

# Chapter 5

## 迴圈與向量優化

Hung-Yuan Fan (范洪源)

Department of Mathematics,  
National Taiwan Normal University, Taiwan

Spring 2017



- 5.1 while 迴圈
- 5.2 for 迴圈
- 5.3 邏輯陣列與向量優化
- 5.6 textread 函式



# Section 5.1

## while 迴圈



## Introduction

- 迴圈 ( loops ) 是一種 MATLAB 架構，允許我們重複執行一連串的宣告式。
- 兩個基本的迴圈架構為：**while 迴圈 ( while loops )** 以及 **for 迴圈 ( for loops )**。
  - while 迴圈: 只要滿足某些條件，將重複執行特定的程式區塊，直到這些條件不滿足後，才結束迴圈，故其執行次數並非固定。
  - for 迴圈: 以一個確定的次數重複執行特定的程式區塊。
- 向量化或是向量優化 ( vectorization ) 是 MATA LB 另一種更快速的方式，用來執行如同許多 for 迴圈一樣的功能。



表 9.3.2 while 迴圈敘述

指令	說明
while 判斷條件 敘述主體 end	當判斷條件為 <b>true</b> 時，會重複執行敘述主體，直到判斷條件為 <b>false</b> 為止

```

while expression
    ...
    ...
    ...
end
    
```

} 程式碼區塊



## while 迴圈的範例

% 本範例計算  $1 + 2 + 3 + 4 + 5$  的總和。

```
i = 1; x = 0;
```

```
while i < 6
```

```
    x = x + i;
```

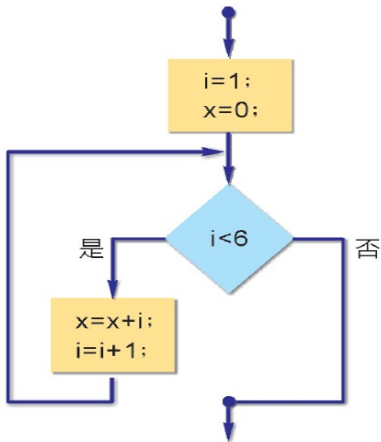
```
    i = i + 1;
```

```
end
```

```
fprintf(' 正整數 1 加到 5 的總和為 %d \n',x);
```



# while 範例的流程圖



## while 迴圈的範例

```
total = 0; num = 2;
while num <= 100
    if isprime(num)
        total = total + num;
    end
    num = num + 1;
end
fprintf('1 到 100 之間的質數總和為 %d \n',total);
```





# 範例 5.1: 統計分析 (1/5)

## 1. 宣告問題

- 計算一組資料的平均數及標準差，假設所有的測量值均為正數或零，並且使用一個負值作為資料輸入的結尾。

## 2. 定義輸入和輸出

- 輸入：一組不知道總個數的正數或零值。
- 輸出：列出輸入資料的平均數、標準差和資料點的總個數。

## 3. 設計演算法

- 這個程式將會分成 3 個主要的步驟來執行：
  - 1 收集輸入資料
  - 2 計算平均數及標準差

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i, \quad s = \frac{\sqrt{N \sum_{i=1}^N x_i^2 - \left( \sum_{i=1}^N x_i \right)^2}}{N(N-1)}$$

- 3 輸出平均數、標準差及資料點數



## 4. 將演算法變成 MATLAB 宣告式

- 虛擬碼 (pseudocode):

將  $n$ ,  $\text{sum\_x}$  及  $\text{sum\_x2}$  初始化為 0。

提示使用者輸入並讀取需要的第一個  $x$  值。

**while**  $x \geq 0$

$n \leftarrow n+1$ ;

$\text{sum\_x} \leftarrow \text{sum\_x} + x$ ;

$\text{sum\_x2} \leftarrow \text{sum\_x2} + x^2$ ;

提示使用者輸入並讀取下一個  $x$  值。

**end**

計算資料點的平均數  $\bar{x}$  與標準差  $s$ 。

輸出資料點的平均數、標準差及資料點數。



## 範例 5.1: 統計分析 (3/5)

- MATLAB 程式碼: (檔名為 stats\_1.m)

```
n = 0; sum_x = 0; sum_x2 = 0;
x = input('Enter first value: ');
while x >= 0
    n = n + 1;
    sum_x = sum_x + x;
    sum_x2 = sum_x2 + x^2;
    x = input('Enter next value: ');
end
x_bar = sum_x / n;
s = sqrt( (n*sum_x2-sum_x^2) / (n*(n-1)) );
fprintf('The mean of data is: %f \n',x_bar);
fprintf('The standard deviation is: %f \n',s);
fprintf('The number of data is: %f \n',n);
```



## 5. 測試程式

```
>> stats_1
```

```
Enter first value: 3
```

```
Enter next value: 4
```

```
Enter next value: 5
```

```
Enter next value: -1
```

```
The mean of data is: 4.000000
```

```
The standard deviation is: 1.000000
```

```
The number of data is: 3.000000
```

```
>> stats_1
```

```
Enter first value: -1
```

```
The mean of data is: NaN
```

```
The standard deviation is: NaN
```

```
The number of data is: 0.000000
```



## 範例 5.1: 統計分析 (5/5)

- 正確的 MATLAB 程式碼: (檔名為 stats\_2.m)

```
n = 0; sum_x = 0; sum_x2 = 0;
x = input('Enter first value: ');
while x >= 0
    n = n + 1;
    sum_x = sum_x + x;    sum_x2 = sum_x2 + x^2;
    x = input('Enter next value: ');
end
if n < 2
    disp('At least 2 values must be entered!');
else
    x_bar = sum_x / n;
    s = sqrt( (n*sum_x2-sum_x^2) / (n*(n-1)));
    fprintf('The mean of data is: %f \n',x_bar);
    fprintf('The standard deviation is: %f \n',s);
    fprintf('The number of data is: %f \n',n);
end
```



# Section 5.2

## for 迴圈



- **for 迴圈 ( for loops )** 是一種可執行程式區塊**特定次數**的迴圈。
- 形式如下：

表 9.3.1 for 迴圈敘述

指令	說明
for 迴圈變數=向量 敘述主體 end	將變數依序設定成向量裡的每一個元素值，然後執行敘述主體
for 迴圈變數=矩陣 敘述主體 end	將變數依序設定成矩陣裡的每一個直行，然後執行敘述主體



## for 迴圈的範例

```
total = 0;
for num = 2:100 % num = 2,3,4,...,100
    if isprime(num)
        total = total + num;
    end
end
fprintf('1 到 100 之間的質數總和為 %d \n',total);
```





## 計算向量元素的總和 (檔名: vec\_sum.m)

```
total = 0;
for num = [-5.6 -1 4 7.2] % num = -5.6 , -1, 4, 7.2
    total = total + num;
end
fprintf(' 向量元素的總和為 %f \n',total);
```

```
>> vec_sum
```

```
向量元素的總和為4.600000
```

```
>> sum([-5.6, -1, 4, 7.2])
```

```
ans =
```

```
4.6000
```



## 迴圈變數被指派為陣列 (檔名: for\_arr.m)

```
for ii = [1 2;3 4] % 矩陣  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$  被指派給變數 ii  
    ii  
end
```

```
>> for_arr
```

```
ii =
```

```
    1
```

```
    3
```

```
ii =
```

```
    2
```

```
    4
```



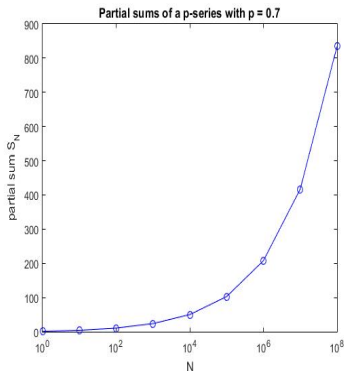
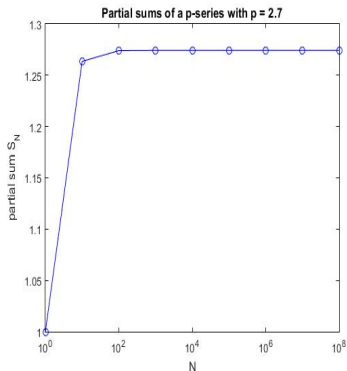
## 計算 $p$ -級數的和 (檔名: p\_series.m)

```
p = 2.7; data = [];  
for k = 0:8  
    N = 10^k;  
    n = 1:N;  
    S_N = sum(1./(n.^p));  
    data = [data;N S_N];  
end  
semilogx(data(:,1),data(:,2),'bo-');  
title('Partial sums of a p-series with p = 2.7');  
xlabel('N');  
ylabel('partial sum S_N');
```



# 程式執行結果 (承上例)

下列圖形顯示  $\sum_{n=1}^{\infty} \frac{1}{n^p}$  的收斂與發散，其中  $p = 2.7$  和  $p = 0.7$ :



## 1. 縮排迴圈的程式本體

- 向後縮排2個以上的空格，以增加程式碼的可讀性。
- 在編輯/偵錯視窗 (Editor) 中，善用“反白 MATLAB 宣告式 → 按下滑鼠右鍵 → 選取 **Smart Indent**”。

## 2. 不能在迴圈的程式本體中修改到迴圈指標

- 因為指標變數在迴圈裡通常被當成計數的工具使用，而修改迴圈指標可能會導致錯誤，或是產生難以找到的錯誤。

```
for ii = 1:10
    ...
    ii = 5;    % Error!
    ...
    a(ii) = <calculation>
    ...
end
```



## 3. 預先配置 (preallocate) 陣列記憶體空間

- 如果陣列在迴圈開始執行之前，就預先配置其所需的最大空間，當程式執行時，就不需要經過複製的過程。
- 這將大幅地增加執行迴圈的速度。例如：

```
square = zeros(1,100);  
for ii = 1:100  
    square(ii) = ii^2;  
end
```

## 4. 向量優化陣列

- 使用向量化的宣告來取代迴圈的過程，被稱為**向量優化 (vectorization)**。向量優化可以大幅地改善許多 MATLAB 程式的執行效率。
- 上述 for 迴圈的範例可以改寫成向量化版本：

```
ii = 1:100;  
square = ii.^2;
```



## MATLAB 動態編譯器

- 在 6.5 的版本以後，增加**動態 (Just-in-Time, JIT) 編譯器**的工具。
- 在執行程式碼之前先**編譯 (compile)** 它，而非直譯程式碼。
- 可以提升 `for` 迴圈的執行速度，幾乎可與**向量優化**一樣快。
- 動態編譯器有許多限制，而且這些限制因所使用的 MATLAB 版本而有所差異。
- 向量優化的方式通常能比動態編譯器更有效率。



## 範例 5.5: 比較迴圈與向量優化

請比較下列寫法所需的計算時間:

- 1 不事先初始化陣列，利用 `for` 迴圈計算 1 至 10000 每個整數的平方。
- 2 先用 `zeros` 函式去預先配置一個陣列，再利用 `for` 迴圈計算 1 至 10000 每個整數的平方 (這將啟動動態編譯器)。
- 3 利用**向量**計算 1 至 10000 每個整數的平方。

表 9.4.1 tic 與 toc 指令

指令	說明
<code>tic</code> 程式敘述 <code>toc</code>	<code>tic</code> 可啟動計時器， <code>toc</code> 則是停止計時器的執行，並顯示執行 "程式敘述" 所需的時間





```
% 檔名: timings.m  
  
% 第一種寫法  
maxcount = 1000;  
tic;  
for jj = 1:maxcount  
    clear square  
    for ii = 1:10000  
        square(ii) = ii^2;  
    end  
end  
average1 = (toc)/maxcount;
```



```
% 第二種寫法
tic;
for jj = 1:maxcount
    clear square
    square = zeros(1,10000);
    for ii = 1:10000
        square(ii) = ii^2;
    end
end
average2 = (toc)/maxcount;
```



```
% 第三種寫法
tic;
for jj = 1:maxcount
    clear square
    ii = 1:10000;
    square = ii.^2;
end
average3 = (toc)/maxcount;

% 顯示三種寫法的計算時間
fprintf('Loop / uninitialized array = %8.5f \n', average1);
fprintf('Loop / initialized array / JIT = %8.5f \n', average2);
fprintf('Vectorized = %8.5f \n', average3);
```



>> **timings**

Loop / uninitialized array = **0.00261**

Loop / initialized array / JIT = **0.00010**

Vectorized = **0.00002**



備註：上述結果的運算環境為

- 機器型號：ASIS Nootebook (G501JW)
- 微處理器：Intel Core i7-4720HQ CPU@2.60GHz  
2.59GHz
- 記憶體數：12 GB
- 作業系統：Windows 8.1
- 軟體版本：MATLAB R2015a



# 使用 break 與 continue 指令

表 9.3.3 break 與 continue 指令 (以 for 迴圈為例)

指令	說明
<pre>for 迴圈變數=向量   敘述主體 1   break   敘述主體 2 end for 迴圈之後的敘述</pre> 	當程式執行到 <b>break</b> 敘述時，即會離開迴圈，繼續執行迴圈外的下一個敘述，如果 <b>break</b> 敘述出現在巢狀迴圈中的內層迴圈，則 <b>break</b> 敘述只會跳離當層迴圈。
<pre>for 迴圈變數=向量   敘述主體 1   continue   敘述主體 2 end for 迴圈之後的敘述</pre> 	當程式執行到 <b>continue</b> 敘述時，即會停止執行剩餘的迴圈主體，回到迴圈的開始處繼續執行。



指令 `break` 的範例 (檔名: `test_break.m`)

```
for ii = 1:5
    if ii == 3
        break;
    end
    fprintf('ii = %d \n', ii);
end
disp(['End of loop!']);
```

>> **test\_break**

ii = 1

ii = 2

End of loop!



指令 `continue` 的範例 (檔名: `test_continue.m`)

```
for ii = 1:5
    if ii == 3
        continue;
    end
    fprintf('ii = %d \n', ii);
end
disp(['End of loop!']);
```

>> **test\_continue**

ii = 1

ii = 2

ii = 4

ii = 5

End of loop!



## 使用巢狀迴圈的注意事項

- 如果一個迴圈內有另外一個迴圈，這兩個迴圈稱為**巢狀迴圈 (nested loops)**。
- 如果 for 迴圈為巢狀結構，它們應該擁有**獨立的指標變數**。
- 如果 break 或 continue 指令在巢狀迴圈內出現，則這些宣告與**包含它們的最內層迴圈有關聯**。
- 如果可能的話，請使用向量優化的宣告式取代巢狀迴圈結構。
- 在使用巢狀迴圈之前，請預先分配記憶體給迴圈內所運行的變數或陣列。





## break 和巢狀迴圈結合的範例 (檔名: nested\_for.m)

```
for ii = 1:3
    for jj = 1:3
        if jj == 3
            break;
        end
        product = ii * jj;
        fprintf('%d * %d = %d \n',ii,jj,product);
    end % for jj
    fprintf('End of inner loop \n');
end % for ii
fprintf('End of outer loop \n');
```



## 輸出結果 (承上例)

```
>> nested_for
1 * 1 = 1
1 * 2 = 2
End of inner loop
2 * 1 = 2
2 * 2 = 4
End of inner loop
3 * 1 = 3
3 * 2 = 6
End of inner loop
End of outer loop
```



# Section 5.3

## 邏輯陣列與向量優化



## 邏輯陣列的特性

- 由邏輯值 `true(1)` 或 `false(0)` 所形成的陣列稱為**邏輯陣列 (logical array)**。
- 邏輯陣列可以當做算術運算的**遮罩 (mask)** 陣列。
- 遮罩陣列是一個可以用來選擇某些陣列的元素，做為算術運算的元素。指定的運算會作用在選擇的元素上，而不會作用在其餘未被選擇的元素上。
- 這是一個**非常快速而聰明的**方式來執行陣列子集的運算，因其**不需使用任何的迴圈或分支結構**。



# 遮罩陣列的範例

```
a = [1 2 3; 4 5 6; 7 8 9]; c = a;
>> a                                >> b = a > 5 % 邏輯陣列
a =                                    b =
    1    2    3                        0    0    0
    4    5    6                        0    0    1
    7    8    9                        1    1    1
>> a(b) = sqrt(a(b))                >> c(~b) = c(~b).^2
a =                                    c =
    1.0000    2.0000    3.0000          1    4    9
    4.0000    5.0000    2.4495          16   25    6
    2.6458    2.8284    3.0000          7    8    9
```



## MATLAB 的原始程式碼

```
a = [1 2 3; 4 5 6; 7 8 9];  
for ii = 1:size(a,1)  
    for jj = 1:size(a,2)  
        if a(ii,jj) > 5  
            a(ii,jj) = sqrt(a(ii,jj));  
        else  
            a(ii,jj) = a(ii,jj)^2;  
        end % if  
    end % jj  
end % ii
```



# 使用邏輯陣列產生等效的 if/else 架構 (2/2)

## 邏輯陣列可產生等效的分支敘述

```
a = [1 2 3; 4 5 6; 7 8 9];
```

```
b = a > 5;
```

```
a(b) = sqrt(a(b)); % 取大於 5 的元素開平方根
```

```
a(~b) = a(~b).^2; % 取小於或等於 5 的元素計算平方
```

```
>> a
```

```
a =
```

```
1.0000    4.0000    9.0000
16.0000   25.0000   2.4495
2.6458    2.8284    3.0000
```



## 邏輯陣列應用於基本列運算上

試用基本列運算 (elementary row operation) 與邏輯陣列 (或是遮罩陣列) · 將矩陣

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

的第一行元素  $a_{21} = 2, a_{31} = 3, a_{41} = 4$  全部消為零。





# MATLAB 程式碼 (1/2)

```
>> A = [1 0 1 0; 2 2 2 2; 3 3 3 3; 4 4 4 4];  
>> jj = logical(A(1,:));  
% 非零元素設為 true(1) · 零元素設為 false(0)  
>> A(2,jj) = (-2) * A(1,jj) + A(2,jj)
```

A =

```
1 0 1 0  
0 2 0 2  
3 3 3 3  
4 4 4 4
```



# MATLAB 程式碼 (2/2)

```
>> A(3,jj) = (-3) * A(1,jj) + A(3,jj)
```

```
A =
```

```
1 0 1 0  
0 2 0 2  
0 3 0 3  
4 4 4 4
```

```
>> A(4,jj) = (-4) * A(1,jj) + A(4,jj)
```

```
A =
```

```
1 0 1 0  
0 2 0 2  
0 3 0 3  
0 4 0 4
```



# Section 5.6

## textread 函式



表 17.1.5 讀取摻雜文字與數據資料的檔案

函數	說明
<code>[a,b,...]=textread('fname','format')</code>	依 <i>format</i> 所記載的格式從檔案 <i>fname</i> 裡讀取資料。 <i>format</i> 常用的格式如下：  %n - 可讀取整數或浮點數  %d - 讀取含正負號的整數  %f - 讀取浮點數的數據  %s - 讀取由空白鍵隔開的字串



## 函式 textread 的使用範例 (1/2)

```
>> type test_input.dat
James Jones 0+ 3.51 23 Yes
Sally Smith A+ 3.28 22 No

>> [first,last,blood,gpa,age,answer] = ...
textread('test_input.dat','%s %s %s %f %d %s');

>> whos
Name      Size  Bytes  Class
age       2 x 1   16     double
answer    2 x 1  234    cell
blood     2 x 1  232    cell
first     2 x 1  244    cell
gpa       2 x 1   16     double
last      2 x 1  244    cell
```



## 函式 textread 的使用範例 (2/2)

```
>> clear all
```

% 函式 textread 允許忽略某些資料欄位不被讀取，例如：

```
>> [first,last,gpa] = textread('test_input.dat',...  
'%s %s %*s %f %*d %*s');
```

```
>> whos
```

Name	Size	Bytes	Class
first	2 × 1	244	cell
gpa	2 × 1	16	double
last	2 × 1	244	cell



MATLAB 支援四種常用的資料檔案格式，如下表所示：

表 17.1.1 檔案類型與相關資訊

檔案格式	說明	讀取函數	寫入函數
MAT	儲存 Matlab 工作區的變數所產生的檔案	load()	save()
CSV	Comma-separated value，即以逗號隔開的數據檔案	csvread()	csvwrite()
DLM	Delimited text，以特定的分隔符號隔開的數據檔案	dlmread()	dlmwrite()
TAB	Tab-separated text，以 Tab 鍵隔開的數據檔案	dlmread()	dlmwrite()



# 以逗點隔開的資料檔案

- 以逗號隔開的資料稱為 **CSV(comma separated value)**。
- 函式 `csvread` 與 `csvwrite` 可用來存取 CSV 型態的資料。

表 17.1.3 存取由逗號隔開的數據資料

函數	說明
<code>m=csvread('filename')</code>	讀取以逗號為分隔符號的數據資料，並以 <b>double</b> 型態儲存到變數 <i>m</i> 裡
<code>csvwrite('filename',m)</code>	將數據資料以 <b>csv</b> 的格式（即以逗號為數據的分隔符號）寫到檔案 <i>filename</i> 裡





# CSV 檔案的寫入與讀取

```
>> A = magic(3) %A 是一個 3 × 3 魔術方陣
```

```
A =
```

```
8 1 6
3 5 7
4 9 2
```

```
>> csvwrite('magic3.csv',A); % 以逗點為分隔符號
```

```
>> type magic3.csv
```

```
8,1,6
```

```
3,5,7
```

```
4,9,2
```

```
>> B = csvread('magic3.csv')
```

```
B =
```

```
8 1 6
3 5 7
4 9 2
```



# 以特定符號隔開的數據處理

- 函式 `dlmread` 與 `dlmwrite` 可存取不是以逗號分隔的數據資料。
- `dlim` 是 `delimiter` 的縮寫，分隔符號之意。

表 17.1.4 處理以特定符號隔開的數據

函數	說明
<code>m=dlmread('filename','dlim')</code>	讀取以 <code>dlim</code> 為分隔符號的數據資料，並以 <code>double</code> 型態儲存到變數 <code>m</code> 裡
<code>dlimwrite('filename',m,'dlim')</code>	以 <code>dlim</code> 為分隔符號來儲存數據資料 <code>m</code>



# DLM 檔案的寫入與讀取

```
>> A = magic(3); %A 是一個 3 × 3 魔術方陣
>> dlmwrite('magic3.dlm',A,'\t'); % 以空白為分隔符號
>> type magic3.dlm

8      1      6
3      5      7
4      9      2

>> B = dlmread('magic3.dlm','\t')

B =
     8     1     6
     3     5     7
     4     9     2
```



表 17.2.1 開檔與關檔的函數

函數	說明
<code>fid=fopen('filename', 'permission')</code>	讀取檔案的內容，其中 <i>filename</i> 為欲開啟的檔案名稱， <i>permission</i> 為檔案的存取模式，並傳回檔案識別碼，由變數 <i>fid</i> 接受
<code>fclose(fid)</code>	關閉檔案識別碼為 <i>fid</i> 的檔案

表 17.2.2 檔案存取模式

存取模式	代碼	說明
讀取資料	r	開啟檔案以供讀取。在開啟前，此檔案必須先存在於磁碟機內。如果檔案不存在，則開檔失敗
寫入資料	w	開啟檔案以供寫入。如果檔案已經存在，則其內容將被覆蓋掉。如果檔案不存在，則系統會自行建立此檔案
附加於檔案之後	a	開啟一個檔案，可將資料寫入此檔案的末端。如果檔案不存在，則系統會自行建立此檔案
讀取與附加	a+	可讀取檔案，也可附加資料於檔案之後



表 17.2.3 檔案寫入與讀取函數

函 數	說 明
<code>fprintf(fid, 'str', e<sub>1</sub>, e<sub>2</sub>, ...)</code>	依格式字串 <i>str</i> 所記載的格式碼，依序將運算式 <i>e<sub>1</sub></i> , <i>e<sub>2</sub></i> 填入 <i>str</i> 裡，並將它寫入檔案識別碼為 <i>fid</i> 的檔案中。下面列出格式字串裡常用的格式碼：  %d：寫入整數 %f：寫入浮點數 %c：寫入字元 %s：寫入字串
<code>fscanf(fid, 'str')</code>	依格式字串 <i>str</i> 所記載的格式碼，讀取檔案識別碼為 <i>fid</i> 之檔案裡的資料
<code>fscanf(fid, 'str', n)</code>	一次讀取 <i>n</i> 筆資料
<code>fscanf(fid, 'str', [m, n])</code>	一次讀取 <i>m</i> × <i>n</i> 筆資料，並以 <i>m</i> × <i>n</i> 的陣列回應讀取的結果



## 計算 $p$ -級數的部分和 (檔名: test\_pseries.m)

```
clc,clear all;

fid = fopen('results.txt','w');
fprintf('      N                S_N \n');
fprintf(fid,'      N                S_N \n');

p = 2.7;
for k = 0:8
    N = 10^k;
    n = 1:N;
    S_N = sum(1./(n.^p));
    fprintf('10^%d %12.6f \n',k,S_N);
    fprintf(fid,'10^%d %12.6f \n',k,S_N);
end
fclose(fid);
```



# 檢視計算結果 (承上頁)

```
>> test_pseries
```

除了螢幕上會顯示  $N$  與  $p$ -級數部分和  $S_N = \sum_{n=1}^N \frac{1}{n^p}$  的計算結果之外，這些數據資料同時也會寫入 ASCII 格式的文字檔案 `results.txt` 中。

```
>> type results.txt
```

N	S_N
10 <sup>0</sup>	1.000000
10 <sup>1</sup>	1.263481
10 <sup>2</sup>	1.274032
10 <sup>3</sup>	1.274260
10 <sup>4</sup>	<b>1.274265</b>
10 <sup>5</sup>	<b>1.274265</b>
10 <sup>6</sup>	<b>1.274265</b>
10 <sup>7</sup>	<b>1.274265</b>
10 <sup>8</sup>	<b>1.274265</b>



# Thank you for your attention!

