6 + i 或者是 idx_xi = j*NZ6+k)

Table 2: 實際使用權重指標型記憶體：double*（其中，F 一律為大寫）

變數	F	次編號	類型	Idx	索引 i						
					0	1	2	3	4	5	6
X	–	0	Para	0	XPara0_d[0]	XPara0_d[1]	XPara0_d[2]	XPara0_d[3]	XPara0_d[4]	XPara0_d[5]	XPara0_d[6]
		2	Para	2	XPara2_d[0]	XPara2_d[1]	XPara2_d[2]	XPara2_d[3]	XPara2_d[4]	XPara2_d[5]	XPara2_d[6]
Y	–	0	Para	0	YPara0_d[0]	YPara0_d[1]	YPara0_d[2]	YPara0_d[3]	YPara0_d[4]	YPara0_d[5]	YPara0_d[6]
		2	Para	2	YPara2_d[0]	YPara2_d[1]	YPara2_d[2]	YPara2_d[3]	YPara2_d[4]	YPara2_d[5]	YPara2_d[6]
Xi	F	3	Para	3	XiParaF3_d[0]	XiParaF3_d[1]	XiParaF3_d[2]	XiParaF3_d[3]	XiParaF3_d[4]	XiParaF3_d[5]	XiParaF3_d[6]
		4	Para	4	XiParaF4_d[0]	XiParaF4_d[1]	XiParaF4_d[2]	XiParaF4_d[3]	XiParaF4_d[4]	XiParaF4_d[5]	XiParaF4_d[6]
		5	Para	5	XiParaF5_d[0]	XiParaF5_d[1]	XiParaF5_d[2]	XiParaF5_d[3]	XiParaF5_d[4]	XiParaF5_d[5]	XiParaF5_d[6]
		6	Para	6	XiParaF6_d[0]	XiParaF6_d[1]	XiParaF6_d[2]	XiParaF6_d[3]	XiParaF6_d[4]	XiParaF6_d[5]	XiParaF6_d[6]
		15	Para	15	XiParaF15_d[0]	XiParaF15_d[1]	XiParaF15_d[2]	XiParaF15_d[3]	XiParaF15_d[4]	XiParaF15_d[5]	XiParaF15_d[6]
		16	Para	16	XiParaF16_d[0]	XiParaF16_d[1]	XiParaF16_d[2]	XiParaF16_d[3]	XiParaF16_d[4]	XiParaF16_d[5]	XiParaF16_d[6]
		17	Para	17	XiParaF17_d[0]	XiParaF17_d[1]	XiParaF17_d[2]	XiParaF17_d[3]	XiParaF17_d[4]	XiParaF17_d[5]	XiParaF17_d[6]
		18	Para	18	XiParaF18_d[0]	XiParaF18_d[1]	XiParaF18_d[2]	XiParaF18_d[3]	XiParaF18_d[4]	XiParaF18_d[5]	XiParaF18_d[6]
XBFL	F	37	BFL	37	XBFLParaF37_d[0]	XBFLParaF37_d[1]	XBFLParaF37_d[2]	XBFLParaF37_d[3]	XBFLParaF37_d[4]	XBFLParaF37_d[5]	XBFLParaF37_d[6]
		38	BFL	38	XBFLParaF38_d[0]	XBFLParaF38_d[1]	XBFLParaF38_d[2]	XBFLParaF38_d[3]	XBFLParaF38_d[4]	XBFLParaF38_d[5]	XBFLParaF38_d[6]
		49	BFL	49	XBFLParaF49_d[0]	XBFLParaF49_d[1]	XBFLParaF49_d[2]	XBFLParaF49_d[3]	XBFLParaF49_d[4]	XBFLParaF49_d[5]	XBFLParaF49_d[6]
		410	BFL	410	XBFLParaF410_d[0]	XBFLParaF410_d[1]	XBFLParaF410_d[2]	XBFLParaF410_d[3]	XBFLParaF410_d[4]	XBFLParaF410_d[5]	XBFLParaF410_d[6]
YBFL	F	3	BFL	3	YBFLParaF378_d[0]	YBFLParaF378_d[1]	YBFLParaF378_d[2]	YBFLParaF378_d[3]	YBFLParaF378_d[4]	YBFLParaF378_d[5]	YBFLParaF378_d[6]
		4	BFL	4	YBFLParaF4910_d[0]	YBFLParaF4910_d[1]	YBFLParaF4910_d[2]	YBFLParaF4910_d[3]	YBFLParaF4910_d[4]	YBFLParaF4910_d[5]	YBFLParaF4910_d[6]
		15	BFL	15	YBFLParaF15_d[0]	YBFLParaF15_d[1]	YBFLParaF15_d[2]	YBFLParaF15_d[3]	YBFLParaF15_d[4]	YBFLParaF15_d[5]	YBFLParaF15_d[6]
		16	BFL	16	YBFLParaF16_d[0]	YBFLParaF16_d[1]	YBFLParaF16_d[2]	YBFLParaF16_d[3]	YBFLParaF16_d[4]	YBFLParaF16_d[5]	YBFLParaF16_d[6]
XiBFL	F	3	BFL	3	XiBFLParaF3_d[0]	XiBFLParaF3_d[1]	XiBFLParaF3_d[2]	XiBFLParaF3_d[3]	XiBFLParaF3_d[4]	XiBFLParaF3_d[5]	XiBFLParaF3_d[6]
		4	BFL	4	XiBFLParaF4_d[0]	XiBFLParaF4_d[1]	XiBFLParaF4_d[2]	XiBFLParaF4_d[3]	XiBFLParaF4_d[4]	XiBFLParaF4_d[5]	XiBFLParaF4_d[6]
		15	BFL	15	XiBFLParaF15_d[0]	XiBFLParaF15_d[1]	XiBFLParaF15_d[2]	XiBFLParaF15_d[3]	XiBFLParaF15_d[4]	XiBFLParaF15_d[5]	XiBFLParaF15_d[6]
		16	BFL	16	XiBFLParaF16_d[0]	XiBFLParaF16_d[1]	XiBFLParaF16_d[2]	XiBFLParaF16_d[3]	XiBFLParaF16_d[4]	XiBFLParaF16_d[5]	XiBFLParaF16_d[6]

二　專論: XPara0_h[6][NX], XPara2_h[6][NX], YPara0_h[6][NY], YPara2_h[6][NY] 的意義

簡單說，這四者均為與插值有關的參數 (權重)，分為以下函數開始分析:

二、1　函數式: Lagrange_6th

```
double Lagrange_6th(
double x,double x_i, //1.待內插點2.輪到誰
double x_1,double x_2,double x_3,double x_4,double x_5,double x_6//內插成員，跳過x_i，只以1,2,3,4,5,6表示
){
  double Para = (x-x_1)/(x_i-x_1)*(x-x_2)/(x_i-x_2)*(x-x_3)/(x_i-x_3)*(x-x_4)/(x_i-x_4)*(x-x_5)/(x_i-x_5)*(x-x_6)/(x_i-x_6) ;
  return Para ; //Para為對應f_{i}的比例權重
}
```

Listing 1: initializationTool.h52

上述程式碼為連乘符號的程式碼具象化，對應在求和符號中的編號 i。

$$\texttt{Lagrange\_6th(x,x\_i,x\_1,x\_2,x\_3,x\_4,x\_5,x\_6)} = \prod_{j=1 j\neq i}^{6} \frac{\texttt{x - x\_j}}{\texttt{x\_i - x\_j}} \qquad (二.1)$$

二、2　函數式: GetParameter_6th

```
void GetParameter_6th(
double* Para_h[7] ,//Para_h[0],Para_h[1],...Para_h[6]作為一個指標陣列 尺寸為7
double Position ,//Position為真正要內插的位置
double* Pos ,//Pos作為一個指向double的指標變數//Pos 的型別 = double*；Para_h的型別 =
    double**
double i , double n//n為處理過後的內插成員index!
){//這是void函數，不會回傳，只需做某些事情
  Para[0][i]=Lagrange_6th(Position,Pos[n+0],Pos[n+1],Pos[n+2],Pos[n+3],Pos[n+4],Pos[n+5
    ],Pos[n+6]) ;
  Para[1][i]=Lagrange_6th(Position,Pos[n+1],Pos[n+0],Pos[n+2],Pos[n+3],Pos[n+4],Pos[n+5
    ],Pos[n+6]) ;
  Para[2][i]=Lagrange_6th(Position,Pos[n+2],Pos[n+0],Pos[n+1],Pos[n+3],Pos[n+4],Pos[n+5
    ],Pos[n+6]) ;
  Para[3][i]=Lagrange_6th(Position,Pos[n+3],Pos[n+0],Pos[n+1],Pos[n+2],Pos[n+4],Pos[n+5
    ],Pos[n+6]) ;
  Para[4][i]=Lagrange_6th(Position,Pos[n+4],Pos[n+0],Pos[n+1],Pos[n+2],Pos[n+3],Pos[n+5
    ],Pos[n+6]) ;
  Para[5][i]=Lagrange_6th(Position,Pos[n+5],Pos[n+0],Pos[n+1],Pos[n+2],Pos[n+3],Pos[n+4
    ],Pos[n+6]) ;
  Para[6][i]=Lagrange_6th(Position,Pos[n+6],Pos[n+0],Pos[n+1],Pos[n+2],Pos[n+3],Pos[n+4
    ],Pos[n+5]) ;
}
```

Listing 2: initializationTool.h61

(1). 如上可以看到，所謂對應關係**:**

1. Position $\longleftrightarrow$ x
2. Pos[n] $\longleftrightarrow$ x_i
3. Pos[n+1] $\longleftrightarrow$ x_1
4. Pos[n+2] $\longleftrightarrow$ x_2
5. Pos[n+3] $\longleftrightarrow$ x_3
6. Pos[n+4] $\longleftrightarrow$ x_4
7. Pos[n+5] $\longleftrightarrow$ x_5
8. Pos[n+6] $\longleftrightarrow$ x_6

他把要內插的各個點用陣列元素取代，其中，第幾個元素的"n" 代表處理過後的 index 編號。所以陣列元素 Pos[n+0,1,2,3,4,5,] 為實際內插成員，而 n 為 Position 最靠近的網格點的編號-3，為什麼是-3，因為六階 Lagrange 內插插中間那個非空間計算點的位置。

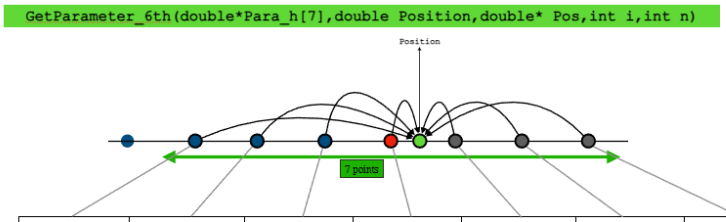
n 是 Position 的最靠近網格點的編號-3

換言之，還有以下結論:

Position 的最靠近網格點 = Pos[n+3]

(2). Para_h[0 6][i] 是什麼**?**

1. Para_h[0][i]：對應位置編號 n+0 的連乘權重係數
2. Para_h[1][i]：對應位置編號 n+1 的連乘權重係數
3. Para_h[2][i]：對應位置編號 n+2 的連乘權重係數
4. Para_h[3][i]：對應位置編號 n+3 的連乘權重係數
5. Para_h[4][i]：對應位置編號 n+4 的連乘權重係數
6. Para_h[5][i]：對應位置編號 n+5 的連乘權重係數
7. Para_h[6][i]：對應位置編號 n+6 的連乘權重係數



	Pos[n+0]	Pos[n+1]	Pos[n+2]	Pos[n+3]	Pos[n+4]	Pos[n+5]	Pos[n+6]
Para_h[0][i]	x_i	x_1	x_2	x_3	x_4	x_5	x_6
Para_h[1][i]	x_1	x_i	x_2	x_3	x_4	x_5	x_6
Para_h[2][i]	x_1	x_2	x_i	x_3	x_4	x_5	x_6
Para_h[3][i]	x_1	x_2	x_3	x_i	x_4	x_5	x_6
Para_h[4][i]	x_1	x_2	x_3	x_4	x_i	x_5	x_6
Para_h[5][i]	x_1	x_2	x_3	x_4	x_5	x_i	x_6
Para_h[6][i]	x_1	x_2	x_3	x_4	x_5	x_6	x_i

Figure 二.1: Para_h[0][i] 的對應關係

$$\text{GetParameter\_6th}(\text{Para}[7], \text{Position}, ...) \implies \begin{bmatrix} \texttt{Para\_h[0][i]} \\ \texttt{Para\_h[1][i]} \\ \texttt{Para\_h[2][i]} \\ \texttt{Para\_h[3][i]} \\ \texttt{Para\_h[4][i]} \\ \texttt{Para\_h[5][i]} \\ \texttt{Para\_h[6][i]} \end{bmatrix} = \begin{bmatrix} \prod_{j=1 j\neq i}^{6} \frac{\texttt{Position - Pos[n+j]}}{\texttt{Pos[n+0] - Pos[n+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{Position - Pos[n+j]}}{\texttt{Pos[n+1] - Pos[n+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{Position - Pos[n+j]}}{\texttt{Pos[n+2] - Pos[n+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{Position - Pos[n+j]}}{\texttt{Pos[n+3] - Pos[n+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{Position - Pos[n+j]}}{\texttt{Pos[n+4] - Pos[n+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{Position - Pos[n+j]}}{\texttt{Pos[n+5] - Pos[n+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{Position - Pos[n+j]}}{\texttt{Pos[n+6] - Pos[n+j]}} \end{bmatrix} \quad (二.2)$$

(3). 指標陣列與一維指標的索引行為

為什麼宣告一個「指標陣列」後，可用 **Para_h[0][i]?**

- 若 Para_h 型別為 double **，它本身是一個「指向指標的指標」。Para_h[0] 等價於 *(Para_h + 0)，取出第一個「列指標」；再索引 [i]，等價於 *(*(Para_h + 0) + i)，取得該列的第 i 個元素。
- 要成立，Para_h 必須先指向一個「指標陣列」(每個元素是一列的起始位址)，且每個列指標也必須指向已配置的連續元素區塊。常見作法：

  - 先配置「連續指標記憶體」：Para_h = (double**)malloc(Ny * sizeof(double*))；
  - 每一元素再配置一塊連續記憶體，每一塊連續記憶體的初始存放位址就是該元素：Para_h[y] = (double*)malloc(Nx * sizeof(double))。
- 如果 Para_h 未初始化或列指標未配置，Para_h[0][i] 只是對隨機位址做兩次間接存取，屬未定義行為。

為什麼只宣告 Pos 指標就能寫 Pos[n]?

- 在 C/C++ 中，Pos[n] 被編譯成 *(Pos + n) 的指標算術；編譯器不會檢查記憶體是否已配置，也不知道有效長度。
- 若 Pos 尚未指向合法的連續區塊（例如尚未 malloc/new，或指向的區塊長度不足），Pos[n] 會解參考未定義位址，仍屬未定義行為，可能讀到垃圾值、撞到保護頁、或靜悄悄破壞他處資料。
- 正確流程：先配置或讓 Pos 指向已存在的陣列，再用索引；並確保 n 在配置長度內。例如

```
double *Pos = (double*)malloc(N * sizeof(double)); // 或指向現有陣列
/* 使用 Pos[0..N-1] */
free(Pos);
```

Listing 3: 函數內宣告的自由性

重點

- 語法允許用 p[n] 是因為它只是指標算術，不代表記憶體已合法。
- 安全性取決於指標是否指向足夠長度的已配置（或靜態存在的）連續記憶體；否則都是未定義行為。
- 在函數括號內所宣告的指標型記憶體，如 GetParameter(double* Pos....)，自動將其映射為一維矩陣 Pos[?] ，其陣列大小由其他宣告之變數決定。

(4). i 是什麼?

i 是 (z(G(A 非物理空間計算點) 的 (最靠近物理空間計算點)) 的 (編號))

```
void GetParameter_6th(
//1.
輸出: 有關於該最近網格點的各個內插權重連乘數(二維陣列描述)(第一格放:內插編號:0~6，第二格放該網格點位置)
//2.
輸入:非物理空間點位置
//3.
內插成員的實際位置陣列的初始存放位址
//4.
最靠近的網格點的編號
//5.
最靠近的網格點的編號-3 //內插起始編號
)
```

Listing 4: GetParameter_6th 函數簽名

舉例: y 方向 (*Stream − wise*) 方向預先配置內插權重

```
//initialization.h217-219
for(int i = 3 ; i < NYD6-3 ; i++){//double*(陣列),double,double*,int,int
    GetParameter_6th(YPara0_h,y_h[i]-minsize,y_h[i-3],i,i-3) ; //y方向預配置內拆權重係數二維陣列
    GetParameter_6th(YPara2_h,y_h[i]+minsize,y_h[i-3],i,i-3) ; //y方向預配置內拆權重係數二維陣列
}
```

- GetParameter_6th 的第一個參數寫成 double* Para_h[7]（或你口述的 double*），但在函式參數位置「陣列形參」會自動退化成指標；double* Para_h[7] 退化成 double** Para_h。因此呼叫時可以傳入 double** （例如 YPara0_h 如果宣告成 double** 或 double *YPara0_h[7]），編譯器會視為「指向指標的指標」，型別就對得上。換句話說，形參看起來像一維指標陣列，但在函式裡實際型別是 double**，所以傳入 double**/指標陣列都是合法的。
- 函數內定義的 double* Para_h[7] 的初始存放位址 = double** Para_h ，但是在 C 語言中，兩者等效，換言之，後續產生的二維陣列的列數，要由自己決定。

上述程式碼你得到什麼?

$$\text{GetParameter\_6th}(...) \Longrightarrow \begin{bmatrix} \texttt{YPara\_h[0][i]} \\ \texttt{YPara\_h[1][i]} \\ \texttt{YPara\_h[2][i]} \\ \texttt{YPara\_h[3][i]} \\ \texttt{YPara\_h[4][i]} \\ \texttt{YPara\_h[5][i]} \\ \texttt{YPara\_h[6][i]} \end{bmatrix} = \begin{bmatrix} \prod_{j=1 j\neq i}^{6} \frac{\texttt{y\_h[i]-minsize - y\_h[i-3+j]}}{\texttt{y\_h[i-3+0] - y\_h[i-3+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{y\_h[i]-minsize - y\_h[i-3+j]}}{\texttt{y\_h[i-3+1] - y\_h[i-3+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{y\_h[i]-minsize - y\_h[i-3+j]}}{\texttt{y\_h[i-3+2] - y\_h[i-3+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{y\_h[i]-minsize - y\_h[i-3+j]}}{\texttt{y\_h[i-3+3] - y\_h[i-3+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{y\_h[i]-minsize - y\_h[i-3+j]}}{\texttt{y\_h[i-3+4] - y\_h[i-3+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{y\_h[i]-minsize - y\_h[i-3+j]}}{\texttt{y\_h[i-3+5] - y\_h[i-3+j]}} \\ \prod_{j=1 j\neq i}^{6} \frac{\texttt{y\_h[i]-minsize - y\_h[i-3+j]}}{\texttt{y\_h[i-3+6] - y\_h[i-3+j]}} \end{bmatrix} \quad (二.3)$$

其中，產生的二維陣列 (我把它當成一維陣列) YPara[A][B] 中，A 代表內插成員之編號，B 為非物理空間計算點的最近物理空間計算點的編號。
(1.) 在 GetParametert_6 一使用時，相對於指定的物理間計算點 (i)，七個內插成員隨之確定，與非物理空間點的位置無關，所以 GetParametert_6 是 be based on "i" 的。(2.) 不過一般而言，Position 的最近物理空間計算點的編號 =i。

(5).GetParameter 函數式的呼叫與統計

1. 產生 x/y/xi 方向預配置連乘權重一維陣列
   - 產生 xi 方向預配置連乘權重一維陣列

```
//(1.) xi/z方向 存在buffer layers
```

```
//(2.) xi/z方向 不取用buffer layers
//(3.) xi/z方向 的buffer layers 不做更新也沒有讀取使用
if(k>=3 && k <+6){//取下邊界靠近的幾個物理空間計算點
  GetParameter_6th(XiPara0_h,pos,Pos,
}else if(k>=NZ6-7 && k<=NZ6-4){//取上邊界靠近的幾個物理空間計算點
}else{
}

```

- 產生 x 方向預配置連乘權重一維陣列
- 產生 y 方向預配置連乘權重一維陣列

二、3 函數式：GetNonuniParameter

目的：讓物理間計算點的距離最小值與晶格節點的距離相等，為此程式碼作區間搜尋法之目的。

二、4 非均勻網格系統之座標映射：tanh 函數

本程式碼唯獨在 Z 方向採用座標映射，形成網格拉伸關係，在此理解上，先不考慮誰是均勻網格系統，非均勻網格系統，以及不管編號，單純先以連續域的區段的映射關係做討論:

$$z = \frac{L}{2a}\tanh\left(\frac{\xi}{2}\ln\left(\frac{1+a}{1-a}\right)\right) \tag{二.4}$$

其中，ξ 有如下定義:

$$\xi = -1 + 2\frac{j}{N} \tag{二.5}$$

對於上述映射關係:

$$z \in [-\frac{L}{2}, \frac{L}{2}] \rightarrow \xi \in [-1, 1] \rightarrow j \in [0, N] \tag{二.6}$$

由公式 (二.4)可知，各參數的名詞意義如下:

- z : 離散化 z 座標
- j : 離散化正規化 z 座標
- L : 物理空間總長度
- a : 控制網格拉伸程度的參數
- N : 計算點包圍的總長度

處理上，會將 j 作為正規化座標系統，作為實際運算上的坐標系，因此在對此區域 ($j \in [0, N/2]$) 做網格劃分時，會以均勻切割做處理。透過上述之映射關係，映射成 z 座標系統，範圍為 $z \in [-\frac{L}{2}, \frac{L}{2}]$ ，此連續區段的區域，通常被當成時真實物理域。若對於變數 $j \in [0, \frac{N}{x}]$ 採用均勻切割，則會在 z 座標系中形成非均勻網格系統。且兩套系統彼此一一對應。

$$j = 0, 1, 2, 3, ..., N \in [0, N] \longleftrightarrow z = z_0, z_1, z_2, z_3, ..., z_N \in [-\frac{L}{2}, \frac{L}{2}]$$

後續為讓 z 方向的第一個 Z 座標點能夠與第一個物理空間計算點重合，再對上式(二.7) 做兩段平移處理，用處有二，第一 : 先讓映射過後的 z 座標從 0 開始計算；第二 : 讓第一個 z 座標

計算點與第一個物理空間計算點重合。因此有下式之關係:

$$z = \frac{L}{2} + \frac{\texttt{LatticeSize}}{2} + \frac{L}{2a}\tanh\left(\frac{\xi}{2}\ln\left(\frac{1+a}{1-a}\right)\right) = \frac{L}{2a}\tanh\left(\frac{-1+2\frac{j}{N}}{2}\ln\left(\frac{1+a}{1-a}\right)\right) \quad (二.7)$$

程式碼如下呈現:

```
#define tanhFunction( L, LatticeSize, a, j, N )
(           \
    L/2.0 + LatticeSize/2.0 + ((L/2.0)/a)*tanh((-1.0+2.0*(double)(j)/(double)(N
    ))/2.0*log((1.0+a)/(1.0-a)))     \
)
```

Listing 5: Hyperbolic Tangent Function

平移 **minsize/2.0** 的意義: 乍看之下，平移此段距離為之意義為在“z 方向“與壁面抱持相距離 minsize/2.0 的距離。然而，minsize/2.0 是插值晶格波茲曼法中，沿單一方向的遷移距離。，所以當物理空間計算點的 z 方向邊界點與壁面在 z 方向上保持相距“半個” streaming” 距離時，就可以對於邊界點上的編號五碰撞後分布函數以“Half-way Bounce Back” 格式處理邊界條件。所以

讓每一個 z 方向下邊界點與壁面保持相距“半個” streaming” 距離

Hyperbolic Tangent Funtion 中的分割數，與實際應用上有所差異: 可是 j 作為正規化座標系統，N 為切割之網格數。因此，不管是正規化座標系統 j，還是映射過後的座標 z，均相對於 N (切割之網格數) 作為網格節點，所以預設上: 由方程式 (二.4)

1. $z_i \in [-\frac{L}{2}, \frac{L}{2}]$ 一共切割為 N+1 個
2. $j \in [0, N]$ 一共切割為 N+1 個

可是經由方程式平移後，(二.7)，j 正規化座標系統第一個點對應到第一個物理空間計算點，為第一個網格中心點。因此，若要定義 Z 座標或者 j 座標對應到網格中心點，則當切割數量為 N 時，公式中的 N 必須改為 N-1，如此，座標點的對應數量才是真正的切割數。
程式碼具體實現如下:

```
// double GetNonuniParameter()
x_temp[0] = tanhFunction(total, minSize, a_mid, 0, (NZ6-7));//他是放NZ6-7不是放
    入NZ6-6，但是Z方向網格數是NZ6-6，他放NZ6-7的意義在於讓真正的網格數量j 一共有NZ
    6-6個。
```

Listing 6: initializationbTool.h17

tanhFunction 的函數簽名：

```
#define tanhFunction(物理空間計算點所包圍的總長度,minsize,
控制網格拉伸的參數,正規化坐標,切割之網格數-1)
```

Listing 7: Hyperbolic Tangent Function 簽名

二、5 minSize 的物理意義：

定義：

```
#define     minSize             ((LZ-1.0)/(NZ6-6)*0.6)
```

從上式僅可看出，minSize 為 z 方向上，在均勻切割下，單一網格的 0.6 倍長。然而，minSize 的物理意義在於:（你怎麼使用它）

- minSize 被定義為 (T(A 非物理空間點) 跟 (G(A 非物理空間點) 的 (遷移後空間點)) 的 (距離))
- minSize 為單一 Lattice 的大小

minSize 的實際定義與物理空間計算點之間距分配:

- minSize : $(LZ - y = 0.0$的山坡高度$)/$(Z 方向物理空間計算點數量$) * 0.6$，因此，此量為預先配置。
- dx ：物理空間計算點間的最小距離
- 物理空間計算點一定作為在真實物理網格之節點
- 物理空間計算點必然為 Lattice 的中心點
- 計算 minSize 所用的網格數目為 Z 方向物理空間計算點數量；
- 計算 dx 所用的網格數目為 Z 方向物理空間計算點數量-1；

如下程式碼可以呈現這一點，關於 z 方向的參數如何選取:

```
double GetNonuniParameter() {
  double total = LZ - HillFunction( 0.0 ) - minSize;
  //選取物空間計算點的最大間距做為分配之總長度
  double a_temp[2] = {0.1, 1.0};
  double a_mid;
  double x_temp[2], dx;
  do{
    a_mid = (a_temp[0]+a_temp[1]) / 2.0;
    //dx = Z方向y = 0.0 非均勻網格下的最小間距 (從y=  0,0取)
    //minSize的定義為以LZ-(y = 0.0下的山坡高度)為總長度均勻切割下的網格大小*0.6
    //判斷標準 最小的網格必須要大於minSize
    x_temp[0] = tanhFunction(total, minSize, a_mid, 0, (NZ6-7));
    x_temp[1] = tanhFunction(total, minSize, a_mid, 1, (NZ6-7));
    dx = x_temp[1] - x_temp[0];
    //物理空間計算點的間距最小值
        if( dx - minSize >= 0.0 ){
            a_temp[0] = a_mid;
        } else {
            a_temp[1] = a_mid;
        }
    } while ( fabs( dx - minSize ) > 1e-14 );
  return a_mid; }
```

所以從上述可以預見，$minSize$ 可以預先配置，而 dx 則必須由 LatticeSize 確定了以後由 (y = 0.0) 下物理空間計算點的最大包圍長度計算分配決定。所以 dx 的分配邏輯為: 1. 物理空間

計算點作為真實物理網格之節點，2. 先決定好分配之真實物理網格之總長度，再分配物理空間計算點，其中，總長度的決定必須滿足 Half-way Bounce Back。所以是" 先定義 minSize"，再分配決定" 物理空間計算點間距"。(最小者為 dx 如上呈現)

二、6 (含山丘) 離散化區域 Z 座標：z_h：邊號從 3 開始，到 NZ6-4 結束

```
//此程式碼為動態定義(U(B物理間計算點)的(真實Z座標))擬合山丘高度，再同一個山丘高度
    下，定義z座標所對應矩陣編號: k = 3
for( int j = 0; j < NYD6; j++ ){
  double total = LZ - HillFunction( y_h[j] ) - minSize;
  for( int k = bfr; k < NZ6-bfr; k++ ){
    z_h[j*NZ6+k] = tanhFunction( total, minSize, a, (k-3), (NZ6-7) ) +
                   HillFunction( y_h[j] );
  }
  z_h[j*NZ6+2] = HillFunction( y_h[j] );
  z_h[j*NZ6+(NZ6-3)] = (double)LZ;
```

Listing 8: initializationTool.h120 定義 z_h

上述對於 tanhFunction 的使用，因為上述 tanhFunction 的簽名是簽入 j，可以發現: 對於真實 Z 座標矩陣，矩陣所對應的編號正是正規化 Z 方向座標系統 j。所以有

z_h[j*NZ+(3→NZ6-4)] = tanhFunction(total,minSize,a,正規化坐標系+3,(NZ6-7))+...

Table 3: z_h 座標系統

中文名稱	(含山丘) 離散化區域 Z 座標
對應關係	z_h [NZ * j + k]
自變數	NZ * j + k
j 定義域	$j \in [0\,,\ NYD6-1]$
k 定義域	$k \in 2\,,\ [3\,,\ NZ6-4]\,,\ NZ6-3$
值域	z_h[NZ * j + k] ∈ Hill(y_h[j]) , [Hill(y_h[j]) + minSize , LZ-minSize] , LZ

二、7 (不含山丘) 離散化無因次化 Z 座標：xi_h

如下為該變數的程式碼定義:

```
for(int i = 3 ; i < NZ6-4 ; i++){
  xi_h[k] = tanhFunction( LXi, minSize, a, (k-3), (NZ6-7) ) - minSize/2.0;
}
```

Listing 9: initializationTool.h130 定義 xi_h

Table 4: xi_h 坐標系統

中文名稱	(不含山丘) 離散化無因次化 Z 座標
對應關係	xi_h [k]
自變數	k
定義域	$k \in [3\ ,\ NZ6-4]$
值域	$xi\_h[k] \in [0\ ,\ 10]$

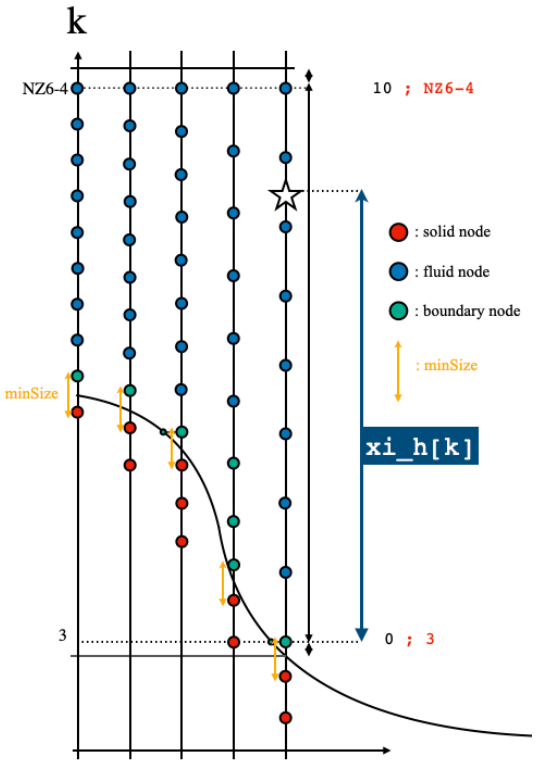


Figure 二.2: xi_h 座標系統示意圖

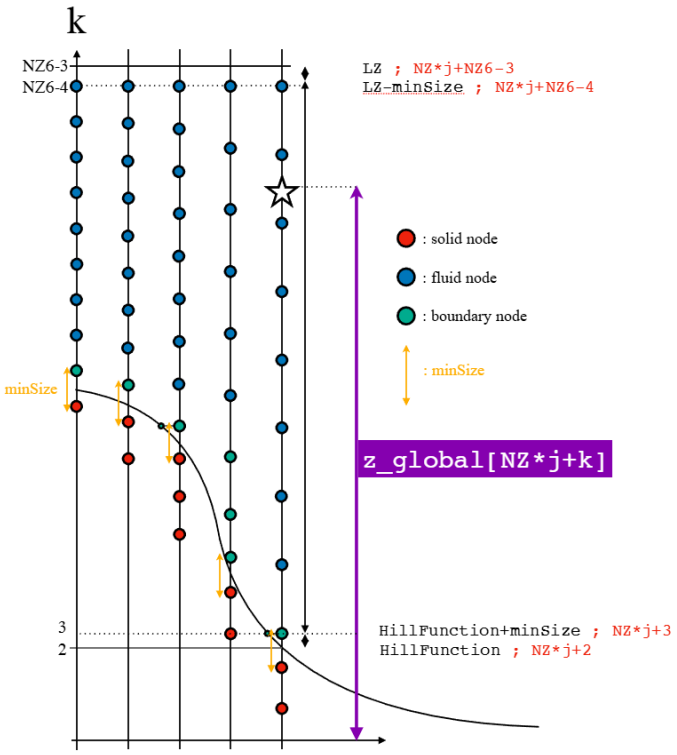


Figure 二.3: xi_h 座標系統示意圖

二、8 (含山丘) 離散化全域 Z 座標：z_global

如下為該變數的程式碼定義:

```
double  y_global[NY6] ; //宣告陣列尺寸之大小但是y_global並非全域變數
double  z_global[NY6*NZ6] ;//定義z_global輸出全域網格
for(int j = 0 ; j < NY6 ; j++){
  double dy = (LY-HillFunction(y_global[j])) / (NZ6-1);
  y_global[j] = j * dy ;
  for(int k = 3 ; k < NZ6-3 ; k++){
    z_global[j*NZ6+k] = tanhFunction(total,minSize,a,(k-3),(NZ6-7)) +
                      + HillFunction(y_global[j]) ;
  }
  z_global[j*NZ6+2] = HillFunction(y_global[j]) ;
  z_global[j*NZ6+(NZ6-3)] = (double)LZ ;
```

```
12 }
```

Listing 10: z_global 座標系統

如圖二.3 變量 z_global[NZ*j+k] 為物理空間計算點的 (含山丘) 離散化全域 Z 座標，當 z_global[NZ*j+k] =z_[3] 時，(定義域起始點為 3) 則其值代表山丘的表面：z_global[3] = HillFunction(y_global[j])
當 z_global[NZ*j+k]=z_[NZ6-3] 時，(定義域終止點為 NZ6-3) 則其值代表離散化區域的最高點 (在計算點區域高半個 minSize)：z_global[NZ6-3] = LZ。
與物理空間計算點的對應範圍為：

if k ∈ [3,NZ6-4] , z_global[NZ6*j+k] ∈ [HillFunction+0.5minSize , LZ-0.5minSize]

Table 5: z_global 坐標系統

中文名稱	**z_global** 坐標系統
對應關係	z_global [NZ * j + k]
自變數	NZ * j + k
j 定義域	$j \in [0\,,\, NY6-1]$
k 定義域	$k \in 2\,,\, [3\,,\, NZ6-4]\,,\, NZ6-3$
值域	z_global[NZ * j + k] ∈ Hill(y_global[j]) , [Hill(y_global[j]) + minSize , LZ-minSize] , LZ

三 專論: XPara0_h[6][NX] , XPara2_h[6][NX] , YPara0_h[6][NY] , YPara2_h[6][NY] 的意義 2

若定義式中 Position 填入 y_h[i]-minSize，則代表內插分成員為前 3 後 4 插。
若定義式中 Position 填入 y_h[i]+minSize，則代表內插分成員為前 4 後 3 插。

並快速給定對應關係：

$$\begin{bmatrix} \texttt{Y0\_0} \\ \texttt{Y0\_1} \\ \texttt{Y0\_2} \\ \texttt{Y0\_3} \\ \texttt{Y0\_4} \\ \texttt{Y0\_5} \\ \texttt{Y0\_6} \end{bmatrix} \longleftrightarrow \begin{bmatrix} \texttt{YPara0\_d[0][i]} \\ \texttt{YPara0\_d[1][i]} \\ \texttt{YPara0\_d[2][i]} \\ \texttt{YPara0\_d[3][i]} \\ \texttt{YPara0\_d[4][i]} \\ \texttt{YPara0\_d[5][i]} \\ \texttt{YPara0\_d[6][i]} \end{bmatrix} \begin{bmatrix} \texttt{Y2\_0} \\ \texttt{Y2\_1} \\ \texttt{Y2\_2} \\ \texttt{Y2\_3} \\ \texttt{Y2\_4} \\ \texttt{Y2\_5} \\ \texttt{Y2\_6} \end{bmatrix} \longleftrightarrow \begin{bmatrix} \texttt{YPara2\_d[0][i]} \\ \texttt{YPara2\_d[1][i]} \\ \texttt{YPara2\_d[2][i]} \\ \texttt{YPara2\_d[3][i]} \\ \texttt{YPara2\_d[4][i]} \\ \texttt{YPara2\_d[5][i]} \\ \texttt{YPara2\_d[6][i]} \end{bmatrix} \tag{三.1}$$

$$\begin{bmatrix} X0\_0 \\ X0\_1 \\ X0\_2 \\ X0\_3 \\ X0\_4 \\ X0\_5 \\ X0\_6 \end{bmatrix} \longleftrightarrow \begin{bmatrix} XPara0\_d[0][i] \\ XPara0\_d[1][i] \\ XPara0\_d[2][i] \\ XPara0\_d[3][i] \\ XPara0\_d[4][i] \\ XPara0\_d[5][i] \\ XPara0\_d[6][i] \end{bmatrix} \begin{bmatrix} X2\_0 \\ X2\_1 \\ X2\_2 \\ X2\_3 \\ X2\_4 \\ X2\_5 \\ X2\_6 \end{bmatrix} \longleftrightarrow \begin{bmatrix} XPara2\_d[0][i] \\ XPara2\_d[1][i] \\ XPara2\_d[2][i] \\ XPara2\_d[3][i] \\ XPara2\_d[4][i] \\ XPara2\_d[5][i] \\ XPara2\_d[6][i] \end{bmatrix} \tag{三.2}$$

如下以 XPara0_h 與 XPara0_d 為例子說明作為二為陣列如何宣告，定義，以及記憶體配置，與使用：

- **XPara0_h[A][B] 與 XPara0_d[A][B] 的宣告：**
  宣告記憶體為宣告一個尺寸為 7 的指標陣列。

```
double //宣告為一維指標陣列
*XPara0_h[7],*XPara0_d[7],*XPara2_h[7],*XPara2_d[7],
*YPara0_h[7],*YPara0_d[7],*YPara2_h[7],*YPara2_d[7],
```

Listing 11: main.cu45

- **XPara0_h[A][B] 與 XPara0_d[A][B] 的記憶體配置為：**

```
for( int i = 0; i < 7; i++ ){
  //memory.h81-86
  CHECK_CUDA( cudaMallocHost( (void**)&YPara0_h[i], nBytes ) );
  //對主機變數
  CHECK_CUDA( cudaMallocHost( (void**)&YPara2_h[i], nBytes ) );
  CHECK_CUDA( cudaMalloc( &YPara0_d[i], nBytes ) );
  //對GPU變數
  CHECK_CUDA( cudaMalloc( &YPara2_d[i], nBytes ) );
}
for( int i = 0; i < 7; i++ ){
  CHECK_CUDA( cudaMallocHost( (void**)&XPara0_h[i], nBytes ) );
  //對主機變數
  CHECK_CUDA( cudaMallocHost( (void**)&XPara2_h[i], nBytes ) );
  CHECK_CUDA( cudaMalloc( &XPara0_d[i], nBytes ) );
  //對GPU變數
  CHECK_CUDA( cudaMalloc( &XPara2_d[i], nBytes ) );
}
```

Listing 12: memory.h81-86,91-96

- **XPara0_h[A][B] 定義與 XPara0_d[A][B] 的傳遞：**
  本部分為先定義主機端二維陣列，作為單一方向預配置一維連乘權重陣列，再將此定義好的主機端記憶體透過指標陣列的方式傳遞給 GPU 端。換言之，二維度係樹之配置為先在主機端定義好，再將主機端的資料往 GPU 端傳遞。

```
void GetIntrplPara_X(){
for(int i = 3 , i <= NX6-4 ; i++){
//GetParametert_6th(指定二為陣列的初始存放位址的位址,非物理空間計算點,七個物
    理空間計算點,指定的物理空間計算點的編號指定的物理空間計算點的編號-3.)
  GetParameter_6th( XPara0_h, x_h[i]-minSize, x_h, i, i-3 );
  GetParameter_6th( XPara2_h , x_h[i]+minSize, x_h, i, i-3);
}
//將主機端的內插權重係數二維陣列傳遞給GPU端
for(int i = 3 ; i <= NX6-4 ; i++){
  CHECK_CUDA( cudaMemcpy(XPara0_d[i], XPara0_h[i], NX6*sizeof(double),
    cudaMemcpyHostToDevice) );
  CHECK_CUDA( cudaMemcpy(XPara2_d[i], XPara2_h[i], NX6*sizeof(double),
    cudaMemcpyHostToDevice) );
}//cudaMemcpyHostToDevice 為 cudaMemcpy()的其中一個指令參數
}
```

Listing 13: initialization.h201

如上可以預見：**XPara0_h[A][B]** 與 **XPara0_d[A][B]** 的意義為：
X 方向預配置一維連乘權重陣列，其中 A+i 代表內插成員的編號，B 代表物理空間計算點的編號。分析函數式：

```
GetParameter_6th( XPara0_h, x_h[i]-minSize, x_h, i, i-3 );
                              ⇓
        XPara0_h[0][i] = 內插成員 1 的連乘權重係數
        XPara0_h[1][i] = 內插成員 2 的連乘權重係數
        XPara0_h[2][i] = 內插成員 3 的連乘權重係數
        XPara0_h[3][i] = 內插成員 4 的連乘權重係數
        XPara0_h[4][i] = 內插成員 5 的連乘權重係數
        XPara0_h[5][i] = 內插成員 6 的連乘權重係數
        XPara0_h[6][i] = 內插成員 7 的連乘權重係數
```

當 i 值掃描到某一個物理座標時 (從 3 到 NX6-4)，此 x 方向預配置連乘權重一維陣列為相應於該物理空間計算點的右側一個 minSize 的距離之非物理空間計算點的 Lagrange 七點內插係數。
分析函數式：

```
GetParameter_6th( XPara2_h , x_h[i]+minSize, x_h, i, i-3);
                              ⇓
        XPara2_h[0][i] = 內插成員 1 的連乘權重係數
        XPara2_h[1][i] = 內插成員 2 的連乘權重係數
        XPara2_h[2][i] = 內插成員 3 的連乘權重係數
        XPara2_h[3][i] = 內插成員 4 的連乘權重係數
        XPara2_h[4][i] = 內插成員 5 的連乘權重係數
        XPara2_h[5][i] = 內插成員 6 的連乘權重係數
```

XPara2_h[6][i] = 內插成員 7 的連乘權重係數

- XPara0_h[A][B] 與 XPara0_d[A][B] 的使用：

四　專論：XiParaF3_d[6][NZ6*NYD6] 的意義:

四、1　分配記憶體

四、2　函數式：GetXiParameter

此函數式為一套建模組函數式：目標為寫入對應輸入的實際位置點 pos_z, pos_y 的 z 方向預配置權重一維連續記憶體的某一個元素，程式碼如下：

```
void GetXiParameter(double** XiPara, double pos_z, double pos_y , double* xi_h,
double IdxToStore , double k){
  double L = LZ - HillFunction(pos_y) - minSize;
  double pos_xi = (LXi/L)*(pos_z - HillFunction(pos_y) - 0.5*minSize ;
  if(k>=3 && k <= NZ6-4){//也是配置單方向上的一維權重陣列
    GetParameter_6th(XiPara, pos_xi, xi_h, IdxToStore,k-3);
  }else if(k<3 && k >= 0){
    GetParameter_6th(XiPara, pos_xi, xi_h, IdxToStore,3);
  }else if(k >= NZ6-3 && k < NZ6){
    GetParameter_6th(XiPara, pos_xi, xi_h, IdxToStore,NZ6-4);
  }}
```

從程式碼可以看出，命名 L 的目的為計算不含山丘無因次化 Z 座標 pos_xi。再利用 Getparameter_6th 配合 (不含山丘) 離散化無因次化 Z 座標 xi_h，寫入 Z 方向預配置權重一維記憶體。

為什麼需要兩個參數: **pos_y, pos_z**？ 在實際使用上，pos_y, pos_z 會填入以下形式：

```
GetXiParameter(XiParaFe_d , z[j*NZ6+k] , y_h[j] + minSize , xi_h , j*NZ6+k , k
    );
```

可以發現，pos_y 填入為 y_h[j] + minSize，而 pos_z 填入為 z[j*NZ6+k]。

實際上寫入 XiParaF3_d[m][i] 元素的函數為 GetParameter_6th(XiParaF3_d,pos_xi,xi_h,...)，換言之

$$\texttt{XiParaF3\_d[0][index]} = \prod_{i=0,\ i\neq 0}^{6} \frac{\texttt{pos\_xi} - \texttt{xi\_h[index-3+i]}}{\texttt{xi\_h[index-3+0]} - \texttt{xi\_h[index-3+i]}} \qquad (四.1)$$

所以參與單一位置點，六項權重係數的寫入實際上還是用到單一方向的量：pos_xi, xi_h[index]，只是因為因為物理空間計算點所包圍的長度 total 隨著不同 y 值而變，作為一個定義域為一維的函數式 total = total(y) 導致同一個高度的 Z 座標因為在不同 y 值，其無因次化 Z 座標 pos_xi 會不同，所以需要兩個參數來決定實際寫入的權重係數，也因此，為了明白起見，我們將方程式(四.1)寫為如下：

$$\texttt{XiParaF3\_d[0][index]} = \prod_{i=0,\ i\neq 0}^{6} \frac{\texttt{pos\_xi(pos\_y, pos\_z)} - \texttt{xi\_h[index-3+i]}}{\texttt{xi\_h[index-3+0]} - \texttt{xi\_h[index-3+i]}} \qquad (四.2)$$

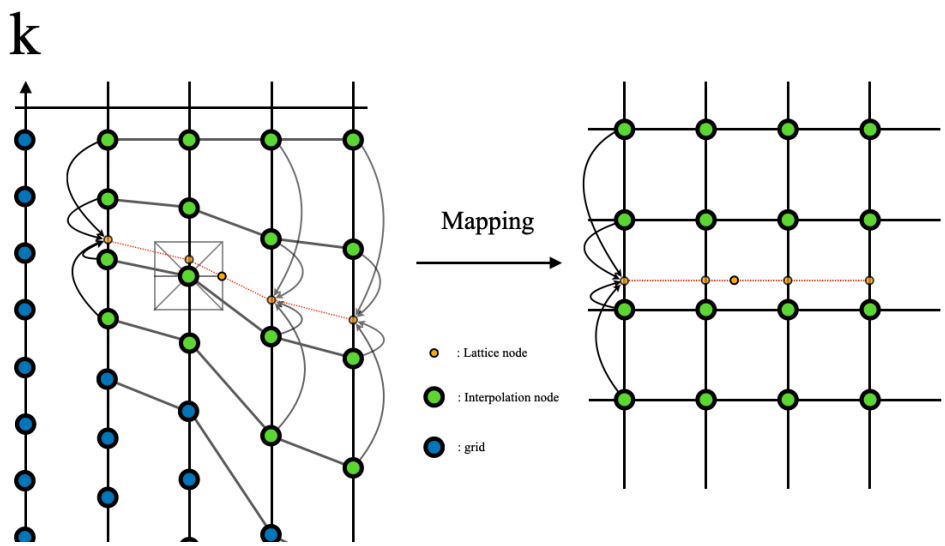


Figure 四.4: Z 方向權重插值映射關係

曲線坐標系下的一維 **Lagrange** 內插

- (Z_1 (A (c 一維 Lagrange 內插) 的 (非物理間計算點)) 的 (座標)) 被定義為 (G 無因次化座標)
- (Z_2 (B (c 一維 Lagrange 內插) 的 (物理間計算點)) 的 (座標)) 被定義為 (Q 離散化無因次化座標)
- (n_1 (B_1 (c 一維 Lagrange 內插) 的 (物理間計算點)) 的 (垂直座標)) 被定義為 (n_2 (B_2 (c 一維 Lagrange 內插) 的 (物理間計算點)) 的 (垂直座標))
- (n_3 (A (c 一維 Lagrange 內插) 的 (非物理間計算點)) 的 (垂直座標)) 被定義為 (n_2 (B_2 (c 一維 Lagrange 內插) 的 (物理間計算點)) 的 (垂直座標))

因此，才有如圖四.4所顯示的無因次化技巧。其中，程式碼操作技巧為：

```
double L = LZ - HillFunction(pos_y) - minSize;//截取該位置的總長度
double pos_xi = (LXi/L)*(pos_z - HillFunction(pos_y) - 0.5*minSize );//此Z座標
    為一維Lagrange內插的基準座標
```

如上，則顯示 Mappig 過程中，的座標轉換。

四、3 實際寫入內容：GetIntrplParameter_Xi

為什麼需要預配置權重陣列的第二個參數尺寸為二維：NZ6*NYD6 因為在 y_z 平面上，每一個物理空間計算點的相應非物理空間計算點的 Z 方向預配置連乘權重一維連續記憶體都不一樣，會受到座標比例影響內插計算結果。所以需要配置二維記憶體，第一個座標為內插成員編號，第二個座標為物理空間計算點在 y_z 平面上的一維座標索引。

```
void GetIntrplParameter_Xi(){
//目的，寫入尺寸為7*(NZ6*NYD6)的二維連續記憶體，將每個位置點各個編號的連乘權重寫各
    個元素
for(int j = 3 ; j <= NZ6-4 ; j++){

}
}
```

四、4　雙重索引結構解析

在 XiParaF3_d 二維陣列中，索引 [m][i] 具有明確的物理與數值意義：

第一層索引 **[m]** **(**$m = 0, 1, \ldots, 6$**)**　選擇 7 個 Lagrange 基函數中的第 $m+1$ 個：

$$\texttt{XiParaF3\_d[0][i]} \to L_1(\xi^*) \text{ 的係數} \tag{四.3}$$

$$\texttt{XiParaF3\_d[1][i]} \to L_2(\xi^*) \text{ 的係數} \tag{四.4}$$

$$\vdots$$

$$\texttt{XiParaF3\_d[6][i]} \to L_7(\xi^*) \text{ 的係數} \tag{四.5}$$

第二層索引 **[i]** **(空間位置)**　線性化的二維空間索引：

$$i = j \times \text{NZ6} + k \tag{四.6}$$

其中：

- $j \in [0, \text{NYD6} - 1] = [0, 38]$：Y 方向格點索引
- $k \in [0, \text{NZ6} - 1] = [0, 69]$：Z 方向格點索引
- $i \in [0, \text{NYD6} \times \text{NZ6} - 1] = [0, 2729]$：線性化索引

四、5　具體數值範例

範例 **1**：邊界格點　考慮 Y-Z 平面的邊界格點 $(j = 3, k = 3)$（底部緩衝區邊界）：

$$i = 3 \times 70 + 3 = 213 \tag{四.7}$$

則：

- XiParaF3_d[0][213] = 第 1 個 Lagrange 基函數在 $(j = 3, k = 3)$ 的值
- XiParaF3_d[3][213] = 第 4 個 Lagrange 基函數在 $(j = 3, k = 3)$ 的值（中心點）
- XiParaF3_d[6][213] = 第 7 個 Lagrange 基函數在 $(j = 3, k = 3)$ 的值

範例 **2**：內部格點　考慮內部格點 $(j = 20, k = 35)$：

$$i = 20 \times 70 + 35 = 1435 \tag{四.8}$$

在 F3 方向進行插值時：

$$\begin{aligned} f_3^{\text{interp}}(20, 35) &= \sum_{m=0}^{6} f_4(\xi_{m+1}, 20, 35) \cdot \texttt{XiParaF3\_d[m][1435]} \\ &= f_4(\xi_1) \cdot \texttt{XiParaF3\_d[0][1435]} \\ &\quad + f_4(\xi_2) \cdot \texttt{XiParaF3\_d[1][1435]} \\ &\quad + \cdots \\ &\quad + f_4(\xi_7) \cdot \texttt{XiParaF3\_d[6][1435]} \end{aligned} \tag{四.9}$$

四、6 物理意義詮釋

F3 方向的特殊性 F3 方向為 $(0, +1, 0)$，代表純 Y 方向的正向流動。但在 ISLBM 中，由於 Periodic Hill 幾何的影響，實際的流線並非直線，而是沿著變換後的 ξ 座標追蹤。

為何需要 **Y-ξ 二維插值** 雖然 F3 的速度向量只有 Y 分量，但由於：

1. Periodic Hill 的高度隨 Y 變化：$H = H(y)$
2. Z 方向網格非均勻：使用 ξ 變換
3. 流線在 Y-Z 平面上彎曲

因此需要在 Y-ξ 平面進行二維插值，而非單純的 Y 方向一維插值。

四、7 記憶體佈局與效能考量

陣列記憶體佈局 `XiParaF3_d` 在 GPU 記憶體中的實際配置：

```
XiParaF3_d[0] -> 連續記憶體區塊 [NYD6×NZ6 個 double]

                 [0][1][2]...[2729]

XiParaF3_d[1] -> 連續記憶體區塊 [NYD6×NZ6 個 double]

                 [0][1][2]...[2729]

  ...
XiParaF3_d[6] -> 連續記憶體區塊 [NYD6×NZ6 個 double]

                 [0][1][2]...[2729]

```

Coalesced Memory Access 在 CUDA kernel 中，當相鄰 thread 存取相鄰的 `i` 值時：

- Thread 0 讀取 `XiParaF3_d[m][i]`
- Thread 1 讀取 `XiParaF3_d[m][i+1]`
- Thread 2 讀取 `XiParaF3_d[m][i+2]`
- ...

由於這些記憶體位址連續，GPU 可以將它們合併為單次記憶體交易（coalesced access），大幅提升效能。

快取友善性 當計算某格點的插值時，需要連續讀取：

$$\texttt{XiParaF3\_d[0][i]}, \texttt{XiParaF3\_d[1][i]}, \ldots, \texttt{XiParaF3\_d[6][i]} \tag{四.10}$$

雖然這 7 個值位於不同的陣列，但若 `i` 相同，它們的記憶體位址模式規律，有利於 L1/L2 快取預取。

四、8　程式碼追蹤

初始化階段（**initialization.h:201–256**）　計算每個格點 (j,k) 的 Lagrange 係數：

```
for j in range(NYD6):
    for k in range(NZ6):
        i = j * NZ6 + k
         _target = GetTargetXi(j, k)  # 目標   座標

        for m in range(7):
            XiParaF3_h[m][i] = Lagrange_6th(
                _target,
                _m,       # 第 m+1 個基點
                _1, _2, ..., _7  # 全部 7 個基點
            )
```

記憶體傳輸（**initialization.h**）　將計算好的係數從 Host 傳至 Device：

```
for m in range(7):
    cudaMemcpy(
        XiParaF3_d[m],           # Device 目標
        XiParaF3_h[m],           # Host 來源
        NYD6 * NZ6 * sizeof(double),
        cudaMemcpyHostToDevice
    )
```

使用階段（**interpolationHillISLBM.h:F3_Intrpl7**）　在 kernel 中執行插值：

```
__device__ void F3_Intrpl7(...) {
    int i = j * NZ6 + k;

    F3_in = f4_old[ _indices[0]] * XiParaF3_d[0][i]
          + f4_old[ _indices[1]] * XiParaF3_d[1][i]
          + f4_old[ _indices[2]] * XiParaF3_d[2][i]
          + f4_old[ _indices[3]] * XiParaF3_d[3][i]
          + f4_old[ _indices[4]] * XiParaF3_d[4][i]
          + f4_old[ _indices[5]] * XiParaF3_d[5][i]
          + f4_old[ _indices[6]] * XiParaF3_d[6][i];
}
```

四、9　與其他方向的對比

F3 vs F5 索引意義差異

所有 ξ 相關方向的索引一致性　F3, F4, F5, F6, F15, F16, F17, F18 這 8 個方向的參數陣列，全部使用相同的索引結構：

- 第一層索引 [m]：基函數編號（0–6）

項目	F3 (Y-ξ 2D)	F5 (ξ 1D)
速度向量	$(0, +1, 0)$	$(0, 0, +1)$
插值維度	2D	1D
[m] 意義	Lagrange 基函數索引	Lagrange 基函數索引
[i] 結構	$j \times \text{NZ6} + k$	$j \times \text{NZ6} + k$
係數計算複雜度	高（2D）	低（1D）
記憶體使用	$7 \times \text{NYD6} \times \text{NZ6}$	$7 \times \text{NYD6} \times \text{NZ6}$

Table 6: F3 與 F5 的索引結構對比

- 第二層索引 [i]：空間位置 $j \times \text{NZ6} + k$

這種一致性簡化了程式碼維護與除錯。

四、10 常見錯誤與除錯技巧

錯誤 **1**：索引順序混淆　錯誤寫法：

```
F3_in = XiParaF3_d[i][m] * f4_old[...]  //  錯誤！
```

正確寫法：

```
F3_in = XiParaF3_d[m][i] * f4_old[...]  //  正確
```

錯誤 **2**：索引範圍越界　檢查點：

- $m \in [0, 6]$：基函數索引
- $i \in [0, \text{NYD6} \times \text{NZ6} - 1] = [0, 2729]$：空間索引
- $j \in [0, 38], k \in [0, 69]$

除錯技巧：列印特定格點的係數

```
  // 在 Host 端檢查
int j = 20, k = 35;
int i = j * NZ6 + k;
printf("Position (j=%d, k=%d), i=%d:\n", j, k, i);
for (int m = 0; m < 7; m++) {
    printf("  L_%d = %e\n", m+1, XiParaF3_h[m][i]);
}

// 驗證 Lagrange 性質：和為 1
double sum = 0.0;
for (int m = 0; m < 7; m++) {
    sum += XiParaF3_h[m][i];
}
printf("  Sum = %e (should be ~1.0)\n", sum);
```

四、11 數值特性與驗證

Lagrange 基函數的性質　對於任意格點 (j,k)，其對應的 7 個係數必須滿足：

$$\sum_{m=0}^{6} \texttt{XiParaF3\_d[m][i]} = 1.0, \quad i = j \times \text{NZ6} + k \tag{四.11}$$

這可作為數值正確性的檢驗。

對稱性檢查　由於 Periodic Hill 的對稱性，某些格點的係數應該呈現對稱模式。例如在 $y = L_y/2$ 處的格點。

四、12 總結

`XiParaF3_d[m][i]` 的雙重索引系統：

1. 第一層 **[m]**：數學意義 - 選擇 Lagrange 基函數
2. 第二層 **[i]**：空間意義 - 指定 Y-Z 平面上的格點位置
3. 數值意義：該格點、該基函數的插值係數
4. 計算效率：預先計算避免 kernel 內重複運算
5. 記憶體優化：連續佈局利於 GPU coalesced access

這個設計體現了 ISLBM 方法的核心思想：將複雜的插值計算前置到初始化階段，換取執行階段的高效能。