中華民國第 55 屆中小學科學展覽會

作品說明書

國中組　數學科

030427

連弟弟都看得懂的白努利級數及其應用

學校名稱：新竹縣立竹東國民中學

作者：	指導老師：
國一 劉樂山	梁馨文

關鍵詞：白努利數、賈憲三角形

摘要

本研究肇始於作者五年級弟弟因練習奧數求連續整數三次和而發問，引發作者對四次五次和六次求和的興趣，以簡單方式找出解法，並寫程式執行並分析比較，根據運算結果，數度來回優化算式，再依據新式子修改程式，最後終於找到方法，以降冪加上 S3(三次和)轉換求 S6，速度甚至快於未優化的 S4。

此外應用二項式定理加上 Telescope Sum 以程式找出四次方到一百次方求和式子的所有係數進而求出其冪次和。最後更發展遞迴程式，又快又精確沒有誤差地找出 n 次和。

本研究還以極為簡單易懂的方法，找到(證明)白努利數及其通式！因發現機械運算太多，故決定以程式證明通式，還因此意外發現白努利數b_k通式!最後用自己早期的式子取代 Telescope Sum，並寫出另外三種遞迴求極數程式！

壹、研究動機

有一次弟弟練習數學奧林匹亞的題目，其中一題要算 $1^3+2^3+\cdots+50^3$之值，因不會所以問我。花了一些時間教會了他，突然一個念頭掃過腦海，"如果是四方怎麼辦?"，於是我馬上動手算!約花了一下午的嘗試，導出式子拿去給爸爸看，爸爸的態度是剛開始說"錯"，然後說"不對"，最後經我努力解釋，他才露出喜悅的笑容，很高興能靠我自己導出此式子，且鼓勵我繼續研究，並利用寒暑假時間，引導我學習 CProgrammingLanguage。

關於白努利級數，網路上雖有不少文獻，然而卻遠遠超出我和弟弟的當時的理解範圍! 透過自己導式子，不但弟弟能懂，還能讓我們對數列、級數有進一步理解。透過 C 程式的撰寫，除了讓我深深體會到降冪對於加速電腦運算的重要，更大大讓我驚訝到數學竟然可以應用於電腦，除了數學之美、程式之美也同樣令人著迷! 第一次看到自己的程式按照自己的意思跑出結果來，是迷人且使人振奮的!現在全世界都在積極鼓勵青少年學習 Coding，很高興我並不缺席!

除此之外，國內外電腦開放課程風氣日盛，坐在家裡透過網路，隨時可以在線上找到適當的課程，免費學習!OCW 的課，我也看了很多，但要以此找到白努利級數和的通解，實在是非常困難! 所以並不預期能找到通解(通式)，然而，透過式子的演算和程式的驗證，期待能找到連弟弟也懂的有效解決方案。

貳、研究目的

本研究以三次級數和為已知，希望找出四次級數和、五次級數和及六次級數和，並以 C 程式驗證其結果並比較速度，從而找出相對最佳解決方法。

參、研究設備與器材

一、個人電腦(配有 UBuntu 12.4 LTS,Win7 作業系統)

二、紙筆

三、網路

四、維基百科

五、GNU C4.9.1 , Visual C++6.0

六、Eclipse Kepler:2.0.0 Build Id 20130613-229

肆、研究過程及方法

一、列出指數為 3 的級數公式

S3= $1^3 + 2^3 + \cdots + n^3$

= $(n * (n + 1)/2)^2$

可以數學歸納法證明，詳細過程請參考均一教育平台，數列與級數單元。

(以 Telescope Sum 方法或 MyRecursive1,MyRecursive2,MyRecursive3 方法亦可)

二、四次級數求和

$$S_4 = 1^4 + 2^4 + 3^4 + \cdots\cdots + n^4 = 1\times 1^3 + 2\times 2^3 + 3\times 3^3 + \cdots\cdots + n\times n^3$$
$$= 1^3 + 2^3 + 3^3 + \cdots\cdots + n^3 \rightarrow S_3$$
$$+ 2^3 + 3^3 + \cdots\cdots + n^3 \rightarrow S_3 - 1^3$$
$$+ 3^3 + \cdots\cdots + n^3 \rightarrow S_3 - 1^3 - 2^3$$
$$+ \cdots\cdots\cdots\cdots$$
$$+ n^3 \rightarrow S_3 - 1^3 - 2^3 - \cdots\cdots - (n-1)^3$$
$$= nS_3 - \sum_{i=1}^{n-1}\sum_{j=1}^{i} j^3 \qquad \text{(式　肆.二.1)}$$

這是本研究最重要的式子,後面甚至可據以推導出通式

三、撰寫四次級數求和 C 程式，第一步

根據(式　肆.二.1) (式　肆.二.2) 撰寫 C 程式。

因為要比較速度，所以 n 要選大一點，幸好 GNU C 提供 unsigned __int128 資料型態，但後來卻發現連 128 bit 整數都不夠用，為了簡化程式，也為了不讓問題失焦，採取如下變通辦法，既可達到目的，又可讓作者這種 C 程式初學者不必為了提供 256 bit 整數運算而焦頭爛額。

(一)unsigned __int128

也就是用 128Bits(16Bytes)整數，而 unsigned 表示沒有 signed bit，簡單講就是沒正負，128Bits 全用來表示整數，所以可以表示 0- 2^{128} 也就是從 0 到 0xffffffffffffffffffffffffffffffff ，但即使用這麼大的整數，當 n 約在 1000110 左右時 S6 就超過範圍了，不過幸好即使超出範圍，尾數

128bits 還是正確，因此若忽視進位，還是可以比較速度，並檢查計算結果是否正確。

而在程式 Output 使用如 sum= xxxxxxxx24924924679e79e79e79e79e80000000

其中 xxxxxxxx 表示超出範圍(進位)之位數 Don’ t Care。

(二)xxxxxxxx 128 bit 進位 Don’ t care 變通辦法測試程式

為了讓進位可以不影響結果, 作者寫了一個測試程式如下：

```
int main(int argc, char* argv[])
{
    unsigned __int128 n=0xfedcba9876543210fedcba9876543210;
    int      *p;// pointer helps to print 128 bit integer __int128
    p= (int*) &n;
    p[3]=0xfedcba98;
    p[2]=0x76543210;
    p[1]=0xfedcba98;
    p[0]=0x76543210;
    printf("\nn=            xxxxxxxx%08x%08x%08x%08x\n\n", p[3], p[2], p[1], p[0]);
    for(int i=0; i<40; i++){
        n*=2;
        printf("n*=2  then n= xxxxxxxx%08x%08x%08x%08x\n", p[3], p[2], p[1], p[0]);
    }

    p[3]=0xfedcba98;
    p[2]=0x76543210;
    p[1]=0xfedcba98;
    p[0]=0x76543210;
    printf("\nn=            xxxxxxxx%08x%08x%08x%08x\n\n", p[3], p[2], p[1], p[0]);
    for(int i=0; i<40; i++){
        n*=16;
        printf("n*=16 then n= xxxxxxxx%08x%08x%08x%08x\n", p[3], p[2], p[1], p[0]);
    }

    p[3]=0xffffffff;
    p[2]=0xffffffff;
    p[1]=0xffffffff;
    p[0]=0xffffffff;
    printf("\nn=            xxxxxxxx%08x%08x%08x%08x\n\n", p[3], p[2], p[1], p[0]);
    for(int i=0; i<40; i++){
        n+=1;
        printf("n+=1  then n= xxxxxxxx%08x%08x%08x%08x\n", p[3], p[2], p[1], p[0]);
    }

    return 0;
}
```

128 bit 整數測試程式

其測試結果如下

```
n=            xxxxxxxxfedcba9876543210fedcba9876543210
n*=2  then n= xxxxxxxxfdb97530eca86421fdb97530eca86420
當n乘2  n進位 xxxxxxxx= 00000001 但尾巴fdb97530eca86421fdb97530eca86420正確
以 xxxxxxxxfdb97530eca86421fdb97530eca86420 表示
   00000001fdb97530eca86421fdb97530eca86420
Don't care 00000001 並不影響結果

n*=2  then n= xxxxxxxxfb72ea61d950c843fb72ea61d950c840
當n乘2  n進位 xxxxxxxx= 00000001 但尾巴fb72ea61d950c843fb72ea61d950c840正確
以 xxxxxxxxfb72ea61d950c843fb72ea61d950c840 表示
   00000001fb72ea61d950c843fb72ea61d950c840
Don't care 00000001 並不影響結果

  n*=2  then n= xxxxxxxxf6e5d4c3b2a19087f6e5d4c3b2a19080
當n乘2  n進位 xxxxxxxx= 00000001 但尾巴f6e5d4c3b2a19087f6e5d4c3b2a19080正確
以 xxxxxxxxf6e5d4c3b2a19087f6e5d4c3b2a19080 表示
   00000001f6e5d4c3b2a19087f6e5d4c3b2a19080
Don't care 00000001 並不影響結果
```

```
n=           xxxxxxxxfedcba9876543210fedcba9876543210
n*=16 then n= xxxxxxxxedcba9876543210fedcba98765432100
n*=16 then n= xxxxxxxxdcba9876543210fedcba987654321000
n*=16 then n= xxxxxxxxcba9876543210fedcba9876543210000
n*=16 then n= xxxxxxxxba9876543210fedcba98765432100000
n*=16 then n= xxxxxxxxa9876543210fedcba987654321000000
n*=16 then n= xxxxxxxx9876543210fedcba9876543210000000
n*=16 then n= xxxxxxxx876543210fedcba98765432100000000
n*=16 then n= xxxxxxxx76543210fedcba987654321000000000
n*=16 then n= xxxxxxxx6543210fedcba9876543210000000000
n*=16 then n= xxxxxxxx543210fedcba98765432100000000000
n*=16 then n= xxxxxxxx43210fedcba987654321000000000000
n*=16 then n= xxxxxxxx3210fedcba9876543210000000000000
n*=16 then n= xxxxxxxx210fedcba98765432100000000000000
n*=16 then n= xxxxxxxx10fedcba987654321000000000000000
n*=16 then n= xxxxxxxx0fedcba9876543210000000000000000
n*=16 then n= xxxxxxxxfedcba98765432100000000000000000
n*=16 then n= xxxxxxxxedcba98765432100000000000000000
n*=16 then n= xxxxxxxxdcba98765432100000000000000000
n*=16 then n= xxxxxxxxcba98765432100000000000000000
n*=16 then n= xxxxxxxxba98765432100000000000000000
n*=16 then n= xxxxxxxxa98765432100000000000000000
n*=16 then n= xxxxxxxx987654321000000000000000000
n*=16 then n= xxxxxxxx876543210000000000000000000
n*=16 then n= xxxxxxxx765432100000000000000000000
n*=16 then n= xxxxxxxx654321000000000000000000000
n*=16 then n= xxxxxxxx543210000000000000000000000
n*=16 then n= xxxxxxxx432100000000000000000000000
n*=16 then n= xxxxxxxx321000000000000000000000000
n*=16 then n= xxxxxxxx210000000000000000000000000
n*=16 then n= xxxxxxxx100000000000000000000000000
n*=16 then n= xxxxxxxx000000000000000000000000000
每次Shift to Left 4 bits xxxxxxxx 有變化但 Don't Care
```

```
n=           xxxxxxxxffffffffffffffffffffffffffffffff
n+=1  then n= xxxxxxxx0000000000000000000000000000000
xxxxxxxx= 00000001 但 Don't Care
尾數00000000000000000000000000000000不受影響

n+=1  then n= xxxxxxxx0000000000000000000000000000001
xxxxxxxx= 00000001 但 Don't Care
尾數00000000000000000000000000000001不受影響

n+=1  then n= xxxxxxxx0000000000000000000000000000002
xxxxxxxx= 00000001 但 Don't Care
尾數00000000000000000000000000000002不受影響
```

(三)重要的 local variables

```
unsigned __int128  nn=1 ,nnn=1 ,sum=0;
unsigned __int128  n=1, nnp1=4;
unsigned __int128 cubicsum=0 ,sum2lastnnn=0, sum2n=0, n2=0, nn3=3, ntmp=0;
//nn   ie n^2     表示 n^2
//nnp1 ie (n+1)^2 表示 (n+1)^2
//nnn  ie n^3     表示 n^3
//sum  為總和
//cubicsum    (n(n+1)/2)^2 即S3 3次級數和
//sum2lastnnn ie sum to last n^3
//sum2n       ie sum to n
//n2          ie n*2  因為  2n is not allowed by compiler
//nn3         ie nn*3 因為 3nn is not allowed by compiler
time_t  start_time, end_time;
//start_time 開始時間
//end_time   結束時間
int      *p;
//p is pointer helps to print 128 bit integer __int128
//因為標準C無法把sum(__int128)完整印出,所以使用p指到sum
//的位址(&sum),再把 p[3],p[2],p[1],p[0]印出即可。
//p= (int*) &sum;
//printf("sum= xxxxxxxx%08x%08x%08x%08x\n", p[3], p[2], p[1], p[0]);
```

(四)方法一 Method1

Method1 不作任何處理，直接以 n*n*n*n 或 n*n*n*n*n 或 n*n*n*n*n*n 累加，因乘法非常耗時，所以速度很慢!

當 n=0x100000000 時，計算 $1^4 + 2^4 + 3^4$ ...+n^4需 143 秒，$1^5 + 2^5 + 3^5$ ...+n^5需 178 秒，$1^6 + 2^6 + 3^6$ ...+n^6需 225 秒。

以下為其 C 程式片段:

```
void Method1(int odr) //方法一,不作任何處理,直接以n^4或n^5或n^6相加
{
    ----------------------
    ----------------------
       if(odr==6){//6次
         for(n=1;n<=N;n++){//n從0到N(0x100000000)
             sum+=n*n*n*n*n*n;//n^6 加總
     }//
         //當n=1 sum=1^6=1
         //當n=2 sum=1^6+ 2^6
         //當n=3 sum=1^6+ 2^6+ 3^6
         //當n=4 sum=1^6+ 2^6+ 3^6+ 4^6
       }
       else if(odr==5){//5次
         for(n=1;n<=N;n++){//n從0到N(0x100000000)
             sum+=n*n*n*n*n;//n^5 加總
         }//
         //當n=1 sum=1^5=1
         //當n=2 sum=1^5+ 2^5
         //當n=3 sum=1^5+ 2^5+ 3^5
         //當n=4 sum=1^5+ 2^5+ 3^5+ 4^5
       }
       else if(odr==4){//4次
         for(n=1;n<=N;n++){//n從0到N(0x100000000)
             sum+=n*n*n*n;//n^4 加總
         }// sum= 1^4+ 2^4+....+N^4= S4
         //當n=1 sum=1^4=1
         //當n=2 sum=1^4+ 2^4
         //當n=3 sum=1^4+ 2^4+ 3^4
         //當n=4 sum=1^4+ 2^4+ 3^4+ 4^4
      }
      ----------------------
  ----------------------
   }
```

(五)方法二 Method2

直接以 n * S3- $\sum_{i=1}^{n-1}(\sum_{j=1}^{i} j^3)$相加

因方法一使用大量乘法,因此甚為耗時，利用(式 肆.二.2)，仍然需要 n*n*n 求$\sum_{i=1}^{n-1}(\sum_{j=1}^{i} j^3)$，所以只能小幅度加速,當 n=0x100000000 時，計算 $1^4 + 2^4 + 3^4$ ...+n^4需 83 秒，其 C 程式重要片段如下:

```
void Method2(int odr)//方法二, 直接以N*S3-ΣΣ n^3相加
{
    ........................
    ........................
    for(n=1; n<N; n++){
       sum+=  (sum2lastnnn);//sum of sum2lastnnn
       sum2lastnnn+= n*n*n; //sum2sum2lastnnn 即為 Σ n^3
    }
    //n=1 sum= Σsum2lastnnn sum2lastnnn=0
    //n=2 sum= Σsum2lastnnn sum2lastnnn=1^3
    //n=3 sum= Σsum2lastnnn sum2lastnnn=1^3+ 2^3
    //n=4 sum= Σsum2lastnnn sum2lastnnn=1^3+ 2^3+ 3^3

    sum+=  (sum2lastnnn);// (1^3+2^3+....+n^3)- sum of sum2lastnn
    //now sum= 1^3 + (1^3+2^3)+ (1^3+2^3+3^3)+......
    //             + (1^3+2^3+....+(n-1)^3)- sum of sum2lastnnn
    sum= cubicsum* N - (sum ); //N*S3 - ΣΣ n^3
    ........................
    ........................
}
```

四,撰寫四次級數求和 C 程式，第二步

(一). 方法三 Method3

方法二有 n*n*n 所以需時 83 秒完成$1^4 + 2^4 + 3^4$ ...+$(2^{32})^4$，而$\sum_{i=1}^{n-1}(\sum_{j=1}^{i} j^3) =$

$\sum_{i=1}^{n-1}(\frac{i(i+1)}{2})^2=\frac{\sum_{i=1}^{n-1} u^2*(i+1)^2}{4}$其中$(i+1)^2 = i^2 + 2i + 1$(可用以降冪)，因此在 for loop 中，

j^2和$(j+1)^2$二者只要算其中之一即可，而且$(j+1)^2$可以加法(2i+1)降冪算出，故可以節省更多時間，以下為其程式片段如下：

```
void Method3(int odr)//方法三 S4= N*S3- Σ(n^2(n+1)^2)/4
{
    ------------------------
    ------------------------
    for(n=1; n<N; n++){
        sum+=  (sum2lastnnn);// (1^3+ 2^3+....+ n^3)- sum of sum2lastnnn
        nn= nnp1;// nnp1 給 nn
        nnp1+=  ( n+n +1) ;// nnp1 ie (n+1)^2= n^2 + 2*n + 1
        sum2lastnnn= (nn*(nnp1));//sum2sum2lastnnn = n^2 * (n+1)^2
    }
    //當n=1 sum=0   nn=1  nnp1=4  sum2lastnnn=4
    //當n=2 sum=4   nn=4  nnp1=9  sum2lastnnn=36
    //當n=3 sum=40  nn=9  nnp1=16 sum2lastnnn=144
    //當n=4 sum=184 nn=16 nnp1=25 sum2lastnnn=400
    sum+=  (sum2lastnnn);// (1^3+2^3+....+n^3)- sum of sum2lastnnn
    sum=   cubicsum* N - (sum /4);//只要在此除4一次即可,可節省大量時間
    ------------------------
    ------------------------
}
```

當 n=0x100000000 時，計算 $1^4+2^4+3^4$ ...+n^4需 50 秒，本方法在某些狀況下會有結果與他法不一致之缺點，但只要 n 不要過大，此法表現堪稱差強人意!

$$S_4 = 1^4+2^4+3^4+\cdots\cdots+n^4 = 1\times1^3+2\times2^3+3\times3^3+\cdots\cdots+n\times n^3$$

$$= 1^3+2^3+3^3+\cdots\cdots+n^3 \rightarrow S_3$$

$$+2^3+3^3+\cdots\cdots+n^3 \rightarrow S_3-1^3$$

$$+3^3+\cdots\cdots+n^3 \rightarrow S_3-1^3-2^3$$

$$+\cdots\cdots\cdots\cdots$$

$$+n^3 \rightarrow S_3-1^3-2^3-\cdots\cdots-(n-1)^3$$

$$= nS_3-\sum_{i=1}^{n-1}\sum_{j=1}^{i} j^3 \qquad \text{(式 肆.二.1)}$$

$$= n * S3- \sum_{i=1}^{n-1}\left(\frac{i(i+1)}{2}\right)^2$$

$$=-n * S3- \frac{\sum_{i=1}^{n-1} i^2*(i+1)^2}{4} \text{(式 肆. 四.1)}$$

(二)方法五 Method5

方法五利用$(j+1)^3 = j^3+3(j^2+j)+1$ 及$(j+1)^2 = j^2+2*j+1$降冪，

技巧簡化 n*n*n，需時 58 秒完成$1^4+2^4+3^4$ ...+$(2^{32})^4$

以下為其 C 程式重要片段：

```
void Method5(int odr)
{
    ----------------------
    ----------------------
    for(n=1; n<N; n++){
       sum+=  (sum2lastnnn);// (1^3+2^3+....+n^3)- sum of sum2lastnnn
       sum2lastnnn+= nnn;//sum2sum2lastnnn ie (i-1)^3
       nnn+=  3*(nn  +n )+1;// for (i+1)^3
       nn+=   (n+n)+1 ;// for (i+1)^3
       //printf("n=%8llu nn=%20llu nnn=%30llu, sum2lastnnn==%40llu\n", n, nn, nnn, sum2lastnnn);
    }
    sum+=  (sum2lastnnn);// (1^3+2^3+....+n^3)- sum of sum2lastnnn
    sum= cubicsum* N - (sum  );
    ----------------------
    ----------------------
}
```

五、撰寫四次級數求和 C 程式，第三步：

方法四 Method4

寫報告時一再碰釘子，也很苦惱。寫不下去時，就一再用筆在計算紙上列式子 某日突然發現

1^3=1 = 1^2

1^3+ 2^3=9 =$(1+2)^2$

1^3+ 2^3 + 3^3=36 = $(1+2+3)^2$

對啊! S3 就是 $(\frac{n(n+1)}{2})^2$ 而$\frac{n(n+1)}{2}$不就是大數學家高斯小時因調皮而被老師罰算的 1+ 2+ 3+.....+ n 嗎!!!

但這又怎樣???

對程式有何幫助???

……???

因為 n*n*n 改成$\frac{n^2(n+1)^2}{4}$雖然讓使用時間從 83 秒降到 50 秒，但 Method3 計算 sum2lastnnn 包括 for loop 還是很麻煩!!!

而$(1+2+3+\ldots.+n)^2$ 不但再降冪(4->3->2)，而且每次進 for loop 裏只要加一行 sumx+=n 即可求出 1+ 2+ 3+.....+ n

式子簡單加上再降冪，正好是程式加速之兩大要件!!!

毫無疑問，這對本已低迷的 Project，注入一股強心動能!!!

而 $1^3+2^3+3^3$ …….+n^3 剛好等於$(1+2+3+\ldots.+n)^2$

這讓我體悟到數字之美!!! 也讓我不禁感動!!!

$$S_4 = 1^4 + 2^4 + 3^4 + \cdots\cdots + n^4 = 1\times 1^3 + 2\times 2^3 + 3\times 3^3 + \cdots\cdots + n\times n^3$$

$$= 1^3 + 2^3 + 3^3 + \cdots\cdots + n^3 \rightarrow S_3$$

$$+ 2^3 + 3^3 + \cdots\cdots + n^3 \rightarrow S_3 - 1^3$$

$$+ 3^3 + \cdots\cdots + n^3 \rightarrow S_3 - 1^3 - 2^3$$

$$+ \cdots\cdots\cdots\cdots$$

$$+ n^3 \rightarrow S_3 - 1^3 - 2^3 - \cdots\cdots - (n-1)^3$$

$$= nS_3 - \sum_{i=1}^{n-1}\sum_{j=1}^{i} j^3 \qquad \text{(式 肆.二.1)}$$

$$= nS3 - \sum_{i=1}^{n-1}\left(\sum_{j=1}^{i} j\right)^2 \qquad \text{(式 肆.五.1)}$$

以下為其 C 程式重要片段：

```
//方法四 S4= N*S3- Σ((Σn)^2)
//因為1^3       =( 1   )^2
//    1^3+2^3    =(1+2  )^2
//    1^3+2^3+3^3=(1+2+3)^2
void Method4(int odr)
{
    ----------------------
    ----------------------
    else if(odr==4){
        cubicsum=N*(N+1)/2;
        cubicsum*=cubicsum;//cubicsum即為S3=1^3+ 2^3+....+N^3=(N*(N+1)/2)^2
        nn=1 ,sum=0;//nn, sum值初始化
        sum2n=0;
        for(n=1; n<N; n++){
           sum+=sum2n*sum2n;//1^3=(1)^2 1^3+2^3=(1+2)^2 1^3+2^3+3^3=(1+2+3)^2= sum2n^2
           sum2n+=n;//(1+2+3) 即為 sum2n
        }
        sum+=sum2n*sum2n;
        sum= cubicsum*N- sum;//n*S3- Σ(Σn)^2

    }
    ----------------------
    ----------------------
}
```

當 n=0x100000000 時， 計算 $1^4 + 2^4 + 3^4$ ...+n^4需 33 秒，是 4 月之前為止最有效之方法!

六、五次級數

$$S_5 = 1^5 + 2^5 + 3^5 + \cdots\cdots + n^5 = 1\times 1^4 + 2\times 2^4 + 3\times 3^4 + \cdots\cdots + n\times n^4$$
$$= 1^4 + 2^4 + 3^4 + \cdots\cdots + n^4 \rightarrow S_4$$
$$+ 2^4 + 3^4 + \cdots\cdots + n^4 \rightarrow S_4 - 1^4$$
$$+ \cdots\cdots\cdots\cdots$$
$$+ n^4 \rightarrow S_4 - 1^4 - 2^4 - \cdots\cdots - (n-1)^4$$
$$= nS_4 - \sum_{i=1}^{n-1}\sum_{j=1}^{i} j^4 \qquad \text{(式 肆.六.2)}$$

(式 肆. 六.1)是早期所導出之五次級數求和公式(上學期)， 在完成前述四次級數求和程式後，並不難發現，要把(式 肆.六.2)寫為 C 程式可不容易!

而且即使做到也絕對不快!因為 S4 並沒有一個簡單的式子可以求出!

同樣地也是在寫報告時一再碰釘子，很苦惱，實在寫不下去時，就一再用筆在計算紙上列式子，突然靈光乍現.....

$$S_5 = 1^5 + 2^5 + 3^5 + \cdots\cdots + n^5 = 1^2\times 1^3 + 2^2\times 2^3 + 3^2\times 3^3 + \cdots\cdots + n^2\times n^3$$
$$= 1^3 + 2^3 + 3^3 + \cdots\cdots + n^3 \rightarrow S_3$$
$$+ 2^3 + 3^3 + \cdots\cdots + n^3 \rightarrow S_3 - 1^3$$
$$+ 2^3 + 3^3 + \cdots\cdots + n^3 \rightarrow S_3 - 1^3$$
$$+ 2^3 + 3^3 + \cdots\cdots + n^3 \rightarrow S_3 - 1^3$$
$$+ 3^3 + \cdots\cdots + n^3 \rightarrow S_3 - 1^3 - 2^3$$
$$+ \cdots\cdots\cdots\cdots \qquad + \cdots\cdots\cdots\cdots$$
$$+ n^3 \rightarrow S_3 - 1^4 - 2^4 - \cdots\cdots - (n-1)^4$$
$$= n^2 S_3 - \sum_{i=1}^{n-1}(2i+1)\left(\sum_{j=1}^{i} j^2\right)^2 \qquad \text{(式 肆.六.3)}$$

一樣不難發現，只要稍加修改即可將方法四改成五次級數求和程式如下。

七、五次級數 C 程式

根據(式 肆. 六.3)，可把方法四修改成如下：

```
void Method4(int odr)
{
    ......................
    ......................
    else if(odr==5){
        cubicsum=N*(N+1)/2;
        cubicsum*=cubicsum;//cubicsum即為S3=1^3+ 2^3+....+N^3=(N*(N+1)/2)^2
        nn=1 ,sum=0;//nn, sum值初始化
        sum2lastnnn=0, sum2n=0, n2=0, nn3=3, ntmp=0;
        for(n=1; n<N; n++){
            sum+=sum2n*sum2n*(n+n-1);
            //根據 (式     肆.六.1)
            //1^3=(1)^2 1^3+2^3=(1+2)^2 1^3+2^3+3^3=(1+2+3)^2= sum2n^2
            sum2n+=n;//(1+2+3) 即為 sum2n
        }///
        sum+=sum2n*sum2n*(n+n-1);
        sum= cubicsum*N*N- sum;////根據 (式     肆.六.1)
    }
    ......................
    ......................
}
```

當 n=0x100000000 時， 計算 $1^5+2^5+3^5$ ...+n^5需時 86 秒， 而 Method1(5)需時 178 秒。

八、六次級數

同理

$$S_6=1^6+2^6+3^6+\cdots\cdots+n^6=1^3\times1^3+2^3\times2^3+3^3\times3^3+\cdots\cdots+n^3\times n^3$$

$$=1^3+2^3+3^3+\cdots\cdots+n^3\rightarrow S_3$$

$$+2^3+3^3+\cdots\cdots+n^3\rightarrow S_3-1^3$$

$$+2^3+3^3+\cdots\cdots+n^3\rightarrow S_3-1^3$$

$$+2^3+3^3+\cdots\cdots+n^3\rightarrow S_3-1^3$$

$$+2^3+3^3+\cdots\cdots+n^3\rightarrow S_3-1^3$$

$$+2^3+3^3+\cdots\cdots+n^3\rightarrow S_3-1^3$$

$$+2^3+3^3+\cdots\cdots+n^3\rightarrow S_3-1^3$$

$$+2^3+3^3+\cdots\cdots+n^3\rightarrow S_3-1^3$$

$$+3^3+\cdots\cdots+n^3\rightarrow S_3-1^3-2^3$$

$$+\cdots\cdots\cdots\cdots$$

$$+\cdots\cdots\cdots\cdots$$

$$+n^3\rightarrow S_3-1^3-2^3-\cdots\cdots-(n-1)^3$$

$$=n^3S_3-\sum_{i=1}^{n-1}(3i^2+3i+1)(\sum_{j=1}^{i}j^2)^2 \qquad \text{(式 肆.八.1)}$$

九、六次級數 C 程式

根據(式 肆. 八.1)，可把方法四修改成如下：

```
void Method4(int odr)
{
    ----------------------
    ----------------------
    if(odr==6){
        for(n=1; n<N; n++){
            n2= n+n;
            sum+=sum2n*sum2n*(ntmp);//根據 (式 肆.八.1)
            ntmp=nn3+n2+n+1;
            //n=1 n2=2 nn=1 nn3=3  sum2n=0 ntmp=3+2+1+1=7= 8- 1
            //n=2 n2=4 nn=4 nn3=12 sum2n=1 ntmp=12+4+2+1= 19= 27- 8
            nn+=n2+1; //(n+1)^2= n^2+ 2n+ 1    n2: ie n*2
            nn3=nn+nn+nn;// nn3= nn*3= 3*n^2
            sum2n+=n;//(1+2+3) 即為 sum2n
        }///
        sum+=sum2n*sum2n*(ntmp);
        sum= cubicsum*N*N*N- sum;//根據 (式 肆.八.1)
    }
    ----------------------
    ----------------------
}
```

當 n=0x100000000 時， 計算

$1^6 + 2^6 + 3^6$ ...+n^6需時 101 秒，比 Method1(4)方法一求四方和(143 秒)還快!

而 Method1(6)需時 225 秒。

十、k 次級數一般解

Sk= 1^k +2^k+⋯.+n^k　　本來想刪掉此式,沒想到在後面卻大有用處

$$= S3*(n^{k-3}) - \sum_{i=1}^{n-1}\left((i+1)^{k-3} - (i)^{k-3}\right)\left(\sum_{j=1}^{i} j\right)^2$$

十一、發現超級棒的式子

春假期間看到 Youtube 上一部片子(參考資料五)以 Telescope Sum 求出四次和如下:

$\sum_{i=1}^{n}((i+1)^5 - i^5) = (n+1)^5 - n^5 + n^5 - (n-1)^5 \cdots\cdots\cdots\cdots -1 = (n+1)^5 - 1$

而$\sum_{i=1}^{n}((i+1)^5 - i^5) = \sum_{i=1}^{n}(5i^4 + 10i^3 + 10i^2 + 5i + 1)$

$= 5\sum_{i=1}^{n} i^4 + 10\sum_{i=1}^{n} i^3 + 10\sum_{i=1}^{n} i^2 + 5\sum_{i=1}^{n} i + n$

$= 5S4 + 10S3 + 10S2 + 5S1 + n$

$5S4 = (n+1)^5 - 1 - 10S3 - 10S2 - 5S1 - n$

$= n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1 - 1 - n - 10S3 - 10S2 - 5S1$

$S4 = \frac{n^5 + 5n^4 + 10n^3 + 10n^2 - 4n - 10S3 - 10S2 - 5S1}{5}$ (式　肆. 十一.1)

因為 S1、S2、S3 為已知,所以就可從而找出 S4。

同理，

$S2=\frac{(n+1)^3-1-n-3S1}{3}$(式　肆. 十一.2)

$S3=\frac{(n+1)^4-1-n-6S2-4S1}{4}$(式　肆. 十一.3)

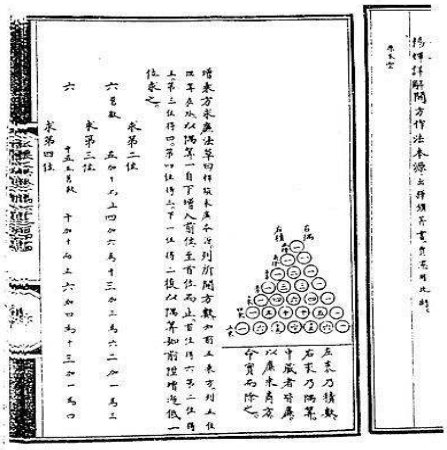
$S5=\frac{(n+1)^6-1-n-15S4-20S3-15S2-6S1}{6}$　　(式　　肆. 十一.4)

$S6=\frac{(n+1)^7-1-n-21S5-35S4-35S3-21S2-7S1}{7}$(式肆. 十一.5)

$Sm=\frac{(n+1)^{m+1}-1-n-C_2^{m+1}(Sm-1)\ldots\ldots-\ldots-C_m^{m+1}S1}{m+1}$(式肆. 十一.6)

雖然只要次方大一些求 Sm 就變得很複雜，但正好可以程式求解！

十二、求係數程式



(了不起的賈憲三角形)

程式先建立賈憲三角形 stri

```
stri[0][0]= 1;
stri[0][1]= 1;
for(i=1;i<=10;i++){
    stri[i][0]= 1;
    for(j=1;j<i+1;j++){
        stri[i][j]= stri[i-1][j]+ stri[i-1][j-1];
    }
    stri[i][j]= 1;
}
```

然後把 S1、S2、S3 已知之係數放入 bern[0][x](S1),bern[1][x], bern[2][x]

根據(式　肆. 十一.1)求出 S4 係數並放入 bern[3][1](S4 之 1 次係數)bern[3][2](2 次係數)，

bern[3][3](3 次係數)bern[3][4](4 次係數)bern[3][5](5 次係數)

有了 S4 就可求 S5……一直到 100 為止。

十三、遞迴求和程式

以前述程式找出係數之後,發現不管是程式找出的係數，或是以手算白努利數套式子找出的係數(以分數表示)，只要數字大一些就有嚴重的誤差(Truncation Error)。為了解決 Truncation Error，試著以遞迴方法求和,求 sum2k(4) (即 S4) 會根據(式 肆. 十一.1)遞迴呼叫 sum2k(3) (即 S3)，一層層做完後再回來求 sum2k(2) (即 S2)及 sum2k(1) (S1)，本程式又快又正確，是本研究五月前之最佳解決方案！

本程式運算迅速精確，當 N=0x1000 以 Method1 求 S10 0x2000000 次，需時 841 秒,用遞迴方式求 S10 0x2000000 次只需 7 秒！

其重要程式片段如下:

遞迴求和程式

```
unsigned __int128 Telescopesum2k(int odr)
{//遞迴求和程式 by Telescoping Sum
    unsigned __int128 sum= 0;
    int i;
    if(S[odr]){//若S[odr]非0 表示Sodr已算過 直接傳回其值即可 S[odr]在main()要先清為0
        return S[odr];
    }
    if(odr!=1){//若odr不等於1
    //則必須Recursive呼叫Telescopesum2k(odr-1),Telescopesum2k(odr-2)...Telescopesum2k(1)
        sum+= (prod[odr+1]-N-1);//prod[odr+1]=(N+1)^odr+1在main()要先填好值
        for(int k= odr-1; k>0; k--){//recursive 呼叫 Sodr, Sodr-1, Sodr-2.......S1
            sum-= stri[odr+1][k+1] * Telescopesum2k(k);
        }
        sum/= (odr+1);//依式子必須除 odr+1
        S[odr]= sum;//以S[odr]存sum
        return sum;//傳回sum
    }
    sum= (N*(N+1)) /2;//若odr等於1 直接傳S1值(N*(N+1)) /2
    S[odr]= sum;//以S[1]存S1
    return sum;//傳回sum
}
```

十四、利用 CAP 程式(Computer Aided Proof)找出並證明通式，同時找出白努利數關係式。

因為牽涉大量機械式運算，所以即使是使用數學歸納法也是舉步維艱，最後終於下定決心寫程式解決問題,大量機械式運算,就該機器去作!真不知道機器證明法 Machine Proof 或 Computer Aided Proof(CAP)也有其他人用過嗎？基本上，利用 Telescoping Sum(式肆. 十一.6)逐次找出 Sm 之n^{m+1},n^m, n^{m-1}, n^{m-2}…即可找到白努利數，並導出通式，利用程式不但省去大量運算，更可以更嚴謹證明出通式，還可更容易、更方便找出埋在式子裡的神祕秩序！例如我是有了此 CAP 程式後才找到b_k關係式。$b_{k+1} = \left(1 - \frac{k+1}{2!} b_k - \frac{(k)(k+1)}{3!} b_{k-1} - \cdots - b1 - \frac{1}{k+2} b0\right)$。此外程式還發現$n^k$與$S_m$中只要 m-k 為定值,則不管是

$Coef_{Sm}^{m-k}$(令$Coef_{Sm}^{m-k}$表示S_m之n^{m-k}係數),$Coef_{m+1}^{m-k+1}$,$Coef_{m-1}^{m-k-1}$, $Coef_1^1$,$Coef_2^2$結果多一致,可以b_{m-k+1}表示,所以程式 Output 如下:

```
Prooof: To prove Coefficient n^ (k-1) of Sk   Coef(k-1, Sk)= C(k+1, 1)*b1/k+1
Coef(m+1, Sm+1)
=( C(m+2, 1)-C(m+2, 2)* Coef(m+1, Sm))/ m+2
=( C(m+2, 1)*(1-(m+1)/2* Coef(m+1, Sm) )/ m+2
=( C(m+2, 1)*(1-(m+1)/2* 1/(m+1)))/ m+2
=( 1  - 1/2 b0 ) C(m+2, 1)/ m+2
Coef(m, Sm)
=( C(m+1, 1)-C(m+1, 2)* Coef(m, Sm-1))/ m+1
=( C(m+1, 1)*(1-(m)/2* Coef(m, Sm-1) )/ m+1
=( C(m+1, 1)*(1-(m)/2* 1/(m)))/ m+1
=( 1  - 1/2 b0 ) C(m+1, 1)/ m+1
Coef(m-1, Sm-1)
=( C(m, 1)-C(m, 2)* Coef(m-1, Sm-2))/ m
=( C(m, 1)*(1-(m-1)/2* Coef(m-1, Sm-2) )/ m
=( C(m, 1)*(1-(m-1)/2* 1/(m-1)))/ m
=( 1  - 1/2 b0 ) C(m, 1)/ m
Coef(1, S1)
=( C(1+1, 1)-C(1+1, 2)* Coef(1, S1-1))/ 1+1
=( C(1+1, 1)*(1-(1)/2* Coef(1, S1-1) )/ 1+1
=( C(1+1, 1)*(1-(1)/2* 1/(1)))/ 1+1
=( 1  - 1/2 b0 ) C(1+1, 1)/ 1+1
Coef(2, S2)
=( C(2+1, 1)-C(2+1, 2)* Coef(2, S2-1))/ 2+1
=( C(2+1, 1)*(1-(2)/2* Coef(2, S2-1) )/ 2+1
=( C(2+1, 1)*(1-(2)/2* 1/(2)))/ 2+1
=( 1  - 1/2 b0 ) C(2+1, 1)/ 2+1
Now we have Sm+1, Sm, Sm-1, S1, S2 ok, so it's proven True
```

CAP程式
Computer Aided Proof
意外發現bk關係式

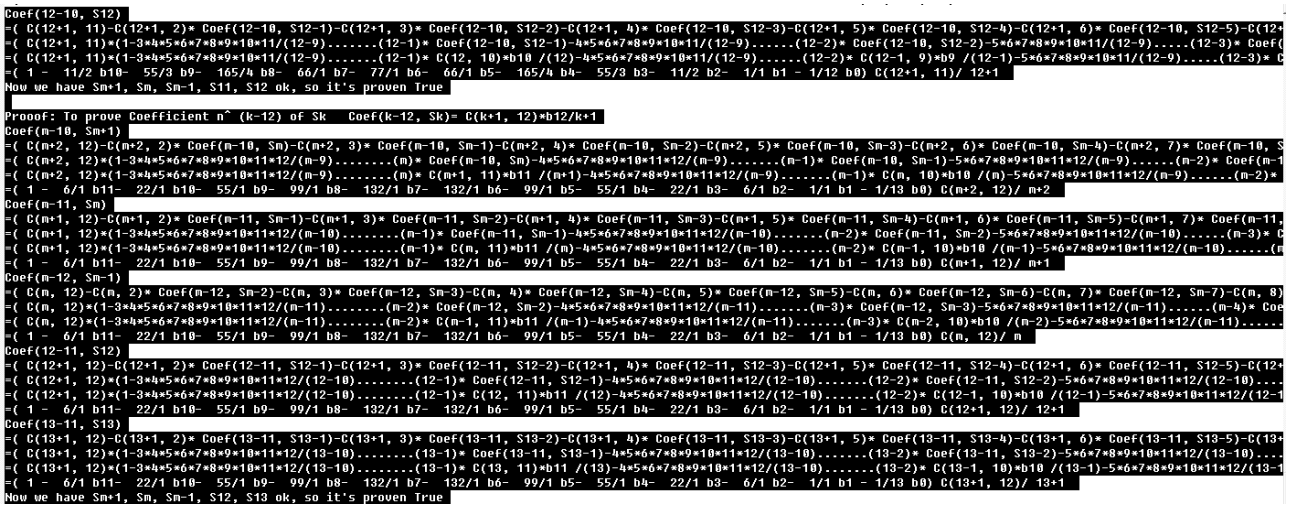
$$b_{k+1} = \left(1 \quad \frac{k+1}{2!} b_k \quad \frac{(k)(k+1)}{3!} b_{k-1} \quad \cdots \quad b1 \quad \frac{1}{k+2} b0\right)$$

bk關係式有遞迴關係
且目前已完成遞迴求
bk程式

遞迴求bk程式
及其output

```
Prooof: To prove Coefficient n^ (k-2) of Sk   Coef(k-2, Sk)= C(k+1, 2)*b2/k+1
Coef(m, Sm+1)
=( C(m+2, 2)-C(m+2, 2)* Coef(m, Sm)-C(m+2, 3)* Coef(m, Sm-1))/ m+2
=( C(m+2, 2)*(1-1* Coef(m, Sm)-(m)/3* Coef(m, Sm-1) )/ m+2
=( C(m+2, 2)*(1-1* C(m+1, 1)*b1 /(m+1)-(m)/3* 1/(m)))/ m+2
=( 1 -  1/1 b1 - 1/3 b0) C(m+2, 2)/ m+2
Coef(m-1, Sm)
=( C(m+1, 2)-C(m+1, 2)* Coef(m-1, Sm-1)-C(m+1, 3)* Coef(m-1, Sm-2))/ m+1
=( C(m+1, 2)*(1-1* Coef(m-1, Sm-1)-(m-1)/3* Coef(m-1, Sm-2) )/ m+1
=( C(m+1, 2)*(1-1* C(m, 1)*b1 /(m)-(m-1)/3* 1/(m-1)))/ m+1
=( 1 -  1/1 b1 - 1/3 b0) C(m+1, 2)/ m+1
Coef(m-2, Sm-1)
=( C(m, 2)-C(m, 2)* Coef(m-2, Sm-2)-C(m, 3)* Coef(m-2, Sm-3))/ m
=( C(m, 2)*(1-1* Coef(m-2, Sm-2)-(m-2)/3* Coef(m-2, Sm-3) )/ m
=( C(m, 2)*(1-1* C(m-1, 1)*b1 /(m-1)-(m-2)/3* 1/(m-2)))/ m
=( 1 -  1/1 b1 - 1/3 b0) C(m, 2)/ m
Coef(2-1, S2)
=( C(2+1, 2)-C(2+1, 2)* Coef(2-1, S2-1)-C(2+1, 3)* Coef(2-1, S2-2))/ 2+1
=( C(2+1, 2)*(1-1* Coef(2-1, S2-1)-(2-1)/3* Coef(2-1, S2-2) )/ 2+1
=( C(2+1, 2)*(1-1* C(2, 1)*b1 /(2)-(2-1)/3* 1/(2-1)))/ 2+1
=( 1 -  1/1 b1 - 1/3 b0) C(2+1, 2)/ 2+1
Coef(3-1, S3)
=( C(3+1, 2)-C(3+1, 2)* Coef(3-1, S3-1)-C(3+1, 3)* Coef(3-1, S3-2))/ 3+1
=( C(3+1, 2)*(1-1* Coef(3-1, S3-1)-(3-1)/3* Coef(3-1, S3-2) )/ 3+1
=( C(3+1, 2)*(1-1* C(3, 1)*b1 /(3)-(3-1)/3* 1/(3-1)))/ 3+1
=( 1 -  1/1 b1 - 1/3 b0) C(3+1, 2)/ 3+1
Now we have Sm+1, Sm, Sm-1, S2, S3 ok, so it's proven True
```

一直就很想要寫這程式卻一直不敢動手，不是沒時間，其實因為怕太難，寫不出來，所以就一直拖，但因為這程式實在太有用了，所以只好硬著頭皮作。沒想到程式根本不難，卻很花時間，運算不多，需要一些簡單分析，大部分需要堆砌，花了好幾天時間，但是卻很值得！



(二)以數學歸納法證明通式

依據式肆.十四.1 至 6可得

$$S1=\ \frac{1}{1+1}n^2+\ b_1\frac{C_1^{1+1}}{1+1}n\ =\frac{1}{2}n^2+\ \frac{1}{2}n$$

$$S2=\ \frac{1}{2+1}n^3+\ b_1\frac{C_1^{2+1}}{2+1}n^2+b_2\frac{C_2^{1+1}}{(2+1)}n\ \ =\frac{1}{3}n^3+\ \frac{1}{2}n^2+\frac{1}{6}n$$

$$S3=\ \frac{1}{3+1}n^4+\ b_1\frac{C_1^{3+1}}{3+1}n^3+\ b_2\frac{C_2^{3+1}}{3+1}n^2+\ b_3\frac{C_3^{3+1}}{3+1}n\ =\frac{1}{4}n^4+\ \frac{1}{2}n^3+\ \frac{1}{4}n^2$$

$$S4=\ \frac{1}{4+1}n^5+\ b_1\frac{C_1^{4+1}}{4+1}n^4+\ b_2\frac{C_2^{4+1}}{4+1}n^3+\ b_3\frac{C_3^{4+1}}{4+1}n^2+\ b_4\frac{C_4^{4+1}}{4+1}n\ =\frac{1}{5}n^4+\frac{1}{2}n^4+\ \frac{1}{3}n^3-\ \frac{1}{30}n^2$$

$$S5=\ \frac{1}{5+1}n^6+\ b_1\frac{C_1^{5+1}}{5+1}n^5+\ b_2\frac{C_2^{5+1}}{5+1}n^4+\ b_3\frac{C_3^{5+1}}{5+1}n^3+\ b_4\frac{C_4^{5+1}}{5+1}n^2+b_5\ \frac{C_5^{5+1}}{5+1}n$$

$$=\frac{1}{6}n^6+\frac{1}{2}n^5+\ \frac{5}{12}n^4-\ \frac{1}{12}n^2$$

$$S6=\ \frac{1}{6+1}n^7+\ b_1\frac{C_1^{6+1}}{6+1}n^6+\ b_2\frac{C_2^{6+1}}{6+1}n^5+\ b_3\frac{C_3^{6+1}}{6+1}n^4+\ b_4\frac{C_4^{6+1}}{6+1}n^3+b_5\ \frac{C_5^{6+1}}{6+1}n^2+b_6\ \frac{C_6^{6+1}}{6+1}n$$

$$=\frac{1}{7}n^7+\frac{1}{2}n^6+\ \frac{1}{2}n^5-\ \frac{1}{6}n^3+\frac{1}{42}n$$

今假設

$Sm=\ \frac{1}{m+1}n^{m+1}+\ b_1\frac{C_1^{m+1}}{m+1}n^m+\ b_2\frac{C_2^{m+1}}{m+1}n^{m-1}+\ldots..+b_{m-1}\ \frac{C_{m-1}^{m+1}}{m+1}n^2+b_m\ \frac{C_m^{m+1}}{m+1}n$ 成立

(式肆.十四.7)

則依據(式肆. 十一.6)

$Sm+1=\frac{(n+1)^{m+2}-1-n-C_2^{m+2}(Sm).....-..-C_{m+1}^{m+2}S1}{m+2}$

如$Coef_{Sm}^{m}$依據(肆.十四.(一).1)一樣方式,也可找到 $Coef_{Sm+1}^{m+1}=\ \frac{1}{2}\frac{C_1^{m+2}}{m+1}=\ \frac{1}{2}=b_1$

如$Coef_{Sm}^{m-1}$依據(肆.十四.(一).2)一樣方式,也可找到 $Coef_{Sm+1}^{m}=\frac{1}{6}\frac{C_2^{m+2}}{(m+2)}\ =\ b_2\frac{C_2^{m+2}}{(m+2)}$

如$Coef_{Sm}^{m-2}$依據(肆.十四.(一).3)一樣方式,也可找到 $Coef_{Sm+1}^{m-1}=0=\ b_3$

如$Coef_{Sm}^{m-3}$依據(肆.十四.(一).4)一樣方式,也可找到 $Coef_{Sm+1}^{m-2}=-\frac{1}{30}\frac{C_4^{m+2}}{(m+2)}=\ b_4\frac{C_4^{m+2}}{(m+2)}$

如$Coef_{Sm}^{m-4}$依據(肆.十四.(一).5)一樣方式,也可找到 $Coef_{Sm+1}^{m-3}=0=\ b_5$

如 $Coef_{Sm}^{m-5}$依據(肆.十四.(一).6)一樣方式,也可找到 $Coef_{Sm+1}^{m-4}=\frac{1}{42}\frac{C_6^{m+2}}{(m+2)}=b_6\frac{C_6^{m+2}}{(m+2)}$

以一樣方式可找到如 $Coef_{Sm+1}^{m-5}=\ b_7\frac{C_7^{m+2}}{(m+2)}$

……

一直到$Coef_{Sm+1}^{3}=\ b_{m-1}\frac{C_{m-1}^{m+2}}{(m+2)}$

唯一比較特殊的是 $Coef_{Sm,}^{1}$因為$Coef_{Sm,}^{1}$還有一個 -1 如下式紅字部分

而$Coef_{Sm+1}^{2}$卻沒有這部分(-1), 所以必須特別證明如下

$.Coef_{Sm}^{1}=\frac{C_m^{m+1}-C_2^{m+1}(Coef_{Sm-1}^{1})-C_3^{m+1}(Coef_{Sm-2}^{1})-C_4^{m+1}(Coef_{Sm-3}^{1})--\cdots-C_{m-1}^{m+1}(Coef_{S2}^{1})-C_m^{m+1}(Coef_{S1}^{1})-1}{(m+1)}$

$=\frac{C_m^{m+1}-C_2^{m+1}(b_{m-1})-C_3^{m+1}(b_{m-2})-C_4^{m+1}(b_{m-3})-\cdots-C_m^{m+1}(b1)-1}{(m+1)}$

$=\frac{C_m^{m+1}-C_m^{m+1}\frac{m}{2!}(b_{m-1})-C_m^{m+1}\frac{m(m-1)}{3!}(b_{m-2})-C_m^{m+1}\frac{m(m-1)(m-2)}{4!}(b_{m-3})-\cdots-C_m^{m+1}(b1)-C_m^{m+1}\frac{1}{(m+1)}}{(m+1)}$

$=\frac{C_m^{m+1}\left(1-\frac{m}{2!}(b_{m-1})-\frac{m(m-1)}{3!}(b_{m-2})-\frac{m(m-1)(m-2)}{4!}(b_{m-3})-\cdots-(b1)-\frac{1}{(m+1)}\right)}{(m+1)}$

$=\frac{C_m^{m+1}(b_m)}{(m+1)}$

$.Coef_{Sm+1}^{2}=\frac{C_m^{m+2}-C_2^{m+1}(Coef_{Sm-1}^{2})-C_3^{m+2}(Coef_{Sm-2}^{2})-C_4^{m+2}(Coef_{Sm-3}^{2})--\cdots-C_m^{m+2}(Coef_{S2}^{1})-C_{m+1}^{m+2}(Coef_{S1}^{2})}{(m+2)}$

$=\frac{C_2^{m+2}-C_2^{m+1}\left(\frac{m}{2!}(b_{m-1})\right)-C_3^{m+2}\left(\frac{m-1}{2!}(b_{m-2})\right)-C_4^{m+2}\left(\frac{m-2}{2!}(b_{m-3})\right)--\cdots-C_m^{m+2}(b_1)-C_{m+1}^{m+2}\left(\frac{1}{2}\right)}{(m+2)}$

$$=\frac{C_2^{m+2}-C_2^{m+1}\left(\frac{m}{2!}(b_{m-1})\right)-C_3^{m+2}\left(\frac{m-1}{2!}(b_{m-2})\right)-C_4^{m+2}\left(\frac{m-2}{2!}(b_{m-3})\right)-\cdots-C_m^{m+2}(b_1)-C_2^{m+2}\left(\frac{1}{(m+1)}\right)}{(m+2)}$$

$$=\frac{C_m^{m+2}(1-\frac{m}{2!}(b_{m-1})-\frac{m(m-1)}{3!}(b_{m-2})-\frac{m(m-1)(m-2)}{4!}(b_{m-3})-\cdots-(b1)-\frac{1}{(m+1)})}{(m+2)}$$

$$=\frac{C_m^{m+2}(b_m)}{(m+2)}$$

以一樣的方式往回找,同樣可找到

$Coef_{Sm+1}^{3}$同$Coef_{Sm}^{2}$也含b_{m-1}

$Coef_{Sm+1}^{4}$同$Coef_{Sm}^{3}$也含b_{m-2}

……

$Coef_{Sm+1}^{m+1}$同$Coef_{Sm}^{m}$也含b_1

所以

Sm+1= $\frac{1}{m+2}n^{m+2}+ b_1\frac{C_1^{m+2}}{m+2}n^{m+1}+ b_2\frac{C_2^{m+2}}{m+2}n^{m}+\ldots..+b_m\frac{C_m^{m+2}}{m+2}n^2+b_{m+1}\frac{C_m^{m+1}}{m+2}n$ 成立

因為 S1,S 2,S3,S4,S5,S6 成立 Sm 成立 Sm+1 也成立，證明我們可以

Sk= $\frac{1}{k+1}n^{k+1}+ b_1\frac{C_1^{k+1}}{k+1}n^{k}+ b_2\frac{C_2^{k+1}}{k+1}n^{k-1}+\ldots..+b_{k-1}\frac{C_{k-1}^{k+1}}{k+1}n^2+b_k\frac{C_k^{k+1}}{k+1}n$

求 $1^k+ 2^k+ 3^k+\cdots\cdots+n^k$ 之和!

依據參考資料二,六,七 我們所找到的 b1 b2 b3 b4 b5 b5 b6 ……. 還不完全是白努利數， 因為 b1= $\frac{1}{2}$ 而 Sm= $1^m+ 2^m+ 3^m+\cdots\cdots+n^m$

$$=\frac{1}{m+1}n^{m+1}+ b_1\frac{C_1^{m+1}}{m+1}n^{m}+ b_2\frac{C_2^{m+1}}{m+1}n^{m-1}+\ldots..+b_{m-1}\frac{C_{m-1}^{m+1}}{m+1}n^2+b_m\frac{C_m^{m+1}}{m+1}n$$

而 Sm - $n^m = 0^m+\cdots\cdots+(n-1)^m$

$$=\frac{1}{m+1}n^{m+1}+ (b_1\frac{C_1^{m+1}}{m+1}-1)n^{m}+ b_2\frac{C_2^{m+1}}{m+1}n^{m-1}+\ldots..+b_{m-1}\frac{C_{m-1}^{m+1}}{m+1}n^2+b_m\frac{C_m^{m+1}}{m+1}n$$

$$=\frac{1}{m+1}n^{m+1}+ (\frac{1}{2}-1)n^{m}+ b_2\frac{C_2^{m+1}}{m+1}n^{m-1}+\ldots..+b_{m-1}\frac{C_{m-1}^{m+1}}{m+1}n^2+b_m\frac{C_m^{m+1}}{m+1}n$$

$$=\frac{1}{m+1}n^{m+1}-\frac{1}{2}\frac{C_1^{m+1}}{m+1}n^{m}+ b_2\frac{C_2^{m+1}}{m+1}n^{m-1}+\ldots..+b_{m-1}\frac{C_{m-1}^{m+1}}{m+1}n^2+b_m\frac{C_m^{m+1}}{m+1}n$$

而 - $\frac{1}{2}, b_2, b_3\cdots b_m$正是我們日思夜想的白努利數!

十五、終於發現自己的式子，也能導出通式，並寫出另外三組遞迴求和程式。

最後我們終於以自己的式子導出：

$$S_4 = 1^4 + 2^4 + 3^4 + \cdots\cdots + n^4 = 1\times 1^3 + 2\times 2^3 + 3\times 3^3 + \cdots\cdots + n\times n^3$$

$$= 1^3 + 2^3 + 3^3 + \cdots\cdots + n^3 \to S_3$$

$$+ 2^3 + 3^3 + \cdots\cdots + n^3 \to S_3 - 1^3$$

$$+ 3^3 + \cdots\cdots + n^3 \to S_3 - 1^3 - 2^3$$

$$+ \cdots\cdots\cdots\cdots$$

$$+ n^3 \to S_3 - 1^3 - 2^3 - \cdots\cdots - (n-1)^3$$

$$= nS_3 - \sum_{i=1}^{n-1}\sum_{j=1}^{i} j^3 \qquad \text{(式 肆.二.1)}$$

$=nS3- \frac{\sum_{i=1}^{n-1} i^2*(i+1)^2}{4}$(式 肆. 四.1) 注意只要補上 S3 就可湊成$\sum_{i=1}^{n} i^2 * (i+1)^2$

$=(n+1)S3- \frac{\sum_{i=1}^{n} i^2*(i+1)^2}{4} = (n+1)S3 - \frac{S4+2S3+S2}{4}$把 S4 移項可得 S4

$=\frac{4(n+1)S3-2S3-S2}{5}$(式十五之 1)而由五次求和中我們發現 S1,S2 也能表示 S4,所以 S4

$=(n+1)^2S2- \frac{\sum_{i=1}^{n} i*(i+1)(2i+1)(2i+1)}{6}$一個(2i+1)是$\sum_{i=1}^{n} i^2$來另外個是$(n+1)^2 - n^2$來

$=\frac{6(n+1)^2S2-8S3-5S2-S1}{10}$(式十五之 2)

$=(n+1)^3S1- \frac{\sum_{i=1}^{n} i*(i+1)(3i^2+3i+1)}{2}$

$=\frac{2(n+1)^3S1-6S3-4S2-S1}{5}$(式十五之 3) 經過整理,得出 Sm 如下:

$Sm=\frac{2(n+1)^{m-1}S1-C_2^mSm-1-C_3^mSm-2-\cdots-S1}{m+1}$

$=\frac{6(n+1)^{m-2}S2-(3C_1^{m-2}+2C_2^{m-2})Sm-1-(2C_3^{m-2}+3C_2^{m-2}+C_3^{m-2})Sm-2-\cdots-(3+C_{m-3}^{m-2})S2-S1}{2(m+1)}$(式十五之 4)

$=\frac{4(n+1)^{m-3}S3-(2C_1^{m-3}+C_2^{m-3})Sm-1-(1C_3^{m-3}+2C_2^{m-3}+C_1^{m-3})Sm-2-\cdots-(2+C_{m-2}^{m-3})S3-S2}{m+1}$(式十五之 5)

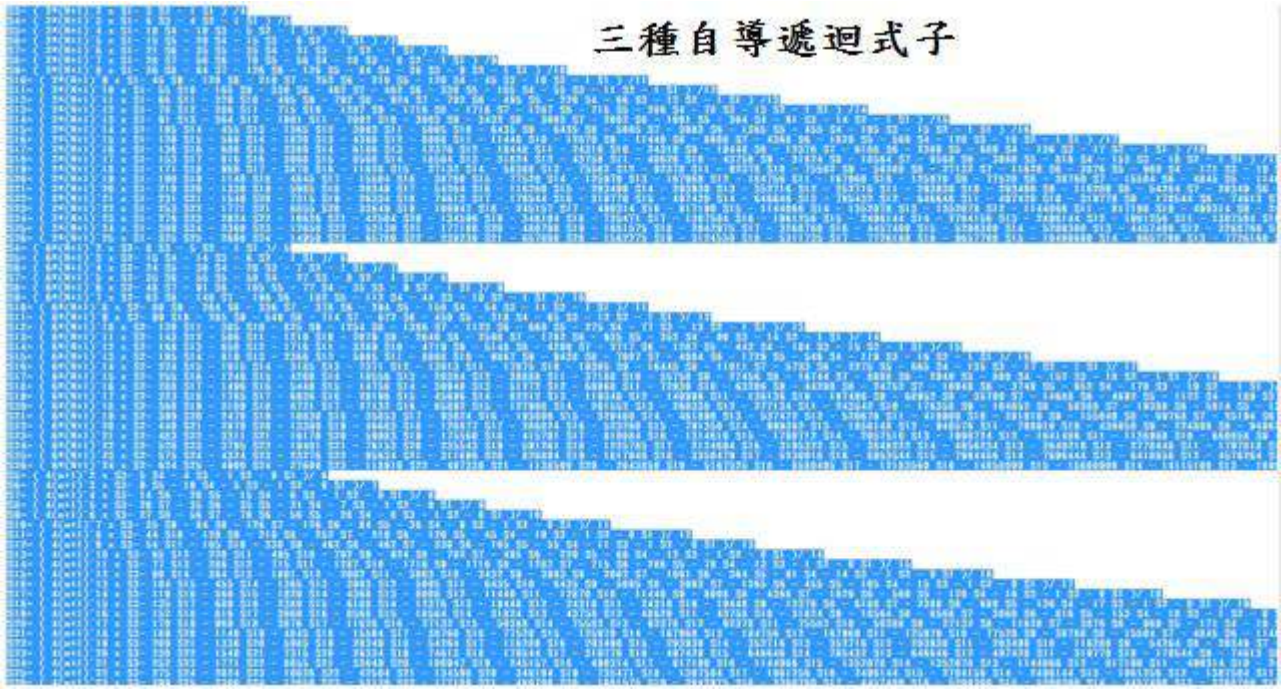
三種方法含意為何，有待更進一步探討！

十六、看文獻八時意外導出的式子

當 k 為大於三之奇數時

$S_{k+1}(n)=(k+1)\int_0^n S_k(x)dx+\int_0^n b_{k+1}dx=(k+1)\int_0^n S_k(x)dx+\sum_{i=2}^{k+2}\frac{(k+1)!}{i!(k+2-i)!}b_{(k+2-i)!}n$ (b_1到b_k為已知)

當 k 是偶數時，$S_{k+1}(n)=(k+1)\int_0^n S_k(x)dx$(文獻八) 故只討論 k 為奇數時的關係。



其遞迴求和程式如下:

```
unsigned long sum2kS1(int odr)
{//遞迴求和程式 by 自己S1法
    unsigned long sum=0;//Visual C++版
    int i;
    if(S[odr]){
    //若S[odr]非0 表示Sodr已算過 直接傳回
    //其值即可 S[odr]在main()要先清為0
        return S[odr];
    }
    if(odr!=1){
        sum= prod[odr-1]*S1*2;
//prod[odr-1]=(N+1)^odr-1在main()要先填好
        for(i=odr-1; i>0; i--){
        //遞迴呼叫Sodr,Sodr-1,Sodr-2....S1
            sum-= P[odr][i]*sum2kS1(i);
        }
        sum/=(odr+1);
//依式子必須除 odr+1
        S[odr]= sum;//以S[odr]存sum
        return sum;//傳回sum
    }
    else{//直接傳值
        S[1]= S1;
        return S1;
    }
}
```

```
unsigned long sum2kS2(int odr)
{//遞迴求和程式 by 自己S2法
    unsigned long sum=0;//Visual C++版
    int i;
    if(S[odr]){
    //若S[odr]非0 表示Sodr已算過 直接傳回
    //其值即可 S[odr]在main()要先清為0
        return S[odr];
    }
    if(odr>2){
        sum= prod[odr-2]*S2*6;
//prod[odr-2]=(N+1)^odr-2在main()要先填好
        for(i=odr-1; i>0; i--){
            sum-= Q[odr][i]*sum2kS2(i);
        //遞迴呼叫Sodr,Sodr-1,Sodr-2....S1
        }
        sum/=((odr+1)*2);
//依式子必須除 odr+1
        S[odr]= sum;//以S[odr]存sum
        return sum;//傳回sum
    }
    else{//直接傳值
        S[1]=S1;
        S[2]=S2;
        return S[odr];
    }
}
```

```
unsigned long sum2kS3(int odr)
{//遞迴求和程式 by 自己S3法
    unsigned long sum=0;//Visual C++版
    int i;
    if(S[odr]){
    //若S[odr]非0 表示Sodr已算過 直接傳回
    //其值即可 S[odr]在main()要先清為0
        return S[odr];
    }
    if(odr>3){
        sum= prod[odr-3]*S3*4;
//prod[odr-3]=(N+1)^odr-3在main()要先填好
        for(i=odr-1; i>0; i--){
            sum-= R[odr][i-1]*sum2kS3(i);
        //遞迴呼叫Sodr,Sodr-1,Sodr-2....S1
        }
        sum/=((odr+1));
//依式子必須除 odr+1
        S[odr]= sum;//以S[odr]存sum
        return sum;//傳回sum
    }
    else{//直接傳值
        if(odr==1)
            S[1]=S1;
        else if (odr==2)
            S[2]=S2;
        else
            S[3]=S3;
        return S[odr];
```

P即賈憲三角形 Q,R為S2,S3相關係數

MyRecursiveS1 MyRecursiveS2 MyRecursiveS3

伍、研究結果與討論

早期程式計算結果輸出(Output1)如下：

```
As N=0x 100000000 M=0x         1
n^6 Method1  (took     225 seconds). sum= xxxxxxxx24924924679e79e79e79e80000000
n^6 Method4  (took     101 seconds). sum= xxxxxxxx24924924679e79e79e79e80000000
n^5 Method1  (took     178 seconds). sum= xxxxxxxx1555555555555554000000000000000
n^5 Method4  (took      86 seconds). sum= xxxxxxxx1555555555555554000000000000000
n^4 Method1  (took     143 seconds). sum= xxxxxxxxb33333338888888888888888000000
n^4 Method2  (took      83 seconds). sum= xxxxxxxxb33333338888888888888888000000
n^4 Method3  (took      50 seconds). sum= xxxxxxxx733333338888888888888888000000
n^4 Method4  (took      33 seconds). sum= xxxxxxxxb33333338888888888888888000000
n^4 Method5  (took      58 seconds). sum= xxxxxxxxb33333338888888888888888000000

As N=0x  10000000 M=0x        10
n^6 Method1  (took     224 seconds). sum= xxxxxxxx924924924921e79e79e79e8000000
n^6 Method4  (took     101 seconds). sum= xxxxxxxx924924924921e79e79e79e8000000
n^5 Method1  (took     178 seconds). sum= xxxxxxxxaaab15555555555554000000000000
n^5 Method4  (took      86 seconds). sum= xxxxxxxxaaab15555555555554000000000000
n^4 Method1  (took     143 seconds). sum= xxxxxxxx3333b33333388888888888888000000
n^4 Method2  (took      83 seconds). sum= xxxxxxxx3333b33333388888888888888000000
n^4 Method3  (took      50 seconds). sum= xxxxxxxxf333b33333388888888888888000000
n^4 Method4  (took      33 seconds). sum= xxxxxxxx3333b33333388888888888888000000
n^4 Method5  (took      58 seconds). sum= xxxxxxxx3333b33333388888888888888000000

As N=0x   1000000 M=0x       100
n^6 Method1  (took     224 seconds). sum= xxxxxxxx49a492492492248f9e79e79e79e800000
n^6 Method4  (took     101 seconds). sum= xxxxxxxx4964924924924 8f9e79e79e79e800000
n^5 Method1  (took     178 seconds). sum= xxxxxxxxab2aaaab155555555554000000000
n^5 Method4  (took      86 seconds). sum= xxxxxxxxab2aaaab155555555554000000000
n^4 Method1  (took     143 seconds). sum= xxxxxxxx00333333b33333888888888888800000
n^4 Method2  (took      83 seconds). sum= xxxxxxxx00333333b33333888888888888800000
n^4 Method3  (took      50 seconds). sum= xxxxxxxx00333333b33333888888888888800000
n^4 Method4  (took      33 seconds). sum= xxxxxxxx00333333b33333888888888888800000
n^4 Method5  (took      58 seconds). sum= xxxxxxxx00333333b33333888888888888800000

As N=0x    100000 M=0x      1000
n^6 Method1  (took     224 seconds). sum= xxxxxxxx2512492c924924924679e79e79e80000
n^6 Method4  (took     101 seconds). sum= xxxxxxxx24924928924924924679e79e79e80000
n^5 Method1  (took     178 seconds). sum= xxxxxxxx002aaab2aaab1555555554000000000
n^5 Method4  (took      87 seconds). sum= xxxxxxxx002aaab2aaab1555555554000000000
n^4 Method1  (took     143 seconds). sum= xxxxxxxx000000033333b333388888888880000
n^4 Method2  (took      83 seconds). sum= xxxxxxxx000000033333b333388888888880000
n^4 Method3  (took      50 seconds). sum= xxxxxxxx000000033333b333388888888880000
n^4 Method4  (took      33 seconds). sum= xxxxxxxx000000033333b333388888888880000
n^4 Method5  (took      58 seconds). sum= xxxxxxxx000000033333b333388888888880000

As N=0x     10000 M=0x     10000
n^6 Method1  (took     221 seconds). sum= xxxxxxxx00002492c925124924921e79e79e8000
n^6 Method4  (took     101 seconds). sum= xxxxxxxxffffe4924924d24924921e79e79e8000
n^5 Method1  (took     183 seconds). sum= xxxxxxxx000000002aab2aab1555555540000000
n^5 Method4  (took      90 seconds). sum= xxxxxxxx000000002aab2aab1555555540000000
n^4 Method1  (took     143 seconds). sum= xxxxxxxx0000000000003333b333888888888000
n^4 Method2  (took      83 seconds). sum= xxxxxxxx0000000000003333b333888888888000
n^4 Method3  (took      50 seconds). sum= xxxxxxxx0000000000003333b333888888888000
n^4 Method4  (took      33 seconds). sum= xxxxxxxx0000000000003333b333888888888000
n^4 Method5  (took      58 seconds). sum= xxxxxxxx0000000000003333b333888888888000

As N=0x      1000 M=0x    100000
n^6 Method1  (took     219 seconds). sum= xxxxxxxx0000000000249a49a49248f9e79e800
n^6 Method4  (took     102 seconds). sum= xxxxxxxxfffffffffffe49249649248f9e79e800
n^5 Method1  (took     185 seconds). sum= xxxxxxxx00000000000002ab2ab155555400000
n^5 Method4  (took      93 seconds). sum= xxxxxxxx00000000000002ab2ab155555400000
n^4 Method1  (took     143 seconds). sum= xxxxxxxx000000000000000333b33888888800
n^4 Method2  (took      83 seconds). sum= xxxxxxxx000000000000000333b33888888800
n^4 Method3  (took      50 seconds). sum= xxxxxxxx000000000000000333b33888888800
n^4 Method4  (took      33 seconds). sum= xxxxxxxx000000000000000333b33888888800
n^4 Method5  (took      58 seconds). sum= xxxxxxxx000000000000000333b33888888800
```

從輸出結果(Output)來看

一、當 N=0x1000 M=0x100000 表示 n=0x1000 求和 0x100000(M)次，舉 Method1(6)為例： 求 $1^6 + 2^6 + 3^6 ...+(2^{12})^6$ 0x100000 次需時 219 秒。

二、使用 N 表示 n，M 表示 loop 次數，是為了讓結果更嚴謹可靠。

三、只要 N*M 值一樣，所需時間大致相同。

四、方法三在 for loop 裏巧妙計算$n^2 * (n+1)^2$ 之方式求和，在和溢位(超過 2^{128})時會有和其他方法不一致的情況，但在 N<=0x100000 則 OK。其他方法無此現象發生。

五、和溢位超過2^{128}，其後面 128bit 不受影響，且不影響執行速度。

六、前述 xxxxxxxx 128 bit 進位 Don’ t care 變通辦法，確實不影響結果，卻可大幅減輕程式處理極大數字運算之負擔，也大大降低程式複雜度。

七、方法一以 n*n*n*n*n*n 之方式極為耗時。

八、方法二以 n*S3-n*n*n 之方式求和確實降低耗時，惟 n*n*n 仍有改善空間。

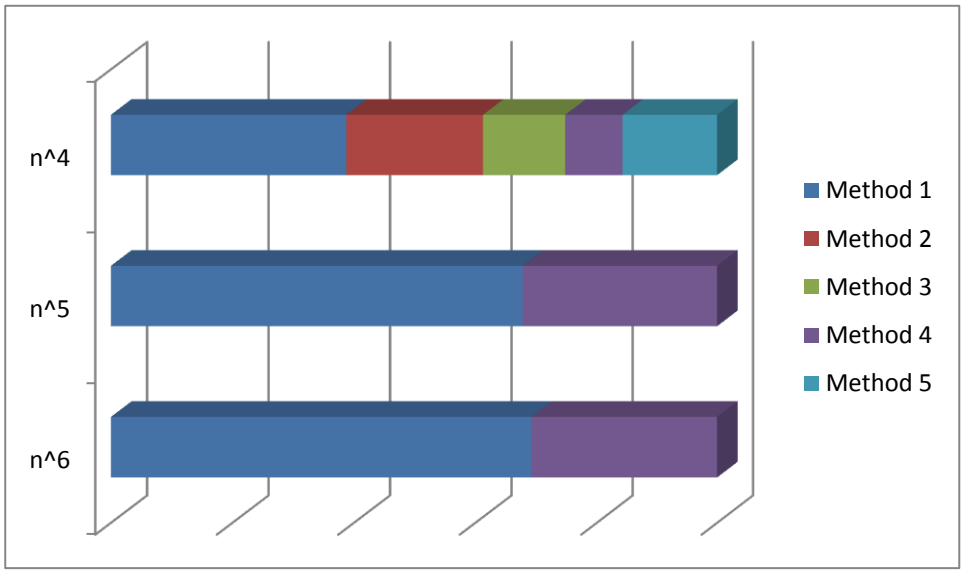
九、方法四為四月之前最快之辦法。

十、乘法固然耗時，加法太多也很花時間，舉 Method5(4)之 for loop 內之

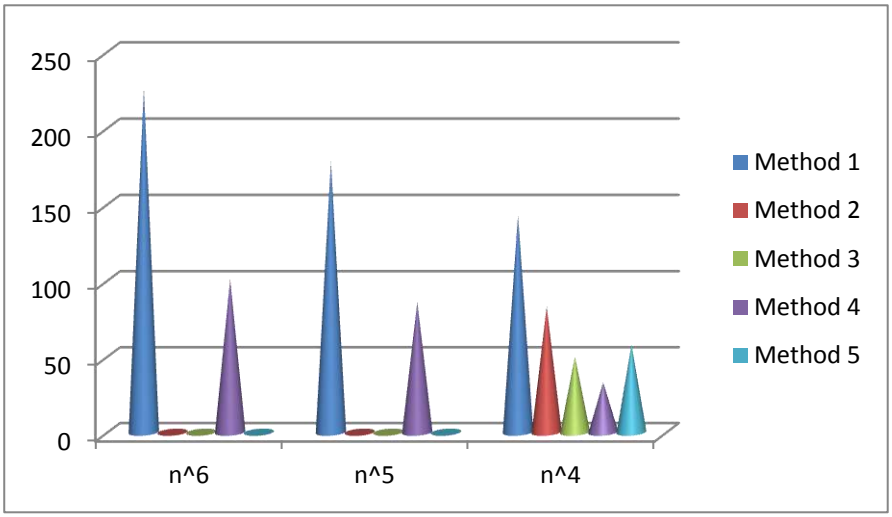
nnn+= 3*(nn +n)+1;為例，若改成

nnn+= nn+nn+nn+n+n +n+1 速度馬上慢到 70 幾秒!

十一、各種方法所需時間之比較長條示意圖：



十二、各種方法所需時間之比較柱狀示意圖：



單位：秒

十三、程式 Output 對照表：

N M	n^6方法1	n^6方法4	n^5方法1	n^5方法4	n^4方法1	n^4方法2	n^4方法3	n^4方法4	n^4方法5
0x100000000 0x1	225	101	178	86	143	83	50	33	58
0x10000000 0x10	224	101	178	86	143	83	50	33	58
0x1000000 0x100	224	101	178	86	143	83	50	33	58
0x100000 0x1000	224	101	178	87	143	83	50	33	58
0x10000 0x10000	221	102	183	90	143	83	50	33	58
0x1000 0x100000	219	101	185	93	143	83	50	33	58
	表示Sum與其他方法不一致								
	表示sum溢位 i.e. sum > 0xffffffffffffffffffffffffffffffff								

單位：秒

十四、遞迴程式 Output

```
===========as n= 0000000000001000  run for (M)= 0000000000200000 times
S10=xxxxxxxx00800d555545555556555554d5555568b took       7 seconds by Recursive Method
S10=xxxxxxxx00800d555545555556555554d5555568b took     841 seconds by Method1

S09=xxxxxxxx0019a19a599998e66666e666666400000 took       6 seconds by Recursive Method
S09=xxxxxxxx0019a19a599998e66666e666666400000 took     751 seconds by Method1

S08=xxxxxxxx000001c79c7c71c714fa4fa888888800 took       5 seconds by Recursive Method
S08=xxxxxxxx000001c79c7c71c714fa4fa888888800 took     610 seconds by Method1

S07=xxxxxxxx0000000020080095555550aaaaac00000 took       3 seconds by Recursive Method
S07=xxxxxxxx0000000020080095555550aaaaac00000 took     546 seconds by Method1
```

速度差距極為懸殊, 遞迴程式不產生 Truncation Error

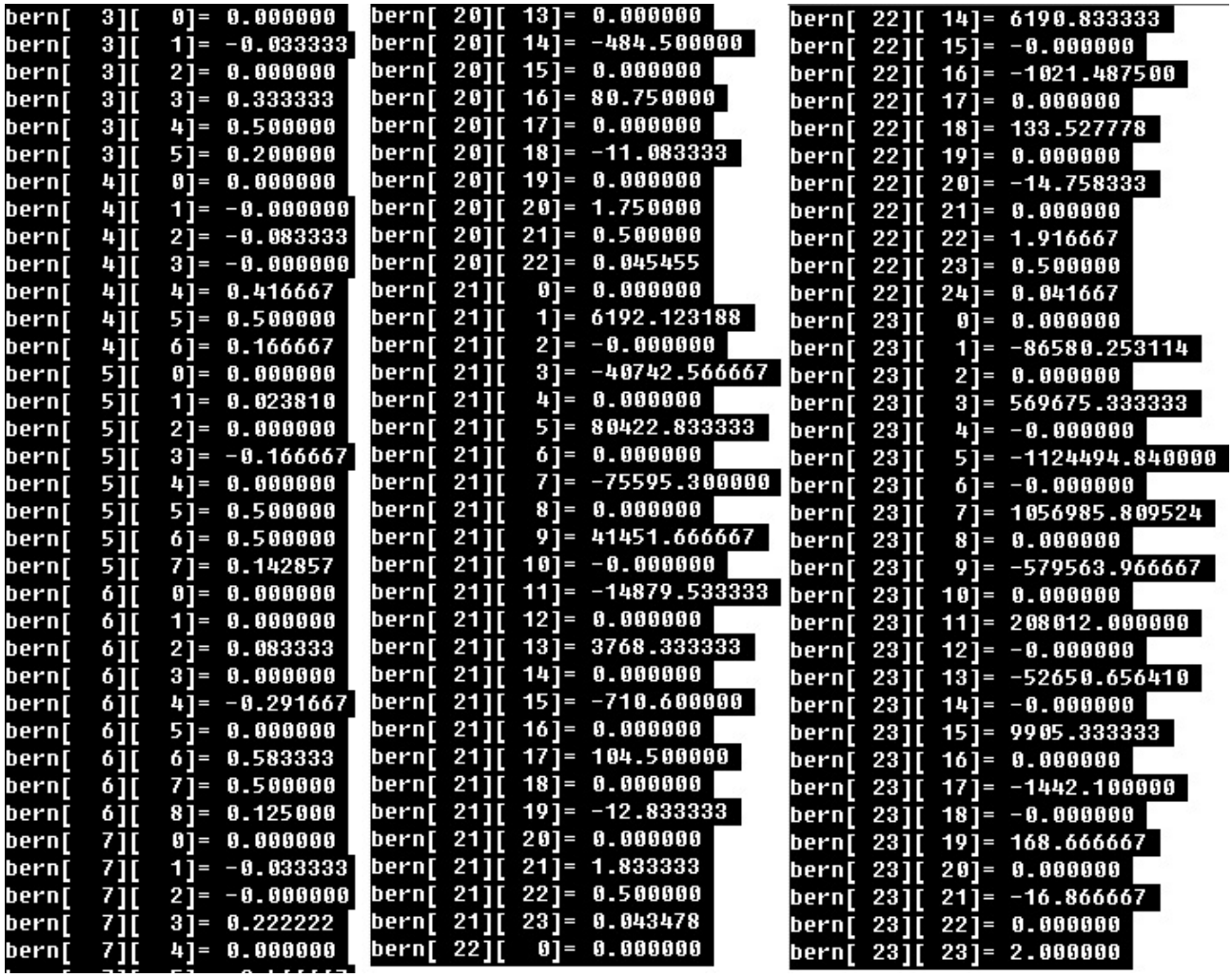
白努利數(式)並不是萬靈丹!!! 因為用在電腦運算上只要數字一大,便會產生極為嚴重的 Truncation Errors。 遞迴程式之所以能避開這個問題應該要從(式肆. 十一.6)來看$(n+1)^{m+1}-1-n$並沒有分數,而所有S_k都從S_1計算出,所以即使最後除 m+1 也不至於產生 Truncation Errors。

```
============as n= 0000000000001000  run for (M)= 0000000000200000 times

S06=xxxxxxxx00000000000249a49a49248f9e79e800 took       3 seconds by Recursive Method
S06=xxxxxxxx00000000000249a49a49248f9e79e800 took     478 seconds by Method1
S06=xxxxxxxx00000000000249a49a49248f9e79e800 took     200 seconds by Method4
S06=xxxxxxxx00000000000249a49a49248f9e79e800 took       1 seconds by Coefficient Method1
S06=xxxxxxxx00000000000249a473f0132060000000 took       1 seconds by Coefficient Method2

S05=xxxxxxxx000000000000002ab2ab155555400000 took       3 seconds by Recursive Method
S05=xxxxxxxx000000000000002ab2ab155555400000 took     342 seconds by Method1
S05=xxxxxxxx000000000000002ab2ab155555400000 took     163 seconds by Method4
S05=xxxxxxxx000000000000002ab2ab155554d55554 took       1 seconds by Coefficient Method1
S05=xxxxxxxx000000000000002ab2b0acfd3aaa0000 took       1 seconds by Coefficient Method2

S04=xxxxxxxx0000000000000000333b33888888800 took       3 seconds by Recursive Method
S04=xxxxxxxx0000000000000000333b33888888800 took     242 seconds by Method1
S04=xxxxxxxx0000000000000000333b33888888800 took      70 seconds by Method4
S04=xxxxxxxx0000000000000000332b33888888800 took       1 seconds by Coefficient Method1
S04=xxxxxxxx0000000000000000332b33888882ea0 took       1 seconds by Coefficient Method2

============as n= 0000000000010000  run for (M)= 0000000000200000 times

S06=xxxxxxxx00002492c925124924921e79e79e8000 took       3 seconds by Recursive Method
S06=xxxxxxxx00002492c925124924921e79e79e8000 took    7646 seconds by Method1
S06=xxxxxxxx00002492c925124924921e79e79e8000 took    3175 seconds by Method4
S06=xxxxxxxx00002492c925124924921e79e79e8000 took       1 seconds by Coefficient Method1
S06=xxxxxxxx00002492c6bf813230000000000000000 took       1 seconds by Coefficient Method2

S05=xxxxxxxx000000002aab2aab1555555540000000 took       4 seconds by Recursive Method
S05=xxxxxxxx000000002aab2aab1555555540000000 took    5462 seconds by Method1
S05=xxxxxxxx000000002aab2aab1555555540000000 took    2603 seconds by Method4
S05=xxxxxxxx000000002aab2aab15555554d5555554 took       1 seconds by Coefficient Method
S05=xxxxxxxx000000002aab3042bd35980000000000 took       1 seconds by Coefficient Method2

S04=xxxxxxxx0000000000003333b333888888888000 took       3 seconds by Recursive Method
S04=xxxxxxxx0000000000003333b333888888888000 took    3923 seconds by Method1
S04=xxxxxxxx0000000000003333b333888888888000 took    1117 seconds by Method4
S04=xxxxxxxx0000000000003332b333888888888000 took       1 seconds by Coefficient Method1
S04=xxxxxxxx0000000000003332b333888884000000 took       1 seconds by Coefficient Method2
```

Coefficient2 Method 產生 Truncation Error 極為嚴重，Coefficient1 Method 使用白努利數(分數方式)計算，Truncation Error 仍然嚴重。

十五、係數程式 Output

理論上本程式可求任何次方之所有係數,但在實務上到了數字一大就有爆掉的情況發生,當然要解決這問題,自然有辦法,但這已不是本研究所該注意之焦點了。



陸、結論

一、本研究確實找到有效改善效率，又簡單又快，連弟弟都看得懂，又能以程式驗證之方法求連續整數冪次和。

二、降冪對於加速電腦運算具至為重要顯著的效果,特別是各種沒有通解的狀況。從方法一 n*n*n*n 到方法二降冪為 n*n*n，求$1^4 + 2^4 + 3^4$ ...+$(2^{32})^4$需時從 143 秒降到 83 秒。方法四更以 for loop 巧妙快速地找到$\mathrm{sum2n}(\sum_{j=1}^{n} j)$,再以$\mathrm{sum2n}^2$取代n*n*n，再次降冪，速度從 83 秒再降到 33 秒。少乘法，code 簡化是程式有效加快之基本原則。

三、遞迴求和程式又快速又正確，沒有 Truncation Error，是以電腦求連續整數冪次和之最有效解決方案。

四、雖然作者本無能力證明或導出白努利級數公式，但最後發現 Telescope sum 方法，參考資料五看懂之後發現，即使無法馬上證明或找出白努利數，至少可用程式去找通式，剛開始只能求算係數。程式發展過程中卻發現其遞迴特性於是才趕緊寫出遞迴程式。程式根據數學式而生，而程式撰寫有助於理解，往往寫好程式後返過來以程式思維修改式子，數度來回之後，才能得到好成果。Method4 是即最佳範例!

五、寫完遞迴程式後，發現白努利數就埋在裡面，而且也以遞迴的方式存在，一個不小心的意外，讓我真正找到他！不禁深深受到感動! 數學真美! 程式也是！

六、透過程式確實真能找出各種係數，白努利 p 級數確實是個有趣的題目！後面還有很多可做(例如 Sm 對 n 微分之後除以 m,去常數項,剛好等於 Sm-1)，

七、一定還有很多未為人知的美麗秩序，依舊靜靜地、悠悠地躺在這些神秘的式子裏，願藉更多的努力，更廣泛的閱讀及各類 OCW 收視，以及思考鍛鍊，能讓我更上一層樓，看得更深，走得更久遠！

八、從科展學到甚麼？不敢高調講作學問的方法，但決不放棄的卻是我充分體會到的！筆記、紙筆不離手，有疑問就要不斷想，不斷思考目前為止是突破困難的一貫妙方！保持樂觀、適當休息第二。

九、雖然有很多相關文獻我依然看不懂，但我很確定這題目雖然已經很多人發表過，但能以這麼簡單易懂的方法導出者，應該很少！

十、目前全世界都在積極鼓勵青少年學習 Coding，真是非常有道理的！台大陳凱風教授即是最好榜樣，我就是看完 134 期科學人雜誌後拿給爸爸看，從此後他就開始教我 C，事實上他是從不管我學校功課的，但卻對我實施 Coding 的魔鬼訓練！而我自學微積分，他是從不鼓勵，也不幫忙的！他只是一再告訴我程式語言是人人該學的第三外國語！

十一、CAP(Computer Aided Proof)是我自創名詞，也不知道是否有別人這麼做，但我的 CAP 程式真的好處多多，還讓我意外找出b_k關係式！當然還有很多改進空間，請拭目以待！

十二、Zeta 函數(負數次方)也可以本研究之數列三角排列，值得想一想！Telescoping Sum 還能解其他問題，有空也值得試一試！

柒、參考資料及其他

一、數列與級數，均一教育平台，http：//www.junyiacademy.org/

二、白努利數楊輝三角形，維基百科 http：//en.wikipedia.org/

三、Bernoulli Numbers，http：//www.youtube.com/watch?v=yGpkB2OoQjk

四、Bernoulli Numbers，http：//www.youtube.com/watch?v=kc-dkOaRf9I

五、Sum of n integers to the power of 4: simple Proof http：//www.youtube.com/watch?v=iR9BTx4tPN8

六、尋尋「冪」「冪」—連續整數冪次和公式解之簡潔表示法，李根瑞，中華民國第51屆中小學科學展覽會，作品說明書。

七、Bernoulli Numbers in PASCAL Triangle，http//www.youtube.com/watch?v=xYuXW9xj2Sc

八、葉東進，一個關於冪次和的定理，*數學傳播季刊*，38(1)，64-71。

九、葉東進，冪次和表為n之多項式的係數律則，*數學傳播季刊*，38(2)，82-87。

十、蘇益弘、胡豐榮、許天維，從連續整數冪次和公式引發之擴充想法，*數學傳播季刊*，29(2)。

十一、吳松霖、李國寧、胡豐榮、許天維，連續整數冪次和公式之指數生成函數探討，*數學傳播季刊*，31(3)，13-16。

十二、陳錦初，多項式根冪次和的新解法，*數學傳播季刊*，23(3)。

十三、何景國，談談求$\sum_{i=1}^{n} i^k$(k=1,2,3)的幾種方法，*數學傳播季刊*，6(4)。

十四、余文卿，一些發散級數的求和法，*數學傳播季刊*，23(4)。

十五、李政豐，連續整數冪次和公式的另類思考，*數學傳播季刊*，26(2)。

十六、Accessing Bernoulli-Numbers by Matrix-OperationsGottfried Helms 8'2009 (3'2006)Version 2.3.2 (small corrections)

十七、Faulhaber’ s formula http://en.wikipedia.org/wiki/Faulhaber’ {}s%20formula?oldid=653008425

十八、溫瑞烘、洪正聰，*資料結構-使用C語言*，全賢圖書。

十九、師明睿譯，*微積分之屠龍寶刀*，天下文化。

二十、師明睿譯，*微積分之倚天寶劍*，天下文化。

二一、陳可崗譯，*質數魔力*，天下文化。

二二、易富國教授，99 普通物理甲，*台大開放式課程 OCW*，前六講。

【評語】030427

作者利用電腦程式編寫求 k 次方和之公式。k 次方和之公式為數學習知之結果，亦有諸多演算法可避開電腦大數運算的誤差問題。作者試圖從頭開始發展自己之想法，精神可嘉亦值得鼓勵。然建議可加深對數學理論之理解，以對問題之背景與發展有更深刻之理解。