第七章 最佳化分析（Optimization）

7.1 簡介
7.2「阻滯最小均方差」法（Damped least squares; DLS）
7.2.1 一般原理
7.2.2 阻滯
7.2.3 約束條件
7.3 變數
7.3.1 邊界條件
7.3.2 偏導數增量(derivative increment)
7.3.3 參數阻滯(variable damping)
7.4 運算元(operands)
7.4.1 元件類別(component classes)
7.4.2 運算元元件語法
7.5 誤差函數之建立
7.5.1 光班均方根值(RMS spot size)
7.5.2 光程均方根值(RMS OPD)
7.5.3 調變函數(MTF)
7.6 內定誤差函數(automatic error function generation)
7.7 多組態下之最佳化(multiconfiguration optimization)
7.8 大域最佳化 (Global optimization)
7.9 一個設計專案
參考文獻
習題

7.1 簡介

光學系統最佳化其意涵是在藉著調動系統中的建構參數（變數）以改善整個光學系統的效能。通常變數指的是鏡面曲率，元件與空氣間隙的厚度、歪斜角度等等。系統效能可以由使用者去定義一個「誤差函數」(有時又叫做「評價函數」（merit function）來評定。誤差函數代表的是對給定系統效能下系統的偏離程度。最佳化的目標是在決定各個變數以使得誤差函數最小。換言之，整個目標是誤差函數的大域最佳化（global optimization）。

除了一些少數簡單情況，事實上很難在很短的時間把誤差函數的大域極小值給定出來。然而，在試著定出大域極小值的過程中通常可以找到一些極好的解，是其它常用的局部手法所難以提供。稍後在本章裡會交待 OSLO Six 中提供一個叫調適性摹擬退火（adaptive simulated Annealing）的大域最佳化的算則。

比較常用在最佳化的算則是遞迴（iterative）極佳化。在這個情況裡，使用者要猜測相關變數的初始值，然後最佳化算則會以此初始值遞迴產生新值，再遞迴計算以降低誤差函數的值。這種算法可從圖 7.1 一眼看出，事實上效果好壞與否跟初始值落點很有關係，若初始值落在區域 A 或 C 內，程式只會是算出局部極小值（x=a or x=c）。

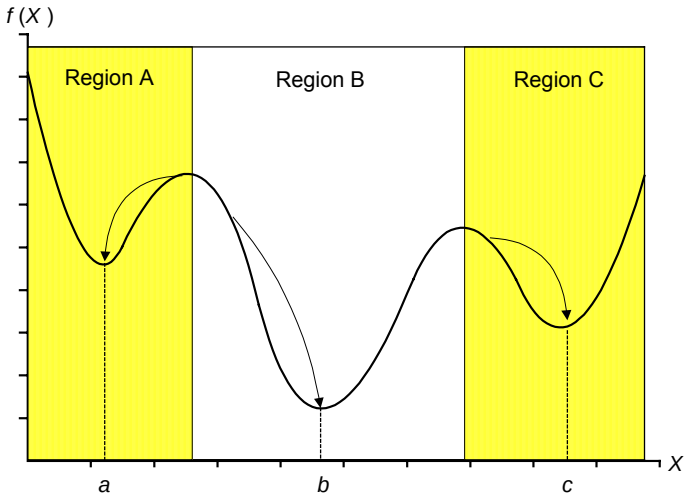

圖 7.1 局部極小值與大域最小值

參考圖 7.1，如果初始點是落在 B 區，則程式結果是會算出大域最小值（落點在 x=b）。在光學系統設計裡所碰到的難題是變數太多，所以所謂的「維度」（dimensionality），亦即獨立變數的數目就很大。如此局部極小值的數目通常會很多，使得要挑個初始點作最佳化變得困難。所以最佳化通常是由現存的系統或設計作參考，或由經驗判斷來挑選適當的初始值。這一類局部極小化是所謂滑坡式（downhill）最佳化方法，文獻有很多。在光學設計裡「最小均方差」（least-square）方法是最常用的。OSLO 軟體已把這類「阻滯最小均方差」（damped least squares）標準最佳化算則涵蓋在內。其原理參見下節。

7.2「阻滯最小均方差」法（Damped least squares; DLS）

OSLO 所用的「阻滯最小均方差」（DLS）最佳化法是光學設計軟體中最常用的算則。幾乎每個光學設計程式都用到 DLS 作最佳化。在這一節中我們討論 OSLO 中有關 DLS 的實用部份。

在 DLS 中誤差函數 $\phi(\bar{x})$ 是以平方權重和方式來表示：

$$\phi(\bar{x}) = \sum_{i=1}^{n} w_i f_i^2(\bar{x}) \qquad (7.1)$$

在此向量 $\bar{x}$ 代表變數 x_i 所成的向量

$$\bar{x} = \langle x_1, x_2, x_3, \cdots, x_n \rangle \qquad (7.2)$$

f 叫做運算元（operands），單獨運算元 f_i 定義如下：

$$f_i = (c_{1i} \oplus c_{2i}) \qquad (7.3)$$

在此 c_{1i}, c_{2i} 叫做分量（components），每個運算元有兩個分量。彼此由一個數算算子（如加法、減法、乘法、除法、指數、大於或小於等等）連接在一起。通常第一個分量是光線位移，而第二個分量是目標值，而算子是減法，所以運算元 f_i 代表偏移量。然而在此（7.3）式定義卻容許更複雜運算元。而 w 是權重，它可以就不同運算元的相對重要性作比例大小調整。在此爲了方便，我們假設所有權重數均爲 1。

誤差函數恆正，故在所有運算元均爲零下，會是極小值。把 $\phi(\bar{x})$ 寫成向量形式是很有用，其形式如下：

$$\phi(\bar{x}) = f^T f \qquad (7.4)$$

在此

$$f = \langle f_1, f_2, f_3, \cdots, f_m \rangle \qquad (7.5)$$

誤差函數的最小化是由運算元與變數相關聯的線性關係來建立。亦即，第 j 個變數所造成的第 i 個運算元改變量形式如下：

$$f_i(x_j + \Delta x_j) = f_i(x_j) + \frac{\partial f_i}{\partial x_j}\Delta x_j \qquad (7.6)$$

當然，這是一個理想化模型。在一個現實情況裡，非線性的效應使得更高階的導數必須納入考量。另外，在真的最佳化問題，因爲物理（非線性）根本就不允許，也有可能無法讓 f_i 爲 0。但無論如何，$\triangle x_j$ 的最小均方差解總是有的，換言之，此時運算元有其容許的極小值。

在這裡要插入一個旁題。在 OSLO 中對誤差函數的表示是以其均方根值來表達。如果所有運算元均是以相同的表示法來表達，則這種均方根差（RMS; root mean square）誤差函數表示式對一個運算元的平均值而言是很合適。

對於多運算元多變數光學系統的最佳化，最好是以矩陣符號來表示。如果 f 代表著是運算元相對於其最小值的改變，則我們可以得到如下對 $\bar{x}$ 改變的掌控方程，它使得誤差函數極小：

$$A \cdot \Delta\bar{x} = -\bar{f} \qquad (7.7)$$

在此 A 是每個運算元對其各自所有的變數的偏導數矩陣：

$$A=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & & \\ \vdots & & & & \\ \frac{\partial f_m}{\partial x_1} & \cdots\cdots\cdots & & & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \qquad (7.8)$$

通常方程式數目比變數多，所以沒有直接解。然而，讓誤差函數極小的最小均方差解是有的。最小均方差求解的算則如下：

(1) 給定初始值 $\bar{x}_0$（先猜測極小值出現的落點 $\bar{x}_0$ ）

(2) 用最小均方差標準方程求解下一個遞迴值 $\bar{x}_1,\bar{x}_2,\cdots$ ，亦即 $\bar{x}_1=\bar{x}_0+\Delta\bar{x}_1,\bar{x}_2=\bar{x}_1+\Delta\bar{x}_2,\cdots$ 或 $\bar{x}_{k+1}=\bar{x}_k+\Delta\bar{x}_k$。其中 $\Delta\bar{x}_k$ 要從最小均方差標準方程

$$A_k^T A_k \Delta\bar{x}_k = -A_k^T f_k \qquad (7.9)$$

定出，在此足碼 k 代表遞迴次數。當然如果（7.9）式是個線性方程，那麼可以用線性代數運算直接解出，不需要遞迴疊代。

（3）如果（7.9）式是非線性，通常用疊代來求解。但由於 $\Delta\bar{x}$ 常會發散，所以加入一個阻滯項以防止 $\Delta\bar{x}$ 發散，整個求解就在「阻滯最小均方差」方程式上，其形如下：

$$\left(A_k^T A_k+\mu_k I\right)\Delta\bar{x}_k = -A_k^T f_k \qquad (7.10)$$

在此 I 是單位矩陣。

高階（如二階以上）的偏導數是可以加入方程式，但對於實際的光學系統設計而言，這樣作並不值得。主要是計算量太大了，另外一方面非線性往往不是只有二階。整個「阻滯最小均方差」方程式求解重點如下：

1、加上阻滯項是爲了防止變量 $\Delta\bar{x}_k$ 發散太快。

2、方程式維度是 n x n 跟變數數目相同。運算元數目一般比變數多（$m>n$，如 7.8 式），但是用 $A_k\cdot A_k^T$ 相乘就成了 nxn。對運算元求和快又有效。

3、阻滯最小均方差求解出的 $\bar{x}_k$ ，跟一般最小均方差求解方程一樣。這是因爲在解點上改變量 $\Delta\bar{x}$ 應爲 0。

第 3 點是相當重要的。因爲它暗示我們可以有個相當大的自由度來調整「阻滯」或該用什麼樣的「阻滯」來加快速度，反正最後解該是一樣。換言之，也沒

有必要說要去限制阻滯項係數 μ k 必須是個常數。亦即我們可以把單位矩陣 I 變成另一個矩陣 D，其形式如下：

$$D=\begin{bmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_m \end{bmatrix}, d_{jj}=\sum_{i=1}^{m}\left(\frac{\partial f_i}{\partial x_j}\right)^2 \qquad (7.11)$$

這是在 OSLO 中用的多變數阻滯項形式。在 OSLO 中允許阻滯係數隨 A^TA 矩陣列的不同而不同。所以對不同變數有不同的阻滯係數。

上面作法是個經驗作法，它沒有什麼漂亮理論，但它很成功。程式設計者可以使用不同的最佳化算則以達到其需求。整個 DLS 流程如圖 7.2 所示。在 OSLO 中，DLS 形式由「ite std」指令來定。阻滯因子的值與遞迴次數由使用者指明。

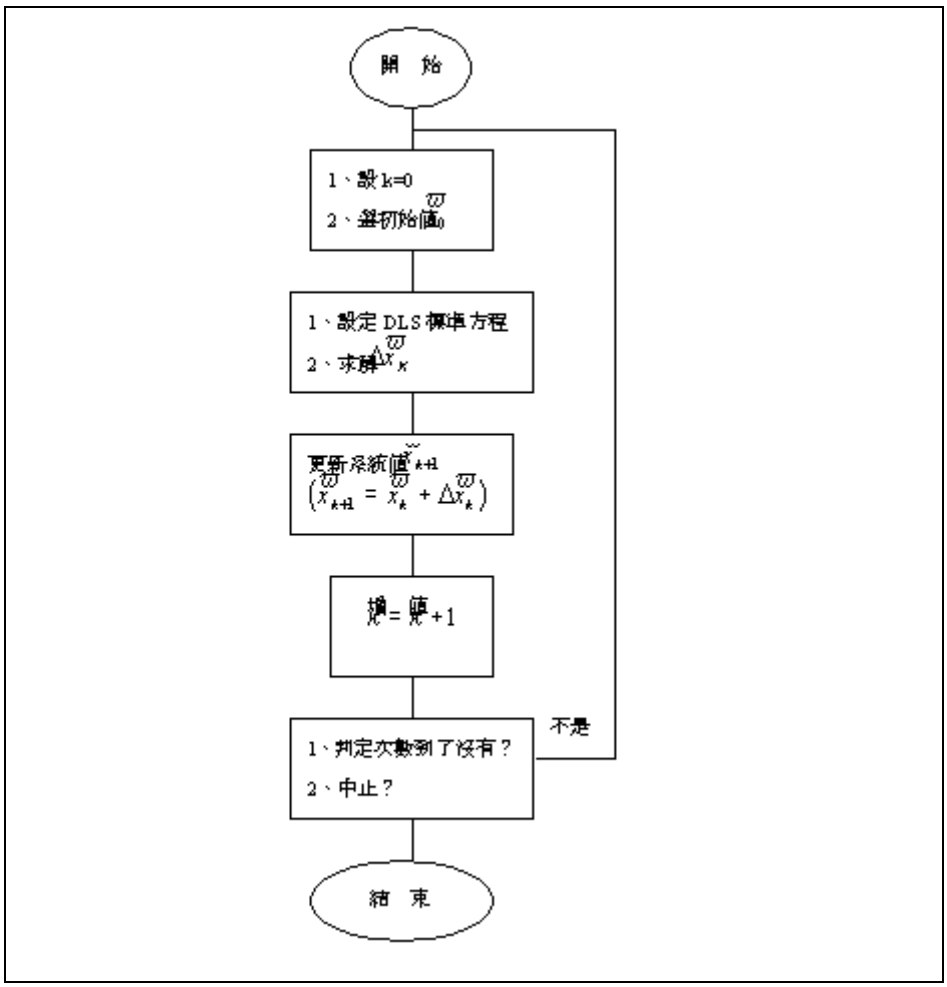

圖 7.2 阻滯最小均方差法求解流程

7.2.1 阻滯（Damping）

如前所述，阻滯因子是用來防止變量 $\Delta\bar{x}$ 增加太快，導致發散。若 μ =0，當

A^TA 具奇異性（singular，它的副作用之一是發散很大）則沒有解。加上阻滯因子，事實上等於對改變量的向量長度平方，當作個比例參數再放進來。所以改變量愈大，阻滯因子效果愈強。

接下來的問題是如何選阻滯因子?如果 μ 太小，而 A^TA 其奇異性又強，則改變向量 $\Delta\bar{x}$ 很快會變大，整個求解變得不穩定且又失效。另外一方面，μ 太大的話，改變向量 $\Delta\bar{x}$ 變化太小，無法使誤差函數 ϕ 在其該陡降時（通常是沿 ϕ 的梯度方向）加快速度以提高計算效率。所以，μ 的選擇關係到穩度性及效率。在 OSLO 中有內值運算自動選擇阻滯因子。這個算則是由「ite ful」指令去界定。每個遞迴都包括一個最佳阻滯因子的搜尋，流程如圖 7.3 所示。

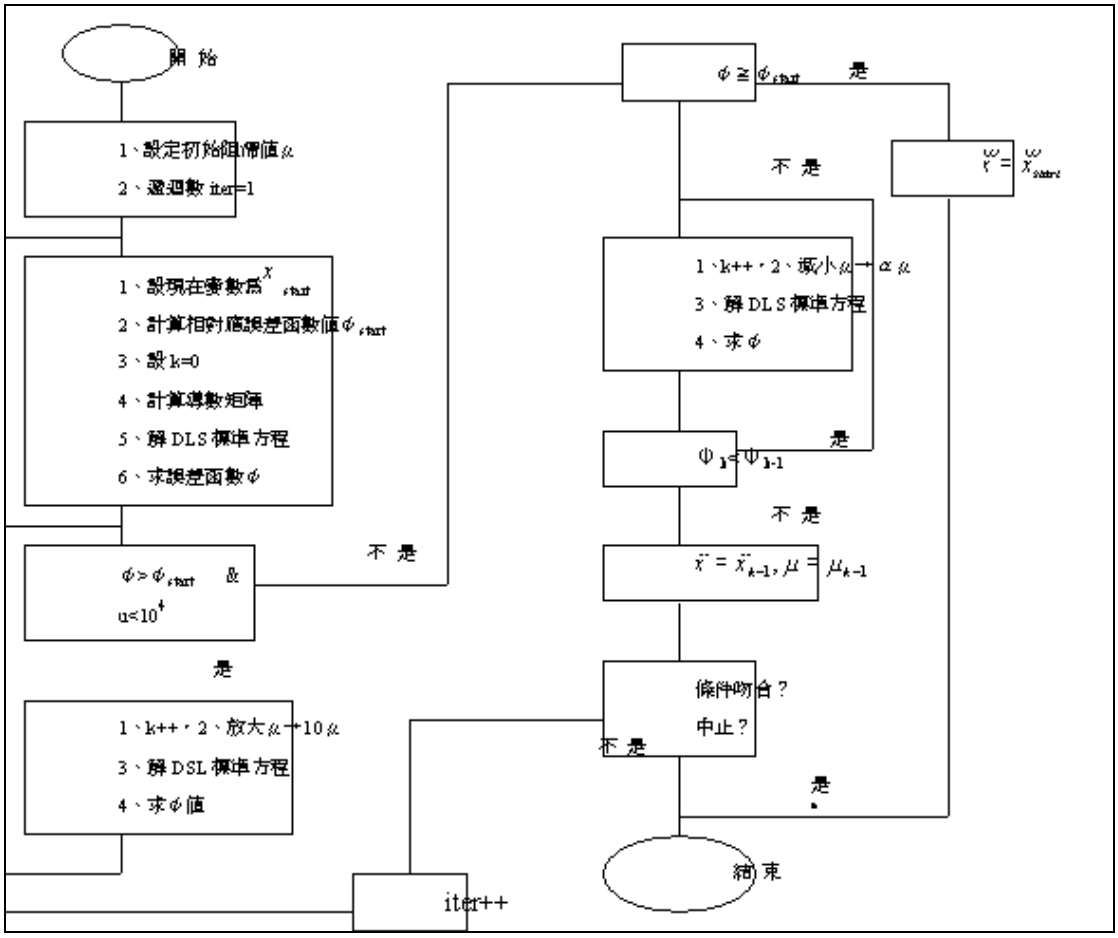

圖 7.3　最佳阻滯值之界定流程

在遞迴開始，阻滯因子值由設定條件的「Opds」指明，通常其值～10^{-8}，隨後程式會拉大此值，除非沒有達到誤差又超過允許值。一旦超過，阻滯因子會被調小（差個因子 α ，由「Opdm」指明），直到找到最佳值爲止。

因爲偏導數矩陣只需算一次，所以找最佳阻滯因子的時間不需要很長。在整個一圈遞迴過程中，只有這個阻滯因子在調整。遞迴會在下面條件成立時停止。

一是超過遞迴圈數設定，二是誤差函數的改變量在二個連續遞迴都沒有超過設定（opst）也會停止。而 opst 是放在「連續全遞回百分比改善」的操作條件裡。

7.2.2. 約束條件（Constraints）

在透鏡組最佳化過程裡，事實上有些值是不能動，變成約束條件。例如有時後焦長就是不可以再變或者透鏡組的速率(或 f 數)必須固定。這些條件可以以下面三種方式來達到。

1、求解（Solves）

求解允許近軸特性被精準保留。例如，對最後鏡面設定「近軸光束角求解」值-0.1，若物在無窮遠即是在要求鏡組的 f 數爲 5。

2、誤差函數的懲罰項（Penalty terms in the error function）

這是在誤差函數外，另外在額外填一個條件（運算元）。這個條件是由使用者選定一項參數，就其上或下限作約束（最佳化在計算過程會試著走滿足這個條件）。例如，在運算元加上「TH(7)>40.0」表示誤差函數計算裡第七鏡面的厚度不可少於 40.0。

3、約束運算元（Constraint Operands）

OSLO 也允許直接寫明約束條件。這個約束運算元的界定與誤差函數無關。約束條件是以 Lagrange 乘數法來引入。換言之，是在阻滯最小均方差方程之求解上，另外加上約束運算元的條件。

這三種方法各有其優劣點。第一個方法（求解），只有在"鏡面可作求解"時且"近軸光束數據要有"才會算得適宜。另外「求解」這個要求可能在最佳化過程中產生不穩定性，這是因爲求解的結果其解出量對變數的反應可能是相當非線性。

至於第二個懲罰項方法，不大容易作。原因是它的有效性往往跟它的權重關聯太大。如果權重太小，則修正效果會以誤差函數其它項爲主，使得約束條件雖然滿足了，但是其它的表現卻不好。

而第三項作法約束運算元，在沒有近軸光束數據或求解變得不穩定或者權重因子無法定下來時會有用。但是主動（active）約束運算元的數目不可以比變數數目多（在此「主動」指的是運算元的值不會是零）。另外，阻滯最小均方差法的作法是先在約束條件已滿足後，再作誤差函數的極佳化。一邊要作誤差函數最佳化，一邊又要約束條件能成立，會使得整個過程較無法預測。這個情況在約束條件的初始值離目標又遠時，會更嚴重。

7.3 變數（variable）

在設計一個光學系統時，系統的部份建構參數可以被調整。這些參數是所謂的「變數」。基本上有兩類變數，一個是自變數（independent variable），它是可以直接由使用者或最佳化之程式來調整，另外一個是因變數（dependent variable），它是因其它變數變動而改變。在 OSLO 中，因變數是那些隨著「求解（solves）」或「選定（pickups）」結果而改變的變數。所以「變數」（variable）指的是自變數。

變數集（set of variable）可以藉由鏡面數據試算表上的變數定義而成的各式物理量來定義，或者在變數試算表加項來達成。如何使用變數可以參考 OSLO 程式參考手冊（programming Reference Manual）。但一些有關最佳化的變數處理的課題放在這裡討論。

7.3.1 邊界條件（Boundary Conditions）

在許多最佳化問題裡，必須指明變數的上限，以避免出來的解在事實上做不到。例如，若有透鏡其邊緣厚度爲負，同一個空間位置有好幾個透鏡共處等等這些事實上都做不到，這些解當然是不合理。有很多方式來避免這種解的產生。但是在使用這些方法之間，應該了解的是過份使用邊界條件會妨礙整個最佳化的運算。事實上，在最佳化過程可能會產生一些中間解，但這些解並不合理，然而稍後中間解可以自發演歷到一個合理的解。如果用邊界條件強制這些中間解不存在，則所期望的最佳化就可能達不到了。

邊界條件可以在變數定義或者誤差函數加上運算元以限制所要變數的範圍。在變數試算表（variable spreadsheet）中「極小」與「極大」兩行代表的是變數的上、下限。如果變數 ν 其值比下限 ν_{min} 小，則一個懲罰項會加到誤差函數。此一懲罰項正比超出下限值，形如：

$$\text{懲罰項(penalty)}=(\textbf{opbw})(\nu_{min}-\nu)^2 \qquad (7.12)$$

同樣地，如果 ν 比上限 ν_{max} 大，也會有一個懲罰項會加到誤差函數上，此時

$$\text{懲罰項(penalty)}=(\textbf{opbw})(\nu-\nu_{max})^2 \qquad (7.13)$$

(7.12)及（7.13）式中的「**opbw**」是變數試算表上叫「邊界條件違反量比例權重」的操作條件。如果變數是界於上、下限之間，就沒有懲罰項加到誤差函數。舉例而言，在實際設計情況下，我們通常對邊界條件裡"厚度"要比"曲率半徑"多

注意。相對應地，一旦厚度用作變數，內定的邊界條件就加諸上去。邊界條件設定值可以在變數試算表上更改。而內定值預設範圍很寬，其主要目的是防止光軸上厚度出現負值的不合理情況。當然，有特定設計約束情況時，就必須修改調整。

邊緣厚度（Edge-thickness）邊界條件必須以運算元方式輸入。一個常用的方法是去決定在每個鏡面上有最大光束高度的光束並要求沿著這個光束在每個鏡面之間的距離必須爲正。這也是 OSLO 內定的誤差函數的作法。雖然這個條件並沒有考慮到光束軌跡通常不跟光軸平行，但是這個作法在多數情況下都行得通。

邊界條件運算元是以大於或小於算子來作。例如，第三面厚度大於 0.5 的寫法爲「TH(3)>0.5」，這類運算元在條件成立時給出 0 值。如果條件違反，則給出兩個分量的差值（在此可以是 TH(3)的值）。更複雜的分量關係（亦即運算元之寫法）可以利用交互參考分量（cross-reference component）來界定。

前面所提的「TH(3)>0.5」的邊界條件運算元是所謂單邊(one-sided)運算元，它應以極小化模式輸入而不是用約束條件來輸入。作法是把權重調整使得可以在原來不允許的範圍內，有小區域仍可動。如果單邊(one-sided)運算元是以約束條件模式輸入，在最佳化過程我們可能會有振盪現象。亦即約束條件一下子強制一下子解除反覆交替。在 OSLO 裡，操作條件提供「one-sided 約束運算元容忍度」（tolerance for one-sided constraint operands）（**opct**）藉此約束條件有個戒護範圍，在這範圍內，事實上運算元振盪約束條件也不會被強制執行。這可以使得我們極佳化的過程比較有效率。

在一般阻滯最小均方差最佳化（DLS optimization）裡最好是沒有邊界條件。定義解的範圍，邊界條件當然是有用，但它們通常阻礙最佳化運算，所以沒有用到時，就應該移除。如果一個變數一直違反其邊界條件，那麼就該設此變數爲一固定值。當然這固定值可以用允許範圍的極值替代。

在「調適摹擬退火」大域最佳化（ASA global optimization）中，邊界條件的角色有些不同。在 ASA 大域最佳化之下，沒有設計者的干涉，放任數仟個系統參數、運算元被求值與分析。在此邊界條件是用來作定義搜尋範圍，對整個算則之操作是極其重要。

7.3.2 偏導數增量（Derivative increments）

在最佳化標準方程式，阻滯最小均方差法必須對各個運算元的偏導數求解，這可以用有限差分近似，其形如下：

$$\left.\frac{\partial f_i}{\partial x_j}\right|_{\vec{x}} \cong \frac{f_i\left(x_1, x_2, \cdots, x_j + fx_j, \cdots, x_n\right) - f_i\left(x_1, x_2, \cdots, x_j, x_n\right)}{\delta x_j} \qquad (7.14)$$

內定的偏導數增量（**opdi**）操作條件決定了計算偏導數中步伐大小（step size）δx_j。最佳步伐 δx_j 的決定是有些爭議。有些說法認爲 δx_j 應該就非線性曲線的斜率（正切值）而定，另外有些人則認爲增量應該就預測的解出向量（solution vector）作參考依據。

在 OSLO 中偏導數增量是依變數形態及入射光束半徑來決定。當變數被確認，偏導數增量可以順便確認而固定下來。但是如果內定偏導數增量設爲零，則程式會依極佳化來調整增量值。

通常最佳化結果跟偏導數增量的大小沒有太大關係，但是如果是一個很高階的非球面鏡（aspheric），情況就不同。在這個例子下，鏡面凹陷（surface sag）差跟孔徑是以 n 次冪次（n^{th} power）成比例，所以非球面係數的偏導數增量必須小心。通常是在最佳化過程對這一類系統把焦距長尺度歸化成 1，以保持不同階的非球面有個最佳平衡。同樣地，這個作法也適用在處理高階多項式相位函數（phase function）的情況。

7.3.4 參數之阻滯（variable damping）

如前（7.11）式所提，A^TA 矩陣的對角各項元素並不需要用相同的阻滯因子。在 OSLO 中允許各個變數以各自方式選用阻滯因子。這些各別變數的阻滯因子自然可以對整個最佳化過程的變數提供較有效控制。通常大一點的阻滯係數對一給定變數可以防止較大的改變。

每個阻滯因子的最佳值必須藉由實驗決定。很不幸，這樣子需要太多的計算量，以致於是不可能真的採行。亦即要把問題化成線段線性（piece-wise linear）模型是有很多有效的近似手法，但與其去求解近似模型的精準解，其實倒不如用近似解點(approximate solution point)算一個相對應偏導數矩陣來得有效。

在 OSLO 中有二個方法來執行變數之阻滯。一個是內定調適演算法，其阻滯因子是取用 A^TA 的行向量(column vector)的各個長度。另一個是在最佳化過程與你互動，讓你決定阻滯因子，很自然地最佳化可以由你調整，往你想要的方向去作，但此種作法很明顯地要有經驗才行。

7.4 運算元（Operands）

OSLO 使用運算元（operands）來表示誤差函數中的項，而誤差函數是各項運算元平方之後再依據權重加起來的和。雖然運算元主要是用來最佳化，它們對諸如公差化規畫、容忍度分析及特定決定的品質評估也可以作爲表現因子加以運用。

運算元是依運算元之定義，以文字形式原始碼輸入，其內在表示法已固定，以達到最大計算效率。所以 OSLO 中的運算元與其說是指令，毋寧說是「表示項」會來得更加貼切。每個運算元包含一個或二個分量，其形如：

$$f = [-1]c_1 \oplus c_2 \qquad (7.15)$$

其中⊕符號代表運算子，其各式定義如下圖 7.4，而 c_1 與 c_2 是分量。

算 子 ⊕	內 容
+	加 法
-	減 法
*	乘 法
／	除 法
**	指 數
<	小於。若 $c_1 \geqq c_2$，則 $f=c_1-c_2$，否則 f=0
>	大於。若 $c_1 \leqq c_2$，則 $f=c_1-c_2$，否則 f=0

圖 7.4 運算元算子的內涵。注意＜與＞。

7.4.1 分量屬類（Component Classes）

因爲運算元分量的值可以是其它運算元。使用前面的雙分量形式，可以建構一個含有多個分量的運算元。每個分量基本上其屬性如下。

系統物性（system）分量

分量可以是透鏡的一個物性。例如曲率半徑、厚度、非球面及 GRIN 係數等等。任何可以用來指定作爲最佳化之變數，同樣也可以用作系統運算分量。另外，鏡面凹陷（surface sag）、邊緣厚度、功率及光軸上長度都可以被指定作分量。

像差與近軸數據（Aberration and Paraxial data）分量

這裡的分量指的是三階、四階像差係數，7 階球差，近軸主光束與軸上（axial）光束之高度與斜率，主要與次要（secondary）色差。在某些系統這種分量也許不適用。

光束（ray）分量

這裡指的是在特定瞳孔上的點，就某個場源的光束對整個系統描光所得的

值。場源之值由場源點集（field points set）界定，而瞳孔上的點由光束集（ray set）指明。

斑點圖（Spot diagram）分量

這裡的分量指的是調變轉換函數（MTF）或波前誤差（RMS wavefront error）的值。它們是由一些特定場源（field points）所形成的斑點圖算出。場源值由場源集（field point set）界定而孔徑分隔數（aperture divisions）或格子大小（grid size）及波長由斑點圖數據表決定。

CCL 及 SCP 指令分量

這類分量指的是由 CCL 或 SCP 運算元函數算出的值。這種分量可以讓我們去計算一些不易界定其形態的分量。

外部（external）分量

這類分量指的是利用動態連接程式庫（dynamic-link library）算出的值。這些程式庫是以高階語言（如 c）所撰寫並編譯的函式庫。這些函數比 CCL 或 SCP 函數執行得更快，但需要較多時間去發展。

交互參考（cross-reference）分量

這類分量指的是運算元集合中其它運算元的值。藉由此類分量形成，我們可以建構出多分量的運算元。

統計（statistical）分量

這類分量指的是一個或多個運算元的平均值或均方根值。它們通常是用在計算光斑大小或波前誤差（註：以均方根型式表示）。

常數（constant）分量

這類的量指的是定常數值。各式屬性的運算元分量在程式參考手冊均有列舉說明。

7.4.2 運算元分量語法（Operand Component Syntax）

每個分量均有一個或多個引數（argument），以整數加上逗號（comma）分隔。當運算元被定義，在 OSLO 中會以其內定格式編譯。之後在顯示視窗上回應的是運算元的編譯版。它也是最有效的語法，而其形式可能與輸入定義時不同。

運算元分量的語法依其屬性不同而異。除了交互參考分量及常數分量之外，每個分量可以有一個或多個整數引數以指明光束（指明光束集的足碼）、波長（指明波長集合的引數）、鏡面、組態形式（configuration）等等。這裡頭有許多引

數已有其內定值不必由使用者再輸入。即使使用者輸入這些引數內定值，在 OSLO 顯示視窗還是不會將它顯示。下圖 7.5 是一些例子。

引 數 （Argument）	內 定 （default）
鏡 面 數	0 （標 示 像 面）
鏡 面 值 中 第 一 鏡 面 數	0 （標 示 物 面）
鏡 面 值 中 最 後 鏡 面 數	0 （標 示 像 面）
組 態 數	1
波 長 數	1
場 點 數	沒 有 （none）
光 束 數	沒 有 （none）
斑 點 圖 數	沒 有 （none）
空 間 頻 率	沒 有 （none）

圖 7.5 運算元中分量的引數及其內定值例子

在分量引數上有些慣用法（conventions）

- 鏡面數爲 0 代表像面，而負值代表相對於像面的鏡面。例如，「cv(-2)」代表在像面之前 2 個鏡面的曲率半徑。若鏡面 7 是像面則「cv(-2)」代表鏡面 5 的曲率半徑。這個慣用法允許我們插入或移除鏡面而不致影響份量表示中真正的鏡面。

- 不存在的波長數、場點數、光束數及斑點圖數可以不用輸入。例如，在場點集合裡只有兩個輸入，則分量「Y(3,1)」代表的是從場點數 3 出來的光束數 1 在像面的 y 切點，其是可以不用輸入。

- 如果一個不存在組態數被輸入，則其分量值爲 0。例如，只有一個組態有被定義，則「TH(5,2)」代表組態 2 的鏡面 5 厚度值會被強制爲 0，如果有新的組態 2 稍後被定義，則「TH(5,2)」會代表組態 2 中的鏡面 5 厚度。

接下來我們敘述各種運算分量的屬性並給出例子。

系統物性運算元分量（System Operand Components）

除了邊緣厚度（ETH）、功率（PWR）及軸長（LN）外，系統物性運算元分量的語法如下。

〈分量〉(面數，組態數)

<component>(surface#,configuration#)

而對於 ETH，PWR 及 LN 三種，則其語法爲

〈分量〉(第一鏡面數，最後鏡面數，組態數)

<component>(first surface#, last surface#, configuration#)

例 1：定義一個分量指明組態 1 的鏡面 5 厚度。若你輸入「TH(5,1)」，OSLO 會顯示成「TH(5)」，原因是組態 1 爲內定。所以可以直接輸入成「TH(5)」

例 2：定義一個分量指明組態 2 的鏡面 5 厚度。若你輸入「TH(5,2)」，OSLO 會保留此定義。

例 3：定義在組態 1 一個從鏡面 1 到像面的軸長之分量答案是 LN(1,0,1)，也可以是 LN(1,0)或 LN(1)。但在 OSLO 中會顯示 LN(1)

例 4：如例 3，但現在是組態 2，答案是 LN(1,0,2)，必須全部寫明。

像差與近軸數據分量（Aberration and paraxial data components）

除了主要軸上色差（primary axial color, PAC），主要橫向色差（primary lateral color, PLC），次要軸上色差（secondary axial color, SAC）及次要橫向色差（secondary lateral color, SLC）外，像差及近軸數據分量其使用語法如下。

〈分量〉(波長數，鏡面數，組態數)

而 PAC，PLC，SAC 及 SLC 分量的使用語法如下：

〈分量〉(鏡面數，組態數)

例 1：定義一個分量以界定在波長，組態 1 下像面的三階球面像差。
答案是「SA3(0,1,1)」由於波長 1、組態 1 及像面是內定值，所以在 OSLO 會以「SA3」表示。

例 2：定義組態 2 下像面之主要橫向色差的分量。答案是 PLC(0,2)

光束運算元分量（ray operand components）

每個光束運算元分量是以特定波長從指定場點（由物面上的相關座標指明）到指定瞳孔上的點（由入射瞳孔上的相關座標指明）到透鏡組，再到像面的描光來計算。波長、物件座標、瞳孔座標在運算元分量並不會被直接指明，而是放在波長集、場點集及光束集裡讓運算元分量參考。因爲單一光束的訊息常被多個運算元分量所引用，這種非直接表達法可以提高效率。在 OSLO 中光束覓跡數目只是作到夠用就不作了。所以要引入任何光束運算元分量必須了解場點集及光束

集是否夠用。至於波長集通常不是最佳化過程裡用，所以不在此討論。

場點集定義在最佳化裡所用的場點。它們並不需要跟透鏡草樣的草樣設定條件的光束一樣。每個場有一數字及 10 個數據項與自此點而出的所有光束相關聯。數據項如下：

FBY，FBX 及 FBZ 是場點在物面上的相對應 x,y,z 座標。它們是以相對於物高的差值（fractional value）來指定。YRF 及 XRF 是自場點出來的參考光束在參考面上的相對座標（已與該鏡面的孔徑半徑作歸一）。FY1，FY2，FX1 及 FX2 則是場點的子午線及弧矢漸暈(Sagittal vignetting)的上、下因子。這些因子允許光束集上光束的瞳孔座標（或參考面座標）可以把漸暈(vignetting)納入而不用改變光束集。如果 FY 及 FX 是給定一光束在光束集內的相對座標值，則自場點出來光束在瞳孔或參考面上真正相對座標值爲：

$$\text{真正描光的 FY}=\frac{[(\text{光束集}FY)+1][FY2-FY1]}{2}+FY1 \qquad (7.16)$$

$$\text{真正描光的 FX}=\frac{[(\text{光束集上}FX)+1][FX2-FX1]}{2}+FX1 \qquad (7.17)$$

FY1，FY2，FX1 及 FX2 的內定值是 -1.0,1.0,-1.0 及 1.0，所以描光下的 FY 與 FX 與光束集內的 FY、FX 一致。漸暈(vignetting)因子的效果是把瞳孔上格子點光束轉成正方形或者把圖形光束轉成橢圓，以藉此近似離軸場點的漸暈(vignetted)瞳孔之形狀。通常要讓 YRF=$\frac{FY1+FY2}{2}$ XRF=$\frac{FX1+FX2}{2}$，此時參考光束大略可以說是通過漸暈(vignetted)瞳孔的中心。

對場點最後一個數據是其權重 WGT，它是在自動產生誤差函數時才會用到。

至於光束集中每個光束有一個數字及一些數據項來指明（1）光束形態（2）孔徑座標餘量（fractional aperture coordinates）及（3）權重。在 OSLO 中，權重通常不會用到。光束形態分二種，一是平常（ordinary），二是參考（reference）。平常光束指的是由入射瞳孔的一定點開始穿到整個透鏡組的描光。參考光束則是不斷遞迴以確保光束是在參考面上。可想而知平常光束的描光比參考光束要快得多。也因此在最佳化的運算中應先考慮。FY 及 FX 是平常光束穿過入射瞳孔或參考光束穿過參考面相對座標值。

光束的形態分量對使用光束作計算的分量而言是重要的，分量的形態要藉此而定。基本上有三類光束相關的運算元型態，一種是兩種光束型態都可以用，一

種是只供平常光束用，一種只給參考光束。

對兩種型態光束皆可用的運算元分量有 X，Y，Z，RVK，RVL，RVM，NVK，NVL，NVM，XA，YA，PL 和 OPL（詳細定義見程式手冊）。這些數據只跟單一光束有關，其語法如下

〈分量〉(場點數，光束數，波長數，鏡面數，組態數)

其它二種型態的光束分量只在像空間才可以用，其語法如下

〈分量〉(場點數，光束數，鏡面數，組態數)

應該一提的是場點集及光束集本身對程式在作運算時沒有加上計算負擔。只有運算元定義中有參考的光束才會真的被覓跡，而如果一個光束被多次引用，這個光束也只會被描光一次，這樣子可以提高速度。

下面 2 個例子是用圖 7.6 的場點集與光束集來算。(由 update > operation conditions > lens drawing 可看到修改場點集或 Optimize>> Error Function Tables >> Ray Sets 看到修改光束集; OSLO LT 5.4 無法執行)

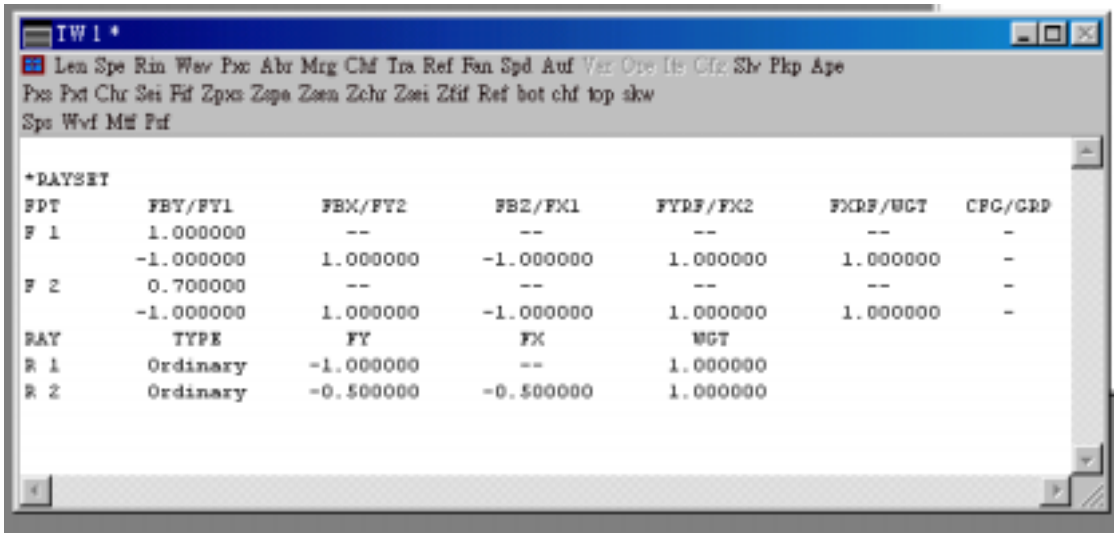

圖 7.6 場點集與光束集

例 1：定義一個運算元分量以量測一自場點（FBY=0.7，FBX=0.0）出來的光束從入射瞳孔到鏡面 5 的光程長。瞳孔座標（FY=-0.5，FX=0.5），波長數爲 3，組態數爲 1。

依據圖 7.6，場點爲 2 (FBY = 0.7，FBX =0.0)，光束爲 2(FY = - 0.5，FX = 0.5)，波長爲 3，鏡面爲 5，組態爲 1，故運算元分量爲 OPL(2,2,3,5,1)。在 OSLO 因組態內定爲 1，故可略化成 OPL（2,2,3,5）。有關 OPL 等指

令可藉由 “Optimize > Operand “而了解。

例 2：定義一個單邊運算元以要求從場點 1 到瞳孔點（光束 1），鏡面 3 到鏡面 4 的路程長（PL）要比 0.5 透鏡單位大。
答案是 PL（1,1,1,4,1）>0.5。在 OSLO 中會簡化成 PL（1,1,1,4）>0.5，注意爲了避免誤解，雖然波長數 1 是內定，但是因爲後面有鏡面數的引數，所以沒有省略。

底下我們展示一個運算元定義的例子。由圖 7.7 可以看到我們要求 OPL(1,1,1,1)是極小(mode 爲 min)。

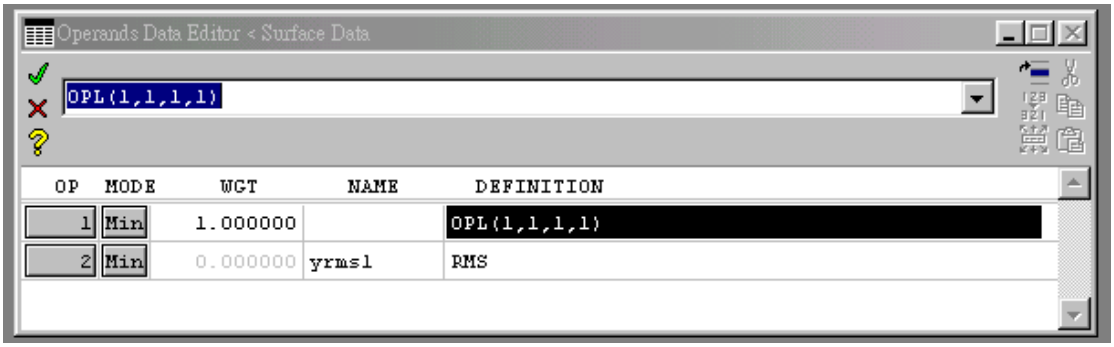

圖 7.7 運算元設定

斑點圖運算元分量（Spot diagram operand components）

在 OSLO SIX 裡可以用斑點圖運算元對調變轉換函數直接作最佳化。這些運算元分量的主要用途是來建構特定的使用者定義的公差化誤差函數(user-defined tolerancing error function)，或者對一個快要達到規格的系統作調變轉換函數（MTF）的微量最佳化（differential optimizations）。在平常情況的最佳化則最好不要用，原因是 MTF 或均方根波前誤差對最佳化之變數而言都是相當大的非線性函數，阻滯最小均方法在這些情況下表現很差。

只有三種斑點圖運算元分量：（1）弧矢(Sagittal) MTF (MTX)，（2）切線 MTF（tangential MTF, MTY）及（3）均方根波前誤差（WVF）。MTX 及 MTY 分量的語法如下。

〈分量〉（斑點圖數，空間頻率，組態數）

WVF 運算元分量其語法爲:

WVF（斑點圖數，組態數）

斑點圖集合試算表（spot diagram set spreadsheet）是用「sde」鍵盤指令來打開。在 OSLO 內定選單裡沒有提供直接進入功能。現時的試算表可用「sds」指令作顯示。就像光束運算元一樣，決定斑點圖要如何畫出的數據（如場點、波

長、孔徑分割）並不是在指令定義裡直接指明，而是在斑點圖集（spot diagram set）界定。

斑點圖集有 5 個數據項，FPT 是場點數，它是一個指明場點集的足碼，斑點圖依此場點集場點作描光。APDIV 是就在瞳孔上斑點圖光束格子盤的孔徑分割數目。FIRST WVL 是畫斑點圖時波長集上第一個用到的波長。NBR WVLS 是斑點圖上會用到的波長。所要描光的波長共(FIRST WVL, FIRST WVL+1,…,FIRST WVL+NBR WVLS-1)。

通常，在誤差函數裡不會只放 MTF（如 MTX 或 MTY）而是 MTF 與目標規格的偏差或繞射極限 MTF 值。

例子：設定一個運算元界定空間頻率爲 10 cycles/mm，場點爲 FBY=1.0，FBX=0.0 下，MTF 偏差值爲（0.9 -多色差橫向 MTF）（多色差橫向 MTF：polychromatic tangential MTF）

作法如下：（就場點 1）設定斑點集輸入（方法參見前段有關「光束運算分量」的作法）。因爲這是一個多波長（polychromatic）斑點圖，所以第一個波長爲波長數 1 並假設有 3 個波長，數據設定應如圖 7.8。

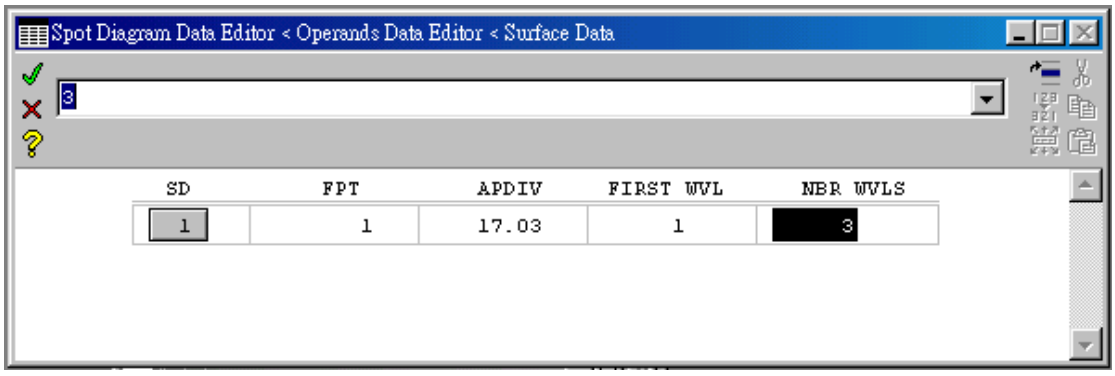

圖 7.8 斑點圖設定

最後，一如前面定義。令運算元如下 0.9 -MTY(1,10)，即爲答案。

CCL 及 SCP 分量

由於內建運算元分量未必合用，CCL 及 SCP 分量的引入，使我們可以在誤差函數裡放入我們想要的項，藉由 CCL 和 SCP 可以將試算表暫存區（spreadsheet Buffer）的 OSLO 指令之數值輸出給叫出。

CCL 及 SCP 分量語法如下:

OCM<陣列元素數>（組態數）

在此<陣列元素數>（array element#）是指向 CCL/SCP 大域陣列變數 OCM 的整數足碼。

設定 CCL/SCP 分量有三步。

1、以 CCL 或 SCP 寫下一個程序（procedure）或者引用附帶的程序來計算將要加入誤差函數的項之值。每一項的值應該被指定到 OCM 陣列的元素。這元素是個內建大域 CCL/SCP 變數。下圖 7.9 是一個 CCL 指令範例。它取材「Callbaks.ccl」檔案(或 op_callbaks.ccl 檔案)，這個檔案是自三個場點的每一個作斑點圖描光並就每個場點情況就其均方根斑點大小(RMS spot size)之值分別指定到大域變數 OCM【0】，OCM【1】及 OCM【2】。

```
cmd oprds_spot_size(void)
{
    set_preference(output_text, off);
    trace_ref_ray(0.0);
    ssbuf_reset();
    spot_diagram(mon, 10.0);
    Ocm[0] = c4;
    trace_ref_ray(0.7);
    ssbuf_reset();
    spot_diagram(mon, 10.0);
    Ocm[1] = c4;
    trace_ref_ray(1.0);
    ssbuf_reset();
    spot_diagram(mon, 10.0);
    Ocm[2] = c4;
    set_preference(output_text, on);
}
```

圖 7.9　CCL 指令範例

2、自步驟 1 設定「CCL/SCP 運算元指令」最佳化操作條件的值作爲 CCL 或 SCP 指令的名字。如果是 SCP 指令，則名字前應加上星號「 *」。就圖 7.9 所列的例子，則其名字應爲 oprds-spot-size。

3、加上對應在 CCL 或 SCP 指令中已指定值的 OCM 陣列元素的 OCM<陣列元素數>運算元分量。以圖 7.9 爲例，加上 ocm1,ocm2 及 ocm3 到運算元。

一旦誤差函數被求值，設定的 CCL 或 SCP 指令對每個透鏡組態都會執行一次。如果必要的話，CCL 或 SCP 函數可以檢查 cfg 大域變數以決定目前的組態數。對所有組態都使用同一 OCM 陣列，所以 CCL/SCP 運算元分量指令需要確認，每個組態使用 OCM 陣列中不同的子集合。例如，運算元集裡有兩個 CCL 運算元，一個就是組態 1，另一個是組態 2。則運算元集有兩個輸入 OCM(0)及 OCM(1)，而 CCL 指令用以計算運算元分量其程式碼如圖 7.10。

```
if (cfg==1)
{
    Ocm[0]=< value of config. 1 operand>;
}
else if (cfg==2)
{
    Ocm[1]=< value of config. 2 operand>;
}
```

圖 7.10 CCL 指令用以計算運算元分量的程式碼

CCL/SCP 運算元分量在執行會先產生文字輸出集 output_text OSLO 指令，等關掉（OFF）在執行完時，再打開（ON），參見圖 7.9 的程式中（set-preference（output_text, off）及 set_preference（output_text_on））這是要避免 CCL/SCP 指令在執行最佳化時打開太多的文字輸出。

外部分量（External components）

外部運算元分量提供一個把需要大量計算又不適合以CCL/SCP指令來完成的結果放入誤差函數的方法。它的作法跟 CCL/SCP 是類似，只是計算以用如 C 的高階語言編譯的動態連接函式庫（dynamic-link library, DLL;在 UNIX 作業系統裡，DLL 叫作 shared libraries）來執行。這方法的好處是速度與彈性，不利的地方除了在 OSLO 中，需要另外一個編譯器，另外也要把 DLL 跟 OSLO 合在一起。

交互參考分量（cross-reference components）

交互參考運算元分量就是分量值是前運算元的結果。其語法是

O〈運算元數〉

在此運算元數指前運算元的編碼。

交互參考運算元提供一個方法來建立多分量運算元。例如，你想要有一個運算元其值是三、五、七階，球面像差（SA）的和。在 OSLO 中是沒有 SA3+SA5+SA7 這種運算元。依定義，必須先寫成兩個部份，一個是中間步驟且設其權重（weight）爲 0，參見表 7.9 的運算元 01，第二個是再將 01 與 SA7 相加。

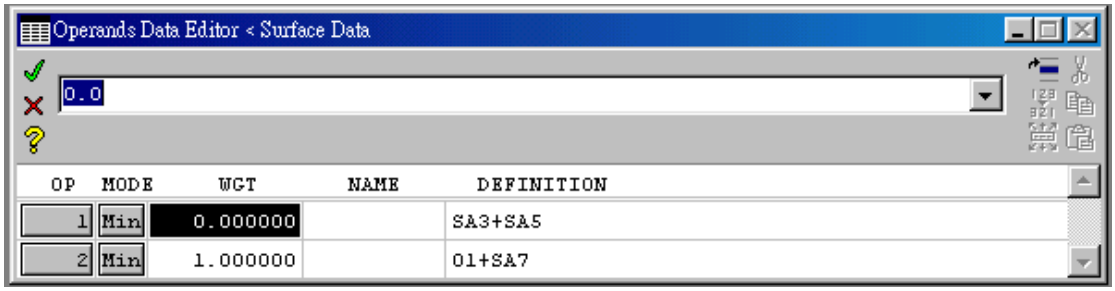

圖 7. 11 交互參考運算元之用法

注意運算元 1 的權重爲 0.0，所以它沒有直接混入誤差函數的計算。而 o1 在運算元 2 的值正如運算元 1 所得一般，不受運算元 1 的權重影響。

統計運算元（statistical operands）

統計運算元是用來對一群連續的運算元之值作平均、標準差及均方根之計算。有兩類統計運算元分量，一是 AVE，二是 RMS。AVE 運算元得到是所有接

續運算元「第一個」分量值的加權和（即依權重作加法），但這接續項在下一個 RMS 運算元前停止。而 RMS 運算元是在前一個 AVE 運算元到 RMS 運算元之間所有運算元的二個分量的均方根值。如果讓 AVE 運算元數是 a，RMS 運算元數是 r（r>a+1），則 AVE 運算元 a 其值爲：

$$f_a = \sum_{i=a+1}^{r-1} w_i c_{1i} \Big/ \sum_{i=a+1}^{r-1} w_i \qquad (7.18)$$

而 RMS 運算元 r 的值是：

$$f_r = \left[\sum_{i=a+1}^{r-1} w_i f_i^{\,2} \Big/ \sum_{i=a+1}^{r-1} w_i \right]^{1/2} \qquad (7.19)$$

統計運算元通常和交互參考運算元合在一起以計算一群運算元的標準差。其形式如下：

$$f_r = \left[\sum_{i=a+1}^{r-1} w_i (c_{1i} - f_a)^2 \Big/ \sum_{i=a+1}^{r-1} w_i \right]^{1/2} \qquad (7.20)$$

例子，建構一個誤差函數來量測從場點（FBY=1.0，FBX=0.0）出來的三道光束在像平面上 y 相交量的均方根值。

第一步：設定光束集及運算元如圖 7.12 所示。

```
*RAYSET
FPT        FBY/FY1        FBX/FY2        FBZ/FX1        FYRF/FX2       FXRF/WGT    CFG/GRP
F 1        1.000000          --             --             --             --          -
          -1.000000       1.000000      -1.000000       1.000000       1.000000     -
RAY           TYPE           FY             FX             WGT
R 1        Ordinary      -0.500000          --          1.000000
R 2        Ordinary         --              --          1.000000
R 3        Ordinary       0.500000          --          1.000000

*OPERANDS
 OP  MODE     WGT       NAME          VALUE  %CNTRB DEFINITION
O 5   M     3.000000                0.037095 100.00 RMS
MIN RMS ERROR:      0.037095
```

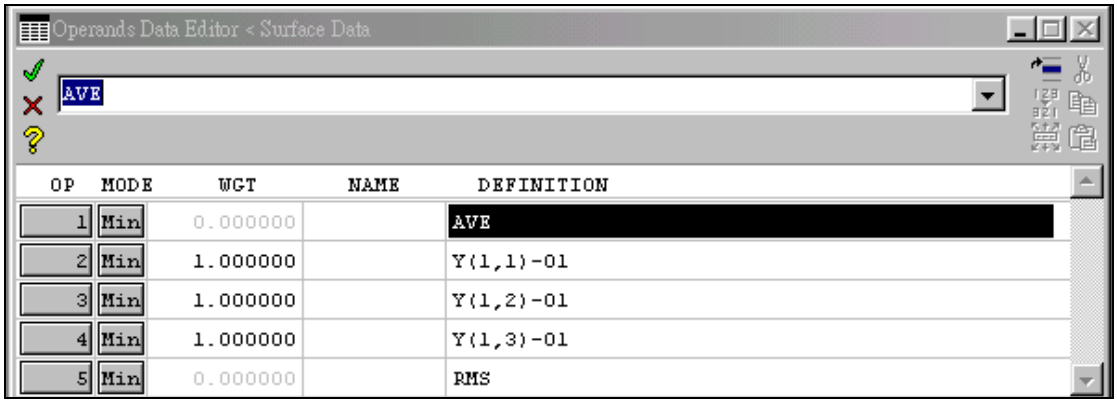

圖 7.12 光束集及運算元設定

注意 AVE 和 RMS 運算元權重均爲 0。如果設爲非零，則這兩個運算元會直接放

入誤差函數來算。在這裡誤差函數只算運算元 2，3，4 的標準差。

在這個例子中，我們以 demotrip.len 的數據來作，如果不同的鏡組，答案自然不同。其圖樣如圖 7.13 所示

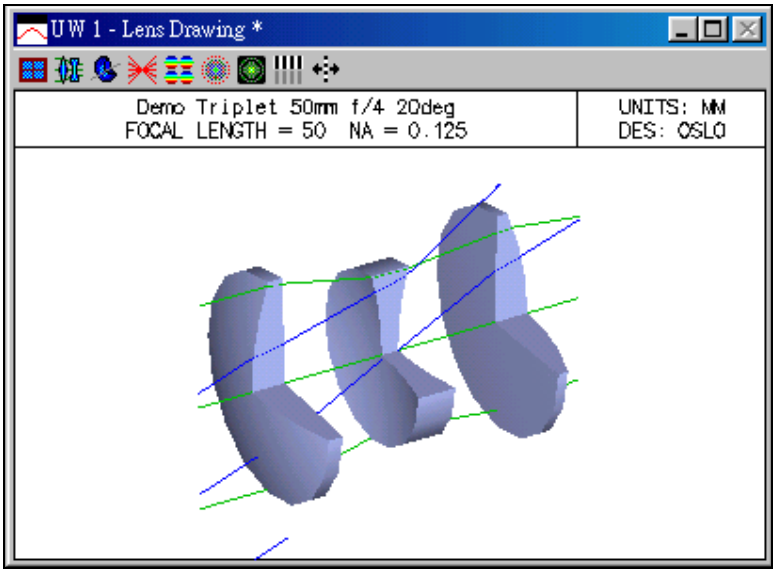

圖 7.13 demotrip.len 圖樣

如果文字輸出指令裡的指令用的是 ope 運算元指令，AVE 運算元及在 AVE 與 RMS 運算元之間的運算元會被隱藏住，而 RMS 運算元部份的權重及百分比貢獻項，事實上只是 AVE 及 RMS 之間運算元的權重及百分比貢獻的總合如圖 7.14 中間部份所示。如果運算元是以「operands all」指令來列出，則所有運算元都會列出，但 RMS 運算元是以零權重方式列出，如圖 7.14 最下端所示：

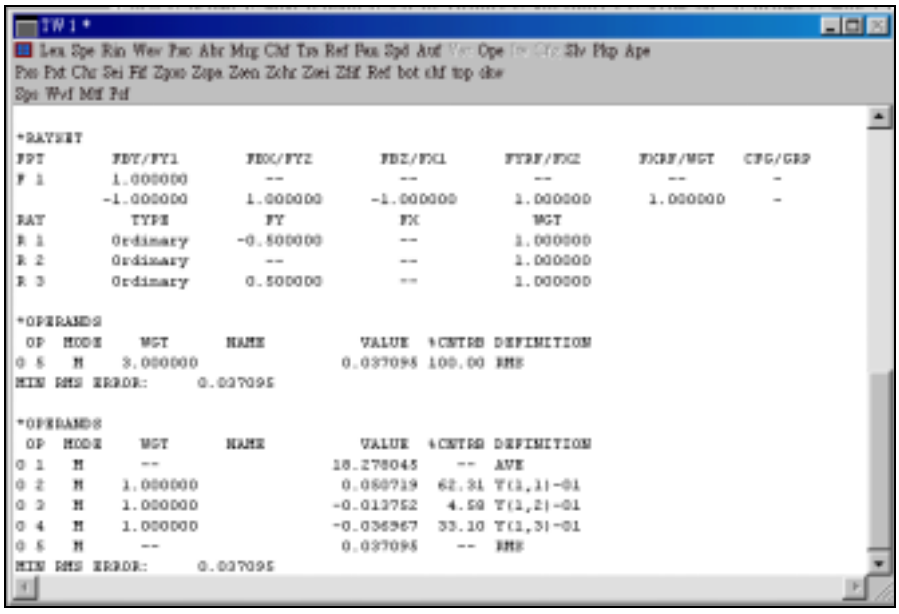

圖 7.14 RMS 運算結果

常數分量（constant components）

常數運算元分量是一些如 27.6 或 1.772×10^{-3} 之類的固定數值。常數分量的使用限制是負常數不可作爲運算元的第二分量。OSLO 不會接受一個運算元定義作「SA3>-0.075」，但是如果改寫成「-SA3<0.075」或「-0.075<SA3」就可以。像運算元「DY(1.1)*-20.0」就得寫成「-20.0*DY(1.1)」。對於那些就是無法簡單改寫的運算元，可以用交互參考運算元方法來作。例如，「CMA3**-0.5」可以寫作「CMA**O1」，而 O1 可以定義成常數分量其值爲-0.5 且其權重爲 0。

7.5 誤差函數的建構（Error function construction）

可以放在誤差函數內以量測效能表現的量有：

- 光效能（例如，調變轉換函數，波前誤差均方根值，斑點大小的均方根值）
- 物理條件（例如，邊緣厚度不應爲零）
- 花費（材料、製程等等）

一般的誤差函數都是用光效能、物理條件作其內含量。而花費通常是用來限制建構透鏡的材料種類、元件個數，不常用的鏡面型態等等。花費也可由製程容忍度來決定，在本段裡，只有光效能與物理條件會被討論。

一個特定設計工作所要的誤差函數之種類當然得依系統種類與規格要求而定。在光學設計的早期階段，最佳誤差函數是由近軸光束數據及像差係數來決定。之所以如此的原因是（1）這種誤差函數不需要對整個系統作描光，非常好用。（2）對一個成功的最終設計，瞳孔位置、孔徑及一般像差分配要能控制好，所以只要系統的對稱性允許像差分析，先用像差係數作初步分析是個很好的主意。

對最終設計而言，引用完整光束覓跡定出的誤差函數以確認效能是適宜的往往是必要的。對一個只有幾個自由度的簡單系統而言，通常要作出最後設計只要用幾道光束即可。但是對於一個較複雜系統，光程差及光斑大小等像質(image quality)量度提供透鏡系統對像面上物點成像較佳的精度估算，因此常用於誤差函數，這些量度屬於基本量度。

7.5.1 光斑均方根值大小（RMS Spot size）

均方根大小是從一個或多個場點出來的一堆光束就整個系統描光之後，再就這些光束與像面交點位置作標準差計算。一個理想系統就給定的場點作聚焦的話，其光斑大小爲零。令 $X(\bar{h},\lambda,\rho)$與$Y(\bar{h},\lambda,\rho)$代表光束在像面上切點的 x 與 y

值，而 λ 代表波長，而比例物座標（fractional object coordinate）爲 $\vec{h}$（$\vec{h}$ 代表物位置（h_x,h_y）），其通過入射瞳孔座標爲 ρ（即瞳孔位置（ρ_x,ρ_y）），則就所有場點平均的光斑大小的平方可表成：

$$s^2 \cong \sum_{i=1}^{\text{場點總數}} \sum_{j=1}^{\text{波長}} \sum_{k=1}^{\text{從場點}i\text{出來的光束數}} w_{ijk} \left\{ \left[X\left(\vec{h}_i,\lambda_j,\rho_k\right) - \bar{X}\left(\vec{h}_i\right) \right]^2 + \left[Y\left(\vec{h}_i,\lambda_j,\rho_k\right) - \bar{Y}\left(\vec{h}_i\right) \right]^2 \right\} \quad (7.21)$$

這裡 w_{ijk} 是光束歸一權重，而自場點出來的光斑中心之座標值給定如下：

$$\bar{X}\left(\vec{h}_i\right) \cong \sum_{j=1}^{\text{波長數}} \sum_{k=1}^{\text{從瑆點}i\text{出來的光束}} w_{ijk} X\left(\vec{h}_i,\lambda_j,\rho_k\right) \Big/ \sum_{j=1}^{\text{波長數}} \sum_{k=1}^{\text{從場點}i\text{出來的光束}} w_{ijk} \quad (7.22)$$

$$\bar{Y}\left(\vec{h}_i\right) \cong \sum_{j=1}^{\text{波長數}} \sum_{k=1}^{\text{從瑆點}i\text{出來的光束}} w_{ijk} Y\left(\vec{h}_i,\lambda_j,\rho_k\right) \Big/ \sum_{j=1}^{\text{波長數}} \sum_{k=1}^{\text{從場點}i\text{出來的光束}} w_{ijk} \quad (7.23)$$

要建構一個可以估計光斑大小的誤差函數，必須找出一個取樣及權重的決定方法。取樣指的是對場點數、場點位置、每個場點出來的光束之覓跡及光束在瞳孔的切點乃至於波長的取樣。在此，OSLO 利用高斯積分提供一組方法來自動選擇樣本位置及權重以建構一個誤差函數。這個方法在下面有關「誤差函數自動產生」（Automatic error function generation）一節特予以細述。

7.5.2 均方根值光程差（RMS Optical Path Difference, OPD）

產生明顯點像(point image)的光學系統另一個特徵是從一給定場點出來的波前形狀是以自場點出來的參考光束與像面之交點爲中心的球。所以一個光學系統其成像品質可以以相對這個參考球的偏差來界定。

從一給定場點出來的單一光束其光程差指的是，沿著光束從參考球面到真正波前的距離乘上所在介質的折射率。爲了就系統整體的成像效能作度量，我們就所有場源作平均量測均方根光程差，並引之作爲誤差函數。其形式如下。

令 $d\left(\vec{h},\lambda,\rho\right)$ 代表波長爲 λ，物座標 $\vec{h}$，通過入射瞳孔位置爲 ρ（注意在此座標均爲比例（fractional）式表示法。所以在平均過所有場點之後，估算的光程差平方值爲：

$$D^2 \cong \sum_{i=1}^{\text{場點總數}} \sum_{j=1}^{\text{波長數}} \sum_{k=1}^{\text{自場點}i\text{出來的光束數}} w_{ijk} \left[d\left(\vec{h}_i,\lambda_j,\rho_k\right) - \bar{d}\left(\vec{h}_i\right) \right]^2 \quad (7.24)$$

同樣地，在此 w_{ijk} 是光束的歸一權重，而自所有場點出來的平均光程差爲：

$$\bar{d}(\bar{h}_i) \cong \sum_{j=1}^{\text{波長數}} \sum_{k=1}^{\text{自場點}i\text{出來的光束數}} w_{ijk}\, d(\bar{h}_i, \lambda_j, \rho_k) \Big/ \sum_{j=1}^{\text{波長數}} \sum_{k=1}^{\text{自場點}i\text{出來的光束數}} w_{ijk} \qquad (7.25)$$

至於估算光斑大小的均方根值，必須有方法去定出合適的場點、波長、瞳孔的分割樣本及權重，換言之，「取樣」方法必須找出來。在下面我們會討論有關高斯積分法，它可以解決「取樣」的難題。

7.5.3 調變轉換函數（MTF）

調變轉換函數（MTF）通常是被指定用來對成像系統作效能量度。但是 MTF 在用於誤差函數時，由於其內在缺點所以限制了其用處。第一是它計算量太高，第二是它通常是一個高度非線性變數，使得它在最小平方最佳化時很不好用。這個非線性也使得在精確估算 MTF 的微分量不容易。由於上述原因，所以用於最佳化時量度效能的量常用的是跟 MTF 有很好關聯的光斑大小或光程差而不是 MTF 本身。不過，在先用光斑大小或光程差作誤差函數進行最佳化之後，再用 MTF 引入誤差函數作最佳化（這叫做最佳化微調，differential optimization）。這個作法可以來改變系統的 MTF 以吻合對系統的規格要求。

7.6 內定誤差函數（Automatic error function generation）

在先前以光束爲主所討論的誤差函數，其光束均是對一堆光束作有限總合來考量。一旦在誤差函數中的光束數變成無窮大，有限合就變成積分。很明顯能用的誤差函數不可以有無窮多條光束，決定光斑大小的積分型式還是最後要變成數值求解。換言之，這裡頭需要選擇一個有效手法將合適光束放入誤差函數計算。

考量一個轉動對稱系統，工作波長爲 λ，而物像高度爲 h。點在系統之瞳孔上以極座標表示，亦即（ρ, θ），在此 ρ 與 h 是以比例方式表示（即是第五章所提的比例座標(fractional coordinates)）。在像面上光束位置的 x 與 y 分量與 ρ, θ, λ, h 有關，亦即 x=x(ρ, θ, λ, h)，y=y(ρ, θ, λ, h)。如此在對整個瞳孔、視野及波長平均的光斑大小的平方可以寫作一個積分式。這個量可以用作誤差函數。

讓我們先在此只考慮對孔徑作積分。對視野及波長的作法，其原理一樣。就單一場點而言，光斑大小的 y 分量之平方可以寫成：

$$\phi = \int_0^{2\pi} \int_0^1 \left[y(\rho, \theta, h) - \bar{y}(h) \right]^2 \rho \, d\rho \, d\theta \qquad (7.26)$$

對實際系統而言，ϕ 的求積必須用數值解法作，而不是解析解求得。積分工作的困難在於積分取樣如何選擇，這就相當於如何選擇具代表性的光束。Forbes 指

出高斯求積法（Gaussian qnadrature methods）非常適合求解這類問題。

高斯求積除了是一個有效的數值積分法外，它有一個很好的性質—它對下述積分給出準確解（exact solution）

$$\int_0^1 \left(a_1\rho + a_3\rho^3 + \cdots + a_{2n-1}\rho^{2n-1}\right) d\rho \qquad (7.27)$$

而 n 在此可以代表取樣點，這很有幫助。舉例而言，如果我們有個軸上（on-axis）光學系統，其 9 階以上的球面像差可以忽略的話，那麼因爲 2n-1<9，所以 n 取 4 即可。換言之，光斑均方根值可以只用 4 個光束準確算出。

要對（7.26）求積，需要同時對 ρ 與 θ 作積分。角度部份的取樣應依角度 $\theta_k=(k-½)\,\pi\,N_\theta$ 來定。而 N_θ 是角度部份在 0 到 π 之間取樣點總數。而徑向部份，取樣點是以高斯積分來作，整個對孔徑求積其形式寫成：

$$\phi \cong \sum_{j=1}^{N_\rho} w_j \sum_{k=1}^{N_\theta} \left[y\left(\rho_j,\theta_k\right) - \bar{y}\right]^2 \qquad (7.28)$$

在圖 7.15，我們列出 N_ρ 與 N_θ 典型值。在圖 7.16 裡，我們畫出高斯求積分的光束分佈圖。圖 7.16 是三個環（ring）三個輪輻（spoke）。注意因爲轉動對稱，光束只需就半個孔徑作描光，同樣地要注意所有光束均是歪斜光束（skew rays），沒有沿 y 軸的光束。

N_ρ, N_θ	j,k	ρ_j	w_j	θ_k
1	1	0.7071	0.500	90
2	1	0.4597	0.2500	45
	2	0.8881.	0.2500	135
3	1	0.3357	0.1389	30
	2	0.7071	0.2222	90
	3	0.9420	0.1389	150
4	1	0.2635	0.0870	22.5
	2	0.5745	0.1630	67.5
	3	0.8185	0.1630	112.5
	4	0.9647	0.0870	157.5

圖 7.15　N_ρ 與 N_θ 典型值

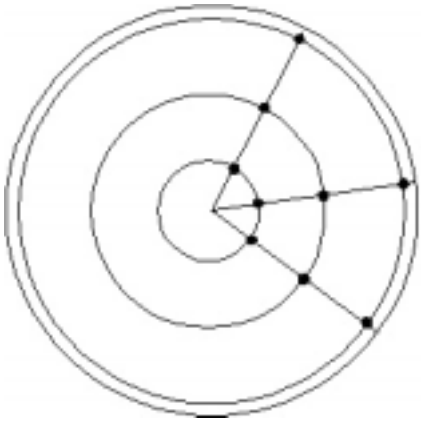

圖 7.16　環（ring）與輪輻(即圓心至圓周上的線段)（spoke）

在圖 7.17 裡，我們就高斯求積法與卡式求積法（Cartesian, square grid）作比較，比較的是求光斑均方根大小。內圖可以看出高斯求積法比較準。不過有點要提醒的是入射瞳孔是視作圓形。而且，這方法是對系統有轉動對稱才合用。如果這兩個條件不見了，卡式求積會比高斯求積較合適。

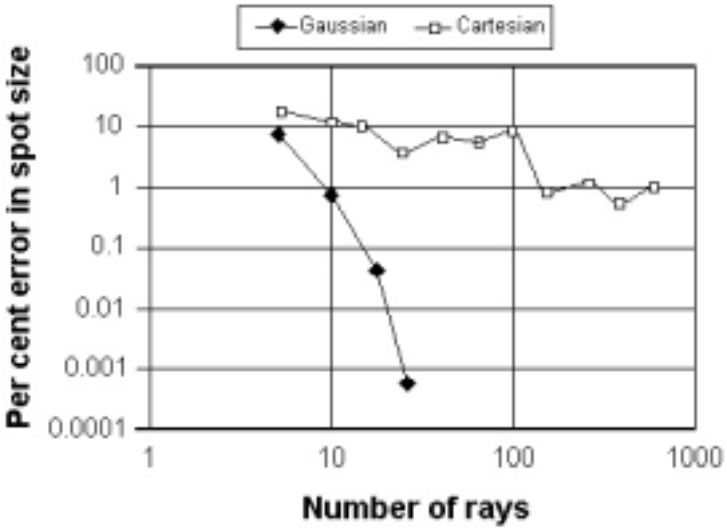

圖 7.17 高斯求積法與方塊求積法比較

就光學設計目的而言，高斯求積法也不是完全合適，原因是它在孔徑中心，沒有取樣點，在孔徑邊緣也是沒有取樣點，而這分別正是參考光束（reference）及邊緣光束（marginal ray）描光通過之處。另外兩種求積法（Radu 法和 Lobatto 法）可以彌補這個缺點。Radu 法是把通過孔徑中心的光束給補上；而 Lobatto 法是二個取樣點都補上。

在 OSLO 中「自動誤差函數產生」(automatic error function generation)[1]選項對上面四種求積法都有提供。預設的是 Lobatto 求積法。原因可從前段敘述了

[1] 我們稱爲"內定"。

解到。而求積是對整個孔徑、場源及波長。程式可以指明環數及輪輻數(見圖 7.16)，也可以視情況自由選定斑點均方根大小（rms spot size）或者波前均方根誤差（RMS wavefront error），另外也有提供讓一個輪輻與 y 軸對齊，如此子午線光束部份就可以被描光。

除了設定光束分佈形態的選項外，自動誤差函數產生器允許選定不同波長作描光的方法，還可以用 Conrady D-d 運算元一個簡化方法來定波長。分散運算元可以來控制形變（distortion）及邊緣厚度，這些不是傳統高斯求積法的範疇。

如果系統的孔徑有漸暈特徵(vignetted)，則求積法必須修改以便近似真正的瞳孔。此時，瞳孔會是橢圓，由漸暈(vignetting)因子界定，此因子放在場點集（field point set）。圓形瞳孔被轉換成漸暈(vignetted)橢圓以決定最後用在誤差函數的光束座標。

整體而言，OSLO 中的自動誤差函數產生器準確度很高，但效率中等。對典型系統用方形格子盤分佈來算誤差函數會比較準，但是相對於之前提及的最佳化所用的誤差函數而言，，卻是用了太多光束。如果需要一個效率高的最佳化，一個有經驗的設計者往往可以自行作出一個誤差函數其速度比 OSLO 內定的還要快。不過，愈來愈少人自己作，這或許是因爲電腦愈來愈快，原本需時甚久，現在已不是問題。

7.7 多組態最佳化（multiconfiguration optimization）

多組態系統意指系統的一部份其透鏡數據因組態不同而不同。伸縮放大透鏡組就是一例。另外像加上透鏡附加物，系統的光路徑因組態不同而不同，也是多組態系統的一例。甚至一些系統看起來只是一種組態由於同時用不同的操作條件最佳化，也算是多組態系統。

多組態最佳化的意思是對一個系統作最佳化使得它在任一個組態中都不是最佳，但是就整個所有可能組態而言（換言之，以統計力學中系統（ensemble）眼光來討論），效能卻是最佳化。

在 OSLO 中有一個二元格式數據結構允許快速切換儲存中多組態數據。基底組態叫做組態 1，而組態數目沒有上限，如果數據沒有定義所在組態，則視作在組態 1。

界定組態的一般數據諸如半徑、厚度、孔徑、玻璃以及近軸操作條件可以藉著設定面數據試算表而直接鍵入。特定的組態數據則可以利用組態數據試算表

（configuration data spreadsheet）來輸入。

在任何給定時段裡，我們都可以選定系統所在組態也可以由程式自動切換。透鏡設定都是在基底組態上執行。而求解（solves）也通常只在這個組態上執行，除非改寫操作條件。而打開面數據試算表或計算運算元都會讓程式自動重設基底組態。

變數可以在任何組態裡指明，也可以設爲大域變數。其作法是在組態 0 指明，換言之，變數其值在所有組態均相同。

多組態系統下的運算元必須指明在那個組態下求值，設有大域運算元，如果沒有指明組態，則內定爲基底組態。

多組態系統的最佳化跟單組態系統情況類似，唯一的差別是在計算誤差函數及設定微分矩陣時，程式是對所有組態空間都一一算過。

7.8 大域最佳化（Global optimization）

設計在最佳化階段之目標是決定出透鏡組最佳組態。這個組態是由依使用者定義的評價函數（merit function）或誤差函數的最小值來決定。而這些函數則是就系統特定參數範圍的約束條件來定。廣義而言，應追求的是這個範圍內的最小值，換言之，是大域的最小值。

概念上，一個簡單的大域最佳化法是格子盤搜尋（grid search）。它的想法是就座標軸固定一長度均分佈上格子點，形成一個格子盤，而評價函數 ϕ 則就各格子點求值，找出最小值 ϕ，ϕ 就是所要找的大域最佳化值。只是這個想法在維度大時會作不下去。舉例而言，在每個軸上有 5 點要求值，每求一次項 1μs。若有 N 個軸，則整個時間要 5^Nμs。假若 N=10，則情況還好，10 秒內完成。但 N=15 需 8 小時以上，N=20 則需「3 年」以上才做得完。可想而知，在 5 點求值這麼疏鬆要求，1μs 這麼快速的假設，格子盤搜尋結果還是不行，所以一定要另有它法才行。

最常用的方法是由設計者選擇一個起始點再用阻滯最小平方法去定出小範圍最小值，如果這個值不夠好，就重換一個新的起始點，直到可以達到要求。這個方法比格子盤搜尋快，但它跟初始點有關，而且通常結果還是跟初始的設計差不了多遠。

如果運用大域最佳化方法，我們可以在某些方面超越局部法的限制。雖然大

域法比起局部法需要更多的計算資源，但是在目前電腦的進展之下，這似乎不再是個問題。因爲大域法與初始值無關，所以當初始點選擇有困難的時候，用大域法會是一個好的解決方案。大域法中「摹擬退火法（stimulated annealing）是個吸引人的算則。「摹擬退火法」在大域法的分類是屬於「有控制的隨機搜尋」（controlled random search）法。這是因爲評價函數的求值是由幾個參數來定範圍並隨機給定。這種方法又被稱作蒙地卡羅（Monte Carlo）法（賭博法），有點像矇眼射鏢槍，但它最後還是比格子點搜尋來得有效率。蒙地卡羅法這類方法也常被用來求解多重積分。

摹擬退火法其名字源自於它是由一個叫做溫度參數 T 來控制，有些像熱退火過程裡的溫度。我們之所以稱摹擬退火是個大域法是因爲它每一次執行是對整個範圍作搜尋，而不是像其它局部法只就初始附近作最陡下坡搜尋。

我們在圖 7.18 列出摹擬退火法的流程。

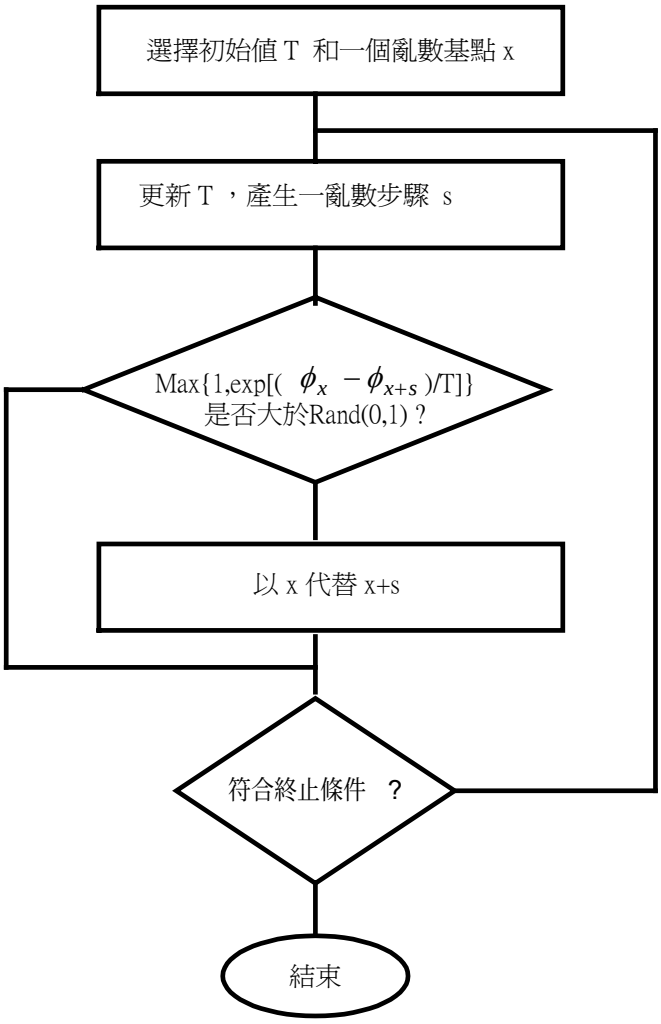

圖 7.18 摹擬退火法的流程

在摹擬退火時，最佳化並不是一直作下坡搜尋（即一直降低評價函數值 ϕ），它也允許 ϕ 場面上昇。ϕ 的上昇多少才是可以由溫度參數 T 值決定。在一開始，T 值會很大，比起 ϕ 的標準差還大，所以整個隨機範圍可以函蓋整個感興趣的範圍。一旦開始，T 會下降（退火），所以 ϕ 的增加會被限小，所以隨機範圍會因之侷限在較低的 ϕ 值（有些像跌入山谷裡）。

如果 T 的下降夠慢，那麼整個隨機運走（random walk）可以在初期跳開那些不夠小的局部極小值，並進而預期它可以稍後走到一個大域極小值。當然如果 T 下降太快，則系統可能很快就侷限在一個局部極小值而不是大域的極小值。這是跟熱退火過程類似。如果一個熱材緩慢冷卻，材質終態的位能會最小，此時原子品格排列更有序。如果冷卻很快，材質通常比較不穩定，亦即有比較高的位能，可以換成其他形態。

目前有很多種不同的摹擬退火，基本上要回答的是：

- 初始 T 如何選？T 如何調變？
- 每一步伐（step）要如何決定？

最佳化調整時大多數算則均要使用者提供參數值，以便確認上面問題。例如，關於溫度 T，使用者要指明一個介在 0 與 1 之間初始因子，藉此 T 可以不斷循環乘上此因子，以調整 T 的變動。另外，還有一個參數要用來控制循環變動該多長。而步伐（step）的調整可能更重要。這些參數因子的決定通常是嘗試錯誤經驗累積，這對簡單問題可用，但複雜問題幾近不可能。另外一件事是對座標作線性轉換成誤差函數（品價函數）既有的一些不變性（invariance），很多算則無法滿足，所以並不是所有摹擬退火算則都可以拿來用。

在 OSLO 中用的是調適摹擬退火（adaptive simulated Annealing, ASA）算則，它在組態空間下對座標或評價函數(merit function)作線性轉換，整個計算結果仍是不變。所有在摹擬退火算則裡參數均可作調適控制（adaptive）。ASA 只須指明一個參數—退火率 ε，它決定 T 下降的平均速率。一旦給定這個參數，其餘就視問題自動調整。ε 降低，則「冷卻」率變小（亦即運算執行時間要拉長），則尋找大域極小值會更徹底。

調適機制的效用與隨機決定步伐長度的統計性的自動控制有相當大的互動。在摹擬退火裡有關步伐（step）的分佈一個基本要求是它必須對稱，所以我們用高斯分佈。以二維爲例，一個廣義高斯分佈像個橢圓狀的雲離開其中心點，愈稀疏，比例方向及橢圓大小是分佈的重要參數。很明顯地，T 值變小，組態空間要檢查的範圍變小。所以步伐長度的高斯分佈必隨退火過程而改變。就任何 T

值，若其步伐長度過大，則幾乎所有嘗試都會失敗，整個最佳化過程就進展不下去。另外一方面，如果步伐長度過小，則隨機運走範圍只涵蓋組態空間的部份，而沒有參數 T 發揮其最佳化效率的影響力。同樣地高斯分佈的形狀、走向亦應隨時間調整以維持最佳效率。在 OSLO 中有個簡單的流程自動調整 n 維高斯分佈在每個退火階段其大小、形狀及走向，以維持最佳效率。

在 ASA 中步伐控制是基於中央極限定理（central limit theorem）。中央極限定理說一群獨立不相關的亂數，其平均的分佈近似於一個高斯分佈。這個結果不受個別亂數的內在分佈爲何，而平均量的變量（variance）正是個別亂數分佈變量的平均。也因爲這樣子，一個多維度高斯分佈步伐長度產生器可以用一群亂數向量作線性組合來完成。

在 ASA 裡，作法是這樣子：對之前 m 個步伐長度作記錄（這些步伐已被接受），再利用這些步伐長度作隨機線性組合並乘上適切的擴張因子（此擴張因子可動態調整）以產生新的步伐長度。這樣子作可以使所產生的步伐跟品價函數與溫度參數 T 直接關聯。這樣子作不僅有效而且簡單。不需要作太多矩陣運算，程式自動調到多維空間的相關地帶（亦即不變性仍能被保留住）。在這透鏡設計裡，不變性（諸如，稜鏡轉動其形狀應不改變等）是相當重要。另外一點是在這透鏡設計裡，組態空間的「距離」（distance）並沒有一個很自然的量度。在組態空間裡座標量指的是折射率、厚度、曲率、非球面係數等等，在這樣的空間裡要定義 2 點之間的距離其實是沒有什麼直覺的法子。

另外一點是在作最佳化時，有時步伐會超出範圍，與其直接拒絕這些步伐不如保留，原因是在範圍邊界附近往往我們讓步伐出現「振盪」一旦超出，就用「反射」反方向縮回，保留這種振盪反而效率比較高。只不過在這種過裡，爲了最佳效率，要確保整個過程的細節其形式在座標線性轉換之下是不變的。這裡有個觀念要小心的是在邊界步伐超出。用「反射」方向反方向調回，必須引入一個特定的「尺度」來定義距離。在 ASA 用的是由橢圓高斯分佈來定義。任 2 點的距離由從一點到另一點所需的步數（該方向的平均大小）決定。由於尺度已與步伐分佈關聯，所以尺度隨退火過程的進展而改變。

而 ASA 有關溫度參數的控制細節是更複雜。不過直覺上，它是這麼作的：給定一評價函數，對 T 會有一特定區間，以使得 T 的下降能有比較好的整體效率。ASA 的調適性是由退火摹擬時所處的階段其前幾個階段的統計行爲來監控，如果相對應變化超出評價函數值太多，就會終止。對任一給定問題，ASA 應該盡可能多跑幾次，而所出現的設計可以藉此再最佳化及微調。這個動作可以藉由「iterate full」或「iterate standard」來執行。

上面提到的是ASA設計的一些基本想法與理念，更多的資料可以參考Jones和Forbes的論文（Journal of global optimization 6, 1-37 (1995)）。當然設計永遠是困難，而成功的自動設計自則還是有賴設計者的指引。不同的是設計者不再被要求給初始值，但是設計者要定出設計允許範圍。

7.9 一個設計專案（A design Project）

本節示範 ASA 如何跟一般阻滯最小平方法合併作設計，目標是設計出有效焦距長（efl）35mm 三鏡，其速率爲 f/3.5，而半視角爲 31.5 度，波長爲可見光。對形變與漸暈(vignetting)沒有要求。

由於 ASA 不需要給定初始設計，所以我們先放三個厚度爲零的玻璃。我們令曲率可以在±0.15 之間，玻璃厚度在 2 到 10，空氣間隔是 2 到 20，折射率可在 1.5 與 1.9 之間變動，色散因子介於 0 與 1 之間，退火速率爲 0.05，這也是計算中止的判定值。誤差函數則是以 Lobatto 求積加上三個場點，9 道光束爲基礎共 50 項。色差用 D - d 控制，焦距長放在誤差函數內，而不是以「求解」作「約束」來進行最佳化，整個透鏡數據如圖 7.20 所示。讀者宜注意「lens data」，「Paraxial setup of lens」,「wavelength」,「variables」及「operands」各個欄位。在這裡我們設定 ASA 產生 20 個設計。在 sparc2 工作站這大概花了三個小時。之後，我們就 ASA 所產生透鏡組作修正以移去玻璃變數（glass variable）。一般阻滯最小平方法再對 ASA 所產生的設計作調整（只改變其曲率及厚度）。整個結果如圖 7.21 所示，所有 20 個設計圖樣及其相關的像差數據見圖 7.22。

*LENS DATA
ASA Triplet 35mm f3.5 31.5 deg

SRF	RADIUS		THICKNESS		APERTURE RADIUS		GLASS	SPE	NOTE
OBJ	--		1.0000e+20		6.1713e+19		AIR		
AST	--		--	V	5.000000	AS	AIR		
2	--	V	--	V	5.000000	S	GLASS1	V	
3	--	V	--	V	5.000000	S	AIR		
4	--	V	--	V	5.000000	S	GLASS2	V	
5	--	V	--	V	5.000000	S	AIR		
6	--	V	--	V	5.000000	S	GLASS3	V	
7	--	V	--	V	5.000000	S	AIR		
IMS	--		--		5.000000	S			

圖 7.20　　透鏡數據圖

*PARAXIAL SETUP OF LENS			
APERTURE			
Entrance beam radius: *	5.000000	Image axial ray slope:	5.0000e-20
Object num. aperture:	5.0000e-20	F-number:	--
Image num. aperture:	5.0000e-20	Working F-number:	1.0000e+19
FIELD			
Field angle: *	31.680000	Object height:	-6.1713e+19
Gaussian image height:	-6.1713e+19	Chief ray ims height:	--
CONJUGATES			
Object distance:	1.0000e+20	Srf 1 to prin. pt. 1:	--
Gaussian image dist.:	-1.0000e+20	Srf 7 to prin. pt. 2:	--
Overall lens length:	--	Total track length:	1.0000e+20
Paraxial magnification:	1.000000	Srf 7 to image srf:	--
OTHER DATA			
Entrance pupil radius:	5.000000	Srf 1 to entrance pup.:	--
Exit pupil radius:	5.000000	Srf 7 to exit pupil:	--
Lagrange invariant:	-3.085652	Petzval radius:	1.0000e+40
Effective focal length:	--		

圖 7.21 近軸數據

*WAVELENGTHS

CURRENT	WV1/WW1	WV2/WW2	WV3/WW3
1	0.587560	0.486130	0.656270
	1.000000	1.000000	1.000000

*VARIABLES

VB	SN	CF	TYP	MIN	MAX	DAMPING	INCR	VALUE	
V 1	1	-	TH	-40.000000	-10.000000	1.000000	1.0000e-07	--	*
V 2	2	-	TH	2.000000	10.000000	1.000000	1.0000e-07	--	*
V 3	3	-	TH	2.000000	20.000000	1.000000	1.0000e-07	--	*
V 4	4	-	TH	2.000000	10.000000	1.000000	1.0000e-07	--	*
V 5	5	-	TH	2.000000	20.000000	1.000000	1.0000e-07	--	*
V 6	6	-	TH	2.000000	10.000000	1.000000	1.0000e-07	--	*
V 7	7	-	TH	5.000000	30.000000	1.000000	1.0000e-07	--	*
V 8	2	-	CV	-0.150000	0.150000	1.000000	1.0000e-07	--	
V 9	3	-	CV	-0.150000	0.150000	1.000000	1.0000e-07	--	
V 10	4	-	CV	-0.150000	0.150000	1.000000	1.0000e-07	--	
V 11	5	-	CV	-0.150000	0.150000	1.000000	1.0000e-07	--	
V 12	6	-	CV	-0.150000	0.150000	1.000000	1.0000e-07	--	
V 13	7	-	CV	-0.150000	0.150000	1.000000	1.0000e-07	--	
V 14	2	-	RN	1.500000	1.900000	1.000000	1.0000e-07	1.600000	
V 15	2	-	DN	--	1.000000	1.000000	1.0000e-07	0.862160	
V 16	4	-	RN	1.500000	1.900000	1.000000	1.0000e-07	1.600000	
V 17	4	-	DN	--	1.000000	1.000000	1.0000e-07	0.862160	
V 18	6	-	RN	1.500000	1.900000	1.000000	1.0000e-07	1.600000	
V 19	6	-	DN	--	1.000000	1.000000	1.0000e-07	0.862160	

圖 7.22 變數設定

由於第 5 解其誤差函數值最小，所以我們用它作進一步額外的最佳化。這個最佳化包括限制光欄在中心元件的前沿（front surface）。PU 要求一從極小值到一個約束模運算元（constraint mode operand）並加上鏡組組合的經驗。在圖 7.23 我們列出結果，圖 7.23（a）用到有改善的玻璃組(improved glasses)，而（b）則是把漸暈(Vignetting)給拿走。David Shafer 建議如果我們把鏡組翻過來，整個設計可以變得更好，其結果放在圖 7.24。最後設計的透鏡數據見圖 7.25。

圖 7.23 進一步的最佳化：（a）用較佳的玻璃組；b）拿走漸暈

圖 7.24 鏡組翻轉

表 7.25 最後設計的透鏡數據

就像這類透鏡設計，最終設計在焦距長的效能不是頂佳，它是因爲 oblique 球面像差無法修正。不過，這透鏡組有好的視野，大的後焦，而且在縮小焦距長下相當合用。

這個設計專案示範 ASA 如何與一般透鏡設計合併以產生一個新的設計。在這一類應用上，追求的並不是大域最佳化。ASA 只是引用作爲一個多樣設計產生器。就這個觀點而言，ASA 比起其它方法都要好，因爲它可以對整個設計可能性都徹底找過。

參考文獻

[1] OSLO Optical Reference version 5 (Sinclair Optics, 1996); chapter 7.
[2] OSLO Optical Reference version 6.1 (Lambda Research Corp, 2001).
[3] OSLO Program Reference version 5 (Sinclair Optics, 1996)

習題

1. 重作 7.9 節之設計專案。