(19)中華民國智慧財產局

(12)發明說明書公開本　(11)公開編號：TW 201248424 A1

(43)公開日： 中華民國 101 (2012) 年 12 月 01 日

(21)申請案號：100119021　(22)申請日： 中華民國 100 (2011) 年 05 月 31 日

(51)Int. Cl.：　*G06F17/18　(2006.01)*　*G06F17/15　(2006.01)*

(71)申請人：陳瑞照 (中華民國)　(TW)

新北市新莊區中正路 510 號

羅國弘 (中華民國)　(TW)

桃園縣平鎮市延平路 3 段 440 巷 20 號

孫天龍 (中華民國)　(TW)

桃園縣桃園市同安街 398 號 8 樓

(72)發明人：陳瑞照 (TW) ；羅國弘 (TW) ；孫天龍 (TW)

(74)代理人：葉建郎

申請實體審查：有　申請專利範圍項數：4 項　圖式數：0　共 33 頁

(54)名稱

一種用以提高計算機硬體計算交叉共變異函數（Ｃｒｏｓｓ－Ｃｏｖａｒｉａｎｃｅ Ｆｕｎｃｔｉｏｎ，ＣＣＦ）與自我共變異函數（Ａｕｔｏｃｏｖａｒｉａｎｃｅ Ｆｕｎｃｔｉｏｎ，ＡＣＦ）計算速度的演算方法

(57)摘要

本發明係一種用以提高計算機硬體計算交叉共變異函數(Cross-Covariance Function,CCF)與自我共變異函數(Autocovariance Function,ACF)計算速度的演算方法。「交叉共變異函數(Cross-Covariance Function,CCF)」與「自我共變異函數(Autocovariance Function,ACF)」是時間序列資料分析中最基本也是最重要的計算。根據 McCullough(1999)指出，現有的演算法都有無法同時兼顧數值計算精確與即時更新(updating)的缺點。本發明提出一個以連續差分為基礎的新計算方法，這個計算方法是利用本發明所提出的加權係數值來做計算，就可以不需要計算平均數而直接計算出 ACF 與 CCF。將本發明所提的加權係數做進一步分解後，更進一步的推導出一個可遞迴又可即時更新計算的方法。本發明所提新方法的計算精確度，以 StRD 資料集和 SAS ver9.0 的 PROC ARIMA 做比較，結果顯示，本發明所提演算法的計算精確度會優於 SAS ver9.0 的計算結果。本發明所提演算法除了能有計算精確度高的優點外，亦改善了過去 CCF 和 ACF 演算法需要先確定資料筆數，以及不能即時更新(updating)的缺點。

六、發明說明：

【發明所屬之技術領域】

本發明係關於一種演算方法，尤指一種用以提高計算機硬體計算交叉共變異函數(Cross-Covariance Function, *CCF*)與自我共變異函數(Autocovariance Function, *ACF*)計算速度的演算方法。

本發明有關在統計分析與時間序列的應用方法中，運用計算機來進行之「交叉共變異函數(Cross-Covariance Function)」與「自我共變異函數(Autocovariance Function)」有關的計算機計算程序並可大量運用於資料分析技術方面之計算方別在：

A. 自我共變異函數(Autocovariance Function)與自我相關函數(Autocorrelation Function)的計算。

B. 偏自我相關函數(Partial Autocorrelation Function)的計算。

C. 交叉共變異函數(Cross-Covariance Function)與交叉相關函數(Cross-Correlation Function)的計算。

D. ARIMA 模式中有關自我共變異矩陣(Autocovariance matrix)與自我相關矩陣(Autocorrelation matrix)的計算。

E. 財務分析與工業工程等領域中，與時間序列資料分析(time series analysis)有關的計算。

本發明之發明領域為上述五個方面的計算精確度，以及即時更新(updating)的計算。

【先前技術】

1. 為有效析述本發明相關之先前技術，於此僅分為 5 項敘述，分述如下：

本發明結構為，第 1 項，說明實務的資料可分為橫斷面(cross sectional)與時間序列(time series)二大類。第 2 項說明橫斷面(cross sectional)與時間序列(time series)二類資料與「離均差平方和(sums of squares of deviation from mean)」有關的計算，並定義自我共變異函數(Autocovariance function)。第 3 項說明 k 階交叉共變異函數(Cross-covariance function at lag k)與 k 階交叉相關函數(Cross-correlation function at lag k)。第 4 項說明即時更新(updating)的意涵。第 5 項，說明和自我相關函數(Autocorrelation Function)與交叉相關函數(Cross-correlation Function)有關的計算問題，及其應用。根據 Anderson *et al.* (2006)的觀點，在實際資料分析中，資料的蒐集可分為橫斷面資料(cross sectional data)與時間序列資料(time series data)二大類。橫斷面資料指的是相同(或幾乎相同)的時間點所蒐集的資料，時間序列資料指的是隨著時間的改變而蒐集的資料，分別說明如下：

(i) 橫斷面資料(cross sectional data)

對某一段特定的時間內所蒐集得的資料。將第一筆資料記為 x_1，第二筆資料記為 x_2，以此類推，第 n 筆資料記為 x_n，將 n 筆資料用集合表示為 $\{x_1, x_2, \cdots, x_n\}$。若以向量方式記錄，則表示為

$$\mathbf{x}_{1\times n}^{\mathrm{T}} = [x_1 \quad x_2 \quad \cdots \quad x_n]^{\mathrm{T}}. \tag{1}$$

(ii)時間序列資料(time series data)

對於連續時間點所蒐集記錄的資料，第 1 個時間點記錄的資料記為 x_1，

5

第 2 的時間點記錄的資料記為x_2，第i個時間點記錄的資料記為x_i，第$i+1$個時間點資料記錄為x_{i+1}，第n個時間點資料記錄為x_n，以此類推，以集合方式記為$\{x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_n, \ldots\}$，將前$n$個連續時間點所蒐集的資料，以向量方式記錄，表示為

$$\mathbf{x}_{1\times n}^{\mathrm{T}} = [x_1 \quad x_2 \quad \cdots \quad x_n]^{\mathrm{T}}. \tag{2}$$

對於固定n筆資料，第(1)和(2)式的向量表示方式雖然相同，但資料的屬性是不同的，第(1)式的資料是沒有先後次序關係的，而第(2)式的資料則是隨著蒐集的時間依序記錄而得。

2. 把固定n筆的資料做加總，然後再除以n，就是n筆資料的算術平均數(mean)，以符號記為

$$\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i . \tag{3}$$

(3)式同時適用於(1)與(2)式二種資料型態的計算。

對於不同的資料型態，計算n筆資料之「離均差平方和(sums of squares of deviation from mean)」，是資料分析領域中相當重要的計算。依橫斷面與時間序列二大類資料分別說明如下：

(i) 橫斷面資料(cross sectional data)

對於固定n筆的橫斷面資料，如(1)式。將每一個資料值(即，x_1, x_2, …, x_n)減去算術平均數，$\bar{x}_n$，再取平方，亦即$(x_i - \bar{x}_n)^2$，稱之為「離差平方(square of deviation)」或稱為「離均差平方(square of deviation from mean)」，再加總後，寫成$\sum_{i=1}^{n}(x_i - \bar{x}_n)^2$，稱為「離均差平方和(sums of square

of deviation from mean)」或「校正平方和(Corrected Sums of Squares, *CSS*)」，本發明以 *CSS* 稱之，以符號記為：

$$CSS_n = \sum_{i=1}^{n} (x_i - \bar{x}_n)^2 . \tag{4}$$

其中，$\bar{x}_n$ 如第(3)式。

(ii) 時間序列資料(time series data)

在固定 n 筆的時間序列資料，如(2)式。把第 1 筆資料到第 $(n-k)$ 資料的序列，以向量表示為：

$$[x_1 \quad x_2 \quad \dots \quad x_{n-k}]_{1\times(n-k)}, \tag{5}$$

把第 $(k+1)$ 筆資料到第 n 筆資料的序列，以向量表示為：

$$[x_{k+1} \quad x_{k+2} \quad \dots \quad x_n]_{1\times(n-k)}, \tag{6}$$

再把第(5)與(6)式的每一個元素，分別減去如第(3)式定義之算數平均數,$\bar{x}_n$ 後，以向量分別表示為：

$$[(x_1 - \bar{x}_n) \quad (x_2 - \bar{x}_n) \quad \dots \quad (x_{n-k} - \bar{x}_n)]_{1\times(n-k)}, \tag{7}$$

$$[(x_{k+1} - \bar{x}_n) \quad (x_{k+2} - \bar{x}_n) \quad \dots \quad (x_n - \bar{x}_n)]_{1\times(n-k)}, \tag{8}$$

將第(7)式的第 1 個元素,$(x_1 - \bar{x}_n)$，乘以第(8)式的第 1 個元素,$(x_{k+1} - \bar{x}_n)$，第(7)式的第 2 個元素,$(x_2 - \bar{x}_n)$ 乘以第(8)式的第 2 個元素,$(x_{k+2} - \bar{x}_n)$，依此類推...，一直計算到第 $(n-k)$ 項，再把它們全部加總，再除以 n，稱之為「k 階自我共變異函數(Autocovariance Function at lag k)」，記為 $C_{XX}(k)$，用數學符號表示為：

$$C_{XX}(k)=\frac{1}{n}\sum_{i=1}^{n-k}(x_i-\bar{x}_n)(x_{i+k}-\bar{x}_n) \tag{9}$$

其中，k小於$n/2$，$\bar{x}_n$如第(3)式。(9)式是根據 Box *et al.* (2008)之定義。

3. 將單變數之離均差平方和(sums of square of deviation from mean)的計算推廣到二變數資料的計算，依橫斷面資料與時間序列資料分別說明如下：

(i) 橫斷面資料(cross sectional data)

對於二變數，固定n筆的成對資料，第 1 筆記為(x_1, y_1)，第 2 筆記為(x_2, y_2)，第 i 筆記為(x_i, y_i)，以此類推，第n筆記為(x_n, y_n)。將第 1 筆到第 n 筆的資料,(x_i, y_i)分別減去$\bar{x}_n$與$\bar{y}_n$，相乘後再加總，稱為「校正積和(Corrected Sums of Products, *CSP*)」，以符號記為

$$CSP_n=\sum_{i=1}^{n}(x_i-\bar{x}_n)(y_i-\bar{y}_n). \tag{10}$$

將CSP_n除以$n-1$就是X與Y的共變異數(covariance)，以符號$\mathrm{cov}(X,Y)$表示，記為：

$$\mathrm{cov}(X,Y)=\frac{CSP_n}{n-1}$$

(ii) 時間序列資料(time series data)

對於二變數的時間序列資料，第 1 個變數以符號x表示，第 2 個變數以符號y表示。將第 1 個時間點資料記為(x_1, y_1)，第 2 個時間點資料記為(x_2, y_2)，第i個時間點的資料記為(x_i, y_i)，第n個時間點的資料記為(x_n, y_n)，以此類推，以集合表示為

$\{(x_1,y_1),\ (x_2,y_2),\ \ldots,\ (x_i,y_i),\ (x_{i+1},y_{i+1}),\ \cdots,\ (x_n,y_n),\ \ldots\}$,

對於變數x，從第 1 筆資料開始，記錄到第$(n-k)$個時間點之資料，以向量表示為：

$$[x_1\ \ x_2\ \ \ldots\ \ x_{n-k}\ ]_{1\times(n-k)}, \tag{11}$$

對於變數y的第$(k+1)$個時間點開始，記錄到第n個時間點之資料，以向量表示為：

$$[y_{k+1}\ \ y_{k+2}\ \ \ldots\ \ y_n\ ]_{1\times(n-k)}, \tag{12}$$

把第(11)式的每一筆資料，x_1，x_2，…，x_{n-k}，分別減去算數平均數,$\bar{x}_n$，以向量表示為：

$$[(x_1-\bar{x}_n)\ \ (x_2-\bar{x}_n)\ \ \ldots\ \ (x_{n-k}-\bar{x}_n)]_{1\times(n-k)}, \tag{13}$$

再把第(12)式的每一筆資料，y_{k+1}，y_{k+2}，…，y_n，分別減去算數平均數,$\bar{y}_n$，以向量表示為：

$$[(y_{k+1}-\bar{y}_n)\ \ (y_{k+2}-\bar{y}_n)\ \ \ldots\ \ (y_n-\bar{y}_n)]_{1\times(n-k)}, \tag{14}$$

將第(13)式的第 1 個元素,$(x_1-\bar{x}_n)$乘上第(14)式的第 1 個元素,$(y_{k+1}-\bar{y}_n)$，第(13)式的第 2 個元素,$(x_2-\bar{x}_n)$乘上第(14)式的第 2 個元素,$(y_{k+2}-\bar{y}_n)$，依此類推...，直到第$(n-k)$項，先把它們全部加總後，再除以n，即所謂的「k階交叉共變異函數(Cross-covariance Function at lag k)」，記為$C_{XY}(k)$，以數學符號表示為：

$$C_{XY}(k)=\frac{1}{n}\sum_{i=1}^{n-k}(x_i-\bar{x}_n)(y_{i+k}-\bar{y}_n), \tag{15}$$

其中，k小於$n/2$，$\bar{x}_n$與$\bar{y}_n$之定義如(3)式。(15)式亦是根據 Box *et al.* (2008)之定義。

4. 所謂的「更新(updating)」，引用 Chan *et al.* (1979)的研究所述，為一個計算公式中，加入一個新的資料時，用新加入的資料之計算結果值，去更新加入新的一筆資料之前的計算結果。而 Higham (2004)對更新公式的定義為：

$$\text{New\_value} = \text{old\_value} + \text{small\_correction} \quad (16)$$

滿足(16)式的計算模式，即稱為「更新(updating)」，用 updating 的方式計算，除了可以節省硬體資源外，還是支援動態資料分析所必需。

5. 許多與時間序列資料分析有關的計算，都是以(9)式和(15)式為基礎，例如，自我相關函數(Autocorrelation Function)，交叉相關函數(Cross-correlation Function)，偏自我相關係數(Partial Autocorrelation Function, *PACF*)，自我共變異矩陣(Autocovariance matrix)或自我相關矩陣(Autocorrelation matrix)等，有關的定義與應用，可以參考 Box *et al.* (2008)。McCullough (1998)指出 *PACF* 在資料分析軟體上的計算，都有無法同時兼顧計算精確與效率的缺點。McCullough (1998)也指出 *ACF* 與偏自我相關係數(*PACF*)的計算有直接計算與牛頓(Newton, 1988)法二種，詳細的說明可參考 McCullough (1998)。此外，Keeling and Pavur (2007)針對 EXCEL XP, EXCEL 2003, JMP 5.0, Minitab 14.0, R1.9.1 SAS 9.1 Splus 6.2 SPSS 12.0, Stat 8.1 和 StatCrunch 3.0 等九種常應用的資料分析軟體的計算精確度研究中，此九種軟體在自我相關函數的計算精確度上，都還有改進的空間，詳細的內容可以參閱 Keeling and Pavur (2007)。

【發明內容】

本發明是根據發明人所發展出以「連續差分(success difference)」為基礎，再乘上發明人所提出的權數(weight), w_{ij} 來做 *ACF* 與 *CCF* 計算的方法。*ACF* 與 *CCF* 的計算公式如(9)和(15)式，這二式各別乘上 n 倍，就是本發明所提出的新計算方法。

本發明的目的是透過 n 階矩陣的分解，先發展出稱之為 k 階延遲連續差分加權式 $\frac{1}{n}\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1}$ 此種可遞迴(recursive)，又可即時更新(updating)的方法，再結合更新項的更新計算，而發明出一個可遞迴，又可做即時更新計算 *CCF* 和 *ACF* 的新方法，此舉解決過去 *ACF* 與 *CCF* 的計算時，均先要先計算平均數的缺點。過去所需要事先計算平均數的方法，都是一種 two pass 的演算法，對所有被計算的資料均需要被讀取二次。而本發明的方法為只需讀取一次即可計算的 one pass 演算法，除了節省硬體儲存空間外，同時也提高計算效率。本發明所提出的結果將可用來改善過去演算法無法同時兼顧即時更新(updating)與計算精確的缺點。

本發明再一目的，是以連續差分為基礎，不需要事先計算平均數，而又能遞迴與更新計算 *CCF* 方法。而 *ACF* 為其單變數的特例。在計算 *ACF* 時，只需將前述演算法中的二個變數，改為一個變數即可。

本發明所提出的方法，除了上述可遞迴(recursive)又能做即時更新(updating)外，另外還擁有計算精確度高的優點。改善應用軟體在計算 *ACF* 與 *CCF* 函數有關的計算之不精確問題。

【實施方式】

針對交叉共變異函數(Cross-covariance Function)簡稱 $C_{XY}(k)$ 與自我共變

異函數(Autocovariance Function)簡稱$C_{XX}(k)$，本發明提出以連續差分(success difference)為基礎，結合一種記為w_{ij}的加權係數，而不需要事先計算平均數(mean)就能計算$C_{XY}(k)$與$C_{XX}(k)$計算方法。

(i) 交叉共變異函數(Cross-covariance Function)

對於交叉共變異函數具備兩變數的n筆資料，第一個變數的第 1 筆記為x_1，第 2 筆記為x_2，以此類推，第n筆記為x_n。定義$sdx_1 = x_1$， 則第一個變數的第i個「連續差分」指的是，用第i個資料值,x_i減去第$i-1$個資料值,x_{i-1}，以符號記為$sdx_i = x_i - x_{i-1}$，其中$i = 2,3,\cdots,n$. 第二個變數的第 1 筆記為y_1，第 2 筆記為y_2，以此類推，第n筆記為y_n。定義$sdy_1 = y_1$， 則第二個變數第i個「連續差分(success difference)」指的是，用第i個資料值,y_i減去第$i-1$個資料值,y_{i-1}，用符號記為$sdy_i = y_i - y_{i-1}$，其中$i = 2,3,\cdots,n$。

首先計算$nC_{XY}(k)$，可表示為

$$nC_{XY}(k) = \frac{1}{n}\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1} \quad + \quad A_{XY} \tag{17}$$

其中，

$$\mathbf{W}_{n\times n} = \left[w_{ij}\right]_{n\times n}, \quad w_{ij} = (n+1)(i+j-1) - ij - \frac{n}{2}(i+j+|i-j|), \tag{18}$$
$$i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, n.$$

$$\mathbf{e}^{\mathrm{T}}_{n\times 1} = \left[sdy_1 \quad sdy_2 \quad \ldots \quad sdy_n\right]_{1\times n},$$

$$\mathbf{d}^{\mathrm{T}}_{1\times n(k)} = \left[\underbrace{0 \quad \ldots \quad 0}_{k\text{個}} \quad sdx_1 \quad \ldots \quad sdx_{n-k}\right]_{1\times n}, \tag{19}$$

其中，

$$\mathbf{d}^{\mathrm{T}}_{1\times n(0)} = \left[sdx_1 \quad sdx_2 \quad \ldots \quad sdx_n \right]_{1\times n},$$

$$\mathbf{d}^{\mathrm{T}}_{1\times n(1)} = \left[0 \quad sdx_1 \quad sdx_2 \quad \ldots \quad sdx_{n-1} \right]_{1\times n},$$

$$\mathbf{d}^{\mathrm{T}}_{1\times n(2)} = \left[\underbrace{0 \quad 0}_{2\text{個}} \quad sdx_1 \quad \ldots \quad sdx_{n-2} \right]_{1\times n},$$

$\ldots$

一直到k小於$n/2$。

而A_{XY}可以表示為下列二種方式：

(a) $$A_{XY} = k\bar{x}_n\left(\bar{y}_k - \bar{y}_n\right) \tag{20}$$

(b) $$A_{XY} = k\left(x_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i \right)\left(\left(y_k - \frac{1}{k}\sum_{i=1}^{k}(i-1)sdy_i \right) - \left(y_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdy_i \right)\right) \tag{21}$$

k階交叉共變異函數(Cross-covariance Function at lag k)的計算，就是把第(17)式的計算結果除以n即可。

對於$\frac{1}{n}\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1}$與$A_{XY}$的迭代計算發展過程，分別說明如下：

(i) $\frac{1}{n}\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1}$的迭代計算實作

對於上述$\frac{1}{n}\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1}$計算，在計算方法的實作上，只需考慮$\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1}$項，最後再除以$n$即可。為了更方便說明不同階$\mathbf{W}_{n\times n}$矩陣之元素，故重新定義$n\times n$矩陣$\mathbf{W}_{n\times n}$的第$i$列、第$j$行元素，記為$w_{ij(n)}$，表示為

$$w_{ij(n)} = (n+1)(i+j-1) - ij - \frac{n}{2}(i+j+|i-j|),$$
$$i = 1, 2, \ldots, n., \quad j = 1, 2, \ldots, n.$$

矩陣$\mathbf{W}_{n\times n}$的分解分為下列二個階段：

階段 1

將 $\mathbf{W}_{n\times n}$ 矩陣分割為二個矩陣的相加，記為

$$\mathbf{W}_{n\times n}=\begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2n} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \cdots & w_{nn} \end{bmatrix}_{n\times n} = \begin{bmatrix} \mathbf{W}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times 1} \\ \mathbf{0}_{1\times(n-1)} & 0 \end{bmatrix}_{n\times n} + \mathbf{G}_{n\times n}, \quad (22)$$

其中，

$$\mathbf{W}_{(n-1)\times(n-1)}=\left[w_{ij(n-1)}\right]_{(n-1)\times(n-1)},$$

$$w_{ij(n-1)}=n(i+j-1)-ij-\frac{(n-1)}{2}\left(i+j+|i-j|\right),$$
$$i=1,2,\ldots,n-1,\quad j=1,2,\ldots,n-1.$$

$$\mathbf{0}_{1\times(n-1)}=\begin{bmatrix}0 & 0 & \cdots & 0\end{bmatrix}_{1\times(n-1)},$$

$$\mathbf{0}_{(n-1)\times 1}=\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix}_{(n-1)\times 1}.$$

$$\mathbf{G}_{n\times n}=\left\lfloor g_{ij}\right\rfloor_{n\times n},$$

$$g_{ij}=\frac{1}{2}(i+j-|i-j|-2),$$
$$i=1,2,\ldots,n,\quad j=1,2,\ldots,n.$$

階段 2

將第一階段所分解出來的 $\mathbf{G}_{n\times n}$ 矩陣做分割，記為

$$\mathbf{G}_{n\times n}=\begin{bmatrix}\mathbf{G}_{(n-1)\times(n-1)} & \mathbf{M}_{(n-1)\times 1}\\ \mathbf{M}^{\mathrm{T}}_{1\times(n-1)} & (n-1)\end{bmatrix}_{n\times n}, \quad (23)$$

其中，

$$\mathbf{G}_{(n-1)\times(n-1)}=\left\lfloor g_{ij}\right\rfloor_{(n-1)\times(n-1)}$$

$$g_{ij}=\frac{1}{2}(i+j-|i-j|-2),$$
$$i=1,2,\ldots,n-1,\quad j=1,2,\ldots,n-1.$$

$$\mathbf{M}^{\mathrm{T}}_{1\times(n-1)} = \begin{bmatrix} 0 & 1 & \cdots & n-2 \end{bmatrix}_{1\times(n-1)}.$$

本發明的第一部份所使用的原理就是將矩陣 $\mathbf{W}_{n\times n}$ 分割的二階段，即(22)和(23)式，代入 $\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1}$，而發展出一個迭代計算公式，透過下列二個步驟獲得：

步驟 1 先將(22)式代入 $\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1}$

定義 $DWE_n = \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1}$，可得：

$$\begin{aligned} DWE_n &= \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{e}_{n\times 1} \\ &= \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\left(\begin{bmatrix} \mathbf{W}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times 1} \\ \mathbf{0}_{1\times(n-1)} & 0 \end{bmatrix}_{n\times n} + \mathbf{G}_{n\times n}\right)\mathbf{e}_{n\times 1} \\ &= \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\begin{bmatrix} \mathbf{W}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times 1} \\ \mathbf{0}_{1\times(n-1)} & 0 \end{bmatrix}_{n\times n}\mathbf{e}_{n\times 1} + \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{G}_{n\times n}\mathbf{e}_{n\times 1} \\ &= DWE_{n-1} + \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{G}_{n\times n}\mathbf{e}_{n\times 1} \end{aligned} \tag{24}$$

步驟 2 再把(23)式代入(24)式等號右邊第二項之 $\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{G}_{n\times n}\mathbf{e}_{n\times 1}$

為了符號表示方便，由(19)式，$\mathbf{d}^{\mathrm{T}}_{1\times n(k)} = \begin{bmatrix} 0 & \cdots & 0 & sdx_1 & \cdots & sdx_{n-k} \end{bmatrix}_{1\times n}$，令 $sdx^*_i = 0$, i=1, 2, …, k., $sdx^*_{i+k} = sdx_i$, i=1, 2, …, n–k., ，則，

定義 $DGE_n = \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{G}_{n\times n}\mathbf{e}_{n\times 1}$，可得：

$$DGE_n = \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{G}_{n\times n}\mathbf{e}_{n\times 1}$$

$$= \mathbf{d}^{\mathrm{T}}_{1\times n(k)}\begin{bmatrix} \mathbf{G}_{(n-1)\times(n-1)} & \mathbf{M}_{(n-1)\times 1} \\ \mathbf{M}^{\mathrm{T}}_{1\times(n-1)} & (n-1) \end{bmatrix}_{n\times n}\mathbf{e}_{n\times 1}$$

$$= \left[\begin{array}{ccc:c} sdx^*_1 & \cdots & sdx^*_{n-1} & sdx^*_n \end{array}\right]_{1\times n}\begin{bmatrix} \mathbf{G}_{(n-1)\times(n-1)} & \mathbf{M}_{(n-1)\times 1} \\ \mathbf{M}^{\mathrm{T}}_{1\times(n-1)} & (n-1) \end{bmatrix}_{n\times n}\left[\begin{array}{c} sdy_1 \\ \vdots \\ \hdashline sdy_{n-1} \\ sdy_n \end{array}\right]_{n\times 1}$$

$$= \mathbf{d}^{\mathrm{T}}_{1\times(n-1)(k)}\mathbf{G}_{(n-1)\times(n-1)}\mathbf{e}_{(n-1)\times 1} + \mathbf{d}^{\mathrm{T}}_{1\times(n-1)(k)}\mathbf{M}_{(n-1)\times 1}sdy_n + sdx^*_n\mathbf{M}^{\mathrm{T}}_{1\times(n-1)}\mathbf{e}_{(n-1)\times 1}$$

$$+ (n-1)sdx^*_n sdy_n$$

$$= DGE_{n-1} + \left(\sum_{i=1}^{n-1}(i-1)sdx_i^*\right)sdy_n + \left(\sum_{i=1}^{n-1}(i-1)sdy_i\right)sdx_n^* + (n-1)sdx_n^*sdy_n \quad (25)$$

把(25)式代入(24)式，即可得：

$$DWE_n = DWE_{n-1} + DGE_n$$

$$= DWE_{n-1} + DGE_{n-1} + \left(\sum_{i=1}^{n-1}(i-1)sdx_i^*\right)sdy_n + \left(\sum_{i=1}^{n-1}(i-1)sdy_i\right)sdx_n^*$$

$$+ (n-1)sdx_n^*sdy_n \quad (26)$$

(ii) A_{XY} 的迭代計算實作

對於 A_{XY}，本發明採用連續差分(successive difference)計算的方法，亦即(21)式，$k\left(x_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i\right)\left(\left(y_k - \frac{1}{k}\sum_{i=1}^{k}(i-1)sdy_i\right) - \left(y_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdy_i\right)\right)$，針對此式的迭代計算，本發明以 $\left(x_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i\right)$ 說明之，如下：

定義 $WSUM\_SDX_n = x_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i$，可得

$$WSUM\_SDX_n = WSUM\_SDX_{n-1} + \sum_{i=1}^{n} sdx_i \quad (27)$$

由於 $k\left(x_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i\right)\left(\left(y_k - \frac{1}{k}\sum_{i=1}^{k}(i-1)sdy_i\right) - \left(y_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdy_i\right)\right)$ 中的 $\left(y_k - \frac{1}{k}\sum_{i=1}^{k}(i-1)sdy_i\right)$ 與 $\left(y_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdy_i\right)$ 和 $\left(x_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i\right)$ 的計算方式是相同的，所以不再贅述。而只要根據(27)的結果，即可做調整項，A_{XY} 的迭代計算。

綜合(26)和(27)式結果，即為 $nC_{XY}(k)$ 的迭代計算方法，此方法可改寫為在硬體上的具體步驟為：

步驟 1：定義硬體上執行之變數

先定義計算用變數，所有變數的計算起始值均設為 0，分述如下：

定義 k 值。

定義 n_cxy_k 表 $nC_{XY}(k)$。

定義 dwe 表示(26)式中的 DWE_n。

定義 dge，表示(26)式中的 DGE_n。

定義 $corrected\_term$ 表(21)式。

第一個變數的連續差分值，記為 $sdx[i]$。再定義 sdx，表(26)式中的 sdx_i^*，

第一次執行時 $i=1$，第二次執行時 $i=2$，以此類推。

第二個變數的連續差分值，記為 sdy，亦即(26)式中的 sdy_i。

第一個變數之連續差分的迭代加權總和，記為 $isum\_sdx$，亦即(26)式中的 $\left(\sum_{i=1}^{n-1}(i-1)sdx_i^*\right)$。

第二個變數之連續差分的迭代加權總和，記為 $isum\_sdy$，亦即(26)式中的 $\left(\sum_{i=1}^{n-1}(i-1)sdy_i\right)$。

定義 n_sum_sdx 表 $\left(\sum_{i=1}^{n}(n-i+1)sdx_i\right)$。

定義 k_sum_sdy 表 $\left(\sum_{i=1}^{k}(k-i+1)sdy_i\right)$。

定義 n_sum_sdy 表 $\left(\sum_{i=1}^{n}(n-i+1)sdy_i\right)$。

定義 sum_sdx 表 $\sum_{i=1}^{n}sdx_i$。

定義 sum_sdy 表 $\sum_{i=1}^{n} sdy_i$ 。

步驟 2：

讀取第 i 筆資料，計算二個變數的「連續差分」值，sdx 與 sdy

sdx 加上 sum_sdx ，再存到 sum_sdx

sum_sdx 加 n_sum_sdx ，再存到 n_sum_sdx

sdy 加上 sum_sdy ，再存到 sum_sdy

sum_sdy 加 n_sum_sdy ，再存到 n_sum_sdy

步驟 3：

條件判斷：

(i) 若 $i \le k$ ，

令 $sdx = 0$,

$k\_sum\_sdy = n\_sum\_sdy$

(ii) 若 $i > k$ ，則 $sdx = sdx[i-k]$

步驟 4：

(i) 將 $(i-1)$ 乘以 sdx 再乘上 sdy

(ii) 把 $isum\_sdx$ 乘以 sdy，$isum\_sdy$ 乘以 sdx

把(i)和(ii)的值全部相加，再加上 dge，再存入 dge 中

步驟 5：

把步驟 4 的 dge 加上 wde 後，存入 wde 中

步驟 6：

做 $isum\_sdx$ 與的 $isum\_sdy$ 遞迴計算

(i) 用 $(i-1)$ 乘上 *sdx*，加上 *isum_sdx* 後，再存入 *isum_sdx*

(ii) 用 $(i-1)$ 乘上 *sdy*，加上 *isum_sdy* 後，再存入 *isum_sdy*

步驟 7：

(i) 用 *k _ sum _ sdy* 除以 *k*

(ii) 用 *n _ sum _ sdy* 除以 *n*

(iii) 用(i)的結果，減掉(ii)的結果乘上 *n _ sum _ sdx*，再乘上 $\frac{k}{n}$，存入 *corrected_term*

步驟 8：

把 *wde* 除以 *i*，再加上 *corrected_term*，再存入 *n _ cxy _ k* 。

當加入第 $i+1$ 筆資料時，回到步驟 2。

上述步驟可以寫成下列在硬體上執行的演算法：

n_cxy_k:=0
dwe:=0
dge:=0
corrected_term:=0
isum_sdx:=0
isum_sdy:=0
n_ sum_sdx:=0
k_ sum_sdx:=0
n_ sum_sdy:=0
sum_sdx:=0
sum_sdy:=0
for *i* := 1 *i*+1

input X_i, Y_i

$sdx[i] := X_i - X_{i-1}$

$sdy := Y_i - Y_{i-1}$

sum_sdx := *sum_sdx* + *sdx*[*i*]
sum_sdy := *sum_sdy* + *sdy*
n_ sum_sdx := *n_ sum_sdx* + *sum_sdx*
n_ sum_sdy := *n_ sum_sdy* + *sum_sdy*

if $i \le k$ **then**
$sdx := 0$,
$k\_sum\_sdy := n\_sum\_sdy$
else $sdx := sdx[i-k]$
$dge := dge + isum\_sdx*sdy + isum\_sdy*sdx + (i-1)*sdx*sdy$
$dwe := dwe + dge$
$isum\_sdx := isum\_sdx + (i-1)*sdx$
$isum\_sdy := isum\_sdy + (i-1)*sdy$
$n\_cxy\_k := dwe/i + (k/n)*(n\_sum\_sdx)*(k\_sum\_sdy/k - n\_sum\_sdy/n)$

(ii) 自我共變異函數(Autocovariance Function)

若將(17)式改為單變數，即為 k 階自我共變異函數(Autocovariance Function at lag k)，記為：

$$nC_{XX}(k) = \frac{1}{n}\mathbf{d}^{\mathrm{T}}_{1\times n(k)}\mathbf{W}_{n\times n}\mathbf{d}_{n\times 1} + A_{XX} \tag{28}$$

其中，$\mathbf{W}_{n\times n}$ 的係數如(18)式，$\mathbf{d}^{\mathrm{T}}_{1\times n(k)}$ 如(19)式，

$$\mathrm{d}^{\mathrm{T}}_{1\times n} = \mathrm{d}^{\mathrm{T}}_{1\times n(0)} = \begin{bmatrix} sdx_1 & sdx_2 & \cdots & sdx_n \end{bmatrix}_{1\times n} \tag{29}$$

而 A_{XX} 可以表示為下列二種方式：

(a) $A_{XX} = k\bar{x}_n(\bar{x}_k - \bar{x}_n)$ (30)

(b) $A_{XX} = k\left(x_n - \frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i\right)\left((x_k - x_n) - \frac{1}{k}\sum_{i=1}^{k}(i-1)sdx_i + \frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i\right)$ (31)

k 階自我共變異函數(Autocovariance Function at lag k)的計算，就是把第(28)式的計算結果除以 n 即可。

交叉共變異數(Cross-covariance function)除了第(17)式以連續差分

(success difference) 為基礎的計算方法外，還有三種計算方法，分別說明如下：

第一種

透過 *CSP* 的計算來計算，表示為：

$$nC_{XY}(k)=CSP_n-\left[(x_{k+1}-x_1)\ \ (x_{k+2}-x_2)\ \ \cdots\ \ (x_n-x_{n-k})\right]_{1\times(n-k)}\begin{bmatrix}(y_{k+1}-\bar{y}_n)\\(y_{k+2}-\bar{y}_n)\\\vdots\\(y_n-\bar{y}_n)\end{bmatrix}_{(n-k)\times 1}$$

$$-\left[(x_1-\bar{x}_n)\ \ \cdots\ \ (x_k-\bar{x}_n)\right]_{1\times k}\begin{bmatrix}(y_1-\bar{y}_n)\\\vdots\\(y_k-\bar{y}_n)\end{bmatrix}_{k\times 1} \tag{32}$$

(32)式是利用 *CSP* 和 *k* 階差分來做計算。

第二種

用 $(n-1)$ 個元素的連續差分向量計算，表示為

$$nC_{XY}(k)=\left[(x_1-\bar{x}_n)\ \ (x_2-\bar{x}_n)\ \ \cdots\ \ (x_{n-k}-\bar{x}_n)\right]_{1\times(n-k)}\begin{bmatrix}(y_1-\bar{y}_n)\\(y_2-\bar{y}_n)\\\vdots\\(y_{n-k}-\bar{y}_n)\end{bmatrix}_{(n-k)\times 1}$$

$$+\left[(x_1-\bar{x}_n)\ \ (x_2-\bar{x}_n)\ \ \cdots\ \ (x_{n-k}-\bar{x}_n)\right]_{1\times(n-k)}\begin{bmatrix}(y_{k+1}-y_1)\\(y_{k+2}-y_2)\\\vdots\\(y_n-y_{n-k})\end{bmatrix}_{(n-k)\times 1} \tag{33}$$

第三種

把(17)式的 *k* 階連續差分向量，改用 *h* 階差分與 $(h-1)$ 階差分的相減來表

示的方向。就是將連續差分寫成

$$sdx_i = (x_i - x_h) - (x_{i-2} - x_h),\ \ h = 1, 2, ..., n.$$

亦即把(19)式改寫為：

$$\mathbf{d}^{\mathrm{T}}_{1\times n(k)} = [\underbrace{0 \ \ldots \ 0}_{k\text{個}} \ \ sdx_1 \ \ (x_2 - x_h) - (x_1 - x_h) \ \ \ldots \ \ (x_n - x_h) - (x_{n-1} - x_h)]_{1\times n},$$

同理，$\mathbf{e}^{\mathrm{T}}_{1\times n}$的第 i 個元素亦可寫成$sdy_i = (y_i - y_h) - (y_{i-2} - y_h),\ \ h = 1, 2, ..., n.$，亦即$\mathbf{e}^{\mathrm{T}}_{1\times n}$可以表示為：

$$\mathbf{e}^{\mathrm{T}}_{1\times n} = [sdy_1 \ \ (y_2 - y_h) - (y_1 - y_h) \ \ \ldots \ \ (y_n - y_h) - (y_{n-1} - y_h)]_{1\times n}.$$

以上三種方法，若改為單變數，即用於自我共變異函數(Autocovariance function)之計算。

本發明所發展出來方法，是以連續差分(success difference)為基礎，不需要事先計算平均數，而又能遞迴與更新計算 *CCF* 方法。而 *ACF* 為其單變數的特例。在計算 *ACF* 時，只需將前述演算法中的二個變數，改為一個變數即可。

另外，本發明在計算精確度的實證上，則是以美國國家標準局(National Institutes of Standards and Technology, NIST)所提供的統計參考資料集(Statistical Reference Datasets, StRD)中的單變量摘要統計量資料集的標準資料驗證值做為實證資料。有關 StRD 的各種詳細的說明，可以參考網址：http://www.itl.nist.gov/div898/strd/ (2011/01/08)。

在計算精確度的衡量方面，根據 Altman *et al.* (2004)指出，所謂的計算

精確，指的正確值與計算結果的非相似(dissimilary)程度，以正確值和計算結果的距離(distance)來表示。對於非相似結果，Altman *et al.* (2004)採用是以 10 為底的對數相對誤差(log relative error, *LRE*)來衡量。對數相對誤差是指正確值減計算結果，再除以正確值之後，再取絕對值，然後再取以 10 為底的對數，*LRE* 值的大小是用來表示 10 進位資料計算精確的有效位數長度。。Altman *et al.* (2004)所定義的 *LRE* 為：

$$LRE=\begin{cases}-\log_{10}\left|\dfrac{exact-output}{exact}\right|, & \text{if } exact\neq 0\\ -\log_{10}|output-exact|, & \text{if } exact=0\end{cases} \tag{34}$$

其中，*exact* 表示正確值，*output* 表示計算的結果。

當計算結果等於正確值時，*LRE* 以「正確(exact)」標示。過去有許多學者用 *LRE* 值來衡量資料分析軟體計算結果的精確度，例如，Wampler (1980)在做最小平方演算法的精確度之衡量, Simon (1985)用來評估 MINITAB 的線性迴歸程序(linear regression procedure)的計算精確度, McCullough (1998, 1999)用來評估 SAS, SPSS, S-PLUS 等軟體的計算精確度， McCullough and Wilson (1999, 2002, 2005)用來評估 EXCEL 的計算精確度,與 Altman *et al.* (2007)在討論 R 與 S-PLUS 二種軟體的精確度，Keeling and Pavur (2007)用來評估九種軟體的計算精確度等等。本發明亦採用 *LRE* 值做計算精確度的實驗結果報告。

實證比較資料：

本發明利用矩陣分割的技巧發展出在硬體上執行，可遞迴又可即時更新計算自我共變異函數(Autocovariance Function)與交叉共變異函數(Cross-covariance function)的新演算法，除了解決了現有演算法無法做即時

更新計算的缺點外，還具備計算精確度高的優點。在實證比較上，本發明所採用的比較資料集為美國國家標準局(National Institutes of Standards and Technology, NIST)提供的統計參考資料集(Statistical Reference Datasets, StRD) (http://www.itl.nist.gov/div898/strd/ (2011/01/08))中的單變量摘要統計量資料集(univariate summary statistics)。

StRD 資料

StRD 是 NIST 的統計工程、數學與計算科學部門資訊技術部份(The Statistical Engineering and Mathematical and Computational Sciences Divisions)的資料館中的一個標準資料集(benchmark datasets)，是提供用來診斷下列五種統計計算的精確度：(a)單變量摘要統計量(univariate summary statistics)；(b)變異數分析(Analysis of Variance, ANOVA)；(c)線性迴歸(linear regression)；(d)蒙地卡羅馬可夫鏈(Markov chain Monte Carlo)；與(e)非線性迴歸(nonlinear regression)。StRD 驗證資料均可於下列網站下載：http://www.itl.nist.gov/div898/strd/ (2011/01/08)。對於本發明所引用的單變量摘要統計量資料集說明如下：

單變量摘要統計量資料集是由九個資料集所組成。每個資料集均提供了正確的 15 位有效位數之平均數、標準差和一階自我相關係數(lag–1 autocorrelation coefficient)供驗證(certification)計算精確度之用。由於本發明的主要目的是在於做 *CCF* 與 *ACF* 的計算精確度，故本將這九個資料集的一階自我相關係數(lag–1 autocorrelation coefficient)轉換為 *ACF*，以 15 位有效位數之驗證值整理如表 1。

表 1. StRD 之單變量資料集之驗證值(certified value)

資料集名稱	資料筆數	自我共變異函數(*ACF*)之驗證值
PiDidits	5000	–0.0291891222080000
Lottery	218	–10244.1567496751
Lew	200	–23517.5956461250
Mavro	50	0.0000001692732800000 00
Michelso	100	0.003307662400000
NumAcc1	3	–0.333333333333333
NumAcc2	1001	–0.009980019801980
NumAcc3	1001	–0.009980019801980
NumAcc4	1001	–0.009980019801980

資料來源：本發明整理

計算精確度的比較

本發明在 Windows XP 作業系統計的 Visual Studio.NET2003 平台下，以 C++程式語言在硬體資源為 Intel Pentium M processor 1.73GHz，2G 的 RAM 下的執行，本發明執行計算的程式如附錄一。

對於計算精確度的比較上，本發明根據 Altman *et al.* (2004)建議的 *LRE* 值做比較，如(34)式。*LRE* 值的大小用來表示計算結果精確的有效位數長度。本發明以附錄一的程式計算後，把計算的結果轉換為 *LRE* 值，並以 10 進位的四捨五入取小數點後一位。若計算輸出的 15 位有效位數與 StRD 所提供的驗證值完全相同，亦即計算結果有 15 位有效位數的精確，則以「正確(Exact)」標示。在計算精確度的比較上，本發明和目前常見的資料分析軟體 SAS ver9.0 的 PROC ARIMA 做比較，結果整理如表 2。

表2. 單變量摘要統計量資料集和SAS ver9.0的*ACF*之*LRE*值比較表

資料集名稱	SAS	本發明	本發明 (round)
PiDidits	Exact	Exact	Exact
Lottery	Exact	Exact	Exact
Lew	Exact	Exact	Exact
Mavro	12.8	12.8	14.2
Michelso	13.2	13.6	13.6
NumAcc1	不提供	Exact	Exact
NumAcc2	7.7	7.7	7.7
NumAcc3	7.7	7.7	7.7
NumAcc4	7.5	7.5	7.7

資料來源：本發明整理

除了計算方法的提出外，本發明所提出以連續差分基礎的方法中，還可以用軟體所提供的四捨五入(round)函數來提高計算精確度，其方法是在計算連續差分的每一次計算，可於連續差分之數值中，自發生估計值位數開始，採取四捨五入之手段，以減少計算平均數與交叉共變異函數的計算誤差。本發明以 C++之實例為

$$sdx = \text{Math} :: \text{Round}(x - x\_1, 4).$$

在連續差分(successive difference)的步驟中使用四捨五入(round)函數，還可以提高最後計算結果的精確度。表 2 的第 4 欄即為在計算連續差分(successive difference)使用四捨五入(round)函數的計算結果。

從表 2 的第二欄和第三欄可知，本發明所提出的計算方法在之 *LRE* 值在 Michelso 資料集的計算是優於 SAS ver9.0，而 SAS ver9.0 在計算 *ACF* 至少需要 6 筆資料以上才能執行，所以對於無法做 NumAcc1 資料集的計算，本發明的計算方法非但沒有這個限制，而且還能得到正確的

結果。此外，在計算連續差分時使用四捨五入(round)函數的話，則還提高了 Mavro 和 NumAcc4 二個資料集的 *LRE* 值，而高於 SAS ver9.0 計算的結果。

附錄一　本發明所用之程式

```
#include "stdafx.h"
#using <mscorlib.dll>
using namespace System;
#include <stdio.h>
#include <float.h>
#include <Math.h>
#define n 1000
#define k 1
/*****************************************************************************/
/**************        這個程式在做自我相關函數的計算      *********************/
/**************        程式撰寫日期：2011/05/31            *********************/
/**************        作者：陳瑞照、孫天龍、羅國弘        *********************/
/*****************************************************************************/

int
main(void)
 {

      FILE  *in_file;
      FILE *out_file;

      double          sdx = 0.0,     sum_sdx = 0.0,    nsum_sdx = 0.0,          dwd = 0.0,          dgd
= 0.0,  css = 0.0;
     double      n_cxx_k = 0.0,       dwd_k = 0.0,       dgd_k = 0.0,    n_w_sum_sdx = 0.0,  k_w_sum_sdy =
0.0,
                 lagk_sdx = 0.0,    isum_sdx = 0.0,  autocov_k = 0.0,     autocorr_k = 0.0, nsum_sdy = 0.0,
n_w_sum_sdy=0.0;
      double x, x_1=0.0, lag[k],a=0.0, b=0.0;
      int i, index;
/*.....................................................................................
.*/
          in_file = fopen ("numacc4.txt","r");
           out_file = fopen ("out_auto_numacc4_NR.txt","w");
           for (i=0; i<= n; i++)
            {
```

```
                fscanf(in_file, "%lf", &x);                    /*  輸入資料                */
                    index = i%k;                               /*  計算指標i 和k值的整數餘數*/
                      sdx = x - x_l;                           /*  計算輸入的連續差分值    */
                  nsum_sdx = nsum_sdx + sdx;                   /*  計算n筆資料的加權連續差分總和  */
                n_w_sum_sdx = n_w_sum_sdx + nsum_sdx;
                     if(i>0)
                         {
                      nsum_sdy = nsum_sdy + sdx;               /*  計算n筆資料的加權連續差分總和  */
                        n_w_sum_sdy= n_w_sum_sdy + nsum_sdy;   /*  計算到第n筆資料的平均數*/
                         }
                     if (i < k )
                     {
                           lagk_sdx = 0;                       /*  讓k階前的連續差分值爲0          */
                             k_w_sum_sdy = n_w_sum_sdy;
                      }
                     else
                        {
                             lagk_sdx = lag[index];            /* 定義lag k的數值，                */
                          }
                    lag[index] = sdx;                          /* 把連續差分放在暫存的陣列中      */

/*****************************************************************************/
/*****          接下來的二行，在做lagk 的 dwd_k 計算                    *******/

                        dgd_k = dgd_k + isum_sdx*sdx + sum_sdx*lagk_sdx + i*sdx*lagk_sdx;
                    dwd_k = dwd_k + dgd_k;
/*****************************************************************************/
/*****          下面二行在做n倍CSS的迭代計算                    ******/

                           dgd = dgd +  2*sum_sdx*sdx + i*sdx*sdx;
                        dwd = dwd + dgd;

/*****************************************************************************/
/*****          以下二行，分別做dgd_l 與dgd的迭代變數計算用的          ******/

                        isum_sdx = isum_sdx + i*lagk_sdx;
                    sum_sdx = sum_sdx  + i*sdx;
/*****************************************************************************/
/****************    計算n倍的k階自我共變異函數，亦即計算n_cxx_k      **********/
                      a = dwd_k/(i+1);
                        b = k*(k_w_sum_sdy/k - n_w_sum_sdy/(i+1))*n_w_sum_sdx/(i+1);
                        n_cxx_k = dwd_k/(i+1) + k*(k_w_sum_sdy/k -
n_w_sum_sdy/(i+1))*n_w_sum_sdx/(i+1);
                 autocov_k = n_cxx_k/(i+1);
```

```
                        css = dwd/(i+1);          /** 計算自我共變異函數      ***/
fprintf (out_file, "i=%d  dwd_k=%.15le  n_cxx_k=%.15e css=%.15e a=%.15e  b=%.15e   autocov_k=%.15e
\n",i, dwd_k, n_cxx_k,css, a, b, autocov_k);

/*************************************************************************/
/**************      計算k階自我相關係數                                  *****/
              autocorr_k =autocov_k / css;

/*************************************************************************/
          x_l=x;                          /** 計算連續差分的迭代變數  **/
             }

         system("PAUSE");
}
```

【圖式簡單說明】

【主要元件符號說明】

201248424

發明專利說明書

（本說明書格式、順序，請勿任意更動，※記號部分請勿填寫）

※申請案號：100119021

※申請日：100.5.31　※IPC 分類：G06F 17/18 2006.01
G06F 17/15 2006.01

一、發明名稱：（中文/英文）

一種用以提高計算機硬體計算交叉共變異函數(Cross-Covariance Function, *CCF*)與自我共變異函數(Autocovariance Function, *ACF*)計算速度的演算方法

二、中文發明摘要：

本發明係一種用以提高計算機硬體計算交叉共變異函數(Cross-Covariance Function, *CCF*)與自我共變異函數(Autocovariance Function, *ACF*)計算速度的演算方法。「交叉共變異函數(Cross-Covariance Function, *CCF*)」與「自我共變異函數(Autocovariance Function, *ACF*)」是時間序列資料分析中最基本也是最重要的計算。根據McCullough(1999)指出，現有的演算法都有無法同時兼顧數值計算精確與即時更新(updating)的缺點。本發明提出一個以連續差分為基礎的新計算方法，這個計算方法是利用本發明所提出的加權係數值來做計算，就可以不需要計算平均數而直接計算出 *ACF* 與 *CCF*。將本發明所提的加權係數做進一步分解後，更進一步的推導出一個可遞迴又可即時更新計算的方法。本發明所提新方法的計算精確度，以 StRD 資料集和 SAS ver9.0 的 PROC ARIMA 做比較，結果顯示，本發明所提演算法的計算精確度會優於 SAS ver9.0 的計算結果。本發明所提演算法除了能有計算精確度高的優點外，亦改善了過去 *CCF* 和 *ACF* 演算法需要先確定資料筆數，以及不能即時更新(updating)的缺點。

三、英文發明摘要：

七、申請專利範圍：

1. 一種用以提高計算機硬體計算交叉共變異函數(Cross-Covariance Function, *CCF*)與自我共變異函數(Autocovariance Function, *ACF*)計算速度的演算方法，其步驟為（1）先算出 k 階延遲連續差分加權式的數值，其計算方法為：以二個變數之連續差分(successive differences)值以及所對應的一加權係數(weighted coefficients)三者的乘積，從第一項數值起開始逐次計算，並將計算結果累加，直到資料輸入結束為止，上述該加權係數為：$(n+1)(i+j-1)-ij-\frac{n}{2}(i+j+|i-j|)$，其中 i,j 為項次(index)，n 為資料總數目，

（2）計算調整項，用前 k 筆的平均數減去 n 筆資料的平均數之差額再乘以 k 筆數所得之積再乘以 n 筆資料的平均數，

（3）將 k 階延遲連續差分加權式的數值與平均數離差調整數相加之和，再除以資料總筆數，即可得交叉共變異函數(Cross-Covariance function)。

2. 如專利申請項第 1 項所述之一種用以提高計算機硬體計算交叉共變異函數(Cross-Covariance Function, *CCF*)與自我共變異函數(Autocovariance Function, *ACF*)計算速度的演算方法，其中計算增加一筆資料之第 n+1 筆 k 階延遲連續差分加權式之值，其值可先利用二個變數的連續差分(successive differences)值以及所對應的 g_{ij} 加權係數三者的乘積，即 $g_{ij}sdx_i^*sdy_j$，從第一項數值起開始逐次計算到第 n+1 筆資料，並將計算結果累加，而得出之數，即 $\sum_{i=1}^{n+1}\sum_{j=1}^{n+1}g_{ij}sdx_isdy_j$，其後加上具 n 筆資料之 k 階延

遲連續差分加權式的 n 倍值，即可得 $(n+1)$ 倍的具 $n+1$ 筆資料之 k 階延遲連續差分加權式之值，其中，該 g_{ij} 加權係數計算法為：

$g_{ij}=\frac{1}{2}(i+j-|i-j|-2)$ ，i,j 為項次(index)，n 為資料輸入筆數。

3. 如專利申請項第 1 項所述之一種用以提高計算機硬體計算交叉共變異函數(Cross-Covariance Function, *CCF*)與自我共變異函數(Autocovariance Function, *ACF*)計算速度的演算方法，其中該計算調整項之步驟，可以用連續差分值與加權係數(weighted coefficients) $l_i=(i-1)$ 相乘，再利用第 n 個資料去做相減的即時更新計算取代之，計算步驟記為：

$\bar{x}_n=x_n-\frac{1}{n}\sum_{i=1}^{n}(i-1)sdx_i$ 。

4. 如專利申請項第 1 項所述之一種用以提高計算機硬體計算交叉共變異函數(Cross-Covariance Function, *CCF*)與自我共變異函數(Autocovariance Function, *ACF*)計算速度的演算方法，其中該連續差分的每一次計算，可於連續差分之數值中自發生估計值位數開始採取四捨五入之手段，以減少計算平均數與交叉共變異函數的計算誤差。

八、圖式：無。

四、指定代表圖：

(一)本案指定代表圖為：無。

(二)本代表圖之元件符號簡單說明：無。

五、本案若有化學式時，請揭示最能顯示發明特徵的化學式：