專題一　資料內插

摘要

實作 Laplace interpolation 方法為專題一主要核心，其目的為修補圖像中全部遺失的數值。

Laplace Interpolation 介紹

Laplace interpolation 是一種用於還原圖像中缺少數據之插值方法，雖然簡單，但在大部分情況下效果良好。

其修補方程式如下：

線性方程式	說明
y_u	為 y 點上方數值
y_d	為 y 點下方數值
y_l	為 y 點左方數值
y_r	為 y 點右方數值
$y_0 = y_0$	在點已知的情況中
$y_0 - \frac{1}{4}y_u - \frac{1}{4}y_d - \frac{1}{4}y_l - \frac{1}{4}y_r = 0$	通用求未知點方程式，為應用均值定理。
$y_0 - \frac{1}{2}y_u - \frac{1}{2}y_d = 0$	求未知點在左右邊界時方程式
$y_0 - \frac{1}{2}y_l - \frac{1}{2}y_r = 0$	求未知點在上下邊界時方程式
$y_0 - \frac{1}{2}y_r - \frac{1}{2}y_d = 0$	求未知點在左上邊界時方程式
$y_0 - \frac{1}{2}y_l - \frac{1}{2}y_d = 0$	求未知點在右上邊界時方程式
$y_0 - \frac{1}{2}y_r - \frac{1}{2}y_u = 0$	求未知點在左下邊界時方程式
$y_0 - \frac{1}{2}y_l - \frac{1}{2}y_u = 0$	求未知點在右下邊界時方程式

數值線性方程式介紹

Gaussian Elimination method:

數值線性方程式中其一算法，可用來為線性方程組求解，求一矩陣時，高斯消去法會產生出一個行梯陣式。

時間複雜度為$O(n^5)$。

Gauss–Seidel method:

Gauss-Seidel method 是數值線性方程式疊代法其中之一，可用來求出線性方程組解的近似值。

時間複雜度為$O(n^3)$。

公式如下：

$$\begin{matrix} a_{11}x_1^{(k+1)} & + & a_{12}x_2^{(k)} & + & \cdots & + & a_{1n}x_n^{(k)} & = & b_1 \\ a_{21}x_1^{(k+1)} & + & a_{22}x_2^{(k+1)} & + & \cdots & + & a_{2n}x_n^{(k)} & = & b_2 \\ \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{n1}x_1^{(k+1)} & + & a_{n2}x_2^{(k+1)} & + & \cdots & + & a_{nn}x_n^{(k+1)} & = & b_n \end{matrix}.$$

This can also be written

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \cdots & a_{nn-1} & a_{nn} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1n} \\ 0 & \cdots & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

That is,

$$(L+D)\boldsymbol{x}^{(k+1)} + U\boldsymbol{x}^{(k)} = \boldsymbol{b},$$

so that

$$\boldsymbol{x}^{(k+1)} = (L+D)^{-1}[-U\boldsymbol{x}^{(k)} + \boldsymbol{b}].$$

Successive over-relaxation:

SOR 是 gauss-seidel 的改進，通常解決大規模系統的線性等式。使用 ω 來加速收斂， 且 ω 需要介於 0~2 之間。

ω (鬆弛因子) 介紹：

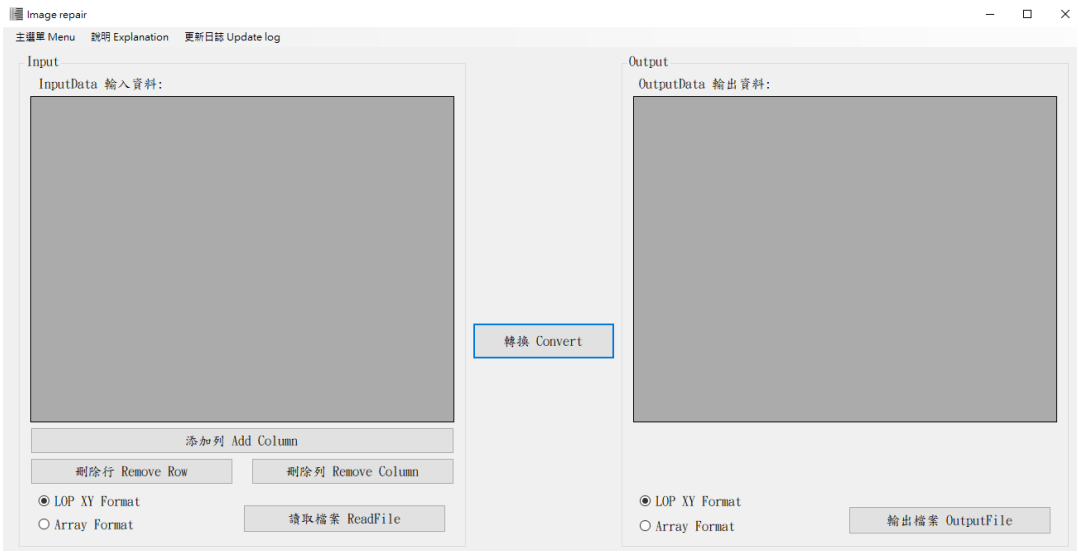
鬆弛因子數值直接影響算法的收斂性及收斂速度，若取到最適當的 ω 可以直接讓發散迭代轉為收斂，因此 ω 在 SOR 方法是否高效率的關鍵，但只有在少數特殊矩陣的類型下才可透過**數學公式**確定，如果是非特定矩陣則可以用下列以下方法取值，**二分比較法**、**逐步搜索法**、**黃金分割法**。

目前我們在程式中寫出 Gaussian Elimination method、Gauss–Seidel method、Successive over-relaxation，並在比較中發現 Successive over-relaxation 速度較快。

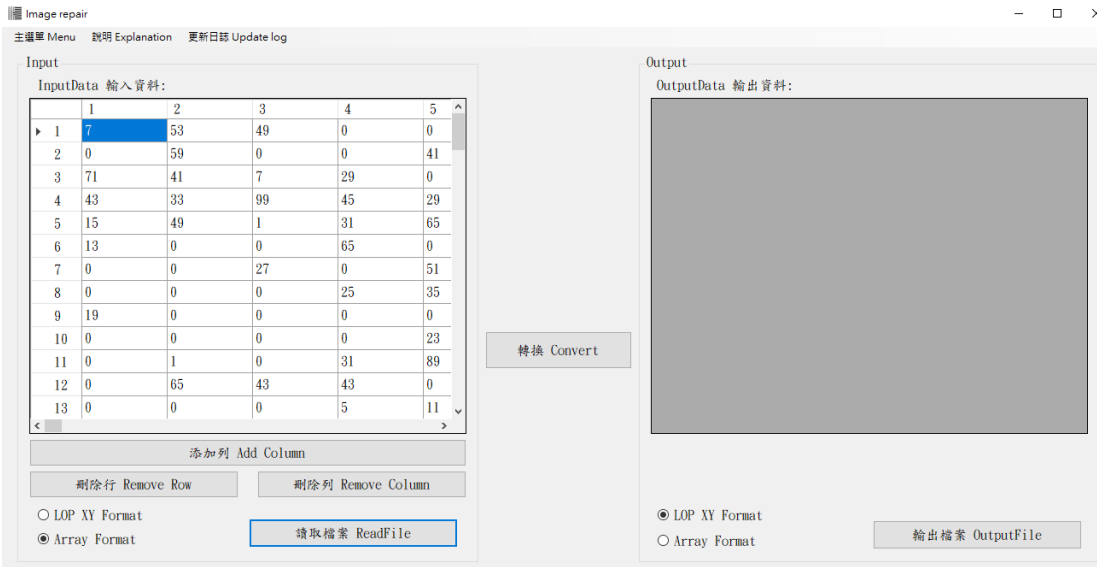
Successive over-relaxation 與 Gauss–Seidel method 比較

Gauss-Seidel	$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right)$
SOR	$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right)$

應用程式介紹

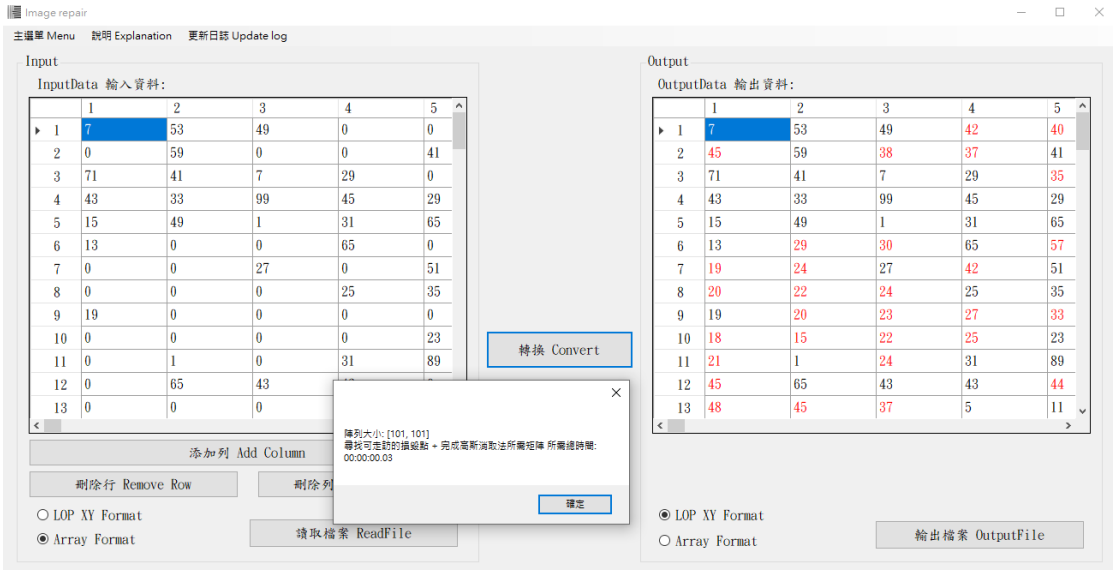


▲應用程式介面



	1	2	3	4	5
1	7	53	49	0	0
2	0	59	0	0	41
3	71	41	7	29	0
4	43	33	99	45	29
5	15	49	1	31	65
6	13	0	0	65	0
7	0	0	27	0	51
8	0	0	0	25	35
9	19	0	0	0	0
10	0	0	0	0	23
11	0	1	0	31	89
12	0	65	43	43	0
13	0	0	0	5	11

▲輸入介面，若資料數量大於 2250000 將不顯示資料



▲輸出介面，若資料數量大於 65535 將不顯示資料，在右方 Output 畫面中紅字即為修補後的數值

此應用程式其餘功能：

1. 可輸入 LOP XY 格式:序列格式
2. Array XY 格式: 二維陣列格式
3. 載入資料，可使用拖曳。
4. 可輸入資料類型為 csv, xlsx ,txt
5. 可輸出資料類型為 csv, xlsx ,txt
6. 添加列需先按按鈕，添加行則是直接輸入數值
7. 數值不需要一定是數字
8. 輸入資料並不需要是正方圖形、可以是任意形狀

時間複雜度比較

根據我們反覆測試之後資料如下，且遇一問題將再結論做說明。

Gauss–Seidel method 運行效率表如下：

矩陣大小	運行時間	作業系統	記憶體使用量
2000	37s	32 位元	≈2GB
3000	79s	64 位元	≈6GB
5000	322s	64 位元	≈16GB
8000	1495s	64 位元	≈16GB

註: Gaussian Elimination method 在 5000 時運行時間已無法再 10min 內執行完畢故不在此加入 Gaussian Elimination method 效率表

Successive over-relaxation 運行效率表

矩陣大小	運行時間	作業系統	ω(omega)	記憶體使用量
2000	23s	32 位元	1.1	≈2GB
3000	44s	64 位元	1.1	≈6GB
5000	136s	64 位元	1.1	≈16GB
8000	743s	64 位元	1.1	≈38GB

可以明顯看出在 Successive over-relaxation 的 ω 為 1.1 時，會比起原先的 Gauss–Seidel method 來的更優秀，在矩陣大小為 5000 時，兩者的運行時間會差距將到一倍以上，只要找到適合的 ω 甚至可以將運行時間優化至兩倍以上。

結論

我們在實現 Gauss–Seidel method 發現兩個問題：

1. 當我們在開一個陣列時候，陣列[0,0] 與陣列[1000,1000] 速度不一致，再呼叫陣列時[0,0] 比 [1000,1000]快了約 10 倍的速度找出，於是我們認為，陣列的大小會影響到我們從陣列提出數值的速度。後來我們在 Gauss–Seidel method 疊代陣列中進行壓縮後，效率比未壓縮效率快約 42 倍。
2. C# 在一維陣列中最多可以有多少元素阿！依據 MSDN 預設，陣列最大範圍只能有 2GB，假設此陣列為 int 時，最大元素最多只能有$2^{32}/4$。可是我們的矩陣只要大於$3000 * 3000$就沒辦法塞入阿！MSDN 有給予些許的解決方法，如果電腦為 64 位元且再 app.config 加入 gcAllowVeryLargeObjects，假如你的 RAM 夠大，就能使陣列擁有 40 億元素，若你的 RAM 無法負載 40 億元素那只能到 RAM 能負載最大值。

在這次的專題研究中我獲得了學習數值線性方程式演算法的機會，並透過教學文章、實作有更深刻的認識。

在實作過程中透過查找網路技術文章獲得了許多 C# 的一些小知識，且這些知識只有在做此專案時才容易去了解，在平常的大學課程中很少要求一個陣列數量大於 $2^{32}/4$，實作此專案時也花了很多時間 Debug 與效率優化；實踐專案來加強自己實力，也需要謝謝惠特科技與老師願意給我機會讓我去嘗試並實作此演算法，讓我對 C# 與工程數學有了更深刻的了解。

資料來源

Iterative Methods for Solving Ax = b - Gauss-Seidel Method

The SOR method