目次

vskip 1cm 別途、prog というファイルの １) ー８) に登場するプログラムが書いてある。９) の内容に対しては、sun.dat 、sunprog というファイルがある。

０．計算機とプログラムの概観

ここでは、計算機を用いて数値計算をする事を想定して、プログラムの概念を説明する。

計算機内部では、何らかの方法で文字や数値を表現している。例えば２進数で１６桁を一つの組合せ記号と考えて、異なる組合せ記号に異なる文字や数値を対応させる。この対応規則は、計算機の内部表現と呼ばれる事がある。ある場合には内部表現を数値と仮定し、別の場合には文字と解釈する。

特に数値表現に限定すると、先ず自然数という概念を最初に思い付く。これは１から始まり、自然数に１を加える事により新しい自然数が作られる。従って自然数は無限に存在する。一方、組み合わせ表現は、利用可能な資源は有限だから、自然数の内の極めて限られた部分しか計算機では取り扱えない。

０と負号を表現出来る様にすると、整数が表現出来る様になる。この段階を基本とする。整数の内どれだけを実際の計算の視野に入れるかは、計算機のハード・ソフトウエアーの設計に依存する。

整数の範囲では、和と差の計算は可能であるが商を求める事は出来ない。そこで、実数を表現する必要がある。整数部と小数点と小数部を組み合わせて実数を表現する事にしておこう。この場合、整数部は (符号付の) 整数である。小数点は、常識的な小数点を内部的に表現する。小数部は、正の整数を想定しておけば良い。実際には、実数の表現には幾らかの工夫があるが、それは計算機の設計をする人が知っていれば良い事であるとしておこう。

計算機の数値を表現する資源は有限であるから、表現出来る整数や実数にも制限がある。例えば、この原稿を書いている PC と GNU を支持する人達が書いた f ７７というプログラムを処理するプログラムを利用して階乗の計算をしてみると以下の様になった。

n	1	2	3	4	5
n!	1	2	6	24	120
n	6	7	8	9	10
n!	720	5040	40320	362880	3628800
n	11	12	13	14	15
n!	39916800	479001600	1932053504	1278945280	2004310016
n	16	17			
n!	2004189184	−288522240			

$n = 13$ で既に計算が破綻している。手元の電卓で計算すると、$13! = 6227020800$ である。即ち、(ある条件では) 関数電卓よりも電子計算機の方が頼りない計算しか出来ない。

$n = 17$ では正であるべき計算結果が負になっている。ここまで来ると、計算機が正しい計算をしていない事がはっきりするだろう。計算結果が負になる理由は、計算機内部で負の整数を如何に表現しているかという知識が無いと理解できないだろう。

次に、実数の例を二つ挙げておこう。

１) 0.1 を１００回加えてみよう。

$$P(n) = \sum_{i=1}^{n} 0.1$$

途中経過と共に表にしてみた。

n	10	20	30	40	50
P(n)	1.0000001192	2.0000002384	2.9999992847	3.9999983311	4.9999976158
n	60	70	80	90	100
P(n)	5.9999966621	6.9999957085	7.9999947548	8.9999980927	10.0000019073

計算結果は、小数点以下７桁目に誤差を持っている。この程度の精度の計算を標準的な計算だと仮定して、計算系が設計されているという事らしい。

実は、この例では 0.1 と指定したのに、 内部では ０．１０００００００１５ が使用されていた。

もう一つの例として、調和級数の部分和を二つの方法で計算してみよう。

$$S(n) = 1 + 1/2 + 1/3 + \cdots 1/n$$

$$T(n) = 1/n + 1/(n-1) + \cdots 1/2 + 1$$

n	10^2	10^3	10^4	10^5	10^6	10^7
S(n)	5.187378	7.485478	9.787613	12.090851	14.357358	15.403683
T(n)	5.187377	7.485472	9.787604	12.090152	14.392652	16.686031
倍精度計算	5.187378	7.485471	9.787606	12.090146	14.392727	16.695311
T(n)−S(n)	−0.000001	−0.000007	−0.000009	−0.000699	0.035294	1.282349

$S(n)$ では、$n = 10^7$ に対して約８％の誤差が発生している。この例でも、１回の計算に対して 10^{-7} の誤差が発生し、それらの誤差が全部相加的に寄与すると、10^7 回繰り返すと約１という大きさの誤差が発生する事になり、表の誤差と大体一致している。一方 $T(n)$ では、$n = 10^7$ に対して 9×10^{-3} と、約１/１００に誤差が小さくなっている。

この例は、ある種の計算機では計算精度が計算手順に依存するという側面を示す為に挙げてみた。実数の内部表現として、固定小数点表現の計算系では $S(n)$ の $T(n)$ 差は出ない可能性があるが、固定小数表現はプログラムを書くのが面倒だから、ほとんど廃れてしまっている。

浮動小数点表現を用いた計算系では、計算結果が大きくなる方向に計算を進めると、逆方向に計算をする場合よりも誤差が小さくなる。これは、数値計算する場合には必ず知っておかねばならない知識である。興味があれば、この理由を考えてみよ。

大量の数値を一連の手続きで処理する時、計算機を用いるならば、全データを一気に処理するのが楽である。一方、紙と鉛筆を用いて計算する場合には、全データを１０個なり２０

個なりの小さな部分に分割し、夫々の部分でのデータ処理をして中間データを作り、全部の中間データが集まると、次に中間データの処理をするだろう。全データを一気に処理するよりも、後者の方がミスが入り込む確率は少ないと想像される。計算機を用いて処理する場合に、高速性に気をとられて、有効数字不足に起因する正確性は忘れられる場合がある。

平均値を計算する場合に、移動平均という概念を用いると言うのは忘れてはならない、生活の知恵である。

実数表現に興味があれば、固定小数点方式、浮動小数点方式、浜田の数値表現等を調べてみよ。計算機の内部表現と言うと、２進数という発想をしがちだが、１６進表現を用いている計算機もあるし、BCD(binary coded decimal) と言って２進化１０進数表現を用いる場合もある。時には、rad50 と呼ばれる５０進で文字を表現したメーカーもある。

計算機の内部表現を直接外部記憶装置に書き出す場合がある。これは、記憶装置の効率的利用に直結する。この内部表現は、書き出し方を含めて、企業秘密になっている場合がある。これは、広い意味での盗聴やデータ漏洩の確率を減らす事につながる。

自分が使用する環境を事前に調べておかないと、悔しい思いをする事がある。

ある時、国産の計算機で正しく働くプログラムを持ってアメリカへ行った。日本では単精度と呼ばれる数値表現の範囲で正しい計算が出来たが、アメリカでは支離滅裂な値が出て来た。そこで、倍精度と呼ばれる精度の高い計算を指示してプログラムを走らせたところ、日本での計算結果を再現した。これは、plag compatible と呼ばれる計算機での例である。電気的には差し替え可能なハードウエアーであるのに、ソフトウエアー的には大違いである。

いわゆる計算機には、以下の機能が備わっている。

外部とのデータのやりとり
加減乗除の計算
条件判定

この３つの機能を旨く組み合わせて、精度と計算速度の意味で効率良く目的を達成する事を目指す。

実数は、１元数とも呼ばれる。この拡張として２元数や４元数もあり得る。２元数は特に複素数という名前で呼ばれる事が多い。複素数は、代数学の基本定理との関係で重要である。逆に言えば、代数方程式を解くならば、複素数迄を視野に入れた計算系を基本的な計算系として取り上げる必要がある。

これからの話では、複素数までを自由に使用する事を想定する。但し、始めの内は実数のみを取り扱う。

計算機では、先に述べたように内部表現を数値として解釈する時、整数と実数とを厳格に区別するという立場をとる。

簡単な計算過程

０) 計算機は、複数種の記憶装置を備えている。計算処理装置に近い部分から並べると、以下の様になる. 近い部分ほど, 高速動作をする傾向にある.

０. １) レジスター：これは記憶装置に含めない人もいるだろうが, ここに書き込まれたデータが通常は処理の対象となる.

０. ２) キャッシュメモリー： レジスターの数は多くはないので, よく使うデータはここに書き込んでおく

０. ３) 主記憶： 通常の記憶装置であり, ここに多くのデータが置かれる. 主記憶装置の個別の部分を識別する番号を, 番地という名前で呼ぶ場合が多い.

０. ４) 補助記憶装置： しょっちゅう使用する訳ではないが, 必要に応じて取り出したいデータを書き込んでおく.

実際に計算機が動作している時には, これらの記憶装置の間でのデータのやりとりが, 頻繁に行われている. 特に, 複数の利用者が, 複数のプログラムを走らせている時には, 補助記憶装置の耐久テストをしているのではないかと疑われる程にデータ転送が頻繁に行われている.

記憶装置内部の最小データ単位には番号が割り振られていて, その番号によりデータが識別される.

複数の最小データ単位をまとめてもう少し大きなデータ単位とする. 例えば, ビット, バイト, ワード, パケット, トラック, シリンダー等の呼び名が付けられる. アトムという単位もある。複数のデータ集合に名前を付けて, これを一まとまりのデータブロックとして利用する場合も多い.

これにも、物理的な記録単位と論理的な記録単位がある。

１) 計算をする前に, 必要な計算手順を記述したデータ (これをプログラムと呼ぶ) と計算対象となるデータとを主記憶装置に書き込む必要がある. プログラムやデータの全体が一度に主記憶装置にある場合もあるし, 一部だけが主記憶装置に書き込まれている場合もある. 従って, データを書き込んだり読みだしたりし, そのデータ番地を正しく管理する必要がある.

停電した場合のデータ保護も非常に大切な仕事である.

利用者の, 利用開始と終了を正しく認識し, 便利な機能を提供する必要がある.

これらの機能を提供するプログラムは Operating System (OS と略称される) の仕事である. 但し, 計算機の起動に際し, OS を計算機の主記憶の正しい位置に書き込み, この書き込んだプログラムを動作させる (通常, 走らせるという語が使われる) というプログラムも必要である. IPL, ROM モニター等の名前で呼ばれる.

OS の管理下で, 補助記憶装置にデータを書き込む手段として, editor と呼ばれるプログラムを使用する. 他の手段としては, 既にある種の媒体 (例えば MT, CD, DVD 等) に書き込まれたデータを別の補助記憶装置に読み込む場合もある.

これからは, 主に editor を通してデータは書き込まれる事を想定する.

２) 計算機の主記憶に書き込まれたプログラムは, 一つずつ読みだし, 命令を解釈し, 実行される.

この場合, 実行されるのは instruction set と呼ばれる命令に限られる.

instruction set は, 個別の計算機 (cpu, central processing unit と呼ばれる) 毎に異なる. どのような instruction set を用意するのが良いかは, 設計者の腕の見せどころである.

この instruction set に含まれる命令は, 複数個組み合わせてマクロ命令とする方が, 利用者には便利である場合が多い.

利用者が和を計算したいとし, A 番地にあるデータと B 番地にあるデータを加えて C 番地に書き込めというプログラムを書くのは不便である. 利用者が直接, instruction set を利用する事は稀である. そこで, instruction set と利用者の希望する動作の間に, 複数のクッションを設ける.

又, 利用者の希望する動作の種類に合わせた instruction set の纏め方も色々とあり得る. 万能の纏め方というアイデアもありうるが, 必ずしも成功しているとは言いがたい. この利用者向けの計算機利用の為の, プログラム記述様式を規定する各種の仕様に対し, プログラム言語という概念が導入されている.

数値計算用に開発されたプログラム言語の代表として,FORTRAN (formula translator) がある. FORTRAN は論理性に欠けるところがあるという理由で ALGOL という言語が開発されたが, 普及していないと思う. 簡易言語としては, BASIC という言語も一時ブームになった事がある. 情報関係者には C という言語も普及しているが, これもブームは去った様である. 個人的には, C は数値計算には不適な言語であると思う. 例えば, 配列の利用には決定的に不利である.

３) 計算をするとは, 次のステップを踏む事であるとする.

３. １) プログラム言語を用いてプログラムを書く. このプログラムは原始プログラムと呼ばれる事もある.

３. ２) コンパイラーというプログラムを用いて, 原始プログラムを中間段階のプログラムに変換する. 変換結果は, 目的プログラムと呼ばれる場合もあるが, これは利用者からみると, 変な名前である. 中間結果は, 補助記憶装置に書き込まれる場合が多い.

計算機からみると, 利用者が書いたプログラムを解きほぐし、instruction set に変換する作業をするコンパイラーも一つのプログラムである.

コンパイラーは, 原始プログラムを利用者が意図した順番とは異なる順番に計算を実行するように, プログラムを内部で変更する場合がある. ある種の場合には, この過程を最適化と呼ぶ. 最適化は, 文字通りの意味ではない事に注意しよう.

３. ３) ある種の基本動作, 例えば三角関数等の初等関数の計算, はコンパイラーが既に知っていて, 利用者はこれを利用する事が出来る.

このコンパイラーが提供する機能を書いた中間出力は事前に補助記憶装置のどこかに書き込まれていて, ライブラリーと呼ばれる.

利用者の目的プログラムにライブラリープログラムを組み込む作業をリンクと呼ぶ.

この作業をするのは linker と呼ばれるプログラムである. 使用者は, linker を起動して 実行可能なプログラムをつくり, これを補助記憶装置に書き込む.

ライブラリーは,OS が提供するもの以外に利用者が用意したものを使う事も可能である. 従って, 個人用のライブラリーを充実する事が可能である.

ライブラリーを編集の対象とする場合もあり, linkage editor という概念が登場する.

大きなライブラリーは, プログラムの実行時に組み込む場合もあり, dynamic link と呼ばれる.

3．4) 補助記憶装置に書き出された実行可能プログラムは, 利用者が呼び出すと, OS 配下の loader プログラムにより, 主記憶装置に書き込まれ, このプログラムの先頭番地から順番に実行される.

プログラム内部の判断基準により, 実行順序が変更される場合もある. この過程を jump と呼ぶ. jump には, 条件判断によるものと, 無条件に jump する場合がある. 後者は, ここまでの準備が出来れば, 三角関数を計算せよという様な場合である. 条件判断による jump の例としては、変数の絶対値 $|x| \leq \pi/4$ ならば $\sin(x)$ を計算し、$\pi/4 < |x| \leq 3\pi/4$ ならば $\cos(x)$ を利用するといったり、$x = 10^{100}$ に対する三角関数を計算せよという無茶な命令を拒む為に行ったりする。

主記憶には, 変数と呼ばれる部分はデータ部に, 実行の手順を書いた部分はプログラム部にという具合に, 分けて書き込まれる.

時として, プログラムが書いてあると勘違いして, データ部のデータを instruction set だと解釈して, 実行しようとする場合がある. 当然, 計算機は誤動作する.

これらのコンパイル・リンク・ロードという一連の作業は一纏めにして、一見したところ、ひとつのプログラムの様に動作させる場合の方が多いとおもう。作業分割のやり方の細かい部分は、OS に依存する。

プログラムを実行する場合、コンパイラーを経由して実行可能プログラムに変換するほうが、実行効率は良くなる。しかし、誤解や操作ミスがあるとプログラム開発にはそれ相当の時間がかかる。そこで、実行効率は低下してもよいから、結果を早く欲しいならば、プログラミングが簡単な方が良いという選択もあり得る。この指導原理の下に開発された言語類を interpreter 言語と呼び、compiler 言語と区別する場合もある。

利用者が複数のプログラムを組み合わせて、目的となる機能を実現する為に、計算機の提供する機能を細分化 (機能分け) するのが良い場合がある。この場合、複数の機能を順番に呼び出す指令手順を job script と呼ぶ人もいる。これは、command interpreter に対する指令である。

これからは、コンパイラー言語の利用を想定した、数値計算を実際に行う場合の know how を提供する事に主眼を置いた話題提供となる。

数値計算においては、結果の信頼性という事は非常に大切な概念である。計算速度よりも、確実に結果を得るという意味で、平野の方法と呼ばれる指導原理があるらしい。これ

を戸山隼人著 ” マトリックスの数値計算 ” オーム社 (昭和５６)pp241 より引用しておこう。高次方程式に関連して書かれているので、一般論としては必ずしも該当しない表現もあるが、プログラムを書き始める時及び、トラブルに出会った時に思い起こして読み返してみる事を強く勧める。

1 **根の近傍に原点を移すと数値計算が容易になる。(桁落ちの危険が少なくなり、計算の見通しが良くなる。)**
2 **” 無視出来るぐらい小さい項 ” は積極的に０と置き、(勿論あとで復活させるが) ” 特に支配的な項 ” を重視する。これにより計算が著しく簡単化される。**
3 **他の大部分の解法では ” 特に支配的な項 ” は低次の項 (大抵は１次項、例えばニュートン法がそうである) であると考えられているが、実際の数値計算の経験では、もっと高次の項が支配的である場合が少なくない。(その場合に従来の１次項しか考えない解法は脱線した)。このため高次、低次にかかわらず、実際の影響力を調べて最も支配的な項を選ぶ。**
4 **検算、修正を重視し慎重にコトを運ぶ。**
5 **計算のどこで誤差が混入し、どこでそれが拡大されるかを考慮し、拡大させる危険のある部分は特別な手当をする。**

プログラムは、以下の様な要素より構成される。

* これ以降がプログラムであるという事を宣言する部分。必要ならば、ここでプログラムに名前を付ける。
* 使用する変数とその属性を宣言する。例えば、(仮想的な) 主記憶上に、実数型の変数をとり、これに A という名前をつける。別の変数は整数であり、５０個を纏めて、I という名前をつけ、前から順番に１から５０迄の番号を付ける。実際にこの変数を利用する場合には、I(１５) という様な書き方をする。又は、２次元的に名前をつけて、I(５、７) という風に書く事も可能だろう。この場合には、第１添字と第２添字の取り得る範囲をどこかでコンパイラーに教える必要がある。
  二つの実数を一組みにして、前半を実部、後半を虚部とする複素数であるとしても良い。この場合には、複素数であるという宣言をして、コンパイラーに教える必要がある。
  別の変数には、文字を入れたり、論理変数として利用する為の宣言もありうる。
  どのような属性の変数を用意するかは、対象とする問題又は作業の種類に依存する。
  数値計算の場合には、精度も大切な概念である。そこで、単精度、倍精度といった変数の属性を修飾する宣言も必要な場合がある。
* 変数に値を代入する必要がある。外部からデータを読み込んだり、変数に値を代入したりする。この操作を変数の値を確定 (定義) するという場合がある。
  値が確定していない場合には、過去に使用された値がゴミとして残っている場合もあ

るし、親切なコンパイラーだと 0 という値を代入していてくれる場合もある。このような作業は、変数の初期化とも呼ばれ、コンパイラーがサービスしてくれる程度は、変数の属性にも依存する。

* 定義された変数値を用いて、ある主の操作をする。例えば、和や差を計算する。
この部分が、これからの話題提供の主要部である。
* 計算された値を必要な補助記憶装置や外部装置に書き出す。
* プログラムの終了を宣言する。

ある種の操作単位、例えば指数関数の計算は、これを一つの塊とし、副プログラムとする。副プログラムは、上に書いた場合とほぼ同じ構成であるが、呼び出しプログラムから、計算に必要なパラメータ (数字) を受取り、計算後、呼び出しプログラムに結果を通知する様な構成にする事ができる。

多くのハードウエアーには、subroutine jump という機構が組み込まれていて、この機構を利用してデータのやりとりと、プログラムの流れの制御を行っている。

勿論、副プログラムで計算に必要なデータを外部から読み込んだり、外部へ書き出したりしても良い。この補助記憶装置を経由するデータ伝達手法は、複数の独立したプログラムの連携に利用可能である。

計算に必要な作業を機能別の副プログラムに分割する事は、プログラミング技術として、非常に大切な概念である。

簡単な入出力や主・副プログラム間のデータ受渡しに関する説明を付録に書いておいた。

先ずはどこかで必要とする簡単なプログラムを通して、計算機利用法を眺めてみよう。

1. GNUPLOT による描画

gnuplot というプログラムを用いて、画面上にグラフを描く手法を 2 次元データに限定してメモする。以下で例示する入力は、 Version 3.8j patchlevel 0+0.01 での使用例である。旧版とは一部で入力仕様が異なっている。

次の様な項目に付いて述べる。

0 開始と終了
1 $f(x) = a\,\cos(b\,x) + c\sin(d\,x + e)$ を想定して、グラフを描く
2 グラフの飾り付け
3 保存
4 eps 形式での出力
5 その他

０) 開始と終了

OS のコマンドとして、gnuplot と入力して、**gnuplot** を起動する。終了は、gnuplot> といった、gnuplot の入力促進表示 (command prompt) に対し、**quit** と入力すればよい。**exit** も同様の命令である。命令の入力は、命令語をタイプ入力した後で、リターンキーを押す。

１) 関数の定義と描画

1-1) 先ず、パラメータ $a.b.c.d.e$ を定義する。

gnuplot> a=1.2; b=2.3; c=3.1; d=-2.5; e=0.123

複数のパラメータに値を代入するには、上の様にセミコロンで区切る。複数の命令を一行で入力できる。勿論、一行に一つずつ入力しても良い。

1-2) 関数を定義する

gnuplot> f(x)=a*cos(b*x)+c*sin(d*x+e)

1-3) 描画命令を入力する

gnuplot> plot f(x)

1-4) x 軸の範囲を $[-5, 6]$ に設定しなおしたい時は、以下の命令を使用する。

gnuplot> set xrange [−5: 6.0]

gnuplot> replot

set 命令だけではグラフの再描画は行われないので、再描画命令を入力する。rep (リターン) という省略形も受け付ける。

y 軸に対する命令ならば、 set yrange [**:**] という形になる事は、想像通りである。

1-5) この場合、グラフの曲線と x、 y 軸の枠の線の太さを比べると、枠の方が太い。曲線の方を太くしたいならば、次の命令を実行する。

gnuplot> f(x) lw 2.3

lw は line width の省略形である。後の 2.3 は、最初の曲線の２．３倍の太さ (公称) で描画せよという意味である。

1-6) 設定したパラメータの値を確認する。

gnuplot> show all

パラメータや関数を再入力すると、上書きされる。

1-7) y 軸を対数表示としたいとすると、

gnuplot> set logscale y

x 軸だとどうすれば良いかはすぐに分かるだろう。

２) 飾り付け

2-1) x 軸に解説文字 (Legend) を付ける

gnuplot> set xlabel ‘This is an x-axis’

大体中央に、 This is an x-axis と書いてくれる。y 軸ならば、set ylabel ’:::’ とすれば良い事は想像通りである。

2-2) グラフ中に見出し文字を入れる

gnuplot> set label 1 “f(x)=acos(bx)+csin(dx+e)” at 1.5,3.3

１番目のラベルという意味で set label 1 としておいた。at 1.5,3.3 は文字の先頭位置を、現在の座標で与えている。

何番目のラベルかが、不明になれば、show label というコマンドで表示させる。

入力文字に間違いを見付けたら、もう一度 set label 1 “***” というコマンドを最初から、入れ直せば良い。history 機能を使用できる。ラベルに１番という番号が付いているので、ラベルが上書きされる。もしもラベル番号を付けてなければ、入力順にラベル番号が内部でつけられる。

文字の入力は、 single quote 又は double quote で文字列を囲む。ラベルを消したければ、unset label 1 とする。

ラベルの為の空白場所を確保したければ、set xrange, set yrange といった命令を入力して、画面の内の描画域を再定義する。

ラベルに上付や下付の文字を利用した場合がある。ギリシャ文字を利用したい場合もあるだろう。

```
gnuplot> set label "A_b+C^d,  {/Symbol sq}" at 1,2
```

と入力して、何が起こるか調べると良いだろう。

2-3) グラフに x 軸を入れてみよう。

gnuplot> set arrow 1 from -5,0 to 6,0 nohead

nohead を付けないと、矢印付の線が描かれる。複数の曲線を描き、こちらの曲線は $f(x)$、もう一方は $g(x)$ と言うような描き方をするならば、通常の矢印付き直線を描き、直線の一方の端に、ラベルを書き込めば良い。

2-4) 画面右上に出力される赤い線と $f(x)$ という凡例文字は消したい。

gnuplot> unset key

標準の凡例よりも丁寧なラベルを付けた時には、凡例が邪魔になる。

３) 保存と再描画

3-1) これまでの作業結果を、保存する。

gnuplot> save ’fx.gnu’

fx.gnu というファイルに描画命令が全て書き込まれる。このファイルは、文字型のファイルであるから、通常の editor プログラムで、内容を確認する事が出来る。興味があるならば、どのような設定がなされているか、目を通しておくと良い。標準設定で、色々な命令が書き込まれている事が分かる。逆に言えば、これらの命令を駆使して各種のグラフを描く事が可能である。

3-2) 先に書いたファイルの再利用をしたい。

gnuplot> load ‘fx.gnu’

この命令で、先に書いたグラフの描画作業を続行、変更して再利用する事が可能である。

４) 他の形式への出力

encapsulated postscript (eps) 形式への変換を例にとろう。

4-1) gnuplot> set term pos enhanced 'Helvetica' 18

この例では、term(inal) (出力先) を X １１形式から、pos(tscript) に変更している。

enhanced は、古い版よりも機能が強化されている事を示していると思う。

使用する文字フォントを Helvetica としている。最後の数値 １８ を宣言して、幾らか標準よりも文字を大きくしている。

ひょっとすると日本語フォントも同時に (default で) 宣言されているかも知れない。これは善し悪しである。もしも日本語フォント宣言は不要ならば、

gnuplot> set locale ""

としておくと良い。

4-2) 次に書き出すべきファイルを宣言する

gnuplot> set output 'fx.eps'

4-3) 実際に書き出す為には、replot 命令を下す。

gnuplot> rep

これで新しく fx.eps というファイルが作られた。この fx.eps ファイルは、ghostview というプログラムを用いて、VDT 端末に表示したり、ワードプロセッサープログラムで利用したり出来る。

５) その他

5-1) 既にあるファイルの数値を描画に利用したい

gnuplot> plot 'x' using 1:2:3 w e

この命令では、x という名前のファイルの第１列目を独立変数 x とし、第２列目を従属変数 y とし、第３列目を y の誤差 δy として、誤差棒付で描画する。

最後の w e は with errorbars の省略形である。

もしも、一つの画面に $f(x)$ と同時に描きたければ、以下の様に入力する。

gnuplot> plot 'x' using 1:2:3 w e,f(x)

$f(x)$ のところは、'y' using 1:4 w lp といった別の plot 対象を指定する命令でも構わない。後者の場合には、y という名前のファイルの第１列と第４列を (x,y) のデータの組だと思い、lp という形式で描画する。lp は、linespoints という二つの形式の重ね合わせであり、lines だけだと与えられた数値データを (x,y) の組だと思い、直線で結ぶ。一方 points は (x、y) の組に対して、点を描く。

因みに、x というファイルの中身は以下の様になっている。先頭が # で始まっている行は、コメント行として読みとばされる。

```
#Ang(deg)   X(exp) (+/-)     X/X_Ruth       Ay(exp  +/-)    Ay
```

```
 36.18 5.5400E-01 2.78E-02 5.6283E-01  -0.34860  0.01820 -0.13470
 41.32 1.4100E-01 7.18E-03 8.8862E-02  -0.47290  0.02620 -0.18394
```

中間のデータは省略した

```
160.70 7.9900E+00 4.02E-01 2.3525E+00   0.98540  0.04950  0.39126
165.53 7.9000E+00 4.03E-01 4.4298E+00   0.94800  0.04820  0.22289
```

5-2) ファイル中のデータを操作して表示したいとする。

```
gnuplot> plot 'x' using 1:($2*$4)
```

この例では、x という名前のファイルの内の第１列を x 軸データとし、第２列のデータと第４列のデータの積を y 軸データとしてグラフを描くという命令である。括弧の中には、定数も使える。

もしも、x というファイルのデータ中に２行以上の空白行があると、gnuplot にこれを独立なデータとして識別させる事が出来る。

gnuplot> plot 'x' index 0 using 1:2 w lines, 'x' index 2 using 1:3:5 w e

この例では、ファイル 'x' の中の第１ (先頭) ブロック (index 0) の第１、２列を描き、更に、x ファイルの第３ブロック (index 2) の第１、３列を (x,y) データとし、y には第５列の数値を誤差として誤差棒付のグラフとする。

5-3) 実験データにあう、関数のパラメータを探したい。

実験データが、ファイル名 x のファイルに入っていて、関数 $f(x) = a * x * *2 + b * x + c$ という関数を想定し、パラメータ a, b, c を決めたいとする。

gnuplot> f(x)=a*x**2+b*x+c

gnuplot> fit f(x) 'x' using 1:2 via a,b,c

特定の初期値を与えなければ、gnuplot が見計らいで探してくれるし、初期値を与えたければ、1-1) の様にすれば良い。

後で、fit.log というファイルの中身を覗いて見るのも良いだろう。

5-4) 多くの命令は覚えられないよ！という人に

gnuplot> help

と入力してみる事を勧める。必要ならば、help の後に キーワードを付けても良い。

5-5) 少し使い込んでみたいという人に、以下のホームページを参照する事を勧める。

http://t16web.lanl.gov/Kawano/gnuplot/

２．乱数の発生

自分で書いたプログラムの動作を確認する方法として、一様乱数を使用してテストデータを

発生させる方法が考えられる。そこで実用的なプログラムを手元に持っておく事は有用であろう。W.H.Press 他著 「Numerical Recipes in C」 、日本語訳は丹慶他、「C 言語による数値計算のレシピ」技術評論社の pp 209 に紹介されている ran2 を引用しておこう。これは $0 < x < 1$ の範囲の３２ビット表現で一様な疑似乱数を発生させる目的で設計されている。乗算合同法を用いて２系列の乱数を発生し、両者を切り混ぜて乱数の発生周期を３２ビット以上の長周期としている。

FORTRAN77 用に書き直した版を、参考の為に書いておく。主プログラムでは乱数を発生させて、πを計算している。先頭の数字が０から９にどのように分布するかも調べている。

CURTIM は 私用に作った、現在時刻を秒単位で返す副プログラムである. 多分、読めば分かるだろう。

```
C      implicit real*8 (a-h,o-z)
       dimension x(100),iy(10)
       do 1 i=1,10
     1 iy(i)=0
       iii=111
       NR=100
       call curtim(it0)
       in=0
       notin=0
       do 2 l=1,100
       do 2 k=1,1000
       call rand2(iii, NR, x)
C
       do 10 i=1,100
       j=x(i)*10+1
    10 iy(j)=iy(j)+1
       do 11 i=1,100,2
       xx=x(i)**2+x(i+1)**2
       if(xx .gt. 1.0) then
          notin=notin+1
       else
          in=in+1
       end if
 11    continue
 2     continue
C
       write(6,20)(i,iy(i),i=1,10)
```

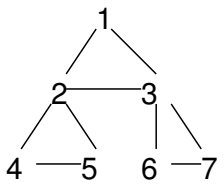
```
 20   format((1h ,i3,i8))
      ratio=real(in)/real(in+notin)
      ref=3.14159265/4.0
      write(6,22)in, notin, ratio,ref
 22   format(1h ,'in=',i12,2x,'Notin=',i12,2x,'Ratio ref=',1p2e14.6)
      call curtim(it1)
      lap=it1-it0
      write(6,30)lap
 30   format(1h ,'lap=',i3,'sec')
      stop
      end
      SUBROUTINE RAND2(I1, NR, RAN)
C      returns table of NR random numbers
C Input I1: seed for the 1st random number
C           use negative value to initialize
C Output RAN(NR): random numbers ranging from 0 to 1
C        NTAB is the size of array IV
      DIMENSION  RAN(NR), IV(32)
      DATA M1, IA1, IQ1, IR1 /2147483563, 40014, 53668, 12211/
      DATA M2, IA2, IQ2, IR2 /2147483399, 40692, 52774,  3791/
      DATA IY, NTAB, EPS /0, 32, 1.2E-7 /
C        initial definition
      MM1=M1-1
      NV =MM1/NTAB+1
      EM1INV=1.0/M1
      RNMAX=1-EPS
C        initialize ?
      IF(IY .GT. 0) GOTO 2
      IF(I1 .LT. 0) I1=-I1
      IF(I1 .EQ. 0) I1=IA2
      I2=IA1
      DO 1 J=NTAB+7, 1, -1
      K=I1/IQ1
      I1=IA1*(I1-K*IQ1)-K*IR1
      IF(I1 .LT. 0) I1=I1+M1
      IF(J .LE. NTAB) IV(J)=I1
    1 CONTINUE
      IY=I1
    2 DO 3 L=1, NR
C        1st random number
```

```
      K=I1/IQ1
      I1=IA1*(I1-K*IQ1)-K*IR1
      IF(I1 .LT. 0) I1=I1+M1
C        2nd random number
      K=I2/IQ2
      I2=IA2*(I2-K*IQ2)-K*IR2
      IF(I2 .LT. 0) I2=I2+M2
      J=IY/NV+1
C         mix two random numbers
      IY=IV(J)-I2
      IV(J)=I1
      IF(IY .LT. 1) IY=IY+M1
      TEMP=IY*EM1INV
      IF(TEMP .GT. RNMAX) TEMP=RNMAX
    3 RAN(L)=TEMP
      RETURN
      END
```

３．並べ変え (ソート)

計算結果を大きい又は小さい順に並べ変えたいとする。ソートと呼ばれるこの作業は、数値計算のなかでは特に大切な位置を占めてはいないので、上に引用した Numerical Recipes in C pp 241 のヒープソートの引用に留める。実数列 $X_i;\ (i = 1, 2, \cdots, N)$ が与えられた時、この数列を小さなな成分順に並べ変えよという問題だとして、簡単な解説を加えよう。

先ず、上に尖った２等辺三角形を一つ作り、上の頂点に１、左下の頂点に２、右下の頂点に３という番号を付ける。一般に N は３よりも大きいだろうから、点２を頂点として更に２等辺３角形を付け、この３角形の左下に番号４、右下には番号５を割り振る。これで足りなければ、更に点３を頂点として同様の作業をする。N 個の要素を各頂点に一つずつ番号順に対応させる。このような、３角形を多数並べると、下の図の様に並ぶ。



この並びでは、頂点の番号を i とすると、左下の番号は (2 i) であり、右下の番号は (2 i + 1) となっている。この頂点に一つずつ、X_i が対応している。

先ず、番号が一番大きな３角形に着目し、この３個の点の番号に対応する X の値の内で一番小さなものを、上の頂点番号の位置に移す。最後の３角形では、比較対象は３個に満た

ないかもしれないが、この操作自体は実行可能である。次は、左隣の３角形に対して同じ作業をする。左隣が全部処理されれば、一段上へ上がり、右から順番に３角形を取り上げて、この３角形の中で最小の値を有するものを上の頂点番号に対応する位置に格納する。このようにして番号１にまで辿り着いた時には、１番の位置には最小値が格納されている。

最小値が X_1 に納まったならば、$X_2, X_3, \cdots, X_N$ を対象とし、同様の手続きを繰り返す。

２回目以降の処理に関しては、X(2)−X(N) を対象として処理するには、別途下請け副プログラムを用意するのが順当なやり方であるが、もしも下請けプログラムを利用しないならば、それなりに処理方法を考えておかねばならない。

大きい順に並べる事を想定した副プログラム HPSDEC でこの事情を見ておこう。N 個のデータを処理する時には、最初は $N-1$ 個のデータを比較し、次は $N-2$ 個のデータを比較する事になる。この流れを制御するのが、最初の DO 10 M=1,N−1 というループである。このループ内で、最初に取り上げる三角形の上の番号は IP から一つずつ減って行く。現在取り上げている三角形の頂点の番号は JP である。即ち DO JP=IP,M,−1 という行は、三角形の頂点を一つずつ指している。この頂点の番号を与えた時、この頂点を共有する底辺は一つしか無い場合と二つある場合が考えられる。一つか二つかを判定するのが JQX という変数である。一つのヒープ内での比較作業は必ず、最後から順番に辿る。この最後から順番に変化するポインターが I である。

このプログラムは非常に短いが、慣れない人には解読が困難かもしれない。コンパイラーの性能にも依るが、実行速度は非常に早いと思う。手元の PC では、 NR=10000 として、昇順と降順の並べ替えを交互に１０回繰り返すのに、5.2 から 5.8 秒かかった。因みに、5.2 秒の方は ifc 、5.8 秒の方は f77 であった。別の、少し遅い PC では、同じ計算が、27.9 秒 (frt)、 18.5 秒 (f77)、15.3 秒 (ifc) であった。この様な単純なプログラムでも、コンパイラーの性能に依存して処理速度が変化するようだ。

```
      parameter (NR=10)
      DIMENSION X(NR)
C       define random numbers for X and Y
      I1=-1
      CALL RAND2(I1,NR,X)
      CALL PUTX(NR,X)
      CALL HPSASC(NR,X)
      CALL PUTX(NR,X)
      STOP
      END
      SUBROUTINE HPSDEC(N,X)
C       reorder array elements by descending order
C       by using heap sort
      DIMENSION X(N)
      IP=N/2
```

```
      ISE=0
      IF(IP*2.NE.N)ISE=1
      ISO=ISE+1
      DO 10 M=1,N-1
      I=N
      JQX=ISO
      DO 9 JP=IP,M,-1
      DO 8 JQ=1,JQX
      IF(X(I).LT.X(JP))GOTO 8
      XX=X(I)
      X(I)=X(JP)
      X(JP)=XX
    8 I=I-1
    9 JQX=2
      ISO=3-ISO
      IP=IP+ISE
   10 ISE=1-ISE
      RETURN
      END
      SUBROUTINE HPSASC(N,X)
C       reorder array elements by ascending order
C       by using heap sort
      DIMENSION X(N)
      IP=N/2
      ISE=0
      IF(IP*2.NE.N)ISE=1
      ISO=ISE+1
      DO 10 M=1,N-1
      I=N
      JQX=ISO
      DO 9 JP=IP,M,-1
      DO 8 JQ=1,JQX
      IF(X(I).GT.X(JP))GOTO 8
      XX=X(I)
      X(I)=X(JP)
      X(JP)=XX
    8 I=I-1
    9 JQX=2
      ISO=3-ISO
      IP=IP+ISE
```

```
10 ISE=1-ISE
   RETURN
   END
   SUBROUTINE PUTX(N,X)
   DIMENSION X(N)
   DO 11 M=1,N
   WRITE(6,10)M,X(M)
10 FORMAT('X(',I3,')=',F8.5)
11 CONTINUE
   WRITE(6,12)
12 FORMAT()
   RETURN
   END
```

４．簡単な数値計算例

多くの初等関数はライブラリープログラムとして、利用者に提供されているので、これを利用すれば良い。しかし、計算技術上知っておいた方が良い事例として取り上げる。一松信著 「近似式」竹内書店を一般的な参考書として挙げておこう。

例えば指数関数の計算に テイラー展開を利用するとしよう。

$$\exp(x) = \sum_n \frac{x^n}{n!}$$

$x > 0$ ならば、各項は正であるから、時間コストさえ気にならないならば、収束するまで強引に展開を進めるのが一つのやり方である。

```
   EPS=1.0E-5; X=5;  DEN=1;  FNUM=1;  SUM=1
   DO 10 N=1,100
   DEN=DEN*N;  FNUM=FNUM*X;  TEMP=FNUM/DEN;  SUM=SUM+TEMP
   IF(TEMP.LE.EPS)GOTO 20
10 CONTINUE
   STOP 'No conv !'
20 Y=EXP(X)
   WRITE(6,22)X,SUM,Y;
22 FORMAT('X Sum Y=',0PF8.3,1P2E15.7)
   STOP;  END
```

これで、以下の出力が得られた。

XSum Y= 5.000 1.4841316E+02 1.4841316E+02

ここで、$1.4841316E+02$ という表現は、148.41316 と解釈する。即ち、最後の方の E+02 というのは、全体を 10^2 倍せよという意味である。

和の途中に負の項が登場する場合には、考えなければいけない事がある。$x < 0$ ならば、上のプログラムは正しく働かない。TEMP と EPS の大小を比較する部分を `ABS(TEMP).LE.EPS` と書き換えなければいけないが、これだけでは精度が保てないという意味である。例示するために `X=-5` として、プログラムを走らせてみると、以下の結果を得た。

X Sum Y= −5.000 6.7362068E−03 6.7379470E−03 4桁目で誤差となっている。勿論、EPS の絶対値を小さくすれば、問題の一部は解決される。別の判定法は、収束判定文を以下の様に置き換える。`IF(ABS(TEMP/SUM).LE.EPS)GOTO20`

即ち、誤差の絶対値に注目するのでなく、相対誤差に注目した。但し、この判定基準は、計算結果の絶対値が0に近くなると破綻を来たし兼ねない。

$\sin(x)$ の計算

色々なアイデアを例示する意味で、取り上げてみよう。

1) 級数展開 1

$|x| < 1$ ならば、奇数次の級数として計算可能である。

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \cdots$$

この式を

$$\sin(x) = x \sum_n a_n \, x^{2n}$$

と書くと、x の偶数巾の計算と展開係数 a_n の計算 及び、両者の積和の計算に分割される。従って、以下の様な手順が一般的であろう。

0) 変数 X に x の値が与えられている。

1) 収束判定の為の小さな数 ϵ を定義する

2) 部分和を書き込む変数を定義し、これに1を代入する。

3) この変数には、SUM という名前を付けておこう。展開係数に対応する変数 A を宣言し、1を代入する。

4) x の2乗と偶数次巾を書き込む変数 X2 、 XPOW を宣言し、X2 には、x^2, XPOW には1を代入する。

5) 和の回数を数える変数として、N を宣言する。

6) 以下の操作を N=1 から何回か繰り返す

6-1) (新しい A) =ー (古い A)/(2 N + 1)

6-2) (新しい XPOW) = (古い XPOW) x X 2

6-3) (新しい SUM) = (古い SUM) + (A) x (XPOW)

6-4) もしも ｜ (A) x (XPOW) ｜ $< \epsilon$ ならば、計算終了

6-5) 収束すら迄、N を１ずつ増やして繰り返す。

7) 最後に SUM に X を掛ける

この操作は、原理的には $|x| < 1$ に限定する必要はないが、大きな $|x|$ に対しては有効でない。その一つの原因は、$|x|$ が大きくなると、部分和の絶対値が１よりも大きくなる事にある。最終的には、部分和が１よりも小さくなる事は、数学的には証明されているが、有効数字が大きく損なわれる場合がある。

例えば、上に書いた例を $x = 4$ に対して約１５桁の有効数字で計算すると、下のようになった。

N	(A) x (XPOW)	SUM
3	−8.5333333E+01	−6.0333333E+01
6	3.6408889E+02	2.0135556E+02
9	−3.6986808E+02	−1.6851252E+02
12	1.4346398E+02	5.5649949E+01
15	−2.6906065E+01	−9.0927707E+00
18	2.8137062E+00	8.4312929E−01
21	−1.8052852E−01	−4.8230261E−02
24	7.6112155E−03	2.1940419E−03
27	−2.2204800E−04	2.8581697E−04
30	4.6670186E−06	3.3642495E−04
33	−7.2993449E−08	3.3544866E−04
36	8.7237735E−10	3.3546278E−04

A x XPOW の値は、N＝６ー１２の範囲で、１００を越えている。この値は、$\sin(4) = 3.35\cdots \times 10^{-4}$ であるから、最終結果に対して約１００万倍大きい。有効数字が７桁の計算を維持していたとすると、最終的には１桁しか正しい値を示さないはずである。

計算技巧として、展開係数や偶数巾乗計算に漸化式を使用している点に注意しておこう。

２) 級数展開 II

ある種の数値計算の本には、以下の様な公式が記載されている。

$$\frac{\sin \pi x/2}{x} \sim \sum_{k=0}^{n} (-1)^k a_k \, x^{2k}$$

ここで、展開係数は以下の表で与えられる。

$$
\begin{aligned}
a_0 &\quad 1.57079\ 63267\ 94817\ 89\\
a_1 &\quad 0.64596\ 40974\ 98520\ 79\\
a_2 &\quad 0.07969\ 26261\ 22389\ 16\\
a_3 &\quad 0.00468\ 17533\ 91417\ 21\\
a_4 &\quad 0.00016\ 04390\ 54586\ 38\\
a_5 &\quad 0.00000\ 35957\ 08013\ 45\\
a_6 &\quad 0.00000\ 00546\ 26236\ 84
\end{aligned}
$$

この展開係数は、$n = 6$ に対して、$|x| \leq 1$ の範囲で、相対誤差の最大値が最小になるように、山下真一郎により決定されという事である。このように、ある区間での誤差の最大値が最小になるような近似法を mini-max 近似とか最良近似と呼ぶ。今の場合、相対誤差の最大値は 8×10^{-14} と与えられている。

この場合の計算手順は、以下の様にするのが良いとされる。

$$\frac{\sin \pi x/2}{x} \sim (((((a_6 \times x^2 - a_5) \times x^2 + a_4) \times x^2 - a_3) \times x^2 + a_2) \times x^2 - a_1) + a_0$$

良いとされる理由は？

今の場合、確かに小さな数から順番に計算しているから、途中に大きな数が登場して、有効数字が計算途中で消える事にはなっていない。

x の冪指数が二つずつ上がる事になっているから、この面からも大きな数字が登場しにくい構造になっている。

事前に x^2 は計算してあるとして、1) で与えた計算手順と計算量を比較してみると、最初の 6 項まで計算するのに、後者では積が 5 回と和又は差が 6 回である。一方前者の計算手順では、繰り返しの中で、積が 3 回、商が 1 回、和が 1 回、これを 7 回繰り返すと、かなり計算量が多い事が分かるだろう。

誤差と計算量の観点から、後者の方が推奨され、ホーナー法と呼ばれている。

これらの方法では x の取り得る範囲がかなり限定されている。もっと広い範囲の $|x|$ に対して計算したい場合もあるだろう。その一つとして、$\cos x$ の級数展開も使用するというアイデアがある。通常は、x から π の整数倍を差し引いて、$|x| < 1$ に取り込んでから上の様な級数展開に持ち込むのが、一般的だろう。

3) 級数展開 III

今の場合、級数は項毎に符号が正、負と入れ替わる交番級数である。この性質を使用すると、収束を加速する事が出来る事をオイラーが説明している。

前進差分演算子 Δ を次式で定義する。

$$\Delta\, a_n \equiv a_{n+1} - a_n$$

この時、

$$\Delta^2 a_0 = \Delta(\Delta a_0) = \Delta a_1 - \Delta a_0 = a_2 - 2\, a_1 + a_0$$

である。一般の場合には、パスカルの３角形が登場する事も分かるだろう。

以下の交番級数を考える。

$$S = a_0 - a_1 + a_2 + \cdots = \sum_{k=0}^{\infty} a_k = \frac{a_0}{2} - \frac{\Delta\, a_0}{2^2} + \frac{\Delta^2\, a_0}{2^3} \cdots$$

k が小さい内は、定義通りの式で計算し、途中から オイラー変換をするのが良い様である。

収束が遅い交番級数に適用すると、驚くべき効果があるという。

少しプログラムが面倒だから、手抜きしよう。

興味があれば、森口繁一著「計算数学夜話」 日本評論社発行 pp ３７を読んでみよう。

ついでに、級数ではないが大量の三角関数を計算したい場合を二つ取り上げよう。

４) 等間隔に沢山計算したい場合 I

$x_n = x_0 + n \times h$ と与えられる $x_n,\ (n = 0, 1, 2, \cdots, N)$ に対して $\sin x_n$ を計算したいとする。 但し、 $|h| << 1$ であるとする。$\sin(x)$ は以下の微分方程式を満足し、$x = 0$ で $\sin(0) = 0,\ d\sin(x)/dx_{x=0} = 1$ を満足する。

$$\frac{d^2 \sin(x)}{dx^2} = -\sin x$$

誤差が h^5 で許されるならば、$s_0 = \sin(0),\ s_1 = \sin(h)$ を初期値として、次の漸化式を用いる。

$$s_n = \frac{2 - 5 * h^2/6}{1 + h^2/12}\, s_{n-1} - s_{n-2}$$

$h = 0.1$ の場合を、倍精度での計算結果を調べてみる。

n	s_n	$\sin(n \times h)$
10	8.4147092E−01	8.4147098E−01
20	9.0929706E−01	9.0929743E−01
30	1.4111936E−01	1.4112001E−01
40	−7.5680288E−01	−7.5680250E−01
50	−9.5892378E−01	−9.5892427E−01
60	−2.7941424E−01	−2.7941550E−01
70	6.5698756E−01	6.5698660E−01
80	9.8935780E−01	9.8935825E−01
90	4.1211669E−01	4.1211849E−01
100	−5.4402275E−01	−5.4402111E−01

そんなに精度を要求しなければ、この方法は使えるかもしれない。

理論的な背景が知りたければ、一松信著、「数値解析」 朝倉書店 pp １１９に書かれた Cowell の３点法を見よ。ここには導き方が書かれていないから不満だというならば、Henrici 著、「Discrete Variable Methods in Ordinary Differential Equations」Wiley 版 pp ２８９ を参照すると良い。その時は、事前に pp １８７以降も読む事を勧める。

５) 複数の等間隔点での計算

三角関数の和の公式を利用する。

$$\sin(x+h) = \sin(x)\cos(h) + \sin(h)\cos(x)$$

$$\cos(x+h) = \cos(x)\cos(h) - \sin(h)\sin(x)$$

即ち、$s_n = \sin(x_n),\ c_n = \cos(x_n)$ と置くと以下の漸化式が成立する。

$$\begin{pmatrix} s_n \\ c_n \end{pmatrix} = \begin{pmatrix} \sin h & \cos h \\ \cos h & -\sin h \end{pmatrix} \begin{pmatrix} s_{n-1} \\ c_{n-1} \end{pmatrix}$$

s_0 と c_0 を初期値とし、$\sin h,\ \cos h$ が計算出来れば、この漸化式はかなり繰り返して使用しても誤差は積み重なり難い。この理由は、右辺に登場する２行２列の行列の行列式が１であるからである。

x が弧度法で与えられずに、通常の度で与えられる時には、ラジアンへ変換するのが常套手段であるだろうが、３度に対する正弦は厳密な式があるからこれを利用するのも良いだろう。即ち、

$$\sin 18^\circ = \frac{\sqrt{5}-1}{4},\ \sin 15^\circ = \frac{\sqrt{6}-\sqrt{2}}{4}$$

という知識と、三角関数の基本的な知識があれば、$\sin 3^\circ$ に関する方程式は解ける。従って、独自に三角関数表を作れるだろう。後は、内挿に関する知識があれば良いだろう。

最後に取り上げた手法は、角度の関数 $f(\sin x, \cos x)$ を台形公式やシンプソンの公式を用いて積分する場合に利用すると良いだろう。

５．内挿

大量の関数値 $f(x)$ を計算したい時には、ある程度の間隔で $f(x_i),\ (i = 0, 1, 2, \cdots, N)$ を計算し、必要に応じて内挿するのが現実的である。特に x_i を等間隔にとっておくと、内挿は簡単である。しかし、関数値が激しく変化する部分は細かく、穏やかに変化する部分は粗い刻みで計算するのが合理的だろう。

Lagrange 補間と３次のスプライン補間の話題を提供しよう。

ラグランジェ補間

区間 $[x_0,\ x_N]$ に両端を含めて $N+1$ 個の点をとり、x_i を代表点とする。y_i をこの点での関数値とする多項式を書き下せという問題である。

先ず、$\Pi_i(x-x_i)$ は、全ての x_i で値０をとる事は明らかである。この式を (x_k-x_i) で割った式 $L_k(x) \equiv \Pi_{i\neq k}(x-x_i)/(x_k-x_i)$ は、$i \neq k$ ならば、$L_k(x_i)=0$ であり、$L_k(x_k)=1$ である。即ち、$L_k(x_i)=\delta(i,k)$ である。従って

$$L(x)=\sum_k L_k(x)=\sum_k y_k \Pi_i(x-x_i)/(x_k-x_i)$$

は、点 $(x_k,\, y_k),\ (k=0,1,\cdots,N)$ を通る N 次多項式である。

この式を直接利用する事は稀であろうが、数値積分や微分方程式の解法には利用される事があるので、知識として知っておけばよいだろう。

この式を外挿に利用してはいけない。定義域の外では激しく変化したり、非常に大きな値をとったりし、Runge の現象とか、Debye の現象と呼ばれる事がある。

高次の多項式の使用は差し控えるのが妥当である。広い範囲を一つの高次多項式で近似するよりも、狭い範囲を低次の多項式で近似する方が落ち着いた結果が得られる様である。

３次のスプライン展開

(N＋1) 個の節点 $(x_i,\ y_i),\ (i=0,1,2,\cdots,N)$ が与えられたとする。但し、$x_i < x_{i+1}$ とする。相隣る４個の点を通る x の３次式は必ず存在するから、これを用いてこの N＋1 個の点に対する内挿は可能である。内挿したい点 x の左右に２点ずつを選び、３次式を作れば良い。

しかし、もう少し近似精度を上げた３次式を作る事が次の条件を設定する事で可能となる。

１) 全ての区間で３次式である。(区分的３次式)

２) 両端を除く全ての節点で、１次及び２次微分が連続である。

区間 $[x_i,\ x_{i+1}]$ でこの事情を式に書き下してみよう。先ず、両端での２階微分を $z_i,\ z_{i+1}$ と書くと、この区間での２階微分は

$$y''(x)=\{z_i(x_{i+1}-x)+z_{i+1}(x-x_i)\}/h_i, \quad h_i \equiv x_{i+1}-x_i$$

と書ける。この式を１回積分すると、

$$y'(x)=\frac{1}{2\,h_i}\{-z_i(x_{i+1}-x)^2+z_{i+1}(x-x_i)^2\}+A$$

更に積分すると、

$$y(x)=\frac{1}{6\,h_i}\{z_i(x_{i+1}-x)^3+z_{i+1}(x-x_i)^3\}+Ax+B$$

$x_i,\ x_{i+1}$ で、 $y_i,\ y_{i+1}$ という値をとるから、

$$A=\frac{y_{i+1}-y_i}{h_i}-\frac{h_i(z_{i+1}-z_i)}{6}$$

$$B=(1-\frac{x_i}{h_i})y_i-\frac{x_i}{h_i}y_{i+1}+\frac{h_i}{6}\{x_i\,z_{i+1}+(h_i-x_i)z_i\}$$

これで、A、B は決定した。点 x_i での 1 階微分は区間 $[x_{i-1}, x_i]$ と $[x_i, x_{i+1}]$ で計算したものが一致する条件を書くと、

$$\frac{h_{i-1}}{6}\{2z_i + z_{i-1}\} + \frac{h_i}{6}\{2z_i + z_{i+1}\} = \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \qquad (S1)$$

この式は、(N-1) 個あるから、未知数 z_i の数より 2 個 条件が少ない。

何らかの方法で、z_0 と z_N を推定出来るならば、残りの $z_i,\ (i = 1, 2, \cdots, N-1)$ は計算可能である。

例えば、$z_0 = z_N = 0$、$z_0 = z_1;\ \ z_N = z_{N-1}$ といった選択が可能である。又は、両端の 3 点を用いて 2 次近似して、$z_0,\ z_N$ を推定する。

(S1) は、次の様にして解くのがよいだろう。簡単の為に、$Z_i = z_i/6$、右辺は δ_i とおくと、

$$h_{i-1}Z_{i-1} + 2(h_{i-1} + h_i)Z_i + h_i\, Z_{i+1} = \delta_i$$

特に、

$$Z_1 = \{\Delta_1 - h_1\, Z_2\}/w_1$$

と書くと

$$w_1 = 2(h_0 + h_1),\ \ \Delta_1 = \delta_1 - h_0\, Z_0 \qquad (S2)$$

一般に、

$$Z_i = \{\Delta_i - h_i\, Z_{i+1}\}w_i \qquad (S3)$$

と仮定し、$i \to (i+1)$ として (S1) に代入すると、次の漸化式を得る。

$$w_{i+1} = 2(h_i + h_{i+1}) + \frac{h_i^2}{w_i},\ \ \Delta_{i+1} = \delta_{i+1} - \frac{h_i\, \Delta_i}{w_i} \qquad (S4)$$

(S2) から出発し、(S4) を用いると、$i = 1, 2, \cdots, N-1$ に対して $w_i,\ \Delta_i$ が計算できる。Z_N の知識を仮定すると、(S3) により、大きい方から小さい方へ全ての Z_i が計算でき、未知数は全て確定した。

先に述べたように、これにより $x_i,\ y_i,\ Z_i$ を利用した内挿公式は簡単に作られる。

$$y(x) = \frac{(x_{i+1} - x)^6}{h_i}Z_i + \frac{(x - x_i)^3}{h_i}Z_{i+1} + \frac{x_{i+1} - x}{h_i}(y_i - Z_i\, h_i^2) + \frac{x - x_i}{h_i}(y_{i+1} - Z_{i+1}h_i^2)$$

ここでは、関数値だけを与えての計算であった。可能ならば、関数値以外に、1 階微分を計算出来る場合もあるだろう。この場合にも、少し変形すれば応用が可能である事は論を待たないだろう。

このスプラインの考え方は、与えられた点を滑らかに結ぶ曲線を描く時にも有用である。但し、この図を描く場合には、新しい独立変数 t を等間隔に導入し、$(t_i,\ x_i)$ と $(t_i,\ y_i)$ とを独立に内挿する方が良い。y' が発散する場合を避けるためである。

実験値の組みが与えられた時、滑らかに実験値を繋ぐ曲線を作りたい時にもこのアイデアは応用が利きそうである。節点の数を増やし、微係数を決定する方程式を最小２乗法の式で置き換えれば良いだろう。

数値微分に利用できるのは、当然だろう。

勿論、数値積分にこの考えを利用する人もいる。

平滑化

上に書いたのは、与えた点を必ず通過する曲線という指導原理であった。しかし、与えた点が実験点であるならば、必ずしも与えた点を通過する必要な無い。与えた点の近くを通り、滑らかであれば良いと考える事も可能である。この指導原理を採用する時には、スプラインとは異なった発想をする事が可能である。例えば、$(x_i,\ y_i),\ (i=1,2,\cdots,N)$ の N 個の点が与えられた時、相隣る５点を取り上げて、この５点を最も良く再現する２次式を最小２乗法を用いて決定する。この様な発想で、実験点を計算点で置き換えた後で、理論的な計算と実験値を比較する、又は実験値の変わりにこれらの値を利用する。この様な発想を平滑化と呼び、内挿とは区別する。簡単だから、特に細かく説明する必要は無いだろうと思ったが、そうでもないかな？。

$(x_i,\ y_i),\ (i=1,2,\cdots,N)$ の最初の５点だけを先ず取り上げる。$i=1,2,\cdots,5$ とする。x_i の精度は高いが、y_i の精度は悪いと仮定する。この５点の近くを通る放物線 $y(x)=ax\ 2+bx+c$ を考えて、係数 a, b, c を決定したい。この時の指導原理として、

$$S(a,b,c)=\sum_i (y(x_i)-y_i)^2=\sum_i \{a\,x_i^2+b\,x_i+c-y_i\}^2$$

を最小にするように係数 a, b, c を決定するのが最小２乗法である。即ち、$S(a,b,c)$ を a, b, c で偏微分して０ とおいた式を解けば良い。即ち、Cramer の公式を用いると、以下の様に表せる。

$$a=\frac{\Delta_a}{\Delta},\ b=\frac{\Delta_b}{\Delta},\ c=\frac{\Delta_c}{\Delta}$$

$$\Delta=\begin{vmatrix} S_4 & S_3 & S_2 \\ S_3 & S_2 & S_1 \\ S_2 & S_1 & S_0 \end{vmatrix},\ \Delta_a=\begin{vmatrix} Y_2 & S_3 & S_2 \\ Y_1 & S_2 & S_1 \\ Y_0 & S_1 & S_0 \end{vmatrix},\ \Delta_b=\begin{vmatrix} S_4 & Y_2 & S_2 \\ S_3 & Y_1 & S_1 \\ S_2 & S_1 & Y_0 \end{vmatrix},\ \Delta_c=\begin{vmatrix} S_4 & S_3 & Y_2 \\ S_3 & S_2 & Y_1 \\ S_2 & S_1 & Y_0 \end{vmatrix}$$

ここで、以下の略号を用いた。

$$S_m=\sum_i x_i^m,\qquad Y_k=\sum_i x_i^k\,y_i$$

この計算で、x_i, 新 y_i, $(i=1,2,3)$ を評価すると、次は x_2, x_3, $\cdots$, x_6 を取り上げて、$(x_4$, 新 $y_4)$ を作る。後は想像出来るだろう。

もしも、x_i の誤差も無視出来る程には小さく無い時には、どの様にこの処方を変更したら良いだろう？

６．数値積分

数値積分という意味では、１次元の直線上での積分や２・３次元積分等も考えられる。基本という意味で１次元積分に重点をおく。積分区間を微細区間に分割し、その区間内の代表点

での関数値と区間幅 (又は面積等) との積和をとり、最大区間の大きさを無限小とした時の積和の極限値を積分値とするのが定義である。

積分区間を N 個に等分割し、各区間の両端のどちらかの端での値をとり、期間幅を掛けて積和をとれば、穏やかな変化をする関数に対しては N を増やすと収束すると期待できる。区間 $[a,\ b],\ (a\ <\ b)$ での被積分関数 $f(x)$ に対して、以下の近似を使用できる。等間隔に分けた時の幅を $h = (b-a)/N$ と書く。

$$I_N = \sum_{i=1}^{N} h\, f(a + i\, h)$$

両端での振舞を修正する為に、以下の方が少しは良いだろう。

$$I_N = h\,\{(f(b) + f(a))/2 + \sum_{i=1}^{N-1} f(a + i\, h)\}$$

この後者の式を積分の近似値として使用する場合の誤差に関して、中間値の定理を用いた評価もあるが、Euler-Maclaurin の和公式の方が後の利用にも便利だと考える。

$$\int_a^b f(x)dx - I_N = -\frac{h}{12}\{f^{(1)}(b) - f^{(1)}(a)\} + \frac{h^3}{720}\{f^{(3)}(b) - f^{(3)}(a)\}$$
$$- \frac{h^5}{30240}\{f^{(5)}(b) - f^{(5)}(a)\} + \frac{h^7}{1209600}\{f^{(7)}(b) - f^{(7)}(a)\}\cdots$$

誤差は、両端での奇数回微分の差にベルヌーイ数で決まる係数を掛ければ良いと言っている。通常は右辺の第 1 項を使用すれば良いだろう。台形公式を用いて、両端での 1 階微分の差を誤差補正をするという補正法もありうるだろう。

両端での微分に差がある時には、次式で与えられる、シンプソンの公式を利用するのも良いだろう。

$$\int_a^b f(x)\, dx \sim [f(a) + f(b) + \{4\, f(a+h) + 2\, f(a+2h)\} + \{4\, f(a+3h) + 2\, f(a+4h)\}$$
$$+ \cdots + \{4\, f(a+h) + 2\, f(a+2h)\} + 4\, f(b-h)]\, h/3$$

両端を除いた時、$f(x)$ にかかる因子は 4 と 2 を繰り返しているが、4 の回数が 1 回多い。従って、N は偶数に限定される。台形公式 I_N は個別の小区間での積分を、被積分関数が直線だと仮定して評価しているのに対し、シンプソンの公式では、連続する二つの小区間でので積分を、$f(x_c - h),\ f(x_c),\ f(x_c + h)$ の値を用いた放物線近似で評価している。従って区間 $[a, b]$ を偶数区間に分ける必要がある。N がシンプソンの公式では偶数にすべき理由がここにある。

4 と 2 という係数の計算を含めて以下の様にすると良いだろう。

```
SUM=f(a)+f(b): W=4
DO ? I=1,N-1; SUM=SUM+F(a+i*H)*W; W=6-W
```

```
? CONTINUE
  SUM=SUM*H/3
```

積分区間と被積分関数が与えられた時、積分公式をどのように選択するかは、考えるべき事である。上に与えた台形公式の誤差評価を見ると、積分区間の両端での奇数次微分の差が誤差を与えている事に注目すべきである。例えば、周期関数の一周期に関する積分では、任意次数の微分に対して $f^{(n)}(b) = f^{(n)}(a)$ であるから、誤差は極端に小さくなる。従って、この周期関数の 1 周期に関する積分の場合には台形公式が高精度の積分結果を与える。別の視点として、$f^{(n)}(a) = f^{(n)}(b) = 0$ であっても、台形公式が高精度の積分公式となる。例えば積分の両端を旨く無限大にとばしてしまう事で、全ての次数の微分を 0 にする様な変数変換を考慮してみるのも良いアイデアである。

これまでは、区間を等分するが、被積分関数にかかる係数は適当に選ぶという指導原理であった。別の指導原理として、被積分関数を計算する x の値は適当に選ぶが、これにかかる係数は一定にしておくという手法もあり、Chevychev の積分公式と呼ばれる。

更に一般的には、x も荷重もどちらも自由に選べるならば、以下の N 点近似ではどこまで精度を上げる事ができるか？ という問いもある。この問いに具体的な手法を挙げて答えたのがガウスである。

$$\int_a^b f(x)\,\rho(x)\,dx \sim \sum_{i=1}^{N} w_i\, f(x_i)$$

ここで登場した荷重関数 $\rho(x)$ は常に正であり、この区間で積分可能な関数である。例えば角度積分で $a = 0,\ b = \pi, \rho(x) = \sin(x)$ といった場合を想定すれば良いだろう。

次の手法に依るのがガウスの言った事らしい。$\rho(x)$ と $N > 1$ を与えた時、先ず次の規格直交関数系 $p_i(x),\ i = 0, 1, \cdots$ を作る。

$$\int_a^b \rho(x)\, p_i(x)\, p_j(x) dx = \delta_{i,j},\ \ (i, j \le N)$$

$p_N(x) = 0$、即ち N 次の関数の全ての 0 点 $(x_i,\ i = 1, 2, \cdots, N)$ を横軸座標とする。荷重の計算は次式による。

$$w_i = \frac{1}{p'_N(x_i)} \int_a^b \rho(x) p_N(x)/(x - x_i) dx$$

もしも、$f(x)$ が $(2N - 1)$ 次式ならば、上のガウスの積分公式は正確な積分値を与える。又は、$f(x)$ を (2N−1) 次近似した積分公式と考える事も可能である。

$f(x)$ を 2N−1 次式、$p_N(x)$ を N 次式だとし、$f(x) = p_N(x)\, G(x) + g(x)$ と書くと $g(x)$ は N−1 次式である。上に与えた N 次式の 0 点を横軸として採用すれば $f(x_i) = p_N(x_i)\, G(x_i) + g(x_i)$ であるから、右辺の第 1 項は 0 であるから、ガウスの積分公式で見ると、$f(x)$ のかわりに右辺で $g(x)$ と書いても良い。積分区間で、N 個の点 $(x_i,\ f(x_i))$ を通る N−1 次式は Lagrange の公式を用いれば良い。w_i の計算式は、このアイデアで計算したものである。

積分区間は $x = (b+a)/2 + t(b-a)/2$ と変換すると t では $[-1, 1]$ に規格化できる。

荷重が 1 ならば、Legendre 関数を直交関数として採用する。他の場合は、直交関数系の簡単な教科書を見れば良いだろう。

Legendre 関数の場合の x_i, w_i を計算するプログラムを以下に例示しておこう。

```
C        test routine
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION XL(50), WL(50)
      DO 20 KK=1,1
      DO 20 N=2,50
      CALL GLCAL(N, XL, WL)
   20 CONTINUE
      S0=0
      S1=0
      DO 10 M=1,50
      S0=S0+WL(M)
   10 S1=S1+WL(M)*XL(M)**2
      S0=S0-2
      S1=S1-2.0D0/3
      WRITE(6,12)n,S0,S1
   12 FORMAT(1H ,'N S0 S1=',I3,1p2E30.20)
      STOP
      END
      SUBROUTINE GLCAL(N, XL, WL)
C  returns abscissa and weights of Gauss-Legendre integration formula
C Input N: order of integration formula
C Output XL(N): abscissa
C        WL(N): weights
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION XL(N), WL(N)
      DATA EPS, PI /1.0D-16, 3.14159265358979D0/
C
      T=PI/(2*N+1)
      DO 4 M=1,N
      X=SIN((N+1-2*M)*T)
      DO 2 ITR=1,10
      P0=1
      P1=X
      DO 1 L=2,N
      P2=((2*L-1)*X*P1-(L-1)*P0)/L
```

```
      P0=P1
    1 P1=P2
      P2=(1-X*X)*P1/(N*(P0-X*P1))
      X=X-P2
      IF(ABS(P2) .LT. EPS) GOTO 3
    2 CONTINUE
      GOTO 91
    3 XL(M)=X
    4 WL(M)=2*(1-X*X)/(N*P0)**2
      RETURN
   91 WRITE(6,92)N,M,P2
   92 FORMAT('(GLCAL) N=',I3,2X,'M=',I3,2X,'Corr=',1PE12.3/
     &'Iteration over time !')
      STOP
      END
```

適応的積分手法

被積分関数は積分区間のある領域では変化が激しく、別の領域では穏やかな変化をする場合がある。従って、効率的に積分を実行するのに、変化が激しい部分では積分区間を小さく分け、変化が穏やかな部分では積分区間を大きめに設定して数値計算をするのが合理的である。

森正武著、FORTRAN77 数値計算プログラミング、岩波コンピュータ・サイエンス、１２章を参考にして、問題に応じて適応的に積分を実行する手法を紹介しよう。

指導原理は、先ず全区間を積分し $Q(1)$ とする。次にこの全区間を二等分し各区間を積分し、$Q(2)$, $Q(3)$ だとする。もしも $|Q(2)+Q(3)-Q(1) \leq \epsilon$ により収束確認が出来れば、計算が終了する。もしも収束が確認出来なければ、２分割した積分区間を個別に ” 全区間 ” だと考えて先頭に戻る。この場合、分割された区間の右区間を ” 全区間 ” としたか、左区間を ” 全区間 ” として計算しているかを覚えておかねばならない。あまり、細かく分割せねばならない時には、エラー表示もしなければならないだろう。例えば不連続領域を一つの積分区間とした時に、この様な事が起こる。

FORTRAN では再帰的なプログラムは書けないので少し工夫が必要である。参考文献では多くの配列を宣言してこの障害を乗り越えているが、記憶領域の利用効率が悪いと思われるので、この部分を書き換えた。又、適応的な積分を考えるならば、等間隔な刻みを利用する必要は無いと思われるので、ニュートン・コーツの方法を使う必要は認められない。ガウス積分を使う方が効率的だと考える。一応６点法を使用する。

積分間隔を適応的に変更する手法を以下に記す。

先ず、全領域を積分し結果を Q(１) に蓄える。次に積分区間を２等分し、それぞれの区間での積分結果を Q(２)、Q(３) に置く。Q(１) と Q(2)+Q(3) の差が小さければ、積分が

実行できたとする。積分結果が収束していないと判断したら、上で定義した小区間のそれぞれを２等分し、結果を比べていく。その内に収束するだろう。

積分区間の分割レベルを定義する。レベル L では全区間が 2^L に仮想的に分割されていると考える。分割レベル L を記憶するために、変数 IW には 2^L を代入しておく。分割レベル L に対応し、積分の下限から上限までの各区間に先頭を 2^L として、番号を付ける。現在の積分区間はこの番号によって識別される。適応的積分であるから、この仮想的に与えられた番号の全ての区間が数値積分の対象になっている訳ではない。ある区間は分割レベルの低い時に積分は終っているし、変化の激しい区間では更に分割レベルを上げなければならない。

上に述べた意味での、収束を確認すべき３個の積分値は配列 Q に納められている。古い値が納められている場所のポインターは JP であり、新しい値は Q(IP) と Q(IP+1) に書き込まれている。更に、IP 番目の積分値が対応する積分区間の番号は ID(IP) に書かれている。即ち、全区間を IW に分けた時、一番下の積分区間を IW として数え挙げた時、積分区間が下からどの位置にあるかを、ID(IP) は指している。

配列 Q や ID は積分実行中に使い回しされているので、ポインター IP だけが、いつも現在の値を保持している。JP の値は、ID(IP) が４の倍数ならば IP−2 であり、そうでなければ IP−1 である。

ISCHK という変数を使用しているが、この変数は和を取った後、上へ上がるかどうかを示している。

プログラムは下限から上限へ向けて、分割レベルを操作して上がったり下がったりしながら、積分を実行している。この分割操作の様子はヒープソートの時に採用されているヒープと非常に似通ている。

参考文献の例、FX 1 に対する動作を以下に記す。積分区間は [-1,1] であり、収束判定の値は、1.0×10^{-5} とした。

$$f_1(x) = \frac{1}{(x+0.3)^2+0.05} + \frac{1}{(x-0.6)^2+0.01} - 2$$

以下の様に、積分区間を小さくして行った。最初の行が全区間であり、下へ行く程分割が細かくなる。数値は、配列 Q のポインターに対応する。先ず１を２と３に分け、２を４と５に分け、５を６と７に分け、収束したから６と７を４に統合し、５を６と７に分け、収束したから６と７を５に統合し、次に４と５を２に統合する。次に３を４と５に分け、４を６と７に分け、６を８と９に分け、収束したから６に統合し、７を８と９に分け、収束したから７に統合し更に６と７を４に統合する。５を６と７に分割し、６は更に８と９に分割する。８は更に１０と１１に分割し、１０を１２と１３に分割する。ここで収束したから１２と１３は１０に統合する。次は、１１を１２と１３に分割し収束したから１２と１３は１１に統合する。１０と１１は８に統合する。次に９を１０と１１に分割し、収束したから９に統合し、更に８と９を６に統合する。最後に７を８と９に分割し収束したから７に統合し、６と

７を５に統合する。４と５を３に統合し、更に２と３を１に統合すると計算は終了である。

```
 IW                  IP
  1                     1
  2        2                           3
  4     4     5                  4                      5
  8    6 7   6 7              6     7              6            7
 16                         8 9   8 9         8            9     8  9
 32                                      10       11     10 11
 64                                   12   13   12   13
```

この場合には、レベル６迄積分区間を分割し、１３個の補助記憶を使用している。186(= 31x6) 回、被積分関数を計算し、積分結果は $Q(1) = 3.624705014512630e + 01$ となった。参考文献の値よりも、2×10^{-10} 大きい。設定誤差よりも極端に誤差が小さいという意味で、誤差評価が十分出来ていないが、かなり良い結果であると思う。

参考文献の試行関数 $f_2(x)$ は、不連続な関数の積分である。参考文献では、レベルが ３０ 迄下ったところで、エラーメッセージを出して、分割を停止している。不連続点を含む時にも、積分を実行するのが妥当だろうか？ここでは、不連続点付近の関数値の情報を提供して、利用者に任せる事としよう。

以下に示したプログラムでは、この区間分割の部分を例示している。引数の意味は次の通りである。

FX　被積分関数を計算する関数副プログラムの名前

A と B　積分領域の下限と上限

EPS　収束を判定する小さな数。このプログラムでは、許容誤差の絶対値を使用している。相対値を使用するというアイデアもあるだろう。

NN　補助的な作業用配列 ID, Q の大きさ。呼び出しプログラムで、配列を確保しておいて、INTHP で使用する。

ID　整数型配列。どこまで積分区間を細分したかを記憶しておく領域。上の例では、１３個使用している。

Q　倍精度型の実数配列。ID と同じ大きさが必要である。ある程度の精度を期待しているので、プログラム全体が倍精度として宣言されている。
このプログラムが正常に終了したならば、Q(1) に回答が書き込まれている。

実際の数値積分を実行するのは、INTHP1(FX, A, B, IW, ID(1), Q(1)) という副プログラムである。

```
      SUBROUTINE INTHP(FX, A, B, EPS, NN, ID, Q)
C         integrate FX fomm A to B and returns to Q(1)
```

```
C         this is just  a control routine to manipulate heap range
C         real integration is made by using subroutine INTHP1
C Input FX: external function, which defines integrand
C      A,B: define range of integration
C      EPS: convergence reference
C       NN: size of the work area of ID, IW, Q
C      ID(NN), Q(NN): work area
C Output Q(1): integrated result
      IMPLICIT REAL*8 (A-H, O-Z)
      DIMENSION ID(NN), Q(NN)
      EXTERNAL FX
C         initialize
      ISCHK=1
      E2=2*EPS
      IW=1
      ID(1)=1
      CALL INTHP1(FX, A, B, IW, ID(1), Q(1))
      ID(2)=2
      IP=2
      NCALL=1
      IPMAX=3
C         start heap control
   10 IF(IP .GT. NN-1) GOTO 91
      IW=IW*2
      CALL INTHP1(FX, A, B, IW, ID(IP), Q(IP))
      ID(IP+1)=ID(IP)+1
      CALL INTHP1(FX, A, B, IW, ID(IP+1), Q(IP+1))
      NCALL=NCALL+2
C
   15 SUM=Q(IP)+Q(IP+1)
      ERR=E2/IW
      IDX=ID(IP)
      JP=IP-1
      IF((IDX/4)*4 .EQ. IDX) JP=IP-2
      IF((ISCHK .EQ. 0) .OR. (ABS(Q(JP)-SUM) .LT. ERR)) GOTO 20
C         is it OK to continue ?
      IF(IW .GT. 5000) GOTO 50
C         subdivide
      IP=IP+2
      ID(IP)=2*IDX
```

```
      IF(IP .GT. IPMAX) IPMAX=IP+1
      GOTO 10
C        converged
   20 Q(JP)=SUM
      IW=IW/2
      IF(JP .EQ. 1) RETURN
C      IF(JP .EQ. 1) GOTO 30
      ISCHK=1
      IF((JP/2)*2 .EQ. JP) GOTO 25
      IP=JP-1
      ISCHK=0
      GOTO 15
C       subdivide
   25 IP=JP+2
      ID(IP)=2*(ID(JP)+1)
      IF(IP .GT. IPMAX) IPMAX=IP+1
      GOTO 10
C      error message
   91 WRITE(6,92)IP, NN
   92 FORMAT(1H ,'(INTHP) IP=',I3,' beyond upper max. of ',I3)
      STOP
      END
```

多重積分

積分結果をもう一度積分するならば、２重 (面) 積分と呼ばれる。この結果を更に何回か積分するというアイデアもあり得る。上に紹介した手法を繰り返して使用すれば、原理的に計算は可能である。個人的には、この手法で４重積分まで実行した経験がある。

多重積分を、乱数を用いて実行するというアイデアがある。この場合、疑似乱数を使用し、モンテカロル法と呼ばれている。この場合の相対誤差は、被積分関数を評価する回数を N とすると $1/\sqrt{N}$ 程度と推定されている。即ち、収束が非常に悪い。

モンテカルロ法を使用した経験は無いが、高橋・室谷著「数値計算とその応用」コロナ社を読むと、疑似乱数ではなく準乱数を使用すると収束率が有意に上がる場合があると説いている。耳を傾ける必要があるだろう。

もう一つのアプローチとして、優良格子点というアイデアもあるようだ。

森正武著、FORTRAN77 数値計算プログラミング、岩波コンピュータ・サイエンス、１３章を参考。テストしてみると、かなり良い成績を示す。

次の d 次元空間での１辺が１の長さの超立方体での対積分を対象とする。

$$I = \int_{[0,1]} f(x_1, x_2, \cdots, x_d) dx_1 dx_2 \cdots dx_d$$

この積分の近似値として、次の量を採用する。

$$I_N = \frac{1}{N}\sum_{k=0}^{N-1} f(x_1^k, x_2^k, \cdots, x_d^k)$$

ここで、$x_i^k = [k\, g_i/N]$ である。$[x]$ が x の小数部を取り出すという逆ガウス記号であり、g_i は d と N に依存する自然数の定数である。関数値を評価する座標は、この式で与えられるある意味での準乱数となっている。N が大きくなると、区間 $[0,1]$ で等間隔に分布すると想像される。その時には、Euler-Maclaurin の和公式的なアイデアを組み込むのが高効率の計算が出来ると期待出来る。

上記の参考書では、積分区間の両端で４階微分までが０となるような変数変換を取り入れている。

優良格子点をどのように決定するのか？という問には現在は答える事が出来ない様だ。現時点では、上に引用した文献を読むのがベストだと思っている。

７．方程式を解く

常識的な１次方程式 $a\,x = b$ ならば、$a \neq 0$ という条件の下に $x = \dfrac{b}{a}$ と解ける。

２次方程式 $a\,x^2 + b\,x + c = 0$、$(a \neq 0)$ でも、実根条件 $D = b^2 - 4\,a\,c \geq 0$ の下では $x_\pm = -b \pm \sqrt{D}/2\,a$ と解ける。複合の内、一方では $-b$ と $\sqrt{D}$ は相加的に働き、他方は相殺的に働く。相殺的に働くと有効数字の桁数が減るから、そちらの解を採用する場合には、分子を有利化しておかねばならない。相殺解は、相加解と根と係数の関係から評価してもよい。

更に次数の高い１元方程式を解くには？というのが問題となるところであろう。

その前に、有効数字に関する注意をしておくのが良いだろう。先ず例え話を一つ。ある数値表示装置では、１から１０００迄の大きさの実数をある桁数で表示していた。次に、表示すべき数値を現在の数値の２乗である１ー１０００００００としなければならなくなったとしよう。当然の事ながら、以前の様な肌理の細かな表示は不可能である。

和や差の計算を実行する前には、小数点の位置合わせをしなければいけない。計算機屋さんは, この操作を正規化と呼ぶ。従って、有効数字が５桁しか無い計算機で計算すると、１０００００００に１を何回加えても１０００００００となる。２乗でも有効数字の桁落ちがある。次数が増えるとどうなる事やら。高次方程式を解く場合にはこの事を肝に銘じておく必要がある。どの程度精度が落ちるか、各自でテストしてみると良いだろう。

高次方程式を解く場合、専門家の書いた参考書には、解きたい方程式のグラフを描いてみよと書いてある。Mathematica の様なプログラムを利用するのが最も良い方法であろう。グラフを描く事は、解くべき関数の関数副プログラムを書いた時にその副プログラムのテストを兼ねてやっておくと良い。

３次方程式には、フェラーリ・カルダノ (タルタリア) の解の公式があり、４次方程式にはオイラーの解法がある。５次以上の次数の方程式には解の公式は無いとアーベルが証明したらしい。４次方程式の解の公式は、覚えていないから、書き下すのが面倒であり、数値計算という意味では、数値的に解いた方が早いと思う。

一般の次数の方程式の解法に移る前に、安直解法を紹介しておこう。$f(x) = 0$ という方程式を解きたい時に、$x = g(x)$ と変形できる場合がある。しかも何かある予備知識のお蔭で、解の近似値が x_0 だと分かっているとしよう。次の漸化式は、解を与える可能性がある。問題を $x_{i+1} = g(x_i)$ と変形する。最初は $i = 0$ から始める。グラフを描いてみれば分かるが、この手法は解を上下に挟みながら収束するので、途中で平均値を採用してみるという工夫もあるかも知れない。ここで登場する x_i を数列とみなして、その収束値を評価するというアイデアもあるようだ。数列に対する収束の加速というキーワードを手がかりとして調べてみよ。

この級数は、$|dg/dx|$ の大きさにより収束したり、収束しなかったりするが関数電卓があれば簡単に解ける手法の一つである。覚えておいても損はしないと思われる。後で説明するニュートン方よりも応用範囲の広い手法だと思う。

例： ある時に、非局所ポテンシャルの等価局所ポテンシャルを計算する為に、次の方程式を解いた。

$$V_{Leq}(r)\,\exp\left[\frac{M\beta^2}{2\hbar^2}(E - V_{Leq}(r))\right] = V_{non-l}(r)$$

この式は一見難しそうだか、r を一つ与えると、$a, b,\ c$ を定数、x を未知数とすると、$x\,\exp\left[a(b-x)\right] = c$ という x に関する方程式である。a の絶対値があまり大きくなかったから、$x = c\,\exp\left[-a(b-x)\right]$ と変形し、x の初期値として c そのものを採用して右辺の値を計算する。計算結果をもう一度右辺に代入する。この操作を繰り返すと数回で収束した。

従って、r を少しずつ変えながら、$V_{non-l}(r)$ を計算し、この $V_{non-l}(r)$ に対応する方程式を解き $V_{Leq}(r)$ を評価する。この操作を繰り返し、r と $V_{Leq}(r)$ の関係表を作成し、必要に応じて、例えば３次スプラインを用いて、内挿すればよい。

ある種のプログラムに $x_{i+1} = g(x_i)$ という式を２回繰り返して書いておいた。即ち、繰り返しが２回で収束が確認出来ている問題であった。プログラムを読んだ知人が、同じ行が連続して出て来るのはバグではないかと問い合わせて来た事がある。

方程式 $f(x) = 0$ の解の存在範囲が $(x_l,\ x_u)$ に追い詰められた場合に使用できる方法に２分法というのがある。領域の中央 $x_c = (x_l + x_u)/2$ での値 $f(x_c)$ を計算し、$|f(x_c)|$ が十分に小さければ、x_c を解とし、そうでなければ、$f(x_l) \times f(x_c)$ の符号を調べる。負ならば 新 $x_l = x_c$ とし、正ならば、新 $x_l = x_c$ とする。解の存在区間を半分ずつに縮小して行く方法である。探査区間での $f(x)$ の連続性を信用するならば、必ず解にたどり着けるという意味で信用のおける方法である。更に繰り返し回数による解の存在範囲の縮小率もはっきりしている。

この方法は、$f(x)$ の値を計算しているにも拘らず、その符号情報しか利用していないと

いう批判を受ける。例えば、x_l, x_c, x_u 3点での値を用いて放物線近似を使えないかというものである。個人的な経験から言うと、失敗する確率が有意にあるから、内挿公式にに頼ってはいけない。例えば、x_u が解の近くに、x_l が解の遠くに位置したとしよう。直線近似や放物線近似が正しく働いた時には、次の点は x_u の近くに予想される。しかしこの点の計算には、遠くの点 x_l の情報も x_u の情報と対等に入っているために、正解に近ずく速度は非常に遅くなる場合がある。即ち、解が端点の近くに来た時に、動きが止まってしまい、なかなか収束しない。対策としては、内挿公式による推定値 x_i が x_l や x_u に近すぎる時には2分法 x_c を採用し、中央付近ならば x_i を採用する。近いか近くないかは、例えば $|x_i - x_l| < |x_u - x_l|/4$ とする。4は3でも5でも良いだろう。

方程式を解くとは、試行値を解に近つけると考えるのでなく、解の存在区間を縮小すると発想すると良い。又、2次元的に考えて、解の存在区間でなく、解の存在面という発想をし、この面を出来るだけ正方形に保ったまま縮小する方向で工夫すると良い。即ち、x 軸と y 軸を固定的に考えずに、時には y 軸を基本にしてグラフを思い描くと良い。区間縮小をする場合には、必ず解に遠い方を解に近つける工夫をしなければならない。このような知識を持った後で、符号だけでなくその値をも利用するアイデアを実行に移そう。

近似解が分かっていて、$f(x)$ の微分が簡単に計算出来るならば、次のニュートン法も利用する価値がある。次の漸化式で近似解を改良すると期待する。

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

この解法も、グラフを描いてみれば直観的に理解できるだろう。このニュートン法も、近似解が分かっているという場合にしか使わないのが賢明である。正解から近い場合に、解を改良する手法として推薦できる。

へんな初期値から出発すると、解の回りをうろついたり、ひどい場合には解から遠くにふっ飛ばされてしまう。2分法と併用し、解の存在範囲を確実に狭くする時の一つの予想値として利用するのが、賢明だろう。

いわゆる2次収束だから、収束しだしたら頼りになる手法である。

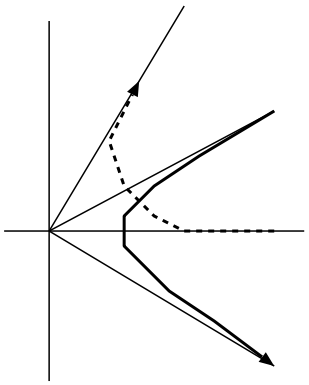
これもグラフに描いて見ると了解出来るが、$f(x)$ が下に凸ならば、ニュートン法では一方から解に近ずく。可能ならば、上からの近似解と下からの近似解を比較して、解の妥当性を確認しておくのも良い習慣だろう。

解の存在範囲を複素数に迄広げる時には、代数学の基本定理を先ず確認しておこう。以下の n 次方程式を考える。

$$f(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 = 0$$

当然の事ながら、$a_n \neq 0$ を仮定している。この方程式は、z が複素数をとるとすると、必ず解 (根) を一つ有する。その解を z_1 とし、$g(z) = f(z)/(z - z_1) = 0$ にこの定理を適用す

ると $n-1$ 次方程式にも 1 個の根がある。結局多重度を認めると、 n 次方程式には n 個の根が存在する事になる。直観的な証明は非常に簡単である。



自明な場合は除外し、簡単化の為に全体を a_n で割っておこう。$z = |z|(\cos\theta + i\sin\theta)$ と表現する。$|z| \to \infty$ という極限を考えると、第 1 項 z^n が支配的な項であるから、この項だけを考えると良い。この極限では、実部のゼロ点はコサインが 0 となる $\pm\pi/2\,n$ とこれに π/n の整数倍を加えた (差し引いた) 角度である。原点から出たこの角度の線上の無限に遠い所では $f(z)$ の実部は 0 である。この $f(z)$ の実部＝0 という軌跡を $|z|$ を変えながら追跡してみる。$|z|$ が大きい時は、最初に仮定した直線に沿って動いていたのが、あるところでずれ始め、ついには $|z|$ が最小と言う点に来る。ここからは $|z|$ を大きくして、更に軌跡を追跡する。最初の出発点が $\theta = \pi/2\,n$ 上にあったとすると、この軌跡は $\theta = 3\pi/2\,n$ 又は、$\theta = -\pi/2\,n$ という直線に漸近的に近つく。例えば、$\theta = -\pi/2\,n$ 上に来たと仮定しよう。

次に、$f(z)$ の虚部＝0 という点の軌跡を考えよう。$|z| \to \infty$ の極限では、$\theta = 0$ 及びこれに π/n の整数倍を加えた (差し引いた) 線上にこの軌跡が存在する事は明らかである。上で仮定したように、実部＝0 という軌跡は $\theta = \pi/2\,n$ の線上を、原点の方に向かい、途中で向きを変えて、$\theta = -\pi/2\,n$ という軸上を辿っていた。そこで、$|z| \to \infty$ という極限では正の実軸上にある、虚部＝0 の軌跡を原点に向けて辿っていこう。この軌跡も、始めの内は $|z|$ が小さくなる方向に辿れるが、途中から向きを変えて $|z|$ が大きくなる方向に向かう。$|z|$ が十分に大きくなった暁にはこの軌跡の偏角は $\pm\pi/n$ のどちらかに向かわなければならない。どちらに向かうにせよ、上で仮定した実部＝0 という軌跡と交わる。この交点では、実部と虚部が同時に 0 になっているから、$f(z) = 0$ の解である。

関数の連続性を利用した非常に直観的な分かりやすい証明である。n 次方程式の解を計算する時にも、大局的にはこの証明のイメージを持っているのが良いと思う。

一般の高次方程式を解く時に、この証明に登場した知識を使用する事があり得る。大きな $|z|$ を持って来て、実部が確実に 0 になる様に $\theta = \pi/(2\,n)$ とし、実部が 0 となるのを確認しながら $|z|$ を減らし、虚部が 0 となる位置を探すとその位置が方程式の解である。原理的には正しいが、なかなかつき合い切れない手法である。$|z|$ が大きいとはどういう事か、少

しずつ $|z|$ を減らすとはどの程度変化させる事なのかといった疑問には答えられない。やはりグラフを描きながら、自分の目で確かめるとしか言えない。

一つの解 z_1 が見付かると、θ を $2\pi/n$ だけ増やして最初から計算をやり直すか、方程式を $(z - z_1)$ で割って、次数を一つ減らした方程式を解くかも判断のしようが困難である。直観的には次数を減らしたくなるが、問題の質が変化する場合もあるらしい。即ち、割算の結果係数の精度が正しく保持されているかどうかを確認する必要があるという事らしい。

n 次方程式の n 個の解を並列的に全部一度に求める方法として、DKA 法がある。簡易型ニュートン法とでも呼んでいいのだろう。収束の最終段階では、ニュートン法と同じ様に 2 次収束であるから、効率の良い方法である。

ニュートン法と同じ様に、初期値をいくらにするかという問題があるが、かなり安易に計算できる方法が提案されていて、それなりに旨く働く様である。

解くべき方程式を $a_n\, z^n + \cdots + a_1\, z + a_0 = 0$ とし、解を $z_i,\ (i = 1, \cdots, n)$ と書くと、解と係数の関係から、解の重心は $\sum_i z_i/n = -a_{n-1}/a_n$ にあるから、この点を中心とする半径 R の円周上に初期値をとる。R は大きめにとる事にして、方程式の全ての項が相加的に働いたと仮定し、最高次の項が、これを全部負担すると考えてみる。即ち、次の式で R を決める。

$$|a_n|\, R^n = \sum_{i=0}^{n-2} |a_i|\, R^i$$

重心を移した後では、z^{n-1} の係数は 0 になっているから、和は $n - 2$ 迄で良い。この式は、以下の様に変形すると、R の上限を推定出来る。

$$|a_{n-2}/(a_n\, R^2)| + |a_{n-3}/(a_n\, R^3)| + \cdots + |a_0/(a_n\, R^n)| = 1$$

左辺の各項が 1 となれば、確かに R の上限であるから、$|a_{n-i}/(a_n\, R^i)| = 1$ を満足する R の最大値を採用すれば、残りの項は 1 よりも小さいから、

$$\hat{R} = m \max_i(|a_{n-i}/a_n|^i)$$

は、確かに R の上限である。ここで、 m は a_i が 0 でない項の数である。解の初期値は、この推定値では大きくなる傾向にある。上で決定した R よりも少し小さめの値 (例えば 8 割) 程度とする事も考えられる。

次に、角度は円周上に初期値が均等に散らばる様にすると、$2\pi/n$ 間隔でばらまける。但し、角度 0 ラディアンは特殊すぎるから避ける事として、$3/2\, n$ を全体的に回転しておく。

これで、n 個の初期値は以下の様に決まった。

$$z_k = -a_{n-1}/(n\, a_n) + R \exp\left(2\, i\, k\, \pi/n + 3i/2\, n\right)$$

これらの解の候補を次の改良公式を適用して、逐次改良していく。

$$(\text{新})z_k = z_k - \frac{f(z_k)}{\Pi_{j \neq k}(z_k - z_j)}$$

修正量が少なくなったら、収束した事にする。右辺の修正量は、正解に近付いた時には、ニュートン法の修正量そのものになっている。

注意： $|a_n|$ が非常に小さいならば、(1) 定数項が大きければ、$1/x$ に関する方程式を先ず解いて、1/x を元に戻すと言う手もある。(2) そのように都合良く行っていないならば、$n-1$ 次方程式を解いてから、解を修正する方が良いかも知れない。

収束判定

収束判定は、補正量が小さくなったらという事が考えられる。一方、関数値の計算途中に登場する最大値の有効数字の桁数から推定するという方法も考えられる。

解の改良過程で、一部の解は他の解よりも急速に収束する場合がある。例えば、実根と複素根が混じっている時、実根だけが先に収束するという経験がある。収束したと確認出来る解を更に改良する必要は無いという考えもあり得る。その時には、収束した解から順番に改良公式の適用をやめると、計算時間が少しは短縮されるだろう。以下の例では、DO 22 I=N0,N と書いてある部分が、この処理に関係する。実際の処理は、５５ の書式番号以降で行っている。

重根があると、修正量を計算する分母が０になるので困る事になりはしないだろうか？

根の改良公式を少し変更して、収束率を２次でなく３次にする工夫もなされている様である。

z_i に対する修正量を $-\dfrac{\dfrac{f(z_i)}{f'(z_i)}}{1-\dfrac{f(z_i)}{f'(z_i)}\displaystyle\sum_{j\neq i}\frac{1}{z_i-z_j}}$ とすれば良い様である。

以下の DKA 法の具体的なプログラムを載せておこう。$\hat{R}$ の評価は乱暴すぎるという意見がある。

```
C           test of DKA
      PARAMETER (NODR=10)
      IMPLICIT REAL*8 (A-H,O-Z)
      COMPLEX*16 A(0:NODR), S(NODR,2), Z, C0, C1, B(0:NODR)
      REAL*4 RAN(500)
      II=-1
      IRN=500
      CALL RAND2(II,IRN, RAN)
      IR=0
      DO 20 KK=1,40
      WRITE(6,12)KK
   12 FORMAT(I3,'-th problem ')
      DO 11 M=0,NODR
```

```
      IR=IR+1
   11 A(M)=RAN(IR)*2-1
      DO 1 M=0,NODR
    1 B(M)=A(M)
      CONV=1.0E-12
C      IF(KK.LE.22)GOTO 20
      WRITE(6,123)A
  123 FORMAT('A-table='/(1P4E15.7))
      CALL DKA(NODR, B, CONV, S)
      DO 5 I=1,NODR
      C0=A(0)
      Z=S(I,1)
      C1=1
      DO 2 J=1,NODR
      C1=C1*Z
    2 C0=C0+C1*A(J)
      WRITE(6,3)I,Z,C0
    3 FORMAT(I2,1P4E15.7)
    5 CONTINUE
   20 CONTINUE
      STOP
      END
      SUBROUTINE DKA(N, A, CONV, S)
C         solve $\sum_{i=0}^N a_i\,z^i =0$ by DKA method
C         where a_n=1 is assumed
C Input N: order
C       A(0:N): complex coefficients
C       CONV: convergence reference
C Output S(N): complex solution
      IMPLICIT REAL*8 (A-H,O-Z)
      COMPLEX*16 A(0:N), S(N,2), C0, C1, C2, Z, ZS
      RCA(Z)=ABS(DBLE(Z))+ABS(DIMAG(Z))
      CABS(Z)=SQRT(DBLE(Z)**2+DIMAG(Z)**2)
C       swap head and tail ?
      ISDEC=1
      F1=RCA(A(0))
      F4=RCA(A(N))
```

```
      IF(F4.GE.F1) GOTO 3
      ISDEC=0
      DO 1 I=0,N
    1 S(I,1)=A(N-I)
      DO 2 I=0,N
    2 A(I)=S(I,1)
C       normalize A(N) ==> 1
    3 Z=1/A(N)
      DO 4 I=0,N-1
    4 A(I)=A(I)*Z
      A(N)=1
C       modify equation by shifting z-a_{n-1}/n
      ZS=-A(N-1)/N
      DO 6 I=0,N-1
      C1=0
      C2=1
      DO 5 J=I,N
      C1=C1+A(J)*C2
    5 C2=C2*ZS*(J+1.0D0)/(J+1-I)
    6 A(I)=C1
C   initial guess of radius R
      R=0
      DO 11 I=0,N-2
      F1=(N*CABS(A(I)))**(1.0D0/(N-I))
   11 IF(F1.GT.R)R=F1
      I=0
      WRITE(6,51)I,R
   51 FORMAT(I4,'-th guess of R =',1PE14.5)
C        decrease R by Newtonial method
      LMAX=100
      DO 13 I=1,LMAX
      F1=0
      F2=0
      F3=1
      DO 12 J=1,N-2
      F4=F3*CABS(A(J))
      F1=F1+  F4
```

```
      F2=F2+J*F4
   12 F3=F3*R
      F3=F3*R
      F2=F2-F3*N
      F1=CABS(A(0))+(F1-F3)*R
      F3=F1/F2
      R=R-F3
      WRITE(6,51)I,R
   13 IF(ABS(F3).LT.1.0E-3)GOTO 14
      GOTO 91
C        initial values
   14 F1=1.5D0/N
      F2=2*3.14159265D0/N
      C1=R*CMPLX(COS(F1), SIN(F1))
      C2=  CMPLX(COS(F2), SIN(F2))
      DO 15 I=1,N
      S(I,1)=C1
   15 C1=C1*C2
C        define convergence reference
      F2=R*CONV
      L=0
C      WRITE(6,53)L,F2,(I,S(I,1),I=1,N)
      WRITE(6,53)L,F2
   53 FORMAT(I4,'-th stage  F2 or R=',1PE13.4:/('S(',I3,')=',1P2E15.7))
C       start of iteration
      K=1
      N0=1
      DO 23 L=1,LMAX
      R=0
      M=3-K
      DO 22 I=N0,N
      Z=S(I,K)
      C0=A(0)
      C1=1
      C2=1
      DO 21 J=1,N
      C1=C1*Z
```

```
      C0=C0+A(J)*C1
      IF(J.NE.I)C2=C2*(Z-S(J,K))
   21 CONTINUE
      C1=C0/C2
      S(I,M)=Z-C1
      F1=RCA(C1)
      F3=RCA(S(I,M))
      F4=RCA(S(I,K))
      F4=ABS(F4-F3)/(F3+F4)
      F1=MIN(F1,F4)
      IF(R.LT.F1)R=F1
      IF(F1.GT.F2)GOTO 22
C        job is over for the I-th solution
      WRITE(6,55)I,N0
   55 FORMAT(I3,'-th sol. converged exchange by ',I3,'-th sol.')
      C1=S(N0,M)
      S(N0,1)=S(I,M)
      S(N0,2)=S(I,M)
      IF(N0.EQ.N) GOTO 25
      S(I,M)=C1
      N0=N0+1
   22 CONTINUE
      WRITE(6,53)L,R,(I,S(I,M),I=1,N)
C      WRITE(6,53)L,R
      K=3-K
   23 CONTINUE
      GOTO 93
C        recover the shifted origin
   25 DO 26 I=1,N
   26 S(I,1)=S(I,K)+ZS
      IF(ISDEC.EQ.1)RETURN
      DO 27 I=1,N
   27 S(I,1)=1/S(I,1)
      RETURN
C
   91 WRITE(6,92)
   92 FORMAT('(DKA) No conv. at the 1st stage !')
```

```
      STOP
   93 WRITE(6,94)LMAX
   94 FORMAT('(DKA) No conv. at 2nd stage Lmax=',I4)
      STOP
      END
```

ここには、乱数発生のプログラムが来る。

高次の有理係数方程式を解く方法には、他に二つ有力と思える手法がある。次にそれらを、簡単に記しておこう。

因数分解法

与えられた方程式の最高次の係数を 1 だとして、方程式を書き直す。

$$f(x) = x^n + b_{n-1}\,x^{n-1} + \cdots + b_1 x + b_0 = 0$$

$n > 2$ だとすると、この方程式は複素数の範囲で必ず解を有する。方程式の係数が実数ならば、その複素共役もやはり解であるから、実係数の 2 次式を積の因子として持つ。即ち、

$$f(x) = (x^2 + p\,x + q)(x^{n-2} + c_{n-3}\,x^{n-3} + \cdots + c_1\,x + c_0) = 0$$

この因数分解が実行できるならば、2 次方程式を解く問題に帰着できるから、問題の複雑度を減らす事が出来る。因数分解した残りの方程式にも、同じ手法は繰り返し適用できるので実質的に問題は解けた事になる。結局、如何にして展開係数 $p,\ q$ を決定するかという事に尽きる。この作業を続けるには、問題を二つに分けるのが良いだろう。一つは、$p,\ q$ の初期値を如何にして見付けるか？であり、二つ目は如何にして初期値を改良するか？

最初の問題に対しては、良い指導原理を知らない。仕方がないから、先に DKA 法で用いた、根の最大値の上限 R と重心を根とする 2 次方程式を採用しよう。これが良い方法かどうかは知らない。少し経験を積む必要がある。

第 2 の問に答えるのは比較的簡単であり、多くの参考書に書いてあるとおり、2 次元のニュートン法を使用すれば良いだろう。

行列の固有値問題に帰着する方法

もう一度方程式を書いておこう。

$$x^n + b_{n-1}\,x^{n-1} + \cdots + b_1 x + b_0 = 0$$

この方程式の解は、次の行列の固有値と一致する。

$$\begin{pmatrix} -b_{n-1} & -b_{n-2} & \cdots & -b_1 & b_0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ & & \cdots & & \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

最初にこの事実に気付いたのは誰だろう？

さてこの事実を確認する必要がある。2次と3次の方程式に対して固有値問題を定義する行列式を展開してみると、$(-1)^{n-1}$ の因子を除いて、問題に一致するから、なる程と納得が出来るだろう。

DKA 法では、本当に x^n を計算するから、非常に大きな範囲の数値を同時に扱わなければならない。一方、後で述べた二つの解法では、高次方程式の係数だけを相手にしている。勿論、冪乗の計算は実質的に解法のどこかでしている訳だが、表には見えない。

一般の方程式 $F(x) = 0$ を解くにはどうしたら良いのだろう？

解の存在が証明されているかどうかを先ず問う必要がある。次には、この節の最初の方に書いた指針を見るのが妥当だろう。

グラフを描く、解の存在範囲を推定する、2分法やニュートン法を適用する。等を覚えておけば良いだろう。

8．線形計算

ここでは、良く利用されるであろう線形計算から、次の項目を取り上げる。

1　ベクトルの直交化
2　連立1次方程式の解法
3　行列の固有値と固有ベクトル
4　線形最小2乗法

1．Gram-Schmidt の直交化

N 元の列ベクトルが N 個、 $\mathbf{a}_i$ $(i = 1, \cdots N)$、 与えられたとする。これらのベクトルを Gram-Schmidt の手法を用いて正規直交化する。但し、与えられたベクトルは線形独立である事を仮定する。

最初の方の手続きを書き下すと以下の様になる。

1　$\mathbf{a}_1$ を規格化する。即ち、内積の平方根 $R_{1\,1} = \sqrt{(\mathbf{a}_1, \mathbf{a}_1)}$ を計算し、これで割る。
$\mathbf{v}_1 = \mathbf{a}_1, \quad \mathbf{q}_1 = \mathbf{v}_1 / R_{1\,1}$

2　$\mathbf{a}_2$ から $\mathbf{q}_1$ 成分を除き、規格化する。

$$R_{1\,2} = (\mathbf{q}_1, \mathbf{a}_2), \quad \mathbf{v}_2 = \mathbf{a}_2 - \mathbf{q}_1 R_{1\,2}, \quad R_{2\,2} = \sqrt{(\mathbf{v}_2, \mathbf{v}_2)}, \quad \mathbf{q}_2 = \mathbf{v}_2 / R_{2\,2}$$

3　$i \geq 2$ に対する一般式は以下の通り。
j=1 から j=(i−1) に対し、

$$R_{j\,i} = (\mathbf{q}_j, \mathbf{a}_i), \quad \mathbf{v}_i = \mathbf{a}_i - \sum_{j=1}^{j=i-1} \mathbf{q}_j R_{j\,i}$$

ベクトル $\mathbf{v}_i$ を規格化する。

$$R_{i\,i} = \sqrt{(\mathbf{v}_i,\, \mathbf{v}_i)}, \quad \mathbf{q}_i = \mathbf{v}_i / R_{i,i}$$

こんな書き方も可能である。$\mathbf{a}_i \to |i>$、内積を $(\mathbf{a}_j, \mathbf{a}_i) \to < j|i >$ と記号化する。

$$(new)|i> = (old)|i> - \sum_{j=1}^{i-1} |j>< j|i>$$

$$(new)|i> = |i> / \sqrt{< i|i >}$$

別の書き方をしてみよう。最初に与えられたベクトルを並べて N×N 行列 A を作る。更に $R_{i\,j}$ で成分が定義される右上三角行列を R、列ベクトル $\mathbf{q}_i$ $(i = 1, \cdots, N)$ を並べて行列 Q を作ると、次の関係がある。

$$A = Q\,R$$

即ち、行列 A は直交行列 Q と 右上三角行列 R の積に分解された。このアイデアは、QR 法の説明で、忘れなければ、もう一度登場させる。

注意深い学生へのメッセージ
N 個のベクトルが与えられた時、この N 個のベクトルの順番を如何に並べ変えてもこの直交化の手法は成立する。ならば、どのような並べ替えをするのが数値計算の意味で有効か？という問が意味を持つ。2 元を持つベクトルで、問題点をもう少し説明しておこう。直交化の結果、ある場合には $\mathbf{e}_1 = (1,0)^T,\ \mathbf{e}_2 = (0,1)^T$ が得られ、他の場合には $\mathbf{n}_1 = (1/\sqrt{2}, 1/\sqrt{2}),\ \mathbf{n}_2 = (1/\sqrt{2}, -1/\sqrt{2})$ が得られたとする。数学的にはどちらの系も規格直交であるが、後者の場合直交性は、$0.5 - 0.5 = 0$ という式で与えられ、前者では $0 - 0 = 0$ という式で与えられる。明らかに前者の方が誤差が少ない数値で表現されるので、得られたベクトルの精度が高い。

多分、大きなノルムのベクトルから順番に番号をつけ替えておくのが妥当だろう。

以下に、実際のプログラムを示す。記憶領域を節約するために, 行列 A と Q は重ね書きされている. 行列 R はここでは保存していない。

```
      PARAMETER (N=10)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(N,N)
      REAL*4 RAN(N*N)
C   generate random numbers
      I1=-1
      NR=N*N
      CALL RAND2(I1, NR, RAN)
      I1=1
```

```
C       define matrix A
        DO 1 I=1,N
        DO 1 J=1,N
        A(J,I)=DBLE(RAN(I1))
      1 I1=I1+1
C         orth-normalize
        CALL GSORTH(N, A)
C           check it
        DO 4 I=1,N
        DO 4 J=1,I
        S=0
        DO 2 K=1,N
      2 S=S+A(K,J)*A(K,I)
        WRITE(6,3)I,J,S
      3 FORMAT('I=',I2,' J=',I2,'  S=',1PE15.7)
      4 CONTINUE
        STOP
        END
        SUBROUTINE GSORTH(N,A)
C       Gram-Schmidt orthogonaliztion
        IMPLICIT REAL*8 (A-H,O-Z)
        DIMENSION A(N,N)
        DO 5 I=1,N
C           |i> -= |j><j|i>
        DO 2 J=1,I-1
        S=0
        DO 1 K=1,N
      1 S=S+A(K,J)*A(K,I)
        DO 2 K=1,N
      2 A(K,I)=A(K,I)-A(K,J)*S
C       orthogonalize
        S=0
        DO 3 K=1,N
      3 S=S+A(K,I)**2
        S=1/SQRT(S)
        DO 4 K=1,N
      4 A(K,I)=S*A(K,I)
      5 CONTINUE
        RETURN
        END
```

連立１次方程式の解法

N 行 N 列の正方行列 A と N 元の列ベクトル $\mathbf{x}$ と $\mathbf{b}$ に対して

$$A\mathbf{x} = \mathbf{b}, \quad det(A) \neq 0$$

が成立し、A と $\mathbf{b}$ が既知ならば、この連立方程式を解けという問題が生ずる。原理的には, 行列 A の 行列式 $|A|$ が 0 でなければ逆行列 A^{-1} が存在し、$\mathbf{x} = A^{-1}\mathbf{b}$ で解は書き下せる。従って、逆行列 A^{-1} が計算出来れば、問題は解けた事になる。

N 個の異なる基本単位ベクトルを $\mathbf{b}$ だと考えて解いた N 個の解ベクトルを並べて N 行 N 列の行列 X を作ると、これが行列 A の逆行列になっている。従って、逆行列の計算は連立１次方程式の解法に含まれる。

数値計算の教科書には、計算資源の浪費以外の何者でもないからという理由で逆行列の計算を禁じる (戒める) 場合が多い。個人的には、自分が書いたプログラムのテストをするために、逆行列を計算するのが早計であると感ずる場合も多い。

ここで問題とした連立１次方程式は ガウスの掃き出し法を利用すると、簡単に解く事が可能である。この同じ問題は、LU 分解という手法でも解く事が可能であり、こちらの方が利用可能性が大きそうだから、こちらを説明する。

実行列の LU 分解と線形方程式の解法

次式で与えられる実の N 元 線形方程式を解く事を考える。

$$A\,\mathbf{x} = \mathbf{b}$$

まず、行列 A を 左下三角行列 L と 右上三角行列 U の積で表現する。但し、L の対角要素は 1 と仮定しておく。$A = LU$ の第 (i, j) 要素を書き下す事により、

$$A_{i,j} = \begin{cases} L_{i,1}U_{1,j} + \cdots L_{i,i}U_{i,j} & (i \leq j) \text{ の時} \\ L_{i,1}U_{1,j} + \cdots L_{i,j}U_{j,j} & (i \geq j) \text{ の時} \end{cases}$$

$i = j$ の時は、上の式も下の式も成立する。この式から、次の漸化式で全ての L と U の成分は計算可能である。

$i = 1$ から $i = N$ まで

j=$i + 1$ から $j = N$ まで (即ち第 i 行ベクトルの成分に対し)、

$$U_{i,j} = A_{i,j} - \sum_{k=1}^{k=i-1} L_{i,k}\,U_{k,j}$$

$j = i$ から、$j = N$ まで (第 i 列成分)、

$$\hat{L}_{j,i} = U_{j,j}\,L_{j,i} = A_{j,i} - \sum_{k=1}^{k=j-1} L_{j,k}\,U_{k,i}$$

L の対角成分を 1 とする作業をする前に、情報落ち対策として、枢軸選びを行う。もしも $U_{j,j} = 0$ ならば、$L_{j,i}$ 行列要素を計算する手続きは破綻する。では、$U_{j,j}$ は 0 では無いが非常に小さな数だと？ この場合には計算は出来るが誤差が大きくなる。その極限として、0 の場合には計算が出来なくなると了解しておく。精度を保つには、$U_{j,j}\, L_{j,i}, \quad (j = i, i+1, \cdots N)$ の中で、絶対値が最大の要素を選び、第 i 行と 第 m 行を入れ換える。この操作を部分的枢軸選びと呼ぶ。

第 i 行ベクトルと 第 m 行ベクトルの全ての成分を入れ換える。列ベクトル $L_{n,i},\ (n = i+1, \cdots N)$ を T で割る。
後の便利のために、$U_{i,i}$ に $1/T$ を保存しておく。

ここまでの作業を全ての i に対して行う。

行列 L, U の各成分は 行列 A に上書き出来る。但し、L の対角要素 1 は、どこにも書き込まれていない。U の対角要素は、その逆数が書き込まれている。
行列 A の行列式の値 $(det(A))$ は、上で登場した T の積に、行を入れ換えた回数に応じて 負号を掛ければ計算出来る。

次に線形方程式を解く。

$$A\,\mathbf{x} = L\,U\,\mathbf{x} = \mathbf{b}$$

であるから、次の二つの線形方程式を解けば良い。

$$L\,\mathbf{y} = \mathbf{b}, \qquad U\,\mathbf{x} = \mathbf{y}$$

先の方程式を $\mathbf{y}$ に関して解くには、前進代入法を利用する。即ち、先ず $y_1 = b_1$ である。
$i = 2$ から $i = N$ 迄、次の手順で y_i を解く。

$$y_i = b_i - \sum_{j=1}^{i-1} L_{i,j}\, y_j$$

但し、A を LU 分解した時に行に関する入れ換えを行ったので、$\mathbf{b}$ もこれに対応した入れ換えを行う必要がある。

後半の線形方程式は、逆代入法で簡単に解ける。即ち、先ず $x_N = y_N/U_{N,N}$ である。
$i = N-1$ から $i = 1$ の順に次の漸化式で x_i を計算する。

$$x_i = \{y_i - \sum_{j=i+1}^{N} U_{i,j}\, y_j\}/U_{i,i}$$

ここで、先に U の対角要素の逆数を保存する理由が分かる。

$\mathbf{b}$ が消えて良いならば、ベクトル $\mathbf{x}, \mathbf{y}, \mathbf{b}$ は重ね書きが可能である。通常は消えても良い事にしておいて、必要ならば、副プログラムを呼び出す前に、コピーをつくっておく。

以下に例示するプログラムでは、計算途中で行成分を直接入れ換えている。ある種の本には、成分の入れ換えは大きな計算量を伴うので、実際には入れ換えずに、どの行とどの行を入れ換えた事にする様なプログラムを推奨している。N＝１００程度 (もっと大きかったかな？) でこのような間接番地指定の場合と例示したプログラムの様に、直接番地指定のプログラムの計算時間を比較してみた。結果は、例示した直接番地指定の方が有意に早かった。

解の改良

問題 $A\mathbf{x} = \mathbf{b}$ を解いて、近似解 $\mathbf{x}_0$ を得たとする。即ち、$A\mathbf{x}_0 = \mathbf{b} + \delta\mathbf{b}$ という事である。数値計算であるから、右辺は $\mathbf{b}$ とならず幾らか誤差 $\delta\mathbf{b}(= A\mathbf{x}_0 - \mathbf{b})$ を伴う。そこで、$A\mathbf{x}_1 = \delta\mathbf{b}$ を解き、$\mathbf{x}_2 = \mathbf{x}_0 - \mathbf{x}_1$ を作ると、この $\mathbf{x}_2$ は $\mathbf{x}_0$ よりも精度の高い近似解となっている。

LU 分解していると、同じ行列 A の連立１次方程式を解くのが非常に簡単になる。ガウスの掃き出し法ではこんな芸当は簡単でない。

条件数の推定

与えられた問題に依って、解の精度は上げられたり上げられなかったりする。解の精度の目安を与えるのに、条件数を使用する。

$$cond(A) \equiv ||A||\,||A^{-1}||$$

即ち、正則行列 A の条件数 $cond(A)$ は、行列 A のノルム ($||A||$) とその逆行列のノルム ($||A^{-1}||$) の積である。条件数は１以上の数であり、これが大きいほど誤差が大きい (解きにくい)。条件数が大きい問題は、条件が悪い (ill conditioned) と言われる事がある。

以下の行列ノルム (フロベニウスノルムでは無い！) を定義する。

$$||A||_1 = \max_j \left(\sum_{i=1}^{N} |A_{i,j}| \right)$$

このノルムに対する条件数の推定値は、以下の様にして可能と言う事だ。

先ず、全ての成分の絶対値が１であるようなベクトル $\mathbf{e}$ を用意し、以下の方程式を解く。

$$A^T \mathbf{y} = \mathbf{e}$$

ベクトル $\mathbf{y}$ の成分の内、絶対値が最大のもの y_m と、ノルムとの積を条件数の推定値として採用する。

この方程式は、上の場合と同じようにして解く事は可能である。

$$A^T\mathbf{y} = U^T\, L^T\, \mathbf{y} = \mathbf{e}$$

$\mathbf{e}$ の各成分の符号の不定性があるから、2^{N-1} の自由度が残る。この自由度を殺すために、$L^T\,\mathbf{y}=\mathbf{v}$ と置いで、$U^T\,\mathbf{v}=\mathbf{e}$ を解く時、$\mathbf{v}$ の各成分の絶対値が大きくなる様に、$\mathbf{e}$ の各成分の符号を選ぶ。

フロベニウスノルムを利用するならば、A の固有値の絶対値が最大 (小) のものを λ_{max} (λ_{min}) とすると、$cond(A)=\lambda_{max}/\lambda_{min}$ と表せる。条件数は、高精度を要求される事は稀だから、λ_{max} は冪乗法、λ_{min} は逆反復法を数回繰り返して推定出来る。

連立 1 次方程式 $A\mathbf{x}=\mathbf{b}$ が与えられたとする。行列 A の固有値 λ_i とこれに対応する固有ベクトル $\mathbf{u}_i$ を考える。$\mathbf{x}=\sum_i x_i\mathbf{u}_i$, $\mathbf{b}=\sum_i b_i\mathbf{u}_i$ と展開すると、$x_i=b_i/\lambda_i$ により、解は表現できる。$\mathbf{b}$ に対し、固有値の大きさで決まる有効数字だけ、x_i は影響を受ける。従って、解の有効数字は最大と最小固有値の比の絶対値分だけ影響を受ける。例えば、$\lambda_{max}/\lambda_{min}=10^{10}$ ならば、解ベクトルのある成分は、他の成分よりも１０桁程度精度が落ちている。この様な問題は解きにくい。従って解きやすさの目安として、固有値の絶対値が最大のものと最小のものの比を考えるのが良い。これが、条件数を導入する直観的な意味である。

以下の LULEQ 副プログラムは、この考え方で書かれている。

引数とその意味を記録しておこう。

CALL LULEQ(N, A, M, B, IP, D, CN, W)

N　問題の次数

A　行列 A(N,N) で定義される。出力時には、左下半分は L、右上半分には、U が返される。但し、L の対角成分は 1 であるが、これはどこにも書かれていない。U の対角成分の位置には、U の対角成分の逆数が返される。

M　解くべき、B の列ベクトルの数

B　B(N、M) として、定義される。出力時には、解ベクトル (列ベクトル) が返される。

IP　部分的な枢軸選びに使用する、N 元の整数型ベクトル

D　行列 A の行列式の値が返される。

CN　行列 A の条件数の推定値が返される

W　N 元の作業用ベクトル。

```
C
      PARAMETER (N=4)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION IP(N), A(N,N), B(N,2), W(N)
C         define matrix A
      DO 2 I=1,N
      DO 1 J=1,N
    1 A(J,I)=0
    2 A(I,I)=4
      DO 3 I=2,N
      A(I-1,I)=1
```

```
    3 A(I,I-1)=1
C define vector B
      DO 6 L=1,2
      DO 5 J=1,N
      S=0
      DO 4 K=1,N
    4 S=S+A(J,K)*K*L
    5 B(J,L)=S
    6 CONTINUE
      CALL PRNTA(N,A)
      DO 7 L=1,2
    7 WRITE(6,124)(B(I,1),I=1,N)
      CALL LULEQ(N, A, 2, B, IP, D, CN, W)
      CALL PRNTA(N,A)
      WRITE(6,122)D, CN
  122 FORMAT('D=',1PE14.5,2X,'CN=',1PE13.4)
      DO 8 L=1,2
    8 WRITE(6,124)(B(I,L),I=1,N)
  124 FORMAT(1P5E15.7)
      STOP
      END
      SUBROUTINE PRNTA(N,A)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(N,N)
      DO 3 I=1,N
      WRITE(6,1)I,(A(I,J),J=1,N)
    1 FORMAT(I2,2X,5F8.4)
    3 CONTINUE
      WRITE(6,4)
    4 FORMAT()
      RETURN
      END
      SUBROUTINE LULEQ(N, A, M, B, IP, D, CN, W)
C  solve m set of real linear equations A(N,N)*X(N,M)=B(N,M)
C  by using L/U decomposition of matrix A
C Input N: size of the problem
C       A(N,N): matrix to be decomposed
C       M     : number of vectors
C       B(N,M): M vectors
C Output A(N,N): diag. of L=1
```

```
C be careful      U(i,i) ==> 1/U(i,i):
C       B(N,M) : solution vectors
C       IP(N): pointer for pivotting
C       D    : det(A)
C       CN   : estimated condition number
C working area W(N)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(N,N), B(N,M), IP(N), W(N)
      TINY=1.0D-20
C       1st Norm
      FNORM=0
      DO 2 I=1,N
      S=0
      DO 1 J=1,N
    1 S=S+ABS(A(J,I))
    2 IF(S.GT.FNORM)FNORM=S
      D=1
      DO 10 I=1,N
      DO 4 J=I+1,N
      S=A(I,J)
      DO 3 K=1,I-1
    3 S=S-A(I,K)*A(K,J)
    4 A(I,J)=S
      T=0
      DO 6 J=I,N
      S=A(J,I)
      DO 5 K=1,I-1
    5 S=S-A(I,K)*A(K,J)
      A(J,I)=S
      S=ABS(S)
      IF(S.LT.T)GOTO 6
      T=S
      L=J
    6 CONTINUE
      IP(I)=L
      D=D*T
      IF(L.EQ.I)GOTO 8
      D=-D
C       swap row(I) and row(L)
      DO 7 J=1,N
```

```
      S=A(I,J)
      A(I,J)=A(L,J)
    7 A(L,J)=S
    8 IF(T.LT.TINY)GOTO 91
      T=1/A(I,I)
      DO 9 J=I+1,N
    9 A(J,I)=A(J,I)*T
   10 A(I,I)=T
C      estimate cond. number CN
      W(1)=A(1,1)
      DO 12 I=2,N
      S=0
      DO 11 J=1,I-1
   11 S=S-W(J)*A(J,I)
   12 W(I)=(S+SIGN(1.0D0,S))*A(I,I)
      CN=ABS(W(N))
      DO 14 I=N-1,1,-1
      S=W(I)
      DO 13 J=I+1,N
   13 S=S-W(J)*A(J,I)
      W(I)=S
      S=ABS(S)
      IF(S.GT.CN)CN=S
   14 CONTINUE
      CN=CN*FNORM
C           solve M set of linear equations
      DO 24 K=1,M
      DO 22 I=1,N
      L=IP(I)
C        swap B(L,K) <==> B(I,K)
      S=B(L,K)
      B(L,K)=B(I,K)
      DO 21 J=1,I-1
   21 S=S-A(I,J)*B(J,K)
   22 B(I,K)=S
C       backward substitution
      DO 24 I=N,1,-1
      S=B(I,K)
      DO 23 J=I+1,N
   23 S=S-A(I,J)*B(J,K)
```

```
24 B(I,K)=S*A(I,I)
   RETURN
91 WRITE(6,92)I,T
92 FORMAT('at I=',' T=',1PE12.3,' too small !')
   STOP
   END
```

このプログラムでは、作業用のベクトル IP と W が使用された。IP は解の改良の場合のように、同一の行列 A に対し別の連立１次方程式を後から解きたいならば、呼出プログラムに返さなければいけないが、W は条件数の推定作業が終れば消えて無くなっても良い。この様な作業場所をわざわざ、呼出プログラムが用意しなければならないのは煩雑である。FORTRAN ７７では仕方が無いが、FORTRAN ９０以降ではこの W を呼出プログラムで用意する必要はない。副プログラム LULEQ の中で次の様にする。

１) 配列宣言のところで、ALLOCATABLE W(:) とする

２) 実行文の最初、又は、配列 W を利用する前に次の文を置く。

ALLOCATE (W(N))

３) W を使い終ったら、以下の文でこれを開放する。

DEALLOCATE(W)

１) は、可変長の１次元実数型配列をこの (副) プログラムで使用する事を宣言している。

２) は、この実行文 (ALLOCATE は関数を実行する実行文である) で、OS が実際に必要な配列を準備してくれる。

N が許容範囲に無ければ、エラーとなる。正常かエラーを知りたければ、W(N) の後ろに、もう一つのキーワード STAT と 変数を書く。

３) OS が準備してくれた記憶領域を、OS に返した事にする。

固有値と固有ベクトル

行列の固有値や固有ベクトルを計算したい場合がある。N 行 N 列で、行列式の値が０でない行列 A と単位行列 E が与えられた時、次の関係を満足する数値 λ と N 元列ベクトル $\mathbf{u}$ をそれぞれ、固有値及びその固有値に対応する固有ベクトルと呼ぶ。

$$(A - \lambda E)\,\mathbf{u} = 0$$

ここで右辺の０は、全ての成分が ０である N 元列ベクトルである。この式から固有値は次の N 次方程式の解であると言える。

$$det(A - \lambda E) = 0$$

従って重複度を考慮すると N 個の固有値があり、これに対応して固有ベクトルも N 個存在する。異なる固有値に属する固有ベクトルは直交するが、固有値が縮退 (重複) している時には、Gram-Schmidt の手法を用いて直交化する事が可能である。固有値と固有ベクトルを規定する式は線形であるから、通常は固有ベクトルのノルムは１に規格化する。

固有値方程式を解く前に、固有値方程式を固有値の意味で等価で、解きやすい方程式に変形する。ここでは、少し一般的な場合を紹介しておこう。

非対称複素行列の変形

与件 $N \times N$ 行列 C が与えられた時、これを三重対角行列に変換する。

先ず、二つのベクトル $\mathbf{x}_1$ と $\mathbf{y}_1$ を何とかして作る。但し、両者の内積は 0 で無い事。ここで、内積の定義は以下の通りとする。

$$(\mathbf{x}, \mathbf{y}) = \sum_i x_i^* \, y_i$$

上付の星印は複素共役を表す。この時、次の恒等式が成立する。

$$(C\,\mathbf{a}, \mathbf{b}) = (\mathbf{a}, C^\dagger \mathbf{b})$$

ここで、上付のダガー (†) は転置行列の複素共役を表す。

$1 \leq k \leq (N-1)$ なる k に対して、逐次、以下の関係式によりベクトルを作る。

$$\mathbf{x}_{k+1} = C\,\mathbf{x}_k - \sum_{j=1}^{k} \mathbf{x}_j \, \alpha_{j\,k}, \quad \mathbf{y}_{k+1} = C^\dagger \, \mathbf{y}_k - \sum_{j=1}^{k} \mathbf{y}_j \, \beta_{j\,k}$$

ここで、係数は $i \neq j$ に対してベクトルが直交する様に取られる。

$$(\mathbf{x}_i, \mathbf{y}_j) = 0, \quad if \quad i \neq j$$

即ち、

$$\alpha_{i\,j} = \frac{(\mathbf{y}_i,\, C\,\mathbf{x}_j)}{(\mathbf{y}_i, \mathbf{x}_i)}, \quad \beta_{i\,j} = \frac{(\mathbf{x}_i,\, C^\dagger\,\mathbf{y}_j)}{(\mathbf{x}_i, \mathbf{y}_i)}$$

証明は、数学的帰納法に依る。

$k = 1$ の時、

$$(\mathbf{y}_1, \mathbf{x}_2) = (\mathbf{y}_1, C\,\mathbf{x}_1 - \mathbf{x}_1\,\alpha_{1\,1}) = (\mathbf{y}_1, C\,\mathbf{x}_1) - (\mathbf{y}_1, \mathbf{x}_1)\,\alpha_{1\,1} = 0$$

$(\mathbf{x}_1, \mathbf{y}_2) = 0$ も同様に示される。

次に、ある k 以下では双直交性が成立していると仮定して、$k+1$ でも成立する事を示す。

$i < k$ として、

$$(\mathbf{x}_i, \mathbf{y}_{k+1}) = (\mathbf{x}_i, C^\dagger\,\mathbf{y}_k) - \sum_j (\mathbf{x}_i, \mathbf{y}_j)\,\beta_{j\,k} = 0$$

仮定に依り、右辺の和は $j = i$ の項だけが 0 でなく、この項は第 1 項と相殺する。

係数、$\beta_{i\,k}$ の定義から明らかだろう。

$$(\mathbf{y}_i, \mathbf{x}_{k+1}) = 0$$

も同様に導ける。

列ベクトル $\mathbf{x}_i$ の内独立なものは N 個存在する。これらの列ベクトルを順番に並べて、行列 X を作る。係数 $\alpha_{i\,j}$ $(i \leq j)$ の作る行列を H_α と書くと、次の行列に関する関係式が成立する。

$$C\,X = X\,H_\alpha, \quad X^{-1}\,C\,X = H_\alpha$$

最後の関係式から、行列 C と H_α の固有値は同じである事が言える。

更に次の関係を導く事が出来る。

$$\alpha_{k\,k} = \beta^*_{k\,k}, \quad \alpha_{k-1\,k} = \beta^*_{k-1\,k}$$

及び、$i+1 < j$ ならば

$$\alpha_{i\,j} = \beta_{i\,j} = 0$$

この関係式から、$\alpha_{i\,i} = \alpha_i$, $\alpha_{i-1\,i} = \beta_i$ と書くと、行列 H_α は以下の様に書ける。

$$H_\alpha = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ 1 & \alpha_2 & \beta_2 & 0 & \\ 0 & 1 & \alpha_3 & \beta_3 & 0 \\ & \ddots & \ddots & \ddots & \beta_{n-1} \\ & \cdots & 0 & 1 & \alpha_n \end{pmatrix}$$

$\alpha_{k\,k} = \beta^*_{k\,k}$ を導く。

$$\alpha_{k\,k} = \frac{(\mathbf{y}_k, C\,\mathbf{x}_k)}{(\mathbf{y}_k, \mathbf{x}_k)} = \frac{(C^\dagger \mathbf{y}_k, \mathbf{x}_k)}{(\mathbf{y}_k, \mathbf{x}_k)} = \frac{(\mathbf{x}_k, C^\dagger \mathbf{y}_k)^*}{(\mathbf{x}_k, \mathbf{y}_k)^*} = \beta^*_{k\,k}$$

$\alpha_{k-1\,k} = \beta^*_{k-1\,k}$ を導く。

$$\alpha_{k-1\,k} = \frac{(\mathbf{y}_{k-1}, C\,\mathbf{x}_k)}{(\mathbf{y}_{k-1}, \mathbf{x}_{k-1})} = \frac{(C^\dagger \mathbf{y}_{k-1}, \mathbf{x}_k)}{(\mathbf{y}_{k-1}, \mathbf{x}_{k-1})} = \frac{(\mathbf{y}_k + \sum_{j=1}^{k-1} \mathbf{y}_j\,\beta_{j\,k-1}\mathbf{x}_k)}{(\mathbf{y}_{k-1}, \mathbf{x}_{k-1})} = \frac{(\mathbf{y}_k, \mathbf{x}_k)}{(\mathbf{y}_{k-1}, \mathbf{x}_{k-1})}$$

$\beta_{k-1\,k}$ も同じような変形が出来、上式右辺の複素共役が登場する。

最後に $i+1 < j$ を仮定して、$\alpha_{i\,j} = \beta_{i\,j} = 0$ を導く。

$$\alpha_{i\,j}\text{の分子} = (\mathbf{y}_i, C\,\mathbf{x}_j) = (C^\dagger \mathbf{y}_i, \mathbf{x}_j) = (\mathbf{y}_{i+1} + \sum_{k=1}^{k=i} \mathbf{y}_k\,\beta_{k\,i}, \mathbf{x}_j) = 0$$

最後の式で、双直交性を用いた。

この変形の仕方と、Gram-Schmidt の直交化を比較してみよう。後者では、新しいベクトルを行列の列ベクトルとして取り出すところを、古いベクトルに行列をかけてつくり出している。複素数が対象だから、一つのベクトルでなく、二つ $C\mathbf{x}_k$, $C^\dagger \mathbf{y}_k$ 作っている。次に、過去のベクトルに平行な成分を抜きだしている。即ち、過去のベクトルと直交化している。ここでは、過去のベクトルを規格化していないので、新ベクトルと旧ベクトルのスカラー積をノルムで割っている。

これまでに, 線形代数の授業を受けただろう。各種の定理を証明する時に、過去のベクトルに線形独立なベクトルを作るのに、行列に過去のベクトルを掛ける事で、新しいベクトルをつくり出していた事を思い出そう。ここでも、その手法が使用されている。

新しいベクトルが、過去のベクトルと平行な成分を取り出すのに、以下の手法を用いていた。

(新ベクトル $\mathbf{b}$) ー (新ベクトルと旧ベクトルの内積,$(\mathbf{b}\cdot\mathbf{a_i})$) (旧ベクトル $\mathbf{a_i}$)

右辺の旧ベクトルに関係する部分、

$$\sum_i (?,\mathbf{a_i})\mathbf{a}_i$$

の部分をベクトル？にかかる演算子と考えて、射影演算子と呼ぶ。

有用な手法には、名前を付けておくと覚えやすい。これからの物理の学習にも登場するはずである。特に線形代数を利用する量子力学では、頻繁に登場する。注意しておくと、量子力学に興味が持てるかもしれない。

もしも対象とする行列 C が実対称行列ならば、どのような簡単化が出来るか考えてみよ。実用的には、実対称行列を相手にする場合の方が多いだろう。

このようにして、与えられた行列 C を三重対角の形を持つ行列 H_α に変換出来た。ここでは、ベクトル $\mathbf{x}$ とその係数 α が作る行列を採用したが、ベクトル $\mathbf{y}$ とその係数 β の作る行列を採用しても同様の議論が出来る。

固有値計算後 C の固有ベクトルを計算したいならば、ベクトル $\mathbf{x}$ は保存しておく必要がある。

C が実対称行列の場合には、別の手法もある。先ず elementary orthogonal matrix という概念を紹介しよう。この行列を、小山と山本は 素直交行列、一松 は基本直交行列と呼んでいる。

$\mathbf{u}$ を実単位列ベクトル、即ち $(\mathbf{u},\mathbf{u})=1$、とし、行列 U をつくる。

$$P = 1 - 2\mathbf{u}\mathbf{u}^T, \text{即ち } P_{i,j} = \delta_{i,j} - 2u_i\,u_j$$

この行列 P は直交行列である。

実行列を相手にしているとしよう。N 次実対称行列 A と直交行列を P を以下の様に分解しよう。

$$A = \begin{pmatrix} c & \mathbf{d}^T \\ \mathbf{d} & B \end{pmatrix}, \quad P_1 = \begin{pmatrix} 1 & 0 \\ 0 & Q \end{pmatrix}$$

ここで、$c = A_{1,1}$、$\mathbf{d}$ は行列 A の第 1 列ベクトルから、$A_{1,1}$ を除いて作られた (N−1) 次の列ベクトルである。Q は $N-1$ 次の基本直交行列であり、$\mathbf{u}$ は $(d_1-\lambda, d_2, \cdots, d_{N-1})^T$ に比例する単位ベクトルである。但し、$\lambda^2 = (\mathbf{d},\mathbf{d}) = \sum_{i=2}^{N} A_{i,1}^2$ であり、計算精度を確保するために、λ は d_1 とは異符号とする。

この時、P_1 で変換された行列 $A_1 = P_1\,A\,P_1$ の第 1 列ベクトルは、(対称行列だから第 1 行も) 最初の二つの成分だけが 0 でなく (c と λ)、残りの成分は全て 0 となる。

A_1 の右下の $N-1$ 行 $N-1$ 列行列を新しい A 行列だと看做して同様の操作を繰り返して行うと、(N−2) 回の直交変換で、A は対称な 3 重対角行列に変換される。

$$C = A_{N-2} = PAP = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & 0\cdots & \\ 0 & \beta_2 & \cdots & & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \beta_{N-1} & \alpha_N \end{pmatrix}$$

ここで、$P = P_1 P_2 \cdots P_{N-2}$ は一つの直交行列である。

直交変換であるから、A と A_{N-2} の固有値は等しい。この変換は、Householder 変換と呼ばれる場合もある。

固有値の計算

上で与えた操作に依り、かなり構造の単純な行列の固有値を計算する問題に帰着出来た。問題の次数が低い場合や、全部の固有値を計算する必要がなければ、ここで代数方程式を解くという選択もあり得る。そのような人への助言として、解の存在範囲に関して ゲルシュゴーリン (Gerschgorin) の定理が存在するという事を指摘しておこう。気が向けば、スペクトル半径という言葉も調べるのが良いだろう。

ここでは、全ての固有値を計算するという場合を想定し QR 法を採用する。

次の事実がある。

1) 与えられた行列 $C(= C_0)$ に対して、QR 分解を行い、これを逆順に掛ける。

$$C_0 = Q_1 R_1, \quad C_1 = R_1 Q_1$$

C_1 を C_0 だと考えて、同様の操作を繰り返し行う。

この時、C_m, $(m \to \infty)$ の主対角成分には左上から右下に向かって、固有値が絶対値の大きな順に並び、左下要素は 0 に収束する。証明は、例えば一松信「数値解析」(朝倉書店) ５１ページを見よ。

QR 法の原理的な手順は、Gram-Schmidt の正規直交化を思い起こせば良い。

2) C_0 が対称な 3 重対角行列ならば、C_1 も対称な 3 重対角行列である。

従って、何回 1) の操作を繰り返しても 3 重対角行列である。

3) ある定数 c と単位行列 E を用い、$C - cE$ という行列をつくる。新しい行列 $C - cE$ の固有値 μ と C の固有値 λ とは $\mu = \lambda - c$ の関係にある。

これは、自明すぎる事実である。

少し解説をしておこう。任意のベクトル $\mathbf{x}$ は、固有ベクトル $\mathbf{u}_i$, $(i = 1, 2, \cdots, N)$ で展開できる。

$$\mathbf{x} = \sum_i x_i \, \mathbf{u}_i$$

この式に左から行列 C を k 回掛けると、

$$C^k \mathbf{x} = \sum_i \lambda_i^k \, x_i \, \mathbf{u}_i$$

右辺は、k が増すと絶対値が一番大きな固有値に対応する成分が抜きん出て来る事を意味する。複数回行列をかけるだけでは、絶対値が最大の固有値しか顔を出さない。そこで、直交化を組み合わせると、次の大きさの固有値も独立に顔を出すようになる。繰り返しに対する収束率は、固有値の大きさの比になり、大きな固有値から順に並ぶのも推察出来るだろう。

収束率を上げる為には、最も小さな絶対値の固有値 λ_{min} を推定してこの推定値分だけ、全ての対角要素から差し引いた行列に対して、QR 法を適用すれば良い。

推定は、右下隅の 2 行 2 列の行列に対する固有値問題 (即ち 2 次方程式を) 解き、右下の行列要素が最小になる方の解を採用し、原点移動をするのが良いようである。 2 行 2 列を取り上げる理由は、固有値が複素数、絶対値が等しい場合を想定するからである。

何度か QR 法の過程を繰り返し、右下隅の値が収束すれば、この部分を切り離し、$N - 1$ 又は $N - 2$ 次の固有値問題を解く。

参考文献 戸川隼人著 マトリックスの数値計算 (オーム社 昭和４６年) pp 210

固有ベクトルの計算

固有値が計算出来ると、固有ベクトルを計算したくなる。これには、逆反復法が最適だろう。先ず、行列 C とその逆行列 C^{-1} の固有値は一方の逆数であり、固有ベクトルは同一であるという事実に着目し、逆行列を頭の中で取り上げる。先に計算されたのは、本物の固有値 λ_i に対する近似値 $\tilde{\lambda_i}$ だと考えて、未知の固有ベクトル $\mathbf{u}_i$ が満足する式を書く。

$$(C - \tilde{\lambda_i})^{-1} \mathbf{u}_i = \{1/(\lambda_i - \tilde{\lambda_i})\} \mathbf{u}_i = \mathbf{x}$$

係数 $1/(\lambda_i - \tilde{\lambda_i})$ は、時には発散するほどの極端に大きな数である。逆行列を右辺に移すと、

$$\mathbf{u}_i = (C - \tilde{\lambda_i}) \mathbf{x}$$

即ち、未知のベクトル $\mathbf{x}$ に右辺で定義された行列を掛けると、固有ベクトル $\mathbf{u}_i$ が計算できるという魔法の様な関係式が得られた。計算精度を上げるために、最初に得られた左辺のベクトルを規格化して、右辺の $\mathbf{x}$ の代用にすると良い。$x_i = 0$ という例外的な場合以外は、２ー３回このプロセスを繰り返すと収束するだろう。

複数の固有値がたまたま一致 (縮退) している場合には、その縮退の多重度だけ、異なる $\mathbf{x}$ から出発して固有ベクトルを計算し、後でこれらの固有ベクトルを直交化しておく。

固有ベクトルの計算には、元から与えられた行列 C を用いても良いし、H_α を用いても良い。C の固有ベクトルを並べて作った行列 U と H_α の固有ベクトルを並べて作った行列 V とは、$U = X\,V$ の関係にある。ここで行列 X は、先に C 行列を H_α に変形するのに導入された。

この固有値計算の部分は、プログラムが長くなるので例示は止めた。

最小２乗法

最小２乗法と関数の極小化

この項目の解説は、ほとんど実験学の講義ノートか計算機を用いた関数の極小化という項目に書いたので、実際的な側面だけを取り上げる。

独立変数 x の関数 $y(x)$ を取り上げる。この関数は、独立変数以外にある種のパラメータ、複数ある事を想定してベクトル $\mathbf{a}$ と書く、を含む。即ち、$y(x, \mathbf{a})$ という関数形の存在を仮定している。

独立変数の各種の値 $x_i,\ (i = 1, 2, \cdots, n)$ に対応する y_i の値を測定したとする。この y_i には誤差 e_i も分かっているとする。誤差が分からない時は、誤差は一定であると考える。更に、x の測定誤差は無視出来ると仮定する。これらの情報から、パラメータ $\mathbf{a}$ の値を推定する為に、最小２乗法を利用する。

最小２乗法では、以下に定義される S を最小にする パラメータの組みを見出せという問題だと了解する。

$$S \equiv \sum_i \left(\frac{y_i - y(x_i, \mathbf{a})}{e_i} \right)^2$$

解法は、理論式 $y(x, \mathbf{a})$ から線形な問題に還元できるか否かで変わって来る。ここでは、線形問題に還元出来る場合を想定しよう。

誤差の公理に、絶対値が大きな誤差は実質的に起こらないというのがある。実際には、大きな誤差を経験する事がある。この場合、非常に大きな誤差が一つでも紛れ込んでいると、上の式の分子に大きな項が登場する。計算結果がこの大きな項に引っ張られてしまい、正しくない結論になってしまう。この恐れがある時には、荷重関数を用意する。例えば $\rho(x) = 1/(a^2 + x^2)\ \ (x = y_i - y(x_i, \mathbf{a}))$ を分母に掛けて、この影響を減らす場合がある。

どうしても、任意パラメータを導入するので、結果にある種の任意性を持ち込んでしまう欠点がある。この種の変形された最小２乗法をロバストな手法と呼ぶ場合がる。

線形最小２乗法

例えば、角度 θ に依存する物理量 $f(\theta)$ を Legendre 関数 $P_\ell(\theta)$ で展開したいとする。角度

θ を独立変数、展開係数をパラメータと思うと

$$S = \sum_i \left(\frac{f(\theta_i) - \sum_\ell a_\ell \, P_\ell(\theta_i)}{e_i} \right)^2$$

を最小にするパラメータの組み a_ℓ, $(\ell = 0, \cdots, n)$ を決定せよという事である。ベクトルの記号を導入しよう。第 i 成分が $y_i \equiv f(\theta_i)/e_i$ で定義されるベクトルを $\mathbf{y}$、第 ℓ 成分が a_ℓ であるベクトルを $\mathbf{a}$ とする。更に、第 i, ℓ 成分が $P_\ell(\theta_i)/e_i$ で定義される行列を P と書く。問題の性質上、実験点の数は、展開パラメータの数よりも多いから、行列 P は縦に長い行列であるとしておく。

最小化すべき対象 S を、ベクトルのスカラー積の形に表現する。

$$S = (\mathbf{y} - P\,\mathbf{a}, \mathbf{y} - P\,\mathbf{a})$$

ここで、行列 P の対角成分から下の部分を全て 0 としてしまうような直交行列 U を持って来る。以下の様に図示出来る。

$$UP = \begin{pmatrix} Q \\ 0 \end{pmatrix}, \quad Q = \begin{pmatrix} Q_{1,1} & Q_{1,2} & \cdots & Q_{1,n+1} \\ 0 & Q_{2,2} & \cdots & Q_{2,n+1} \\ \vdots & \cdots & \ddots & \vdots \\ 0 & \cdots & \cdots & Q_{n+1,n+1} \end{pmatrix}$$

即ち、行列 $U\,P$ の上半分は、$(n+1)$ 行 $(n+1)$ 列の正方行列 Q であり、その行列要素は対角要素を含み、その右上のみが 0 でなく左下は全て 0 であるとする。

行列 $U\,P$ の下半分には、実験点の数を m とすると、$m - (n+1)$ 行、$(n+1)$ 列の 0 行列がくる。

行列 U の存在証明は無視して、この 行列を用いて以下の様な変形をする。

$$S = (\mathbf{y} - P\,\mathbf{a}, \mathbf{y} - P\,\mathbf{a}) = (U\mathbf{y} - UP\,\mathbf{a}, U\mathbf{y} - UP\,\mathbf{a})$$

ベクトル $U\mathbf{y}$ の上側 $(n+1)$ ヶの部分を $\mathbf{r}$、下の残りの部分を $\mathbf{s}$ と書くと、最小化すべき関数は

$$S = (\mathbf{r} - Q\,\mathbf{a}, \mathbf{r} - Q\,\mathbf{a}) + (\mathbf{s}, \mathbf{s}) \geq (\mathbf{s}, \mathbf{s})$$

即ち、$\mathbf{r} - Q\,\mathbf{a} = 0$ の時に S は最小値 $(\mathbf{s}, \mathbf{s})$ をとる。

まとめると、m 個の観測点データ (x_i, f_i, e_i) を与え、第 i 成分が f_i/e_i で表せる ベクトル $\mathbf{y}$ 及び、線形性の仮定から定義される行列 P を計算し、最小 2 乗法の副プログラムにお願いして、未知パラメータとその時の残差 $(\mathbf{s}, \mathbf{s})$ を返してもらえば良い。

行列 U を作り、P を変形する方法は、実験学の講義ノートか計算機を用いた関数の極小化に書いたから、そちらを参照して欲しい。

注意： 問題により 行列 P が与えられる。問題を解くのにこの行列 P を直接使用するのも良いが、少し考えてみるのも良い。非常に大きな成分と、非常に小さな成分が入り乱れて

いる可能性を疑ってみる。P の第 i 行成分とベクトル $\mathbf{a}$ の第 i 成分を同時に定数倍しても問題の答えは変化しない。従って、その行の最大成分でその行の全ての成分と、ベクトルの対応する成分を割っておくと、突出した数値がなくなると期待される。この操作を正規化 (スケーリング) と呼ぶ場合がある。

Gram-Schmidt の正規直交化のところでも書いたが、P の列ベクトルは並べかえておいた方が計算精度が上がるかもしれない。一つの指導原理は、ノルムが大きい列ベクトルから順番に下半分を 0 にしていく事である。

下のプログラムでは、正規化と列の入れ換えをやっている。ノルムと入れ換え順の記憶は、外部からは見える必要はないので、副プログラムの中で、記憶領域を宣言している。この作業は、FORTRAN 7 7 では不可能だから、作業領域を HLSQ の呼出側で宣言しておく必要があるかもしれない。

```
      SUBROUTINE HLSQ(M, N, IPR, P, Q)
C   linear least squares equation P*X=Q  by using Householder transform.
C Input M,N: define size of the problem (M>N)
C       IPR: size of the 1st index of a matrix P defined in the calling
C       P(IPR, N): a matrix
C       Q(M)     : a vector
C Output Q(N)    : solution vector
C        Q(N+1)  : residue
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION P(IPR, N), Q(M)
      ALLOCATABLE PN(:), IP(:)
C
      IF(IPR.LT.M .OR. M.LT.N) GOTO 91
      ALLOCATE (PN(N), IP(N))
C         scaling row vectors
      DO 3 J=1,M
      A=0
      DO 1 I=1,N
    1 IF(ABS(P(J,I)).GT.A)A=ABS(P(J,I))
      IF(A.LT.1.0E-10)GOTO 93
      A=1/A
      DO 2 I=1,N
    2 P(J,I)=P(J,I)*A
    3 Q(J)=Q(J)*A
C       norms of col. vectors
      B=0
```

```
      DO 5 I=1,N
      A=0
      DO 4 J=1,M
    4 A=A+P(J,I)**2
      IF(B.GT.A)GOTO 5
      B=A
      IP(1)=I
    5 PN(I)=A
C       each col. vectors
      DO 20 I=1,N
      A=0
      K=IP(I)
      DO 11 J=I,M
   11 A=A+P(J,K)**2
      IF(A.LT.1.0E-10)GOTO 95
      B=SQRT(A)
      IF(P(I,K).GT.0)B=-B
      P(I,K)=P(I,K)-B
      C=1/(B*P(I,K))
      IF(I.EQ.N)GOTO 16
C       modify rest of matrix P
      E=0
      DO 15 L=1,N
      DO 12 II=1,I
   12 IF(IP(II).EQ.L)GOTO 15
      A=0
      DO 13 J=I,M
   13 A=A+P(J,K)*P(J,L)
      A=A*C
      D=0
      DO 14 J=I,M
      P(J,L)=P(J,L)+A*P(J,K)
   14 D=D+P(J,L)**2
C        define col. vector for the next loop
      D=D/PN(L)
      IF(E.GT.D) GOTO 15
      E=D
```

```
      IP(I+1)=L
   15 CONTINUE
C        modify vector Q
   16 A=0
      DO 17 J=I,M
   17 A=A+Q(J)*P(J,K)
      A=A*C
      DO 18 J=I,M
   18 Q(J)=Q(J)+A*P(J,K)
   20 P(I,K)=B
C         backward substitution
      DO 22 I=N,1,-1
      J=IP(I)
      A=Q(I)
      DO 21 K=I+1,N
      L=IP(K)
   21 A=A-P(I,L)*Q(K)
   22 Q(I)=A/P(I,J)
C        residue
      A=0
      DO 23 J=N+1,M
   23 A=A+Q(J)**2
      Q(N+1)=A
C           reorder the result
      DO 24 I=1,N
      J=IP(I)
   24 P(J,1)=Q(I)
      DO 25 I=1,N
   25 Q(I)=P(I,1)
      DEALLOCATE (PN, IP)
      RETURN
C       error messages
   91 WRITE(6,92)IPR,M,N
   92 FORMAT('(HLSQ) your IPR=',I4,2X,'M and N=',2I4/
     +       8X,'check the relative size !')
      STOP
   93 WRITE(6,94)J,A
```

```
 94 FORMAT('(HLSQ) at row J=',I3,' max. elem.=',1PE12.3)
    STOP
 95 WRITE(6,96)I,A
 96 FORMAT('(HLSQ) rank(P)<',I3,' A=',1PE13.4)
    STOP
    END
```

線形の問題を解く時、展開係数は複素数であるかもしれない。直交行列を、ユニタリー行列と置き換えて読むと良い。

ラベル１１近くにある、素直交行列を構成するベクトルの符号に関する注意が必要かもしれない。先頭要素と逆位相に選んでおくと良い。

共役勾配法

パラメータ $\mathbf{x}$ の正値関数 $f(\mathbf{x})$ の極値探索として、微分 $g = \nabla f(\mathbf{x})$ が使用できるならば、共役勾配法の採用は、考慮してみる価値がある手法である。共役勾配法は、可変計量法 (variable metric method) と呼ばれる場合もある。

R.Fletcher の論文 The Computer Journal vol.13(1970) pp 317 を参照しながら簡単に紹介しよう。

$G_{i,j} = \partial^2 f / \partial x_i \, \partial x_j$ を (i,j) 成分とする行列 G の逆行列 H を推定する手法である。極小値を与えるパラメータを $\mathbf{x}_{min}$ とし、$f(\mathbf{x})$ をベクトル $\mathbf{x}$ の２次形式だと仮定し、$\mathbf{x}_{min}$ の近くでテイラー展開すると、

$$f(\mathbf{x}_{min} + \delta\mathbf{x}) = f(\mathbf{x}_{min}) + \delta\mathbf{x} \cdot \mathbf{g} + \frac{1}{2}(\delta\mathbf{x}, G\delta\mathbf{x})$$

であるから、

$$\delta\mathbf{x} = -G^{-1}\,\mathbf{g}$$

と計算出来る。この事実が、$H(= G^{-1})$ を推定しようとする意味である。

共役勾配法の手法は以下の通りである。

初期ベクトル $\mathbf{x_0}$ を与えてあるとする。H の初期は、単位行列としておく。最初の直線探索方向は $\mathbf{d} = -\nabla f(\mathbf{x}_0) = -\mathbf{g}_0$ としておく。

１) $\mathbf{d}$ 方向に直線探索し、到達点を $\mathbf{x}_1$、変位ベクトルを $\mathbf{r} = \mathbf{x}_1 - \mathbf{x}_0$、両点での勾配の差 $\mathbf{s} = \mathbf{g}_1 - \mathbf{g}_0$ を作る。

２) 変化量、$|f_0 - f_1|$ や $|(\mathbf{r},\mathbf{r})|$ が小さければ収束したとする。

３) H の改良は次の手順による。

$(\mathbf{r},\mathbf{s}) < (\mathbf{s}, H\mathbf{s})$ ならば以下の H_0 を使用し、逆 ならば H_1 を採用する。

$$H_0 = H + \frac{\mathbf{r}\,\mathbf{r}^T}{(\mathbf{r},\mathbf{s})} - \frac{H\mathbf{s}\,(H\mathbf{s})^T}{(\mathbf{s}, H\mathbf{s})}$$

$$H_1 = \left(1 - \frac{\mathbf{r}\,\mathbf{s}^T}{(\mathbf{r}, s)}\right) H \left(1 - \frac{\mathbf{r}\,\mathbf{s}^T}{(\mathbf{r}, s^T)}\right)^T + \frac{\mathbf{r}\,\mathbf{r}^T}{(\mathbf{r}, \mathbf{s})}$$

4) 新しい $\mathbf{x}_1$ を $\mathbf{x}_0$ に移し、次の直線探索の方向は $\mathbf{d} = -H\,g_1$ とし、先頭に戻る。$\mathbf{d}$ は単位ベクトル化しておく方が理解しやすい。

ここで、ベクトルや行列の上付き 'T' は転置を表す。

直線探索に付いて、

$f_0(= f(\mathbf{x}_0),\ g_0 = \nabla f(\mathbf{x}_0) \cdot \mathbf{d}$ とし、$f(\lambda) = f(\mathbf{x} + \lambda\,\mathbf{d}),\ g(\lambda) = \nabla f(\mathbf{x} + \lambda\,\mathbf{d}) \cdot \mathbf{d}$ を λ の関数と考えて、$f(\lambda)$ の極小値を与える λ を探す。端的に言えば、$g(\lambda) = 0$ となる λ を、$f(\lambda)$ を λ の 3 次関数だと仮定すれば、推定は簡単である。ここで、もう一度平野の指導原理を思い起し、計算の初期には、直線探索を慎重に行うべきである。

一方、N 次元の問題に対し、(1-2) ×N 回 H を改良した後では、$\lambda = 1$ と固定しても、暴走しにくいアルゴリズムとなっているのが、この手法の特徴である。

直線探索において、深追いはさけるのが良い。f_0 よりも小さい関数値を与える点が見付かったならば、ヘッシアン H の改良に進むのが効率的な処方である。

この手法が失敗する時の最大のミスの原因は、$\nabla f(\mathbf{x})$ の評価ミスであるようだ。

実例のところでもう一度登場するが、H の評価は、決定したパラメータの誤差評価に使用できる。

但し、誤差に関してはいくらか考えておく必要がある。即ち、$f(\mathbf{x})$ に $\mathbf{x}$ の各成分が対等な寄与をしていると仮定しているが、実際にはそうでない場合も多い。パラメータの線形結合を作り、全てのパラメータが対等に $f(\mathbf{x})$ へ寄与するように考えておくのが良い。

実は、この手法の発明者である Davidon が共役勾配法を可変計量法がと呼ぶ理由はここにある。

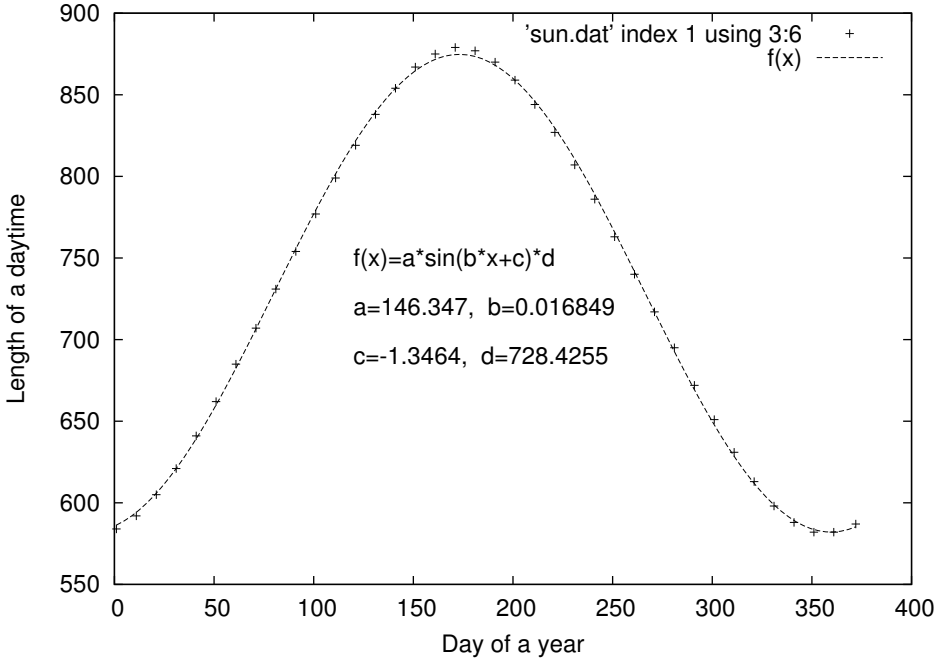
９．実例 水戸での日の出と日の入り

この節で使用する入力データ及びプログラムは、sun.dat 及び sunprog というファイルに納めてある。sunprog には、実対称行列の固有値と固有ベクトルを QR 法で計算するプログラムも付録につけておいた。非常に大きくはない、密行列には使用できると思う。但し、それなりの知識がないと読めないだろう。

sun.dat ファイルの先頭部分には、２００３年１月１日から２００４年１月７日迄の水戸での日の出と日の入りの時刻が、理科年表から引用されている。sun00.f ではこのファイルからデータを読み込み、年初からの積算日数を x 軸データとし、日の出及び日の入り時刻を分に変換し、y 軸データに変換している。更に、太陽が出ている時間を分に換算して出力している。この出力は、sun.dat の第２群データとしてある。年初の１月１日が $x = 1$ である。

先ず、昼の長さをグラフ表示するために、gnuplot を起動し、以下の入力をする。

gnuplot> plot 'sun.dat' index 1 using 3:6



図の点で描かれた部分が、一日の昼の長さの積算日依存性のデータである。かなり良い、三角関数の様に振舞っていると想像される。実線は、このデータを gnuplot の機能を利用して $f(x) = a \times \sin(b \times x + c) + d$ という４個のパラメータで再現した物である。事前に、初期値を $a = 150,\ b = 0.017,\ c = 1.2,\ d = 700$ と見繕って与えておいた。結果のパラメータは、図に書いてあるが、以下の様になった。

a	b	c	d
146.347338887527	0.0168499242513315	−1.34642060203403	728.425486882186
145.213073609541	$2\pi/365.25$	−1.40807345555148	731.237371960864

こんなに有効数字は無いだろう！ c が負になっているのは気に食わない。多分、内部ではかなり乱暴な手法でパラメータを決定しているのだろう。

さて、１年の長さを計算してみると、$2\pi/b = 372.891..$ である ！小学校では１年の長さは３６５日と５時間４６分だと教わったのだが？？どこかで、計算間違いをしたのだろうか？入力にミスはないと仮定して話を進めよう。

１年間の日の出や日の入り時刻を丹念に測定しても、数日の誤差があるという事だろう。上に引用した、１年間の長さを秒以下の精度で決定する事に寄与した人々の努力に頭を下げておきたい。

ところで、$b = 2\pi/365.25$ と固定してもう一度挑戦すると、上の表の下の行の様に変更される。因みに両数値の表す曲線をグラフに描いて比較すると、夏の日が少し長くなり冬の日がいくらか短くなるが、あまり大きな変化は感じられない。

このデータから、１年の長さを正確に決定するのは無理なのだろうか？パラメータの誤差が ± １週間という事だと、どこにも矛盾は無いのだが、実際の b パラメータの誤差の大きさはどの程度だろう？

gnuplot は画面に結果を表示する以外に、途中経過と共にフィットの計算は、fit.log というファイルに出力している。中身を見ると、$b = 0.0168499 \quad +/- \quad 6.916e-05$ となっている。相対誤差は、$6.916 \times 10^{-5}/0.0168499 = 4.1 \times 10^{-3}$ となる。約、１．５日である．どうも外れ過ぎの感を免れない。

new(c) は、初期値を +1.2 と入力したのに、負の値に変更して来る。どんな理論や手法を用いて、データ処理をしているのだろう？あまり深く信用しすぎ無い方が無難だと思う。これらの事を考えると、他人の仕事は信用しすぎずに、自分で確かめながら進む方がよさそうだ。

２００３年の暦には、以下の記述がある。

名称	春分	夏至	秋分	冬至
月 日	3月２１日	6月２２日	9月２３日	１２月２２日

グラフや数値を見ながら、この事実を確認してみよ。又、これらの単語の定義を思い返してみよ。

２００３年のデータは古すぎるという意見があるだろう。では、もっと新しいデータを自分で捜し出して、どのような結果となるか挑戦してみよ。

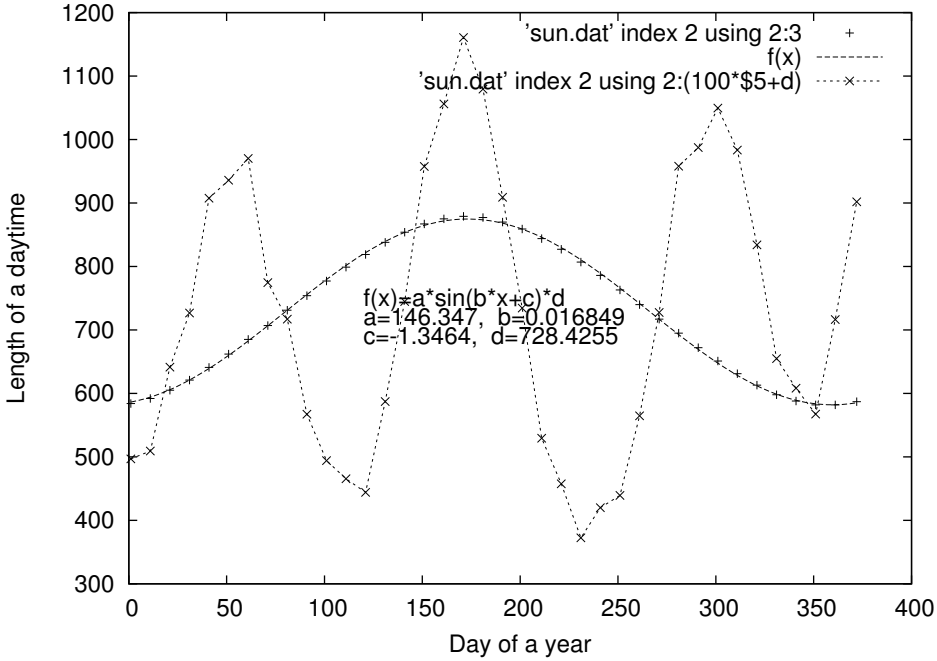
ところで、何故昼の長さが季節により異なるのだろうか？振幅 $a = 145 - 146$ 分と言うのは何を表しているのだろうか？

太陽光線が無限遠方から射していて、水戸は北緯 θ_N に位置し、地球の自転軸が公転面に対し α だけ傾いていると近似すると、一番短い昼の時間を、

$$(1440/2\pi) \times 2 \times \sin^{-1}\left(\frac{\sqrt{1 - \dfrac{\sin^2\theta_N}{\cos^2\alpha}}}{\cos\theta_N}\right) \text{分}$$

と推定してみた。$\theta_N = 36°22''$, $\alpha = 22.5°$ とすると大体正しい値を得るようだ。

もう少し突っ込んで考えてみよう。理科年表のデータと計算値との差の１００倍をグラフにしてみた。

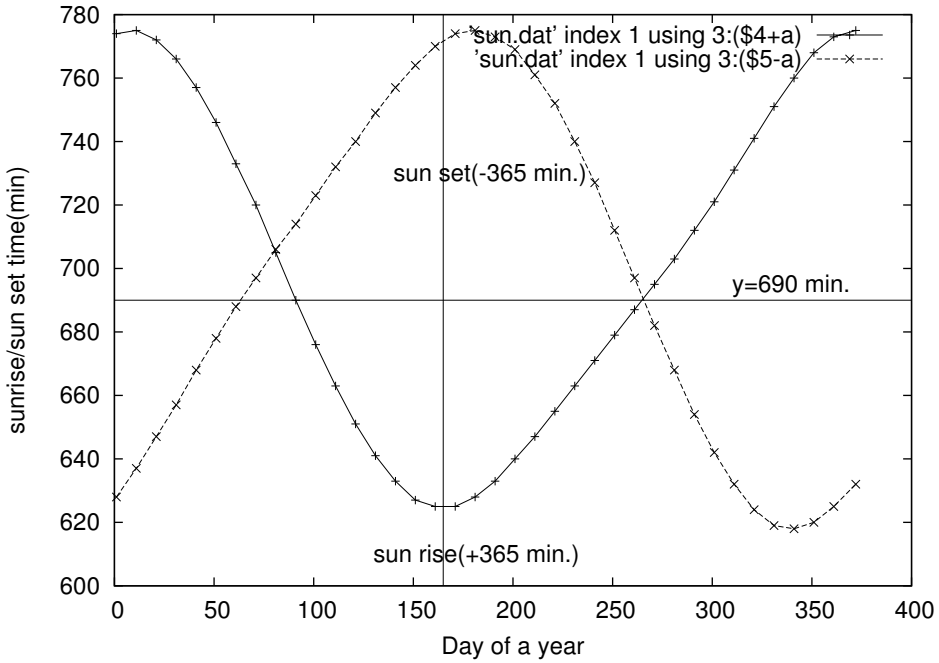


ほぼ、１/３年周期の誤差が見られる。何故、１/３年周期の誤差が顕著になるのだろう？１/２ 年周期の成分は元もと振幅が小さいのだろうか？知識が増えると、疑問も増えるものだ。

元来、昼の長さは日の出と日の入りの時刻の差として定義されている。従って日の出や日の入り時刻の周期性を調べるのが先だと思える。

日の出入り

先ずはグラフを描いてみよう。



図では、１月１日からの積算日 (通日とプロは呼ぶらしい) を x 軸に、日の出入りの時刻を、中心を３６５分ずつずらして、y 軸に描いた。時刻は、全部分に換算している。中央の x 軸に平行な線は、$y = 690$ 分を示す。２本の曲線は、$x =$ ２７０日程度に対して $y = 290$ 分で交わっているのに、春に交わる点は、$y = 705$ 分程度で交わっている。約１５分ほど、春と秋では非対称性が見られる。

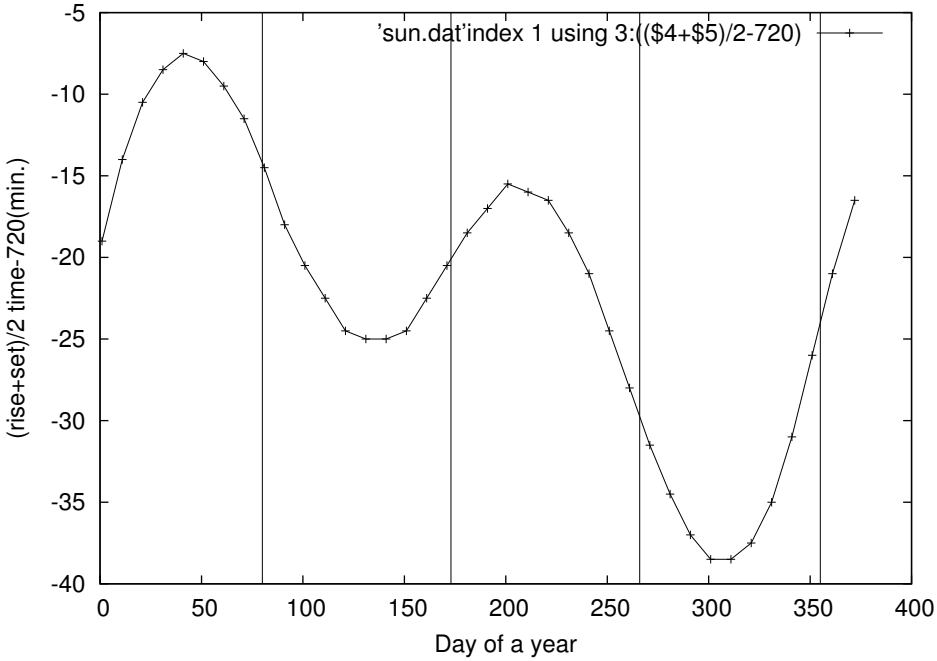
更に縦の線は、$x =$ １６０日の線である。この線が一年で一番早い時刻に日が出る。一方、日が沈むのが一番遅いのは $x =$ １８０日程度であり、両者の間には約半月の差がある。これらの x の値と、夏至の積算日を比較してみよ。

即ち、日の出や日の入りは, １年周期の対称性からそれなりに崩れている。

この非対称の原因は何であろうか？当然、現代の天文学では理解できている事であるから、どこかの本で調べれば良いと言うのも一つの態度であるが、やはり気になる事ではある。

良く見ると、日没時刻の春の振舞いと、日の出時刻の秋の振舞いとは、傾きが逆で、傾向は似ている様に思える。この日の出入りの非対称性は、昼の長さにおいては、全体として打ち消されてかなり対称性の良い結果を与えていた。昼の長さがかなり良い対称性を与えているからと言って、地球の運動がかなり単調だという事を意味しない点に注意しておこう。

この非対称性は日の出入りの時刻は出る、入るという特殊事情を反映している可能性もある。例えば、日の出は太陽の中心が水平線に位置した時ではなく、縁の部分が水平線にかかった時刻である。従って、太陽の大きさ分だけ非対称性が導入されている可能性もある。そこで、南中時刻に対応する量として、両者の平均値が正午 (１２：００) よりどれくらいずれているかを調べる方が良いと思える。下の図は、日の出入りの時刻の平均値を描いてみた。

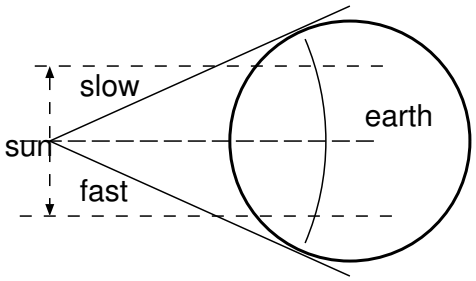


平均値が負にずれている事は、水戸が東経１３５度の地点よりも東に位置する事を示すと解釈しておけば、多分正しいだろう。

非対称な山が二つある理由は何だろう？

図の縦線は、３/２１、６/２２、９/２３、１２/２１ に対応し、春分、夏至、秋分、冬至を想定している。何となくこの４本の縦線が、図の曲線の変曲点に対応しているような気がする。

こんなアイデアで説明出来ないだろうか？



図は、遠くの方から太陽が地球を照らしている様子を描いている。地球上の照らされた部分が昼であり、光円錐と影の部分で昼夜を分けている。地球を止めて考えると、太陽 (黄道) は赤道を中心として南北の回帰線迄、図の上下に動く。この上下の動きと、地球の自転の速度が一定ならば、単一周期の周期的な振舞いが予想される。

ところで、地球の軌道は楕円である事が分かっている。北半球が夏の時には、地球は遠日点の方にあり、公転速度は遅い。逆に南半球上に太陽がある時には、地球は近日点の方にあり、公転速度は早い。従って、二つの周期運動が重なってこのようなグラフが描かれているのではないだろうか？

本当かな？理科年表には、地球ー太陽間の距離を天文単位で記載している。７月６日に最大値 １．０１６７、１２月３３日に最小値 ０．９８３３０をとる。約１．５％の振幅がある。１日の１．５％は約２２分となる。図の振幅と比較すると、大きさ的には何となく妥当である。

天球上の恒星の位置を表す座標系として赤道座標系がよく使用される。その定義を記録しておこう。

先ず、地球の中心 O を原点とする。地球の自転軸考え、自転軸と天球との交点の内、地球の北側にある方を天の北極 P 、反対側の交点を天の南極 P ’ と呼ぶ。線 P、P ’ に直交し、点 O を通る面を考え、この面と天球が交わる円を天の赤道と呼ぶ。

天球上の点 X を指定する角度は、以下の様にする。３点 P、X、P ’ を通る面と天球との交わる大円を考える。天の赤道からこの大円に沿って点 X 迄の角度 δを赤緯と呼ぶ。

天の赤道上の基準点として、春分の日の太陽の位置をとる。この位置を春分点 (γ) と呼ぶ。３点 P、γ、P ’ を通る面と、上に述べた P、X、P ’ で決まる面迄の方位角を赤経 α と呼ぶ。

従って、極座標による表現を思い出すと、赤緯は P、P ’ を極軸とする極角とはπ / ２だけ測り方が異なるが、赤経の方は、方位角そのものである。

この赤道座標系は、恒星にくっついた座標系であるから、地球の自転に伴って動いていく。

問題の設定

上の推測が正しいならば、南中時刻を１年間にわたり、毎日はかれば地球の公転軌道の離心率が判るのだ！

面白そうだから、このデータのフーリエ解析を取り上げて、数値計算の演習問題としよう。

独立変数 t を元旦からの積算日 (通日) とする。従属変数は、水戸での日の出と日の入りの時刻の平均値 $y(t)$ とする。

$$y(t) = C_0 + \sum_{n=1}^{2} S_n \sin(2\pi\, n\, t/T) + C_n \cos(2\pi\, n\, t/T)$$

と展開した時、５個のパラメータ、$C_0,\ C_1,\ C_2,\ S_1,\ S_2$ を最小二乗法の意味で決定せよ。但し、$x_i,\ (i = 1, 38)$ には誤差は無く、$y_i,\ (i = 1, 38)$ の誤差は一定であると仮定する。
１年の長さの近似として $T = 365.24$ を採用しておこう。

基本方針：以下の目的関数を、上記５個のパラメータの関数であると考えて、この関数の最小値を与える様にパラメータを決定する。

$$S = \sum_i [y(t_i) - \{C_0 + \sum_{n=1}^{2} (S_n \sin(2\pi\, n\, t_i/T) + C_n \cos(2\pi\, n\, t_i/T))\}]^2$$

作業の方針

1　先ず、これらのパラメータの初期値を推定しなければならない。
2　目的関数を最小とするように、パラメータの値をすこしずつ改良する。
3　結果の確認。

細かい作業の手順。

初期値は、どのようにして決めれば良いのだろう。

１ー０) １： 直観的には、目視できめる。２：目視できめてグラフを描き、少しずつパラメータを動かしながら見繕っていく。３： フーリエ展開を使用する。

フーリエ展開は既に習っているだろうから、３の手法に依ろう。

１ー１) 以下の積分を実行すれば良い事になった。

$$\int_0^T y(t)\{\sin(2\,n\,\pi\,t/T), \text{又は} \cos(2\,n\,\pi\,t/T)\}dt$$

さて、$y(t_i)$ のデータは３８個の跳び々の t_i に対して与えられている。t_i はほぼ等しい間隔であるが、きっちりとは等間隔ではない。

$1 < t < 370$ なる t に対して、与えられた点のデータを内挿する式を作っておこう。色々なやり方があるだろうが、ここでは３次のスプラインを利用する。

１ー２) 上で作ったスプライン関数を用いて、$y(t)$ に三角関数をかけて数値積分を実行して、初期値を決定する。

１ー３) 初期値がほぼ正しい値を与えているかを、グラフを用いて確認する事。$n = 1,\ 2$ に対する $C_n,\ S_n$ の初期値を決定するたびに、与えられたデータと残差を計算し、その振舞いをグラフに描いてみると、何をしているかが理解し易い。

２ー０) パラメータの精度を上げる作業はどのようにして行おうか？

この時点では想像だが、フーリエ展開を用いるならば、初期値の推定精度が高いはずから、この作業は難しい理論を使用せずとも良いと思う。

思い付くままに、処方を書き出してみよう。

２ー１) ５個のパラメータの一つ を取り上げ、このパラメータを x と呼ぶ。

他の４個のパラメータは 固定しておく。初期値 x_0 を中心として、小さな数 $\delta\,x$ を考え、$x_0,\ x_\pm = x_0 \pm \delta x$ での S の値 $S_0,\ S_\pm$ を計算する。$S_0 < S_\pm$ ならば、x の２次式で S を近似して、初期値を改良する。ここで深追いはしてはせずに、別のパラメータを取り上げる。

もしもそうでなければ、$S_\pm$ の内の小さい方の値を与える $x_\pm$ を改良された x の値 (新 x_0) とする。この場合には、δx を幾らか増やして、もう一歩先へ進むのも良いだろう。

この手順を収束するまで繰り返す。

必要と思うならば、各パラメータに対してパラメータの現在の改良値を覚えておいて、次回に利用しても良いだろう。

２ー２) 目的関数の微分 dS/dx を計算して初期値を改良するという案もあり得る。２ー１) に於いて、x_0 の関数値とその微分を計算するならば、$\partial S/\partial x = 0$ を解く問題に帰着する。解は必ず存在するから、 ２分法に持ち込むのが、賢明だろう。即ち、微分が小さくなる方に少しずつ、$\partial S/\partial x$ の符号が逆転するまで、x の値を動かし、解の存在範囲を確定し、次はその範囲を半分ずつに絞りこんでいく。

問題が非線形であるから、特に５個のパラメータが絞り込まれていない段階で、一つのパラメータだけを目の敵にしてはいけない。

２ー３) ５個パラメータが作る５次元空間を想定するというアイデアもあり得る。関数の値だけを目安にしてパラメータを決定したいならば、シンプレックス法が最も良いだろう。

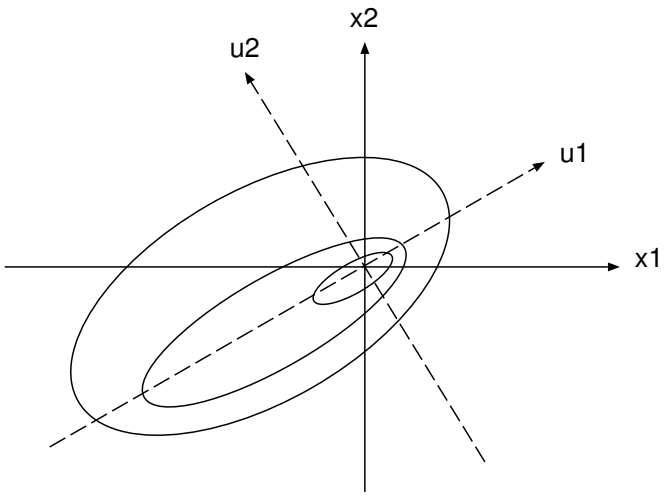
更に、微分 (∇S) を利用しようとするならば、共役勾配法を推奨する。

３) パラメータが決まったならば、決定の過程で評価された目的関数 S の値と、決定されたパラメータを用いて別途計算した値とが一致するかをチェックしておこう。

更に気が向けば、少しだけパラメータを動かして、目的関数 S が定留置をとることも確認するのが良いだろう。

計算されたパラメータの決定精度も気になるところであるだろう。一つの確認法は、極小点での２階微分を評価する事であろう。$\partial^2 S/\partial x_i^2$ は、５次元空間での S の値の x_i 軸断面での切口の谷の切れ込み具合を表す。もしも谷の切れ込みが深ければ、このパラメータは良く決まっていると考えられるし、切れ込みが浅ければ少しぐらいパラメータを変更しても、目的関数に変化が少ないという事だから、パラメータの決定精度は悪いと言える。

更に言えば、$\partial S/\partial x_i\,\partial x_j,\ (i \neq j)$ も評価するのが良い。



図は２次元面内での S の等高線を描いたものだとする。水平と垂直の線が $x_1,\ x_2$ 軸だとする。この場合、直観的に言って、図に示した $u_1,\ u_2$ を用いて、谷の切り込み深さの議論をするのが妥当だろう。$x_1,\ x_2$ 軸を用いて解いた問題から、$u_1,\ u_2$ 軸を探り出す問題は主軸問題と呼ばれる。例えば、慣性能率と慣性乗積という剛体の力学に登場した概念である。

もしも線形数学の知識があれば、以下の成分で定義される実の対称行列

$$H_{i,j} = \partial^2 S / \partial x_i \, \partial x_j$$

の固有ベクトル $\mathbf{u}_i$ がこの軸方向を与えるベクトルであり、この固有値が、谷の曲率を与える物理量になっている事が判るだろう。

理解を助ける為に、少しばかり数式を用いて説明しておこう。目的関数を極小点 $\mathbf{x}_0$ でテイラー展開する。

$$S(\mathbf{x}_0 + \delta\mathbf{x}) = S(\mathbf{x}_0) + \nabla S(\mathbf{x}_0) \cdot \delta\mathbf{x} + \frac{1}{2!}(\delta\mathbf{x}, \nabla\,\nabla S\,\delta\mathbf{x}) + O(\delta\mathbf{x}^3)$$

右辺第２項は、極小点の定義により０である。そこで、第３項を調べる。

$$(\nabla\,\nabla S)_{i,j} = \frac{\partial^2 S}{\partial x_i \, x_j}$$

を (i,j) 成分とする行列を H とし、この行列の固有値を $\lambda_i,\ (i = 1, 2, \cdots, N)$、対応する固有列ベクトルを $\mathbf{u}_i$ とする。更に固有値を対角成分とし、非対角成分は全て０である行列を Λ、固有列ベクトルを横に並べて作る行列を U とする。即ち、

$$H\,U = U\,\Lambda, \quad 又は\ H = U\,\Lambda U^{\dagger}$$

従って、

$$(\delta\mathbf{x}, H\,\delta\mathbf{x}) = (U^{\dagger}\delta\mathbf{x}, U^{\dagger}HU\,U^{\dagger}\delta\mathbf{x}) = (\delta\mathbf{u}, \Lambda\,\delta\mathbf{u}) = \sum_i \lambda_i\,\delta u_i^2$$

ここで、

$$U^{\dagger}\,U = U\,, U^{\dagger} = 1$$

を使用した。更に、

$$\delta\mathbf{u} = U^{\dagger}\delta\mathbf{x}, \quad 又は \quad \delta u_i = \sum_j U^{\dagger}_{i,j}\delta x_j$$

即ち、交差項 $\delta x_i\,\delta x_j,\ (i \neq j)$ を含んでいた第３項の交差項が全部消えて、２乗の項だけになり、式がかなり簡単になった。これが主軸変換である。

初期値の決定

先ず、$x(i),\ y(i),\ \ (i = 0, 1, \cdots, , n)$ というデータを読み込んでおく。

１）３次のスプライン関数による内挿関数を作る
次の作業を行えば良い。

$(n+1)$ 個の節点 $(x_i,\, y_i),\ \ (i = 0, 1, \cdots, n)$ が与えられたとする。$h_i = x_i - x_{i-1}$、$\delta_i = (y_i - y_{i-1})/h_i$ と置いた時、$(n-1$ 個の未知数 $y_i''/6 = z_i$、$(i = 1, 2, \cdots n-1)$ に関す

る次の連立方程式を解く。

$$h_i\, x_{i-1} + 2(h_i + h_{i+1})\, z_i + h_{i+1}\, z_{i+1} = \delta_{i+1} - \delta_i$$

区間 $[x_{i-1},\, x_i]$ での内挿３次式 $F_i(x)$ は次と通りとする。

$$F_i(x) = y_{i-1}\, t + y_i\, s + h_i^2\{z_{i-1}\, t(t^2-1) + z_i\, s(s^2-1)\}$$

ここで、$t = (x_i - x_i)/h_i,\ \ s = (x - x_{i-1})/h_i = 1 - t$ と置いた。

両端での２階微分に対応するパラメータは、相隣る３点を２次式で近似して評価するとしよう。即ち、

$$z_0 = \frac{\delta_2 - \delta_1}{3(x_2 - x_0)},\ \ z_n = \frac{\delta_n - \delta_{n-1}}{3(x_n - x_{n-2})}$$

ここから、作業を開始する。
$z_0,\ z_2$ を用いて、 z_1 を解くと、

$$z_1 = \frac{\{\delta_2 - \delta_1 - h_1\, z_0\} - h_2\, z_2}{2(h_1 + h_2)} = \frac{p_1 - h_2\, z_2}{w_1}$$

最右辺の式は、$p_1,\, w_1$ の定義とする。即ち、$p_1 \equiv \{\delta_2 - \delta_1 - h_1\, z_0\},\ \ w_1 \equiv 2(h_2 + h_1)$。一般に以下の漸化式が成立すると仮定する。

$$z_i = \frac{p_i - h_{i+1}\, z_{i+1}}{w_i}$$

この式は、以下の様な漸化式が成立すれば良い。

$$p_i = \delta_i - \delta_{i-1} - \frac{h_i\, p_{i-1}}{w_{i-1}},\ \ w_i = 2(h_i + h_{i-1}) - \frac{h_i^2}{w_{i-1}}$$

プログラムとしては、p_1, w_1 を与えて、$i = 2$ から $n - 1$ 迄、順次 $p_i,\, w_i$ を計算する。

次に z_n の知識を用いて、$i = n - 1$ から $i = 1$ まで、順次 z_i を i が小さい方へ計算すれば、全ての (２階微分)/6 が計算できた事になる。

後は、この $z_i,\ (i = 0, 1, \cdots n)$ を用いて、個別の区間毎に内挿公式を用いて $y(x)$ を計算すれば良い。

例示したプログラムでは、p_i は、２階微分の計算結果を書き込む配列に一次的に保存されている。

プログラムのテストをするためには、例えば $y(x)$ として２次式を用いてみると良い。２階微分は定数になる。

２) 台形公式を用いて、積分を行う。
オイラーとマクローリンの和公式を知っていれば、周期関数の１周期にわたる積分では、台形公式が高精度の計算公式であると判断出来る。

展開係数は以下の通りとする。

$$C_k = \frac{2}{T}\int_0^T y(t)\cos(\frac{2\,k\,\pi\,t}{T})dt; \quad C_0 = \frac{1}{T}\int_0^T y(t)dt$$

$$S_k = \frac{2}{T}\int_0^T y(t)\sin(\frac{2\,k\,\pi\,t}{T})dt$$

C_0 は、全体の平均値である。計算の精度確保の為に、近似式の $y(t_i)$ から C_0 を差し引いた量を $y(t_i)$ の変わりに使用して積分する。台形公式では、両端は因子 $1\,/\,2$ がかかるが、初期値の計算だから、この因子は無視しておこう。

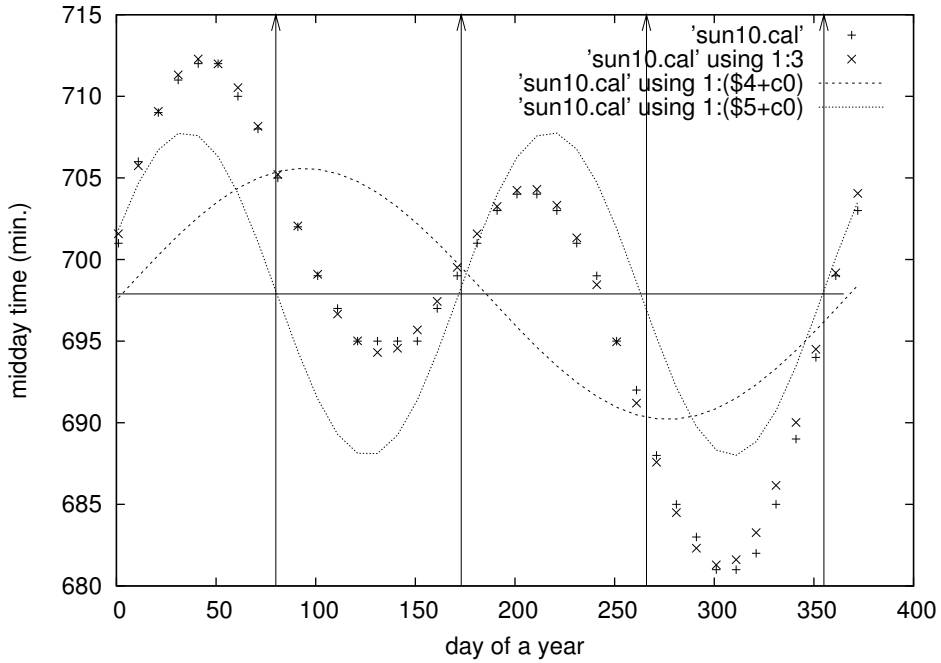
三角関数の計算は、先に例示したように等間隔の場合の漸化式が使用できる。更に倍角公式も使用しよう。

プログラムは、SUN10 と言う名前で、例示してある。

計算結果は、以下のとおりであった。当然単位は分である。

```
C0 C1 C2=   6.97895E+02  -3.74988E-01   3.61153E+00
   S1,S2=                 7.68125E+00   9.24007E+00
```

以下の様になグラフに描いてみよう。



入力データは × 印、計算結果は + 印である。これらの展開係数は特に修正の必要が無いくらいに、入力データを正しく展開している様に見える。

C_0 の値を時刻に直すと、水戸での南中の平均は１１時３８分である。水戸の東経は１４０度２２分だから、5.37° だけ、東経１３５度よりも東にある。$5.37 \times 60/15 = 21.47$ 分だ

け進んでいて欲しい。１分以下は誤差だと考えると、誤差の範囲で一致している。よかったよかった。

$\sqrt{C_1^2+S_1^2}=7.69,\ \sqrt{C_2^2+S_2^2}=9.92$ 分である。南中時刻で見ると、２次の高調波の方が１次の高調波よりもほんの少し、振幅が大きい。

C_1 は S_1 よりもかなり小さい。$tan^{-1}(S_2/C_2)=21.3^\circ$ であるから、２次の高調波の０点はいくらか御正月からずれる。

１次と２次の高調波成分がどれくらいあるかも、図示しておいた。

南中時刻を正確にはかると、東経何度に我々がいるかを推定できる。但し、２月と１０月では３０分程度南中時刻が異なっているから、短期間の測定では精度は悪い。例えば、半年間真面目に南中時刻を記録しても、半年間の平均で $C_2,\ S_2$ の項は消えるけれども、C_1 や S_1 は消えないので S_1 が５分以上あるから、５分 $\times$ １５度/６０分＝１．３度程度の誤差がある。

初期値の改良

上の結果を見ると、かなり良く入力を再現しているので、ファイトが失せる。多分、３次のスプライン近似が非常に強力であったのだろう。

さて、最初に述べた方針を変更し、作戦をたて直そう。５個のパラメータに対して、初期値 x_0 を中心として、$x_0\pm 2h,\ x_0\pm h$ という５点を代表点にとり、$G(x_0\pm k\,h)$ での５個の値を２次式で近似し、この極小値を与える x の値を x_0 の改良値として採用しよう。

この場合、次の手順となる。

1 C_0 から S_2 迄のどれかを一つ取り出し、x_0 とする。

2 小さな増分 h を決定し、これから決まる５点での目的関数の値 $G(k),\ (k=-2,1,\cdots,2)$ を評価する。

3 ５点の値 $(x_k,\ G(k))$ を最小２乗法の意味で、２次近似し極小値を与える x の値を決定する。この計算手法は後で説明する。

4 １ に戻り、新しいパラメータに対して同様の処理を行う。

5 全てのパラメータの改良幅 $|x-x_0|$ 又は $|G(x_0)-G(x)|$ が無視できる程度に小さくなったら、処理を終える。

手順 3 での計算処方

$$G_k=G(x_0+k\,h)=a\,(x_k-x_0)^2+b\,(x_k-x_0)+c$$

と仮定し、$a,^{;}\,b,\,c$ を決定するには、次式を用いれば良い。

$$a=\frac{T^2-2\,h^2\,T^0}{14\,h^2},\ b=\frac{T^1}{10\,h^2},\ c=\frac{17\,h^2\,T^0-5\,T^2}{35\,h^2}$$

ここで、次の略号を用いた。

$$T^2 = \sum_k (x_k - x_0)^2\, G(k),\ \ T^1 = \sum_k (x_k - x_0)\, G(k),\ \ T^0 = \sum_k G(k)$$

導き方は簡単だから、各自でやってもらいたい。

この目的で、PFIT 副プログラムを追加した。
h としては、パラメータの初期値の２％＋０．１をとった。

	C_0	C_1	C_2	S_1	S_2
初期値	697.89474	-0.37498790	3.6115345	7.6812504	9.2400674
終値	697.72278	-0.54205463	3.2606749	7.6738420	9.3446860

各段階毎に目的関数は、以下の様に改良されていった。

繰り返し	初期値	終値
1	1.1021E+01	8.8069E+00
2	8.4142E+00	7.3164E+00
3	6.5713E+00	6.3232E+00
4	6.3051E+00	6.2671E+00
5	6.2654E+00	6.2622E+00
6	6.2621E+00	6.2618E+00
7	6.2618E+00	6.2618E+00
8	6.2618E+00	6.2618E+00

目的関数の値が、約５７％にまで減っている。初期値が良かったから、かなり単純な作業でパラメータが確定した。

これらのパラメータの信頼性を確認するには、個別のパラメータをいくらか変化させて、目的関数がどの程度増加するかを調べれば良いと考えてもよいだろう。しかしここでは最初の作戦通りに処理をしよう。

ここで２次近似をするのに、３点の計算で充分なのに、５点を使用した理由は判るだろうか？

次は、パラメータがどの程度の幅を持っているかを推定したい。
先に作業指針により、２階微分 $\partial^2 S/\partial x_i\, \partial x_j$ を計算する。
ここは、数式を直接使用する。

$$\frac{\partial^2 S}{\partial C_i\, \partial C_j} = \sum_n \cos(i\, x_n) \cos(j\, x_n), \text{等}$$

線形の問題だから、誤差行列には y_i の値は関係しなくなってしまった。
次にする事は、固有値と固有ベクトルを計算する事である。
先ず、行列 H そのものを見よう。

```
7.6000000E+1  2.9370683E+0  2.8984595E+0  2.0513304E-1  4.0701353E-1
2.9370683E+0  3.9449230E+1  2.8858986E+0  2.0350677E-1  4.0378100E-1
2.8984595E+0  2.8858986E+0  3.9373397E+1  1.9864796E-1  3.9412310E-1
2.0513304E-1  2.0350677E-1  1.9864796E-1  3.6550770E+1  5.1169756E-2
4.0701353E-1  4.0378100E-1  3.9412310E-1  5.1169756E-2  3.6626603E+1
```

特徴として、対角優位である。最初の 3 行 3 列が、非対角要素が割合に大きい。定数項と、余弦に関する部分がかなり影響しあっているのだろう。

当然の事であるが、対称行列であるから、固有値は実数である。Gershgorin の定理を用いて、固有値の存在範囲を調べよう。即ち、各行に対し、対角要素を中心値とし、非対角要素の絶対値の和を固有値の存在範囲とする。

行	下限	中央値	上限
1	6.95523E+01	7.60000E+01	8.24477E+01
2	3.30190E+01	3.94492E+01	4.58795E+01
3	3.29963E+01	3.93734E+01	4.57505E+01
4	3.58923E+01	3.65508E+01	3.72092E+01
5	3.53705E+01	3.66266E+01	3.78827E+01

第 1 行から決まる固有値だけが $[69.5, 82.5]$ の区間に分離されているが、残りの 4 個は、範囲が重なっているから、$[32.99, 45.87]$ に残りの 4 個が混みあって存在しているとしか言えない。一つだけ固有値が突出しているから、一つの固有値だけならば冪乗法で処理しても良いが、2 番目と 3 番目及び 4 番目と 5 番目はくっついているので、冪乗法を採用しても固有値を得るのは困難だろう。1 番目は定数項 C_0 に対応し、C_0 の値は他の値よりも突出しているので、最初の固有値だけが大きいのは納得出来る。

どのような手法で、固有値を計算しようか？

常識的には、実対称行列だから、Householder 変換してみよう。

実対称行列の固有値と固有ベクトルの計算

先ず与えられた実対称行列 H を Householder 法で、3 重対角な対称行列 T に変換する。行列 H と T の固有値は共通である。

先ず、任意の実単位ベクトル $\mathbf{u}$ を用いて行列 P を次式で定義すると、P は対称な直交行列である。

$$P = 1 - \mathbf{u}\mathbf{u}^T, \quad P_{i,j} = \delta_{i,j} - 2u_i\, u_j, \quad P^2 = 1$$

行列 H と直交行列 Q を次の様に書こう。

$$H = \begin{pmatrix} A & B^T \\ B & C \end{pmatrix}, \quad Q = \begin{pmatrix} 1 & 0 \\ 0 & P \end{pmatrix}$$

この表現を用いて H を Q で変換する。

$$QHQ = \begin{pmatrix} A & B^T P \\ BP & PCP \end{pmatrix}$$

A の部分は変化せず、B, C の部分が変化する。ここで、A は既に対称な 3 重対角行列であり、B を以下の様な列ベクトルの並びだと考える。即ち、

$$A_{i,i} = \alpha_i, \quad A_{i,i+1} = A_{i+1,i} = \beta_i, \quad \text{他の成分は} 0$$

$$B = (0, 0, \cdots, \mathbf{b})$$

$\mathbf{u}$ を以下の様にとると、$P\mathbf{b}$ の第 1 成分 $(= b')$ のみが 0 でなく、残りの成分は全て 0 となる様に出来る。

$$\mathbf{u} = (\mathbf{b} - b'\mathbf{e})/(2Z)$$

$\mathbf{e}$ は第 1 成分だけが 1、残りは 0 という基本単位ベクトルであり、規格化定数は $2Z^2 = b'(b' - b_1)$ である。但し、b_1 はベクトル $\mathbf{b}$ の第 1 成分であり、$(b')^2 = (\mathbf{b}, \mathbf{b})$ は、ベクトル $\mathbf{b}$ の大きさの 2 乗である。b' の符号は、b_1 とは逆の符号にとる。$(\mathbf{b}')^T = (b', 0, \cdots, 0)^T$ であるから、QHQ の 3 重対称性が一つ増える。

残りの部分は、以下の手順により計算すれば良い。

$$\text{新 } C = C - \mathbf{u}\mathbf{q}^T - \mathbf{q}\mathbf{u}^T$$

ここで、

$$\mathbf{p} = C\mathbf{u}, \quad \mathbf{q} = \mathbf{u} - (\mathbf{u}, \mathbf{p})\mathbf{u}$$

QHQ を新しい H だと思い、この新しい H に対して同様の変換を施していくと、当初の H の次元を N とした時、$(N-2)$ 回の変換で 3 重対角行列に変換できる。

この様な指導原理で書いたのが SUN12 の HTRANS 副プログラムである。少しだけ、プログラムの中身を解説しておこう。

H は対称な行列であるから、左下半分にだけ行列要素が定義してあればよい。即ち、計算の途中でも、左下半分だけを計算していけば、重複する計算はしなくて良い。

行列 H の第 i 列の下の方を 0 ベクトルに変換する直交行列を P_i とすると、P_i を作るベクトルは行列 $H_{j,i}$, $(j = i+1, i+2, \cdots, N)$ に書き込まれる。このベクトルは、上の記号で言えば、$\mathbf{b} - b'\mathbf{e}$ に対応し、規格化係数に関係する $1/2Z^2$ は、対角要素 $H_{i,i}$ に保存してある。これにより、後から全体の直交行列を再現することが可能になる。但し、例示したプログラムでは $b' = 0$ と近似出来る場合には $H_{i,i}$ にゴミが残っている。例えば 0 を代入しておいて、後からこの問題に対処出来る様にしておくのが良いだろう。

上で計算した行列要素を、ファイルから読み込んで、３重対角行列に変換して、その対角要素を１次元配列 A に、副対角要素を１次元配列 B に書き込んで、呼び出しプログラムに返す。

その結果、次の様に変換できた。

7.6000000E+01	−4.1515271E+00	0	0	0
−4.1515271E+00	4.2367319E+01	−6.1737632E−04	0	0
0	−6.1737632E−04	3.6581920E+01	−6.7701488E−03	0
0	0	−6.7701488E−03	3.6525761E+01	2.8298856E−12
0	0	0	2.8298856E−12	3.6525000E+01

$T_{4,5} = 0$ と考えてよいから、一つの固有値は 36.5250 として良い。好運と言うべきか？後半の３個も大体固有値は、対角要素に等しい。即ち、Householder 変換を採用する事で、固有値の存在範囲をかなり狭い範囲に閉じ込める事が出来た。

固有値の計算

上の計算結果に、Gerchgorin の定理を適用する。４個の固有値の存在範囲は次の様になる。

$$[71.84, 80.16],\ [38.21, 46.52],\ [36.576, 36.589],\ [36.518, 36.533]$$

最後の一つは確定だ。最初の 4 つの固有値を計算しよう。この場合には、STURM の定理を利用するのが良いだろう。

対称な実３重対角行列の対角要素を $\alpha_i,\ (i = 1, 2, \cdots, N)$、副対角要素を $\beta_i,\ (i = 1, 2, \cdots, (N-1))$ と書こう。固有値方程式は、以下の様に書ける。

$$D_N(\lambda) = |T - \lambda I| = \begin{vmatrix} \alpha_1 - \lambda & \beta_1 & 0 & 0 & 0 \\ \beta_1 & \alpha_2 - \lambda & \beta_2 & 0 & 0 \\ 0 & \ddots & \cdots & \ddots & 0 \\ & & \cdots & & \\ 0 & 0 & \cdots & \beta_{N-1} & \alpha_N - \lambda \end{vmatrix} = 0$$

この行列式は、以下の漸化式で計算出来る。

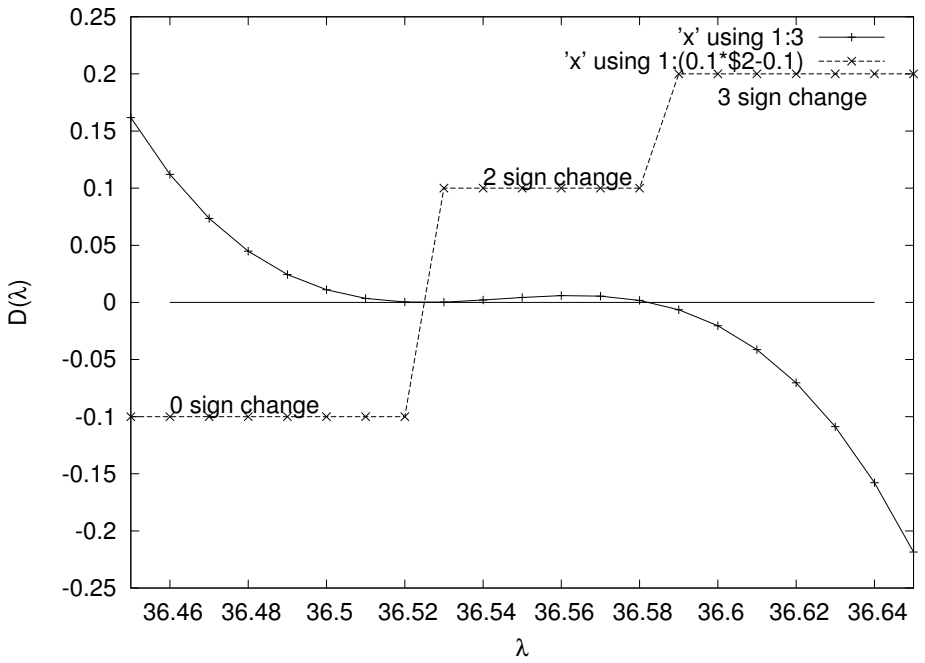
$$D_0(\lambda) = 1,\ D_1(\lambda) = \alpha_1 - \lambda,\ \ D_i(\lambda) = (\alpha_i - \lambda)D_{i-1} - \beta_{i-1}^2 D_{i-2}(\lambda)$$

この漸化式を λ で微分すると、

$$D_0' = 0,\ \ D_1' = -1,\ \ D_i' = -D_{i-1} + (\alpha_i - \lambda)D_{i-1}' - \beta_{i-1}^2 D_{i-2}'$$

この行列式に関する漸化式を D_1 から D_N まで計算する途中に符号が変化する回数を $L(\lambda)$ とすると、区間 $[\lambda_a, \lambda_b]$ の間には、$L(\lambda_b) - L(\lambda_a)$ 個の固有値が存在する。

例として小さい方の固有値３個が密集している付近の計算してみよう。λ を $\delta\lambda = 0.01$ 刻みに増やしながら計算してみると、３６．５以下では符号変化の回数は０であるが, ここから上では, 符号変化の回数が２となり, 更に増やしていくと, ３６．５８あたりで符号変化が３となる。少し上の記述と異なる様なそぶりを見せているので, グラフを描いてみた。



$L(\lambda)$ の怪しい祖ぶりは, ３６．５２付近で $D(\lambda)$ 曲線が λ 軸に接してせいだと思われる。即ち、ここいらで二つのほぼ縮退した固有値を持つ。３６．５２５に一つの固有値が存在する事は分かっているから, ５次方程式を解くのでなく４次方程式を解けば良い。上の記号で言えば、$D_5(\lambda) = (\alpha_5 - \lambda) \times D_4(\lambda)$ という形をしているから、$D_5(\lambda)$ は $(\alpha_5 - \lambda)$ という因子で因数分解出来る。

ここでは、上に調べた４個の区間にある固有値を一つずつ、ニュートン法と２分法を併用しながら計算しよう。即ち、区間 $[\lambda_0,\ \lambda_1]$ に固有値が一つある事を知っているから、その中央で $D_N(\lambda_2),\ D'_N(\lambda_2)$ を評価する。もしも、$|D_N|$ が小さければ λ_2 が固有値である。収束していなければ、ニュートン法での解の修正値 λ_3 を計算する。もしも、λ_3 が $[\lambda_2, \lambda_1]$ 又は $[\lambda_0,\ \lambda_2]$ に入っていれば、λ_3 を修正値として採用し、この区間から外れるようならば、$\lambda_3 = (\lambda_0 + \lambda_2)/2$ 又は、$\lambda_3 = (\lambda_1 + \lambda_2)/2$ を採用する。但し、$D_N \times D'_N$ の符号に依りどちらかを採用する。

４個の固有値は、簡単に計算出来た。

$$76.504874 \quad 41.862445 \quad 36.582725 \quad 36.524956 \quad 36.525000$$

最後の二つがほとんど縮退している。最初の数値データに誤差があるためにかろうじて固有値が異なっているという解釈もありうるだろう。

固有ベクトルの計算

元の行列と固有値が与えられたとする。逆反復法を利用するのが一般的である。次の知識を仮定する。逆行列の固有値と元の固有値は逆数の関係にあり、固有ベクトルは共通である。対角成分から定数を差し引いて作られた行列の固有値は、元の固有値から同じ定数を差し引く事で計算出来る。

任意のベクトル $\mathbf{a}$ に対して以下の連立１次方程式を解く。

$$(H - \lambda_i)\mathbf{x} = \mathbf{a}$$

この解 $\mathbf{x}$ は固有値 λ_i に対する固有ベクトルのかなり良い近似ベクトルである。旨く修正できていないかも知れないから、このベクトル $\mathbf{x}$ を $\mathbf{a}$ だとして、もう数回繰り返すともっと良い固有ベクトルが得られる。

ここでは、別のやり方をしてみよう。先ず先に登場した対称な３重対角行列 T を思い出す。そこで $\beta_4 = 0$ が非常に良い近似で成立していた。図らずも、一松信著 " 数値解析 " (朝倉書店) ページ４４縮退した固有値が別々の小集団に分散して解きやすくなるという場合に遭遇した訳である。$\lambda_5 = 36.525$ に対応する T の固有列ベクトルは、$(0, 0, 0, 0, 1)^T$ であり、残りの４本の固有ベクトルの第５成分は０である。即ち、４行４列の対称な３重対角行列の固有ベクトルを計算すれば良い。

もう一度, 固有ベクトルの定義式を思い出そう。

$$\begin{pmatrix} \alpha_1 - \lambda & \beta_1 & 0 & 0 & 0 \\ \beta_1 & \alpha_2 - \lambda & \beta_2 & 0 & 0 \\ 0 & \ddots & \cdots & \ddots & 0 \\ & & \cdots & & \\ 0 & 0 & \cdots & \beta_{N-1} & \alpha_N - \lambda \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{pmatrix} = 0$$

固有値は既に与えられている。一番上の行に関する式から次の式が書き下せる。

$$x_1 = -\frac{\beta_1}{p_1} x_2, \;\; p_1 = \alpha_1 - \lambda$$

一般に、

$$x_{i-1} = -\frac{\beta_{i-1}}{p_{i-1}} x_i$$

が成り立つと仮定して、第 i 行の関係式に代入すると、

$$p_i = \alpha_i - \frac{\beta_i^2}{p_{i-1}}$$

という漸化式を得る。

従って、計算手順は次の様になる。

$p_1 = \alpha_1 - \lambda$ とおき、p_i に関する漸化式を用いて、$p_1, p_2, \cdots, p_{n-1}$ を計算する。次に、$x_n = 1$ とおいて、漸化式 $x_{i-1} = -(\beta_{i-1}\, x_i)/p_{i-1}$ により、x_n から順に x_1 まで計算する。

最後に、ベクトル $\mathbf{x}$ のノルムを１に規格化する。全体の位相は任意である。この方法は、途中で０で割るような事がなければ、もっとも計算が単純であるという長所がある。今の場合ならば、手で計算するのも簡単である。

上で計算した固有ベクトルは 三重対角行列 T の固有ベクトルであり、H の固有ベクトルではない。この T 行列の固有ベクトルを H 行列の固有ベクトルに変換する作業が残っている。

Householder 法を用いて行列を変換したとすると、$PHP = T$ という変換であったから、H の固有ベクトル $\mathbf{u}$ は T の固有ベクトル $\mathbf{x}$ を用いて、$\mathbf{u} = P\mathbf{x}$ により計算出来る。そのために、Householder 変換をするときの直交行列を作るベクトルは行列 H の左下半分に保存しておいた。

ここでは、計算出来た事にして先へ進もう。下の表は、全体では５列あるが、一番上の行が固有値であり、その下の５個の数値が固有ベクトルを与える。

76.50487	41.86244	36.58272	36.52500	36.52495
9.926863E−1	−1.207223E−1	−1.207500E−5	−9.015457E−14	−1.417526E−6
8.540712E−2	7.022982E−1	3.577200E−2	−5.210233E−2	−7.039094E−1
8.428395E−2	6.930375E−1	1.199299E−1	5.199733E−2	7.039234E−1
5.965901E−3	4.910201E−2	−4.459280E−1	8.928481E−1	−3.903544E−2
1.183717E−2	9.742343E−2	−8.862761E−1	−4.443014E−1	8.648349E−2

第１列を見ると、第１成分が突出しているから、C_0 は、他の成分とはほぼ独立にパラメータが決まっている。第２列と第５列を見ると、第２、３成分が頭一つ、他の成分より大きくなっている。$C_1 \pm C_2$ という方向がほぼ独立な方向である。C_1 軸と C_2 軸のほぼ中央を走っている軸方向の固有値は 41.86、これに直交する軸に対しては固有値が 36.52 である。$S_1,\ S_2$ 軸方向には第３列と第４列ベクトルが大きな値を持っている。固有値もほとんど等しいので、$S_1,\ S_2$ は他の成分ともほとんど独立に決定されていると言って良いだろう。

今の場合、南中時刻をフーリエ展開したが、フーリエ級数はお互いに直交する基底であるから、相互の独立性が良かったのだと想像される。もしも、t/T の級数展開をしたならば、もっと酷い結果が得られたと思われる。

固有値と固有ベクトルが計算出来たところで、$C_0, \cdots, S_2$ の決定精度に関する議論をしよう。決定精度を 1/(固有値の平方根) で定義すると、第１、３、４列のベクトルには、際だって大きな成分があるから、この成分は１で、残りの成分は０と近似すると、分単位で表して、

$$C_0 = 697.90 \pm 0.11, \quad S_1 = 7.68 \pm 0.17, \quad S_2 = 9.24 \pm 0.17$$

とするのが妥当だろう。ここで、誤差は二桁を有効とした。各パラメータは、この誤差に併せて誤差の桁と合うところで四捨五入した。

$C_1,\ C_2$ の誤差は、少し注意が必要である。第２、５列のベクトルの２、３成分がの大きさ

が $1/\sqrt{2}$ と近似すると、以下の式が成立する。即ち、C_1, C_2 方向の単位ベクトルを $\mathbf{e}_1$, $\mathbf{e}_2$ と書き、(C_1+C_2), (C_1-C_2) 方向の単位ベクトルを $\mathbf{v}_1$, $\mathbf{v}_2$ と表すと、両者には以下の関係がある。

$$\mathbf{e}_1 = (\mathbf{v}_1+\mathbf{v}_2)/\sqrt{(2)}, \quad \mathbf{e}_2 = (\mathbf{v}_1-\mathbf{v}_2)/\sqrt{(2)}$$

$\mathbf{v}_1$, $\mathbf{v}_2$ 方向の誤差ベクトルの大きさを、$\delta V_1\mathbf{v}_1$, $\delta V_2\mathbf{v}_2$ と書くと、$\mathbf{e}_1$ 方向の誤差ベクトルは $\delta E_1\mathbf{e}_1 = \delta V_1\mathbf{v}_1 + \delta V_2\mathbf{v}_2$ である。$\delta V_1 = 1\sqrt{41.86244}$, $\delta V_2 = 1/sqrt36.525$ を代入すると、$\delta E_1 = \sqrt{(1/41.86244+1/36.525)/2} = 0.16$ これらより、

$$\delta C_1 = 1.62 \pm 0.16, \quad C_2 = -1.99 \pm 0.16$$

と計算される。

１０．入・出力

変数に数値を代入する、又は計算結果の数値を出力するには、READ 文か WRITE 文を利用する。ここでは、変数又は配列とファイルの間でのやりとりを想定する。ファイルは、論理機番 (logical unit number, 略して LUN) という数値で識別される。LUN はプログラム中では、OPEN 文で定義される。プログラムの外で、job control language を用いて、定義する人もいる。

一つのファイルのなかで、前から順番に並んだデータを想定する。ファイル上のデータは、通常のエディターで読み書きする事を想定し、文字データファイル (ascii file) であるとする。

次の文は、LUNIN という論理機番のファイルからデータを入力し、LUNOUT という論理機番のファイルに出力する。

```
      DIMENSION A(10),IARRAY(20,2)
      LUNIN=1
      OPEN(UNIT=LUNIN, FILE='infile',STATUS='OLD',ERR=90)
      READ(LUNIN,10)I1,I2,F1
   10 FORMAT(2I3,F12.5)
      READ(LUNIN,*)IARRAY
      READ(LUNIN,11)(A(I),I=1,5)
   11 FORMAT(5F10.0)
      CLOSE (LUNIN)
C
      LUNOUT=2
      OPEN(UNIT=LUNOUT,FILE='outfile',STATUS='UNKNOWN')
      WRITE(LUNOUT,21)I1,I2
   21 FORMAT('I1=',I3,2X,'I2=',I3)
```

```
      DO 25 I=1,2
      J0=1
   22 J1=J0+3
      IF(J1.GT.10)J1=10
      WRITE(LUNOUT,23)I,(J,IARRAY(J,I),J=J0,J1)
   23 FORMAT('I=',I2,2X,4('IARRAY(',I2,',I)=',I4,2X:))
      J0=J1+1
      IF(J0.LE.10)GOTO 22
   25 CONTINUE
C
      WRITE(LUNOUT,26)A
   26 FORMAT('A=',5F9.4)
C
C      some useful jobs
C
      STOP
C   error message
   90 WRITE(6,91)
   91 FORMAT('infile open failure !')
      STOP
      END
```

前半では、infile という名前のファイルを開き、そこからデータを読み込んでいる。

もしも、ファイルを開く事に失敗すると、９０というラベルの位置に飛んで、エラー文を出力して、プログラムは終了する。ここで使用された論理機番６というのは、標準的な出力に割り当てられている場合が多い。

infile というファイルの第１行目にある整数２個を、３文字の整数と仮定し、その次の１２文字は、通常の実数表現で書かれた数値として読み込む。この実数の書式は、１２文字の内、後ろから６文字目は小数点である事が仮定されているが、実際の文字列での小数点の位置を優先して読み込む。従って、ここでの書式 F12.5 は、 F12.0 であっても、正常に働く。即ち、入力時には、Fw.d と書いた時の d には意味がほとんど無い。

整数型配列 IARRAY の読み込みには、添字が使用されていない。この場合には、IARRAY の全部の配列要素にデータが読み込まれる。IARRAY へデータが読み込まれる順番は、添字を書くと以下の通りである。(1,1), (2,1), (3,1), ···, (18,1), (19,1), (20,1), (1,2), (2,2), ··· (19,2), (20,2)

FORTRAN では、複数次元の配列のデータは、前の添字が先に動く様に並んでいる。

配列 A は１０個の要素があるが、そのうちの前半５個にのみ、データを読み込んでいる。

CLOSE 文は、LUNIN という論理機番のファイルとプログラムの関係が切れることを意

味する。

次に、読み込んだ内容をファイル名が outfile というファイルに書き出している。最初は I1, I2 という２変数の値が３桁で出力される。次に、IARRAY の一部が一行に４個ずつ出力される。この FORMAT 文では、コロン記述子が使用されている。このコロン記述子の動作を確認したければ、コロンがある場合と無い場合の出力の様子を比較すると良いだろう。

最後に、配列 A のデータ５個が１行に書き出される。ここでは、5F9.4 という書式を使用している。５個の数値を、それぞれ９桁で、そのうち小数部分は４桁、書き出せという意味である。プログラマー最初の５個だけを出力してこれでよしとしている。しかし配列 A の大きさは１０あるので、５個のデータを出力すると、指定された書式を使い尽くす。この場合には、もう一度前頭から、書式と出力データを対応させる。即ち、書式だけ繰り返して使用する。計算機は A(6) から A(10) 迄を 5F9.4 という書式で次の行に出力する。

論理機番 LUNOUT で示されるファイルは明示的には閉じられていない。プログラムが STOP 文まで正常にたどり着けば、どこかで、誰かが後始末をしてくれる。通常は OS と呼ばれるプログラムの仕事である。OS に任せても良いし、明示的にプログラム中で CLOSE 文を用いてファイルを閉じても良い。

同一のプログラムで再度 infile というファイルを使いたいならば、もう一度 OPEN 文を書けば良い。このような事情は、計算の途中経過をファイルに書き出す時に起こる。CLOSE と OPEN を行う代わりに、以下の様な手法を使う場合もある。

```
C      insted of writing following statements
       OPEN(UNIT=LUNOUT, FILE='outfile',.. )
C        write some data onto 'LUNOUT'
       CLOSE(LUNOUT)
       OPEN(UNIT=LUNOUT, FILE='outfile',.. )
C      one can use these statements
       OPEN(UNIT=LUNOUT, FILE='outfile',.. )
C        write some data onto 'LUNOUT'
       ENDFILE (LUNOUT)
       REWIND (LUNOUT)
C        now you can read the calculated results
C        from LUNOUT file
```

上の例で、一行読んだ後で、もう一度同じ行を読みたい場合には、BACKSPACE (LUNOUT) を使用すればよい。例えば C 言語で言えば ungetc で１文字読み戻しておく様な場合である。

実数の出力に科学的表現を用いたいならば、F という文字の所を E という文字で置き換え、Ew.d という書式とする。この場合には、数値表現が (符号) + (文字 ０) + (小数点) +

(整数) + (文字 E) + (符号) + (2 桁の整数) となるので、$w \geq d+7$ としておく必要がある。整数部が 0 となるのは、表示上の無駄だと思うならば、桁移動子 P を利用して、nPEw.d と書くのが良い。ここで、n が 1 ならば (文字 0) と書いた部分は一桁目の整数が入る。例えば E14.5 という書式で −0.12345E+02 と出力されていたならば、1PE14.5 の書式では −1.23452E+01 と出力される。科学的書式では、表現は変化するが、数値の値が変化するわけではない。桁移動子を用いて、m 個のデータを出力したいときには、 nPmEw.d という記法を使う。

桁移動子の作用は、一つの FORMAT 文中では、後続の書式にも影響を及ぼす。問題が発生するのは、Fw.d 型の記述が桁移動子の後に使用された場合である。この場合には、桁移動は有効であるが、指数部がないために、数値の補正をする機会がない。従って桁移動子の後では、必ず桁移動子の効果を元に戻しておく必要がある。mFw.d ではなく 0PmFw.d と書いておく必要がある。

例示する。E=−1.6021E−19, P=3.14156 だとし、FORMAT(1PE12.4,2X,F8.4) と、FORMAT(1PE12.4,2X,0PF8.4) の二つの書式で変数 E と P を書き出された結果は、以下の様になる。

前者の `FORMAT  -1.6021E-19   31.4156`
後者の `FORMAT  -1.6021E-19    3.1416`

後者では、最終桁が四捨五入されている。

実数の書式で、F 形式と E 形式のどちらが良いか分からないときには、Fw.d と書く代わりに、Gw.d としておくと良いだろう。もっと手抜きをしたい人は、5F9.4 という表現を星印で置き換えても良い。

書式の修飾

次に、FORMAT 文の変形を行ってみよう。ラベル 2 1 の FORMAT 文では変数 I1, I2 は整数型で、3 桁の文字として出力している。もしも I1 は 5 桁の大きさ (10000 以上又は −1000 以下の数) ならば、この FORMAT 文では出力が出来ない。以下の様にして FORMAT 文自体をプログラムで作り出すと、このようなトラブルは回避出来る。説明を簡単にするために、I 1 だけを処理してみよう。

```
CHARACTER FMT*4
FMT='(I3)'
KETA=LOG10(FLOAT(ABS(I1)))+1.1
IF(I1.LT.0) KETA=KETA+1
IF(KETA.GT.3)WRITE(FMT(3:3),'(I1)')KETA
WRITE(LUNOUT,FMT)I1
```

先ず、文字型の変数 FMT を宣言する。この宣言文により FMT という変数には最大 4 文字まで文字が書き込める。

次の文は、変数 FMT に文字列 (今の場合４個の文字) ” 左括弧 ”、文字 ”Ｉ”、文字 ” ３ ” 及び ” 右括弧 ” を書き込む。

次の２行で、変数Ｉ１を文字に変換した時の文字数を KETA という変数に定義している。LOG10(x) は x の常用対数を計算する。対数の定義により、引数は正の実数でなければならないから、ABS(I1) により、絶対値に変換し、FLOAT（）文で、整数を実数に変換している。1 を加えれば良いところへ 1.1 を加える理由は、非常に小さな誤差のせいで、整数に変換した時に１だけ小さな数値になる可能性を打ち消すためである。

次の IF 文は、Ｉ １が負の場合、負号を書くために１文字文増やす必要があるから、KETA ＝ KETA ＋１を実行する。

次の行は、KETA が４以上ならば、FMT の第３字目 (３という文字が書いてある部分) を 変数 KETA に定義された文字で置き換えている。

(I:J) という表現は、文字型変数の先頭から数えて、I 文字目から J 文字目までを指定する。今の場合、KETA が１０以上の数字ならば破綻を来す。

最後の行は、変数 FMT の中身を FORMAT 文の中身だと解釈して、変数Ｉ１を LUNOUT で示されたファイルに書き出している。

WRITE(FMT(3:3),’(I1)’)KETA の様に、論理機番を書くべき所に、文字型の変数 (文字型の配列でも可能) を書くやり方を内部ファイルへの出力文という。内部ファイルの入・出力文を使うと、数字を文字列に又は逆に文字列を数字に変換する事が非常に簡単に行える。C 言語での atof, itoa といった関数の利用に対応している。

プログラムの実行中に FORMAT 文の中身を変更して良いという事は、FORTRAN コンパイラーは、FORMAT 文の中身をコンパイルしていない事を意味する。この機能は非常に便利であるが、半面、出力の書式変換に時間がかかる事を意味する。

高速なプログラムを書こうと思うならば、不必要に書式変換をしない方が良いとも言える。

もう少し文字列の取り扱いに付いて書こう。内部ファイルへの入出力が可能という事は、非常に広い応用の可能性を意味する。例えば、入力文字を一字ずつ判読して、プログラムの使用者がプログラムに伝えたい事を、言葉で表現する可能性があることを意味する。この事は、プログラム言語がアセンブラーから FORTRAN への発展に際して経験した事である。

個人的には、FORTRAN 4 の時代から元素記号を入力して、原子番号や質量数を識別し、補助的なデータを用いて、原子 (核) 質量の正確な値を定義している。これにより、入力数値のミスがなくなった。

書式なしファイル

計算結果を予備的なファイルに保存する場合、計算結果の有効数字は全て後の計算に利用したい。計算機の内部表現を、人間が見やすい形式に変換する時に、有効数字の一部が失われてしまうだろうし、変換に時間がかかる。更に、計算機内部では１語で表現されていたものが、複数文字で表現されるから、ファイルの有効利用という面からも好ましくない。このような事を勘案すると、計算機の内部表現をそのままファイルに書き出すという方法も利用したくなる。このようなファイル形式を書式なしファイルと呼ぶ。これまでのファイルは、

書式ありファイルである。書式なしファイルの入・出力にも READ 文や WRITE 文が使われるが、FORMAT 指定が不要である。FORMAT 指定をしてはいけないと言った方がよいかな。

書式なしファイルは、どのような変数やデータが、どのような順番で書き出されたかという判断が出来ないから、書き出しと読みだしに絶対的な自信がある場合以外はしないほうが良い。一つのファイルで、一部は書式あり、残りは書式なしという様な混用は出来ない。書式の有無は ファイルの OPEN 文で指定する。

OPEN(UNIT = lun, FILE='filename',FORM='FORMATTED') と指定すると、書式ありのファイルである。FORM='UNFORMATTED' と書くと、書式なしファイルとなる。これまでの例では、FORM='format' という記述を省略していた。省略すると、FORM ='FORMATTED' という記述がなされていると、どこかで判断していたのである。このように、指定しないと何某の指定がなされたものとするという事を、default は何某であるという表現をする。

計算の途中だけで使用し、後から利用する可能性が無いファイルを使用したい場合には、OPEN 文の STATUS パラメータに SCRATCH と指定する事が出来る。SCRATCH ファイルは、OS 側で準備をしてくれて、使用後は OS 側が引き取るので、(すぐに消して再利用する) 個人が用意する (用意できる) ファイルよりも大きなファイルを利用可能かも知れない。この場合には、ファイルには名前は無く、ただ論理機番があるのみである。

buffering

外部記憶装置の動作は、電子よりも極端に質量が大きな物体が動くために、非常に遅い。外部記憶装置にデータを書き込む際には、出来るだけ効率的なプログラムを組む必要がある。その単純な指導原理として、入・出力の回数を減らし、規則的に行うというのがある。OS のファイルシステムと媒体の基本的な構造によるが、1 KB とか 8 KB とかの切りの良い数字での入・出力が基本であろう。

これに会わせるために、プログラム中では対応する記憶域を宣言し、この記憶域へ順次データを書き込み、この記憶域が一杯になったら、外部記憶装置に書き出す。

外部記憶装置が動作中にも、データ生産が続けられている状況ならば、複数の記憶域をプログラム中にとり、これらの複数の記憶域を循環的に使用する。このような概念を ring buffer と呼ぶ事もある。

実際の運用に於いては、OS がこの作用 (の一部) を分担している場合もあるし、外部記憶装置そのものにも、入出力の記憶領域を持っている場合があるだろう。

ring buffer という概念は、大きな記憶域を必要とするプログラムのポインターにも応用可能である。すなわち、記憶域にあるデータを動かさずに、記憶域へのポインターを動かす訳である。

順編成と乱編成ファイル

これまでの説明では、ファイルの内容は書き出された順番に読み出す。このようなファイルは、順編成 (sequential access file) ファイルと呼ばれる。時には、書き出した順番と読み出したい順番とが一致しない場合もあるだろう。この要求に対応するファイルとして、乱編成 (random access file) という概念がある。これらの単語は、標準的な用語ではない。

外部記憶装置に一定の大きさの記録領域を複数個確保し、この一定の記憶領域に、記録番号と一緒にデータを書き込んでいく。読み出す場合には、記録番号を指定して読み出す。従って、書き出す場合に何番の記録番号の位置には、どんな内容の記録が書き込まれているかという情報をどこかに記録しながら、書いていく必要がある。

乱編成ファイルを、テープ上に書く事は実用的でない。磁気ディスク上にとる場合には、磁気ディスクの物理的な制約を知っておく必要がある。例えば、１シリンダーには幾らのトラックがあり、一つのトラックに書き込める文字数は幾らであるか等である。一人の利用者が利用できる記憶装置の大きさも、計算機の運用規則で制限されている場合が多い。さらに、OS の制限を受ける場合もあるだろう。

個別のディスク装置毎に最適な記録領域の大きさがあるので、メーカーや保守担当者に確認しておかないと、非常に不経済な書き込み方をしてしまう場合がある。記憶領域の大きさは OPEN 文の RECL パラメータで指定する。FORTAN にはバイトという単位は想定されていないが、物理的な記憶装置はバイトという単位を使用している場合が多いだろう。

乱編成である事は、OPEN 文の ACCESS パラメータで指定する。

乱編成ファイルのデータを読む場合、先ず指定された記録番号の位置を探さねばならない。ディスクならば、当該位置まで磁気ヘッドを動かさねばならない。これにはかなり時間がかかる。そこで、READ 文が実行される前に、あらかじめ磁気ヘッドをその位置まで動かしておく事が出来れば運用効率が上がるかも知れない。このために、FIND 文がある。但し、複数の利用者や、複数のプログラムが同じ磁気ディスクを利用していると、磁気ヘッドの取り合いをするかも知れない。

運用に依るが、乱編成ファイルは主記憶上に置かしてくれるかも知れない。この場合には、非常に高速な補助記憶装置として使用できる。

一般論はやりにくいので、文法、運用規則、物理的な実体等を良く調べておく必要がある。

FORTRAN プログラムでは、A=B+C という文は、変数 B で示される記憶域の内容と、変数 C で示される記憶域の内容を加えた、その結果を変数 A で示される記憶域に書き込めという事であった。

即ち、変数と我々が呼んでいるのは、記憶域番地とその大きさ (文字列、整数型、実数型、複素数型等でその記憶域の大きさは異なる可能性がある) である。配列に対してもこの記憶域を指し示すという事情は同じである。

副プログラムを呼び出す場合を考えてみよう。

呼び出し側：

```
DIMENSION X(50)
  N=29
```

```
CALL SUBA(N, X(21), Y, Z)
```

副プログラム側： `SUBROUTINE SUBA(M, P, Q, R)`

```
DIMENSION P(M)
```

呼び出し側では、X を大きさ５０の配列と宣言している。当面 Y、Z は無視しておこう。実引数の２番目では X(21) となっているから、第二変数の意味する所は、X(２１) で示される記憶域の位置 (メモリー上の番地) である。

副プログラム側では、第二変数 (引数) の名前は P であり、P は大きさが M (＝２９) の配列と宣言されている。配列 P の先頭番地は、配列 X の２１ 番目の番地を指す事になる。従って、呼び出し側と副プログラム側では、以下の様な記憶域の共有が起こっている。

配列名									
X	20	21	22	⋯	47	48	49	50	51[a)]
P	−[a)]	1	2	⋯	27	28	29	−[a)]	−[a)]

a) 対応する記憶域は存在しない

コンパイラーに依存するが、P(35) や P(−2) が使用できる場合がある。特に、I=−2, P(I) と言うように変数 I を経由して、配列を利用すると、意図しない動作をする場合がある。

配列宣言された時、例えば呼び出し側での配列 X に対する記憶域の割り当ては文法では特に規定がないから、例えば X(15) と X(16) の間で記憶域が連続している必要は無い。この事は、EQUIVALENCE 文を使用する時に注意が必要である。

一方、共通領域内に配列をとった場合には、文法規則により、共通領域内にある全ての変数に対する記憶域は連続している必要がある。この点で、配列宣言をプログラム単位で行うのと、共通領域で行うのとは訳が違う。

ある種のコンパイラーでは、変数の種類により奇数 (偶数) 番地から開始する方が計算効率が上がる様な構造になっている。その場合には、共通領域にわざとに空き領域を差しはさむ場合がある。これを pudding と呼んでいる。

これらの知識があると、OS の paging(swapping) による時間のロスを最小限にして、記憶域の効率的な利用法を考える事が可能になるだろう。

１１ 有理数表現での計算

Clebsch-Gordan 係数や Racah 係数は、厳密な式が知られている。従って、有理数で数値を表現できるならば、厳密な数値計算が出来るかもしれない。

例えば Clebsch-Gordan 係数の Racah による式は以下の通りである。

$$
\begin{aligned}
&< j_1\, m_2\ j_2\, m_2 | j_3\, m_3 >= \Delta(j_1,\, j_2,\, j_3) \\
&\times \sqrt{(2j_3+1)\,(j_1+m_1)!\,(j_1-m_1)!\,(j_2+m_2)!\,(j_2-m_2)!\,(j_3+m_3)!\,(j_3-m_3)!} \\
&\qquad \times \sum_{\nu} (-1)^{\nu} / \{\nu!\,(j_1+j_2-j_3-\nu)!\,(j_1-m_1-\nu)!\,(j_2+m_2+\nu)! \\
&\qquad\qquad \times (j_3-j_2+m_1+\nu)!\,(j_3-j_1-m_2+\nu)!\}
\end{aligned}
$$

ここで、

$$\Delta(a,\,b,\,c) = \sqrt{\frac{(a+b-c)!(b+c-a)!(c+a-b)!}{(a+b+c+1)!}}$$

Racah 係数に対する Racah の式は以下の通りである。

$$W(a,\,b,\,c,\,d;\,e,\,f) = \Delta(a,\,b,\,e)\,\Delta(e,\,d,\,c)\,\Delta(b,\,d,\,f)\,\Delta(a,\,f,\,c)$$
$$\times \sum_{\nu} \frac{(-1)^{a+b+c+d+\nu}(\nu+1)!}{\nu-a-b-e)!\,(\nu-e-d-c)!\,(\nu-b-d-f)!\,(\nu-a-f-c)!}$$
$$\times \frac{1}{(a+b+c+d-\nu)!\,(a+d+e+f-\nu)!\,(b+c+e+f-\nu)!}$$

ここで大切な件は、沢山の階乗が登場するから、大きな数字が登場する事である。そこで、分子や分母に登場する数値を素数の積で表現し、この冪指数を配列に書き込む事にしよう。例えば、１００=$2^2 \times 5^2$ とする訳である。

この時、数を表現するには、項数・符号・冪指数の表を用意すれば良い。そうすると、以下の様な機能を有する副プログラムを準備すれば良い。

0　素数表を作る。
1　整数を与えて、素数の積に変換する
2　素数の積で与えられた二つの数の和を計算する
3　素数の積で与えられた二つの数の積を計算する
4　素数の積で与えられた二つの数の商を計算する
5　素数の積で与えられた数を、通常の実数に変換する

素数表は、IPTAB(n) という配列に小さい順に書き込まれる。素数表の大きさは、NSIZE というパラメータ文で定義している。

和を作る方法だけ、記録しておこう。素数積表現した二つの数 $a,\,b$ を与えたとする。この両者の冪表現から、共通部分 c を取り出す。即ち、$a = c \times A,\,b = c \times B$ と書いた時、$A,\,B$ はどちらも整数である。この $A + B = C$ を計算し、整数 C を素因数分解し、共通因子との積を計算すれば良い。

添付したプログラムでは 上で示した １００ を以下の配列 IA で表現している。IA(-1)=3, IA(0)=1,　IA(1)=2,　IA(2)=0,　IA(3)=2

解釈は、IA(-1) が項数であり、最大素数 5 に対するポインターが３であるから、IA(-1)=3 とする。１００は正の数であるから IA(0)=+1 とする。負の数ならばここが -1 とする。但し数０ならば、IA(-1)=0 としている。IA(n) は小さい方から n 番目の素数の指数が書かれている。この数値は負の整数が書かれる場合もある。

絶対値が１である数は、全ての冪指数が０であるが、これは IA(-)=1,IA(0)=±1, IA(1)=0 と規格化する。

添付のプログラムは働くと思うが、テストは充分ではない。

速度を上げる工夫や、非常に大きな素数が登場した時にお手上げとならに工夫が必要である。