前言

大概有將近 10 年的時間，開始有人對於如何把數學和統計學應用到日常工作與生活中感到興趣。這是為什麼呢？這是否與被哈佛商業評論稱為「二十一世紀最令人垂涎的工作」（*https://oreil.ly/GslO6*）的「資料科學」日益增加的興趣有關？或者它是機器學習和「人工智慧」會改變我們生活的承諾？還是因為新聞頭條充斥著研究、民意調查和研究結果，但我們不確定如何審查這些說法？或者它是「自動駕駛」汽車和機器人在不久的將來實現工作自動化的一種承諾？

我能證明，數學和統計學學科已經引起廣泛興趣，因為資料的可用性越來越高，我們需要數學、統計學和機器學習來理解它。是的，我們確實有科學工具、機器學習和其他像警笛一樣召喚我們的自動化。我們盲目相信這些「黑箱」、裝置和軟體；儘管不理解它們，但還是會使用它們。

雖然很多人相信電腦比我們更聰明（而且經常有人如此堅持），但恰恰相反，這在很多層面上都與事實脫節。在開發人員和任何人都無法解釋演算法及人工智慧如何做出決定的情況之下，您還有辦法信任用這項技術執行刑事判決或駕駛車輛嗎？可解釋性（explainability）是統計計算和人工智慧下一個要努力的目標，但只有打開黑箱並解開數學之謎後，這件事才有可能發生。

您可能還會問，開發人員怎麼可能不知道他們自己的演算法如何運作？我們將在本書的後半部分談到機器學習技術時討論這一點，並強調為什麼需要了解黑箱背後的數學。

另一方面，我們在日常頻繁地使用互相連接的裝置，而得以大規模蒐集資料。不只是桌上型或筆記型電腦，現在連智慧型手機、汽車和家用裝置都在使用網際網路，生活中無所不在。這隱然促成過去 20 年的轉變，讓資料從一種操作型工具，演變成為了不明確目標而蒐集和分析的數據。智慧型手錶不斷蒐集與我們的心率、呼吸、步行距離和其他相關資料，再將這些資料上傳到雲端中，以便與其他使用者一起分析；電腦化的汽車也正在蒐集我們的駕駛習慣，製造商再以此資料來啟用自動駕駛。就連「智慧型牙刷」也正在進入藥局，讓藥局得以追蹤我們的刷牙習慣，並將資料儲存在雲端中；但智慧型牙刷資料是否有用或有其必要性？這又是另一個值得討論的問題！

這些資料蒐集都已滲透到我們生活的每一個角落，某些層面來說令人難以承受，可以寫出一整本關於隱私問題和道德的書。但是，這種資料的可用性，也替數學和統計學創造出在學術環境以外的新興使用方法機會，我們可以更了解使用者體驗、改進產品設計和應用、優化商業策略。如果您認同本書提出的想法，您將能夠理解資料儲存基礎設施所蘊含的價值。這不表示資料和統計工具是解決世界上所有問題的靈丹妙藥，但它們的確提供了可以使用的新工具；有時，光意識到某些資料專案就像兔子洞一樣令人感到不知所措就值得了，因為這讓我們知道最好將精力放在其他地方上。

這種不斷增長的資料可用性，讓資料科學和機器學習成為熱門專業。我們將基本數學（essential math）定義為對機率、線性代數、統計學和機器學習的接觸。如果您正在尋找資料科學、機器學習或工程領域的職業，這些主題是必要的。我將提供足夠的大學數學、微積分和統計資料，以讓您更能了解將會遇到的黑箱程式庫中的內容。

透過這本書，我的目標是讓讀者了解適用於真實世界問題的不同數學、統計和機器學習領域。前四章涵蓋基礎數學概念，包括實用微積分、機率、線性代數和統計學；最後三章將繼續介紹機器學習。教授機器學習的最終目的是要整合我們所學的一切，並展示在使用機器學習和統計程式庫時的實用洞察，以超越黑箱式的理解。

遵循範例所需的唯一工具是 Windows / Mac / Linux 電腦和您選擇的 Python 3 環境。我們需要的主要 Python 程式庫是 `numpy`、`scipy`、`sympy` 和 `sklearn`。如果您不熟悉 Python 的話，它是一種友善且容易上手的程式語言，背後有海量的學習資源。以下推薦一些學習用書：

Data Science from Scratch, 2nd Edition，Joel Grus 著（O'Reilly）

> 這本書的第二章是我看過最好的 Python 速成課程。即使您以前從未編寫過程式碼，Joel 也出色地完成他的工作，讓您盡可能在最短的時間內，有效啟動並執行 Python。這也是一本很棒的書，可以放一本在書架上，隨時運用其中的數學知識！

第一章

基本數學和微積分複習

我們將在第一章介紹數字的定義，以及如何在笛卡爾（Cartesian）系統上運作變數和函數，接著將介紹指數和對數，並學習微積分的兩個基本運算：微分和積分。

在深入探討基本數學的應用領域（例如機率、線性代數、統計學和機器學習）之前，我們也應該回顧一些基本的數學和微積分概念。在您丟下這本書並尖叫著跑掉之前，別擔心！我將以您在大學裡可能沒有學過的方式，來介紹計算函數的微分和積分法則。我們身邊有 Python 在，而不是鉛筆和紙，即使您不熟悉微分和積分，也不必擔心。

我會讓這些主題盡可能地緊湊和實用，只會關注在後面的章節中對我們有幫助的內容，以及屬於「基本數學」的範疇。

這不是完整的數學速成課程！

這絕不是對高中和大學數學的全面複習。如果您想要的是那種書，推薦你 Ivan Savov 的 *No Bullshit Guide to Math and Physics*（請原諒我說粗話）。前幾章是我見過最好的高中和大學數學速成課程。Richard Elwes 博士所著的 *Mathematics 1001* 一書也有一些很棒的內容，而且也解釋地很簡單。

數論

什麼是數字？我保證不會在這本書中過於哲學化，但是數字不是我們已經定義好的結構嗎？為什麼我們的數字只有從 0 到 9？為什麼我們有分數和小數，而不僅僅是整數？思考數字以及為何要以某種方式來設計它們的這個領域，可稱為數論。

數論（*number theory*）可以追溯到遠古時代，當時數學家研究不同的數字系統（number system），形成今日最常見的模式。以下是您可能認識的各種數字系統：

自然數（*natural number*）

指數字 1、2、3、4、5……等等。這裡只包括正數，它們是已知最早的系統。自然數非常古老，穴居人就已在骨頭和洞穴牆壁上刻出計數標記，以求記錄。

非負整數（*whole number*）

「0」的概念後來被接受並添加到自然數中；我們稱這些為「非負整數」。巴比倫人還提出了有用的想法，也就是為大於 9 的數字（例如「10」、「1,000」或「1,090」）上的空「行」使用占位符。這些 0 表示沒有值占用該行。

整數（*integer*）

整數包括正的、負的自然數以及 0。現在看來理所當然，但古代數學家對負數的概念十分懷疑。但是當您用 3 減去 5 時會得到 –2，這在衡量損益的財務方面尤其有用。西元 628 年，一位名叫 Brahmagupta 的印度數學家，證明負數對於使用二次式來算術有其必要性，因此有了整數的概念。

有理數（*rational number*）

任何可以表示為分數的數字，例如 2/3，都是有理數。這包括所有有限小數和整數，因為它們也可以表示為分數，例如 687/100 = 6.87 和 2/1 = 2。它們被稱為**有理**（*rational*），因為它們是**比率**（*ratio*）。有理數相當重要，因為時間、資源和其他數量並不總是以離散的單位來衡量。牛奶並不總是以 1 加侖，它有可能只占 1 加侖的一部分；如果我跑步跑了 12 分鐘，共跑了 9/10 英哩，也沒辦法直接用「1 英哩」來測量。

無理數（*irrational number*）

無理數無法以分數表達。這包括著名的 π、某些數字的平方根（例如 $\sqrt{2}$），和我們後面會介紹的歐拉數（Euler's number）e。這些數字有無限多的小數，例如 3.141592653589793238462…

無理數背後有一段有趣的歷史。希臘數學家畢達哥拉斯（Pythagoras）認為所有數字都是有理數。他堅信不疑到因此創立了一個向數字 10 祈禱的宗教。「祝福我們，神聖的數字，您創造了神和人！」他和他的追隨者會向 10 祈禱（我不知道為什麼他覺得「10」這麼特別）。據說，他的追隨者之一希帕索斯（Hippasus）只用 2 的平方根，就證明並非所有數字都是有理數，而這嚴重破壞畢達哥拉斯的信仰體系，於是他在海上淹死了希帕索斯。

無論如何，我們現在知道，並非所有數字都是有理數。

實數（*real number*）

實數包括有理數和無理數。實際上，當您進行任何資料科學工作時，您可以將用到的任何小數視為實數。

複數（*complex number*）和虛數（*imaginary number*）

取負數的平方根時就會遇到這種數字類型。雖然複數和虛數會和某些類型的問題相關，但我們大多會避開它們。

在資料科學中，您會發現大部分工作將使用非負整數、自然數、整數和實數。在更進階的使用案例中可能會遇到虛數，例如我們將在第 4 章討論的矩陣分解。

複數和虛數

如果您真的想要更了解虛數，可見以下 YouTube 的播放清單，非常棒：*Imaginary Numbers are Real*（*https://oreil.ly/bvyIq*）。

運算順序

希望您已經熟悉了**運算順序**（*order of operations*），也就是數學運算式中每個部分求解的順序。這裡簡單複習一下，先計算括號中的成分、然後是指數、然後是乘法、除法、加法和減法。您可以透過助記碼 PEMDAS（Please Excuse My Dear Aunt Sally）來記住運算的順序，它依序對應到括號（parenthesis）、指數（exponent）、乘法（multiplication）、除法（division）、加法（addition）和減法（subtraction）。

以這個運算式為例：

$$2 \times \frac{(3+2)^2}{5} - 4$$

首先計算括號（3 + 2），結果等於 5：

$$2 \times \frac{(\mathbf{5})^2}{5} - 4$$

接下來求解指數，也就是剛剛相加後所得的 5 的平方，25：

$$2 \times \frac{\mathbf{25}}{5} - 4$$

接下來是乘法和除法。這兩者的順序可以交換，因為除法也是乘法（使用分數）的一種。相乘 2 和 $\frac{25}{5}$，得到 $\frac{50}{5}$：

$$\frac{50}{5} - 4$$

接下來執行除法，把 50 除以 5，得到 10：

$$\mathbf{10} - 4$$

最後，執行任何加法和減法。當然，10 – 4 將得到 6：

$$10 - 4 = 6$$

如果用 Python 來表達這一點，也會得到 6.0 的值，如範例 1-1 所示。

範例 1-1　在 Python 中求解運算式

```
my_value = 2 * (3 + 2)**2 / 5 - 4

print(my_value) # 印出 6.0
```

這是基本且關鍵的概念。在程式碼中，就算沒有使用括號，仍然可以得到正確的結果，但在複雜運算式中大量使用括號，才可以掌控計算的順序。

在這裡，我把運算式的分數部分放在括號中，這有助於把它和範例 1-2 中的其餘運算式區分開來。

範例 1-2　在 Python 中使用括號以更清晰

```
my_value = 2 * ((3 + 2)**2 / 5) - 4

print(my_value) # 印出 6.0
```

雖然這兩個範例在技術上都是正確的，但後者對容易混淆的人來說比較清楚。如果您或其他人更改程式碼，括號能提供您在更改時運算順序的簡單參考。這為程式碼的更改提供一道防線，以防止出錯。

變數

如果您使用 Python 或其他程式語言編寫過一些腳本，您就會知道何謂變數。在數學中，變數（*variable*）是用於未指定或未知數字的命名占位符。

您可能有一個代表任何實數的變數 x，您可以把這個變數相乘而無須宣告它是什麼。在範例 1-3 中，我們從使用者那裡獲取一個變數輸入 x 並把它乘以 3。

範例 1-3　一個 Python 中的變數，然後把它相乘

```
x = int(input("Please input a number\n"))

product = 3 * x

print(product)
```

某些變數類型有一些標準的變數名稱。如果您對這些變數名稱和概念不熟悉，請不用擔心！有些讀者可能會注意到我們使用 theta θ 來表示角度，使用 beta β 來表示線性迴歸（linear regression）中的參數。希臘符號讓 Python 中的變數名稱很難用，因此我們可能會在 Python 中把這些變數命名為 `theta` 和 `beta`，如範例 1-4 所示。

範例 *1-4*　*Python* 中的希臘變數名稱

```
beta = 1.75
theta = 30.0
```

另請注意，可以用下標（subscript），讓變數有多個名稱。在實務上，只需把它們看作是單獨的變數。如果遇到變數 x_1、x_2 和 x_3，只需把它們看作是 3 個個別的變數，如範例 1-5 所示。

範例 *1-5*　在 *Python* 中表示下標變數

```
x1 = 3  # 或 x_1 = 3
x2 = 10 # 或 x_2 = 10
x3 = 44 # 或 x_3 = 44
```

函數

函數（*function*）是定義兩個或多個變數之間關係的運算式。更具體地說，函數接受**輸入變數**（*input variable*，也稱為**定義域變數**（*domain variable*）或**自變數**（*independent variable*）），把它們插入到運算式中，產生**輸出變數**（*output variable*，也稱為**因變數**（*dependent variable*））。

看看這個簡單的線性函數：

$$y = 2x + 1$$

對於任何給定的 x 值，我們使用這個 x 來求解運算式以找到 y。當 $x = 1$ 時，則 $y = 3$。當 $x = 2$ 時；$y = 5$。當 $x = 3$ 時，$y = 7$，依此類推，如表 1-1 所示。

表 1-1　$y = 2x + 1$ 的不同值

x	2x + 1	y
0	2(0) + 1	1
1	2(1) + 1	3
2	2(2) + 1	5
3	2(3) + 1	7

函數很有用，因為它們能對變數之間的可預測關係建模，例如在 x 溫度下我們可以預期會發生 y 次火災。我們將在第 5 章中使用線性函數來執行線性迴歸。

變數 y 的另一個慣例，是把它外顯式地標記為 x 的函數，例如 $f(x)$。因此，除了將函數表達為 $y = 2x + 1$，我們也可以將之表達為：

$$f(x) = 2x + 1$$

範例 1-6 顯示如何宣告一個數學函數，並在 Python 中對其進行迭代。

範例 1-6　在 Python 中宣告一個線性函數

```
def f(x):
    return 2 * x + 1


x_values = [0, 1, 2, 3]

for x in x_values:
    y = f(x)
    print(y)
```

在處理實數時，函數的一個微妙但重要的特徵是它們通常具有無限數量的 x 值和生成的 y 值。你可以想一下：我們可以透過函數 $y = 2x + 1$ 輸入多少 x 值？為什麼只是 0, 1, 2, 3……而不是如表 1-2 所示的 0, 0.5, 1, 1.5, 2, 2.5, 3？

表 1-2　$y = 2x + 1$ 的不同值

x	2x + 1	y
0.0	2(0) + 1	1
0.5	2(.5) + 1	2
1.0	2(1) + 1	3
1.5	2(1.5) + 1	4
2.0	2(2) + 1	5
2.5	2(2.5) + 1	6
3.0	2(3) + 1	7

或者，為什麼 x 不是每 1/4 為一步？還是 1/10 為一步？我們可以讓這些步長變得無限小，有效地展示了 $y = 2x + 1$ 是一個**連續函數**（*continuous function*），其中對於 x 的每個可能值都有一個 y 值。這讓我們將函數視覺化為一條線，如圖 1-1 所示。

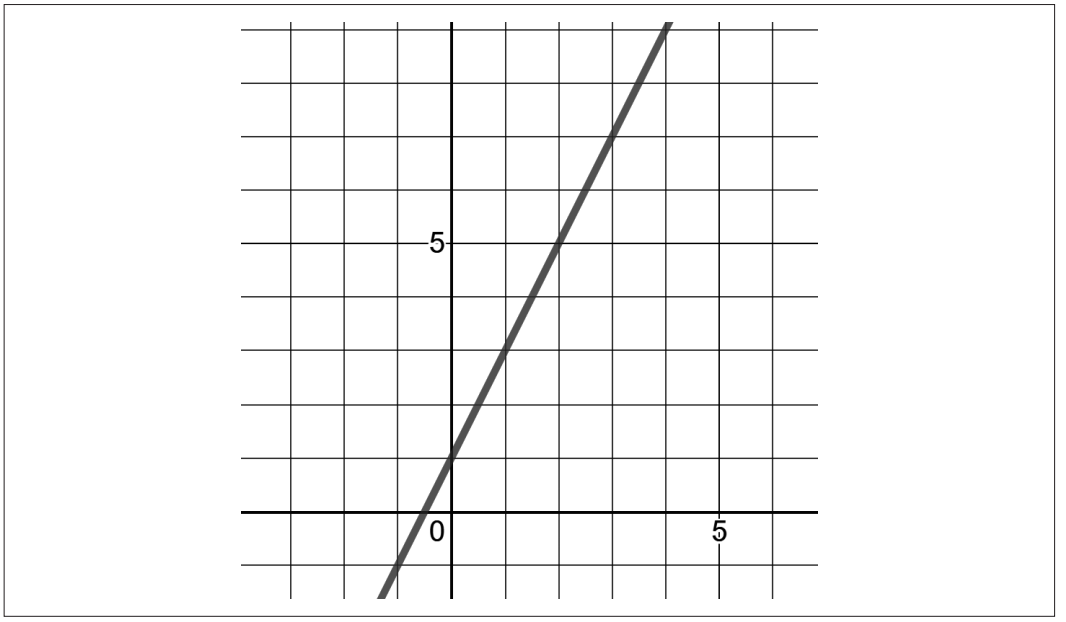

圖 1-1　函數 $y = 2x + 1$ 的圖形

在具有兩條數軸（每個變數一個）的二維平面上繪圖時，即是**笛卡爾平面**（*Cartesian plane*）、**x-y 平面**（*x-y plane*）或**坐標平面**（*coordinate plane*）。追蹤給定的 x 值，然後查找對應的 y 值，並把它們的交叉點繪製為一條線。請注意，由於實數（或小數，如果您喜歡的話）的本質，存在著無限數量的 x 值。這就是為什麼繪製函數 $f(x)$ 時，會得到一條沒有中斷的連續直線。在那條線上，或那條線上的任何部分上，會有無數個點。

如果您想使用 Python 來繪製它，有許多圖表程式庫可以使用，從 Plotly 到 matplotlib 都是。在本書中，我們將使用 SymPy 來完成許多任務，我們首先要使用的是繪製一個函數。SymPy 使用的是 matplotlib，因此請確保您已安裝該軟體套件。否則它會在您的控制台上印出一個基於文字的醜陋圖表。之後，您只需使用 `symbols()` 來將 x 變數宣告為 SymPy、宣告您的函數、然後如範例 1-7 和圖 1-2 所示繪製。

範例 1-7　使用 SymPy 在 Python 中繪製線性函數

```
from sympy import *

x = symbols('x')
f = 2*x + 1
plot(f)
```

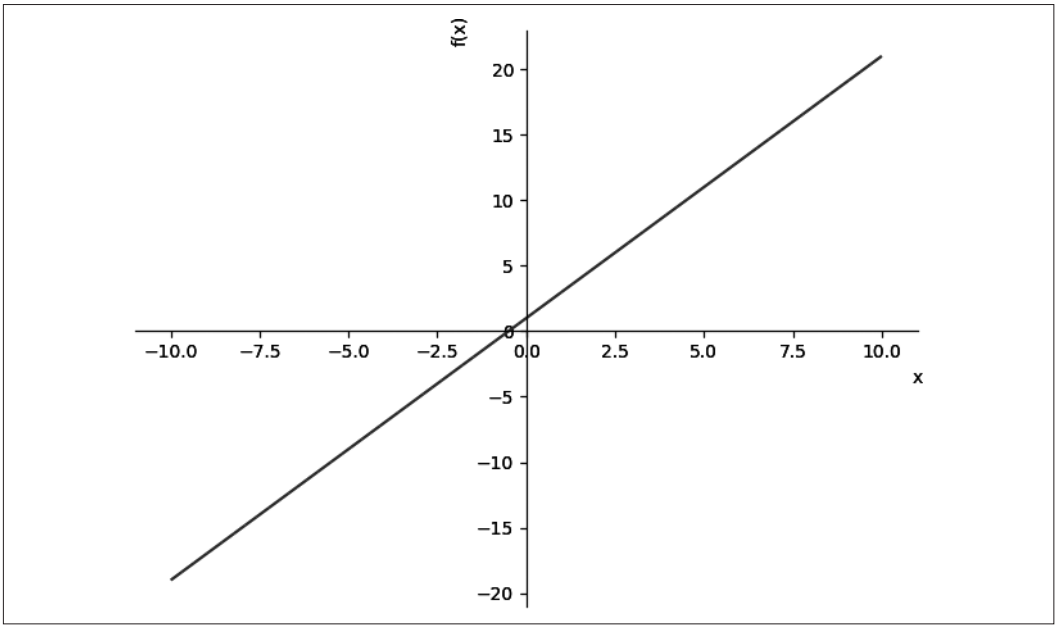

圖 1-2　使用 SymPy 來繪製線性函數

範例 1-8 和圖 1-3 是顯示函數 $f(x) = x^2 + 1$ 的另一個範例。

範例 *1-8*　繪製指數函數

```
from sympy import *
x = symbols('x')
f = x**2 + 1
plot(f)
```

請注意，在圖 1-3 中，我們得到的不是一條直線，而是一條稱為拋物線的平滑對稱曲線。它是連續的但不是線性的，因為它不會產生位於直線上的值。像這樣的曲線函數在數學上更難處理，但我們將學習一些技巧來應付它。

曲線函數

當一個函數是連續但彎曲的，而不是線性和直線時，我們稱它為曲線函數（*curvilinear function*）。

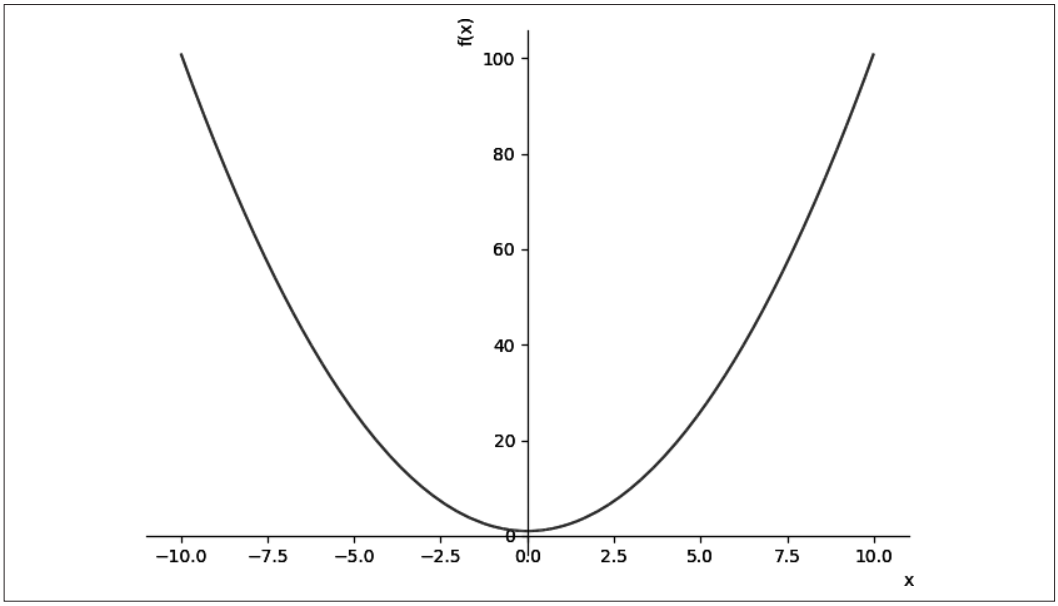

圖 1-3　使用 SymPy 來繪製指數函數

請注意，函數可以使用多個輸入變數，而不僅僅是一個。例如，我們可以有一個具有自變數 x 和 y 的函數。請注意，這裡的 y 不像前面的範例中那樣是一個因變數。

$$f(x, y) = 2x + 3y$$

由於我們有兩個自變數（x 和 y）和一個因變數（$f(x,y)$ 的輸出），所以我們需要在三個維度上繪製此圖，以生成一個值的平面而不是一條線，如範例 1-9 和圖 1-4 所示。

範例 1-9　在 Python 中宣告具有兩個自變數的函數

```
from sympy import *
from sympy.plotting import plot3d

x, y = symbols('x y')
f = 2*x + 3*y
plot3d(f)
```

無論您有多少個自變數，您的函數通常只會輸出一個因變數。當您求解多個因變數時，您可能要為每個因變數使用單獨的函數。

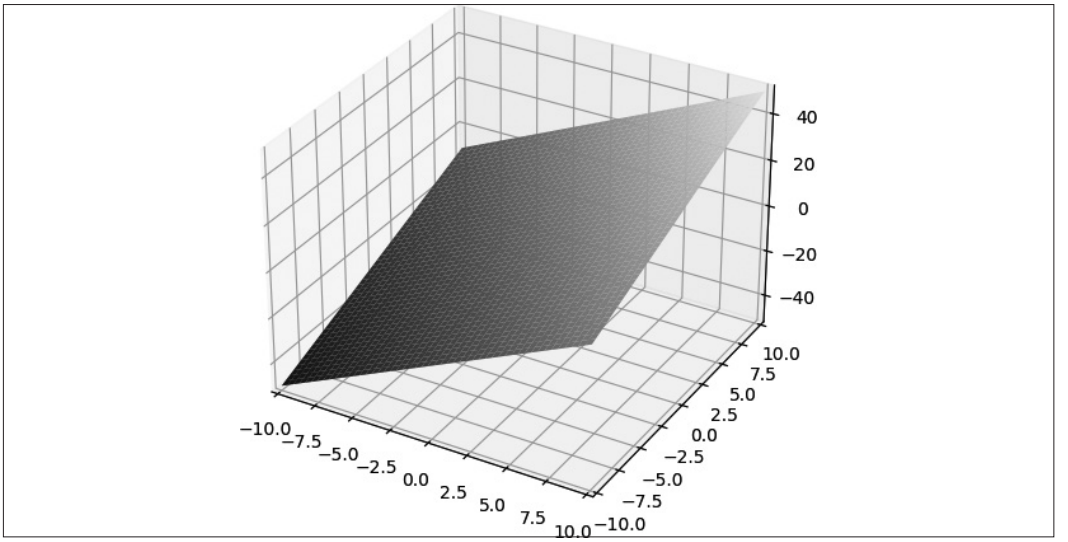

圖 1-4　使用 SymPy 來繪製三維函數

加總

我承諾過不會在本書中使用充滿希臘符號的方程式。然而，其中有一個是如此常見和有用，所以如果不介紹它就是我的怠惰，那就是 Σ（sigma），用來將所有元素相加的**加總**（*summation*）。

例如，我想迭代數字 1 到 5，並將每個數字乘以 2，然後對它們求總和，以下是我利用加總來表達的方式。範例 1-10 則顯示如何在 Python 中執行此運算。

$$\sum_{i=1}^{5} 2i = (2)1 + (2)2 + (2)3 + (2)4 + (2)5 = 30$$

範例 1-10　在 Python 中執行加總

```
summation = sum(2*i for i in range(1,6))
print(summation)
```

請注意，*i* 是一個占位符變數，代表我們在迴圈中迭代的每個連續索引值，把它乘以 2 再加在一起。當您迭代資料時，您可能會看到用 x_i 這樣的變數來指出集合中索引 *i* 處的元素。

range() 函數

回想一下，Python 中的 range() 函數是排除結尾的，這意味著如果您呼叫 range(1,4) 時，它會迭代數字 1、2 和 3。它把 4 當作是上邊界而排除在外。

我們很常使用 *n* 來代表集合中的項目數，例如資料集中的紀錄數。以下是一個這樣的範例，我們迭代一組大小為 *n* 的數字、將每個數字乘以 10 再把它們相加：

$$\sum_{i=1}^{n} 10x_i$$

在範例 1-11 中，我們使用 Python 來對 4 個數字的集合執行此運算式。請注意，在 Python（以及常見的大多數程式語言）中，我們通常會參照從索引 0 開始的項目，而在數學中我們會從索引 1 開始。因此，我們在迭代中會相對應地移動到從 range() 中的 0 開始。

範例 1-11　Python 中的元素加總

```
x = [1, 4, 6, 2]
n = len(x)

summation = sum(10*x[i] for i in range(0,n))
print(summation)
```

這就是加總的要旨。簡而言之，加總 Σ 表示「把一堆東西加在一起」，並使用索引 *i* 和最大值 *n* 來表達輸入到加總的每次迭代。本書將會一直使用這種表達方式。

SymPy 中的加總

當您了解有關 SymPy 的更多資訊時，請隨時返回此邊欄。我們用來繪製函數的 SymPy 實際上是一個符號式數學程式庫；我們將在本章後面討論它的意義味。但請注意並供將來參考，SymPy 中的加總運算是使用 Sum() 運算子來執行。在下面的程式碼中，我們從 1 到 *n* 來迭代 *i*、把每個 i 相乘、然後把它們相加。但隨後我們使用了 subs() 函數來將 *n* 指定為 5，然後對從 1 到 *n* 的所有 *i* 元素進行迭代並求和：

```
from sympy import *

i,n = symbols('i n')

# 對從 1 到 n 的所有 i 元素進行迭代，
# 然後相乘並相加
summation = Sum(2*i,(i,1,n))

# 指明 n 為 5,
# 從數字 1 到 5 進行迭代
up_to_5 = summation.subs(n, 5)
print(up_to_5.doit()) # 30
```

請注意，SymPy 中的加總是惰性的（lazy），這意味著它們不會自動地計算或簡化。所以請使用 `doit()` 函數來執行運算式。

指數

指數（*exponent*）是一個數字和本身相乘指定的次數。當您把 2 提高到三次方（使用 3 作為上標表達為 2^3）時，就是把三個 2 乘在一起：

$$2^3 = 2 * 2 * 2 = 8$$

底數（*base*）是我們要進行指數運算的變數或值，**指數**是我們把底數相乘的次數。對於運算式 2^3 來說，2 是底數，3 是指數。

指數有一些有趣的性質。假設我們要將 x^2 和 x^3 相乘。請觀察當我用簡單的乘法來擴展指數，再把它們合併成一個指數時會發生的事情：

$$x^2x^3 = (x * x) * (x * x * x) = x^{2+3} = x^5$$

當我們把具有相同底數的指數相乘時，我們只需把指數相加即可，這就是所謂的**乘積規則**（*product rule*）。讓我強調一下，所有相乘指數的底數必須相同，才能應用乘積規則。

接下來讓我們來探索除法。當我們把 x^2 除以 x^5 時會發生什麼事呢？

$$\frac{x^2}{x^5}$$

$$\frac{x * x}{x * x * x * x * x}$$

$$\frac{1}{x * x * x}$$

$$\frac{1}{x^3} = x^{-3}$$

如您所見，當我們把 x^2 除以 x^5 時，我們可以消去分子和分母中的兩個 x，剩下 $\frac{1}{x^3}$。當分子和分母中都存在同一個因子時，我們可以消去此因子。

您想知道 x^{-3} 是什麼意思嗎？這是引入負指數的大好時機，它是在分數的分母中表示指數運算的另一種方式。作為示範目的，$\frac{1}{x^3}$ 和 x^{-3} 其實是相同的：

$$\frac{1}{x^3} = x^{-3}$$

回到乘積規則，我們可以看到它也適用於負指數。為了更了解這背後的邏輯，讓我們以不同的方式來解答這個問題。我們可以把 x^5 的指數「5」設為負數，然後把它和 x^2 相乘，來表達兩個指數的除法。當您添加一個負數時，它實際上是在執行減法。因此，相乘指數相加時的指數乘積規則仍然成立，如下所示：

$$\frac{x^2}{x^5} = x^2\frac{1}{x^5} = x^2x^{-5} = x^{2+-5} = x^{-3}$$

最後的關鍵是，您明白為什麼任何底數，一旦指數為 0 時，都會是 1 嗎？

$$x^0 = 1$$

獲得這種直覺的最好方法是推斷任何數字除以本身都是 1。如果您有 $\frac{x^3}{x^3}$，那麼在代數上很明顯它會化簡為 1。但是該運算式也會計算成 x^0：

$$1 = \frac{x^3}{x^3} = x^3x^{-3} = x^{3+-3} = x^0$$

透過遞移（transitive）性 —— 也就是如果 $a = b$ 且 $b = c$ 時，則 $a = c$—— 我們知道 $x^0 = 1$。

使用 SymPy 來化簡運算式

如果您對化簡代數運算式不那麼自在，您可以使用 SymPy 程式庫來完成這項工作。以下是化簡之前範例的方式：

```
from sympy import *

x = symbols('x')
expr = x**2 / x**5
print(expr) # x**(-3)
```

現在分數指數又是如何呢？它們是表示根的另一種方法，例如平方根。簡單複習一下，$\sqrt{4}$ 是在問「什麼數字乘以它本身會得到 4？」這當然是 2。請注意，$4^{1/2}$ 和 $\sqrt{4}$ 是相同的：

$$4^{1/2} = \sqrt{4} = 2$$

立方根類似於平方根，但它們會求一個數乘上本身三次以得出結果。8 的立方根可表達為 $\sqrt[3]{8}$，若是問「什麼數字乘以本身 3 次會得到 8？」答案將會是 2，因為 2 * 2 * 2 = 8。在指數中，立方根會表達為分數指數，而 $\sqrt[3]{8}$ 可以重新表達為 $8^{1/3}$：

$$8^{1/3} = \sqrt[3]{8} = 2$$

再把它拉回來，當您把 8 的立方根乘上三次時會怎樣呢？這將撤消立方根並產生 8。或者，如果我們將立方根表達為分數指數 $8^{1/3}$，很明顯地我們會把指數相加而得到指數 1，這也會撤消立方根：

$$\sqrt[3]{8} * \sqrt[3]{8} * \sqrt[3]{8} = 8^{\frac{1}{3}} \times 8^{\frac{1}{3}} \times 8^{\frac{1}{3}} = 8^{\frac{1}{3}+\frac{1}{3}+\frac{1}{3}} = 8^1 = 8$$

最後一個屬性：指數的指數會把指數相乘，這稱為**冪次規則**（*power rule*）。所以 $\left(8^3\right)^2$ 會簡化為 8^6：

$$\left(8^3\right)^2 = 8^{3 \times 2} = 8^6$$

如果您感到懷疑，請嘗試展開它，您會看到加總規則清楚地表明：

$$\left(8^3\right)^2 = 8^3 8^3 = 8^{3+3} = 8^6$$

最後，當我們有一個分子不是 1 的分數指數時，例如 $8^{\frac{2}{3}}$，這意味著什麼？嗯，這是取 8 的立方根，然後再對其進行平方。看一下這個：

$$8^{\frac{2}{3}} = \left(8^{\frac{1}{3}}\right)^2 = 2^2 = 4$$

還有，無理數可以用來當作指數，例如 8^{π}，也就是 687.2913。這可能感覺不太對勁，但可以理解！由於時間關係，我們不深入研究，因為它需要一些微積分。但本質上，我們可以透過用有理數的逼近來計算無理數指數。這實際上就是電腦會做的事，因為它們無論如何只能計算到那麼多的小數位。

例如說 π 有無限個小數位。但是如果我們取前 11 位數字 3.1415926535，我們可以將 π 近似為有理數 31415926535 / 10000000000。果然，結果大約是 687.2913，它應該和任何計算器（calculator）的計算結果大致匹配：

$$8^{\pi} \approx 8^{\frac{31415926535}{10000000000}} \approx 687.2913$$

對數

對數（*logarithm*）是一種數學函數，可以找到特定數字和底數的冪次（power）。一開始可能聽起來並不有趣，但它實際上可應用在許多方面。從測量地震到管理立體聲音響的音量，對數無處不在。它還大量地進入機器學習和資料科學領域。事實上，對數將會是第 6 章中邏輯迴歸的關鍵部分。

一開始請先自問，「2 提高到**什麼冪次**會得到 8？」以數學方式來表達這一點的其中一種方法，是使用 x 作為指數：

$$2^x = 8$$

我們直觀地知道答案，$x = 3$，但如果要用更優雅的方式，來表達這種常見的數學運算，就可以用 *log*() 功能。

$$log_2 8 = x$$

正如您在前面的對數運算式中看到的那樣，我們有一個底數 2，並且正在尋找可以給我們 8 的冪次。更一般地說法，我們可以把變數指數重新表達為對數：

$$a^x = b$$
$$log_a b = x$$

從代數上講，這是一種隔離 x 的方法，這對於求解 x 很重要。範例 1-12 展示如何在 Python 中計算這個對數。

範例 1-12　在 Python 中使用 log 函數

```
from math import log

# 2 的幕次要是多少才能得到 8?
x = log(8, 2)

print(x) # 印出 3.0
```

如果沒有為 Python 等平台上的 `log()` 函數提供底數參數，它通常也有預設底數。在某些領域，如地震測量，log 的預設底數是 10。但在資料科學中，log 的預設底數是歐拉數 e。Python 使用後者，我們將在之後討論。

就像指數一樣，對數在乘法、除法、取冪（exponentiation）等方面也有幾個屬性。考量到時間和本書重點，我只在表 1-3 中介紹。重點關注的關鍵思想是對數會尋找給定底數的指數，以產生特定數字。

如果您需要深入研究對數的屬性，表 1-3 排列出指數和對數的屬性，您可以以此為參考。

表 1-3　指數和對數的屬性

運算子	指數屬性	對數屬性
乘法	$x^m \times x^n = x^{m+n}$	$log(a \times b) = log(a) + log(b)$
除法	$\frac{x^m}{x^n} = x^{m-n}$	$log\left(\frac{a}{b}\right) = log(a) - log(b)$
取冪	$(x^m)^n = x^{mn}$	$log(a^n) = n \times log(a)$
零指數	$x^0 = 1$	$log(1) = 0$
倒數	$x^{-1} = \frac{1}{x}$	$log(x^{-1}) = log\left(\frac{1}{x}\right) = -log(x)$