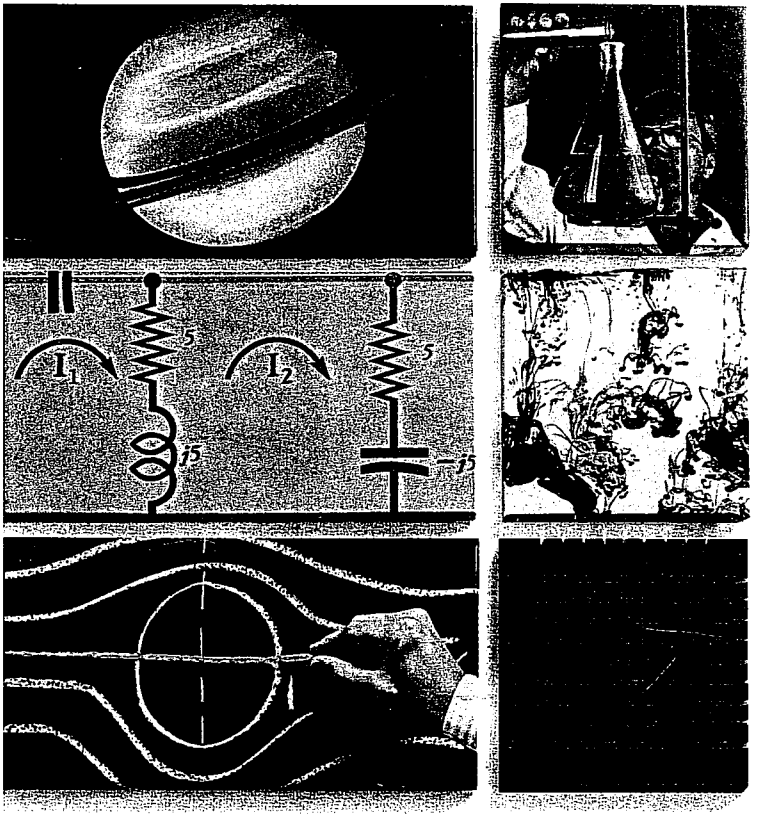
横河・ヒューレット・パッカード株式会社

HP-15C

操作ハンドブック



御　注　意

YHPはこの本に記載したキー操作順序やプログラム内容に関しましては，どんな場合にでも必ず正しい答を求められるとは保証いたしておりません．特殊な場合には別のキー操作や別のプログラムが必要になることがあります．この本のキー操作順序やプログラムはこのような計算のときにはこのようにするのだという一例にすぎません．従って取引上の計算などにはご使用の方が広範囲にテストした上で利用するかどうか決めてくださるようお願いいたします．またそのためにプログラム等を改良する必要がありましたときは，ご使用の方の責任と費用で改良してくださるようお願いいたします．ご使用の方がこの通りに操作したために発生した逸失利益・金銭上の損害や第三者からの請求を受けることがありましても，YHPはその責任を負いませんので御了承くださるようお願いいたします．

横河・ヒューレット・パッカード株式会社

HP-15C
操作ハンドブック

00015-90008 Rev. B

Printed in Canada

この本の内容

HP-15C をお買い上げくださいましてありがとうございました。当社の計算機は初めてだという方も，当社の計算機には慣れている方でも，今度お買い上げくださった HP-15C は従来の計算機では想像もできなかったような性能に気が付かれると思います。不揮発性メモリと低消費電力だけでなく，次のような最新技術を採用しています。

- プログラム用メモリーの容量を 448 バイト（一つの命令で 1～2 バイト使用）にして，条件ジャンプや無条件ジャンプ，サブルーチン，フラグ，それにプログラム編集機能などのように高級なプログラム能力を用意。
- 高等数学用の機能（複素数計算，行列演算，求根計算，数値積分の 4 種）の内蔵。
- 数値記憶用のレジスタを 67 個用意して直接指定または間接指定で操作可能。
- 電池の長寿命化。

この本はお使いになる人の経験などを考慮して 3 部に分けました。第 1 部は HP-15C の基礎知識で，HP-15C の基本機能とその操作法です。第 2 部はプログラムの作成と利用で，章ごとに仕組み，例，詳細説明の三つに分けて説明するので，使う人の必要度に応じて読んでください。第 3 部は高等数学用機能で，4 種の高等数学用の機能の説明です。*

各章を読み始める前に，HP-15C の操作法やプログラム作成法を知っておきたい人は，13 ページからの HP-15C 操作法入門を読みながら実際に操作してください。この本だけを読むのではなくて，HP-15C も一緒に並べてこの本の通りに操作すれば，理解が早くしかも確実に覚えられます。

＊当社の計算機に慣れている人は，第 1 部と第 2 部を飛ばして，直接第 3 部へ進んでも良いでしょう。しかし [SOLVE] と [∫] を使うには HP-15C のプログラム作成の知識が必要です。

付録の部分は本文中以外の計算機の操作，保証や修理について記載しました。巻末の機能の要点と索引とプログラム用機能の要点と索引は各キーの記号の意味と使用法の要点で，それ以上の説明は該当のページを見てください。

この本以外に HP–15C Advanced Functions Handbook（英文版だけ約 220 ページ，製品番号 00015–90011）もあって，操作ハンドブックの第 3 部のもっと詳しい説明や応用例とかプログラム例を記載してあるので，HP–15C をもっと活用したい方にお勧めできると思います。（申し訳ありませんが現在のところこの本を和訳する予定はございません。）

目　　次

HP-15C 操作法入門

HP-15C は科学技術者用のプログラム可能な高級計算機で，種々の関数を内蔵していてしかも操作し易くて携帯に便利な頼りになる計算機です。大切なプログラムやデータは不揮発性メモリー中に記憶していて，それを消すまで消えません。それにプログラム作成の経験や，プログラムについての知識は不要です。

HP-15C のもう一つの特長は電力消費が非常に小さいことです。そのため計算機全体を軽量で小型にまとめて，重くて邪魔な AC アダプターが不要になりました。HP-15C を普通に使っていると 6～12 か月位は電池交換が不要です。電池が使えなくなる前に電圧低下の表示で予告します。

更に HP-15C は電池の寿命を伸ばすために，計算が終ってもそのままにしておくと自動的にスイッチが切れるようにしてあります。しかしスイッチが切れても重要なデータなどは HP-15C の不揮発性メモリー中に記憶しているので慌てる必要はありません。

ENTER の要点

当社の計算機は他社とは違って ENTER キーを使った独特な計算方式（日本語方式，正しく言えば逆ポーランド方式，RPN）を採用しています。ENTER キーの使い方に慣れると，複雑な計算も少ないキー操作で手早く計算できるようになります。そのためにちょっとキー操作をしてみましょう。

まず簡単な加減乗除をやってみましょう。最初に計算機に数値をキー入力する必要があります。計算機のスイッチが入っていますか？ 入っていなかったら ON を押してください。表示が 0 になっていますか？ 表示を 0 にするには g CLx，つまり g を押してから ← を押します*。計算をするには第 1 の数値をキー入力してから，ENTER

＊今までに当社の計算機を使ったことのない方は，大部分のキーに 3 種の記号が付いているのに気が付いたと思います。キーの上面の白文字(第一機能と呼びます)を使いたいときは，そのキーだけを押します。黄文字や青文字（第二機能と呼びます）を使いたいときは，その文字の色に応じて黄色の f キーまたは青色の g キーを先に押してから目的のキーを押します。

を押して第 1 の数値と第 2 の数値を区切り（第 1 の数値を入れ終ったと合図し），次に第 2 の数値をキー入力してから目的の [+]，[−]，[×] や [÷] を押します。目的の計算キーを放すと直ちにその答を表示します。

この本では特に明示したところ以外は [FIX] 4（小数点の右側に数字を 4 桁だけ表示する指定）になっているものとして説明を進めます。もしこのようになっていなかったら [f] [FIX] 4 と押すと例題の表示と同じになります。

手 計 算

次の 2 数の計算をやってみてください。計算ごとに前の計算の答を消す必要はありません。もし数字を入れ間違えたら，間違えたところまで [←] を押し，それから正しい数字をキー入力してください。

計算の目的	キー操作	表　示
9 − 6 = 3	9 [ENTER] 6 [−]	**3.0000**
9 × 6 = 54	9 [ENTER] 6 [×]	**54.0000**
9 ÷ 6 = 1.5	9 [ENTER] 6 [÷]	**1.5000**
$9^6 = 531{,}441$	9 [ENTER] 6 [y^x]	**531,441.0000**

上の 4 題で覚えて頂きたいことは次の 3 点です。

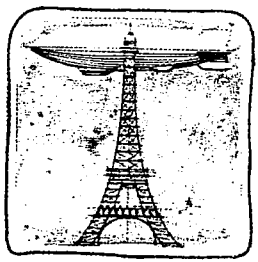
- 計算機に二つの数値をキー入力してから，計算目的のキーを押す。
- [ENTER] は二つの数値を区切るために使うので，第一の数値をキー入力してからこのキーを押し，それから第二の数値をキー入力する。
- 計算目的のキー（上例では [−]，[×]，[÷] と [y^x]）を押すと直ちにその計算をして答が見られる。

手計算とプログラム計算の関係を知るために，まず一つの問題を順番にキーを押す手計算でやってみましょう。次に同じ問題をプログラム計算でやってみて，ついでに別の数値についても計算してみましょう。

空気の抵抗を無視すると，ある高さから品物を落としたときに地面に落ちるまでの時間は次の計算式で求められます。

$$t=\sqrt{\frac{2h}{g}}\ ,$$

ここで t = 所要時間（秒）
h = 高さ（m）
g = 地球の重力で 9.8m/s²



例　パリのエッフェル塔の頂上（300.51 m）から小石を落としたときの所要時間を計算してみましょう。

キー操作	表　示	
300.51 [ENTER]	**300.5100**	高さのキー入力。
2 [×]	**601.0200**	$2h$ の計算。
9.8 [÷]	**61.3286**	$2h/g$ の計算。
[√x]	**7.8313**	落下時間（単位は秒）。

プログラムを使った計算

種々の高さからの落下所要時間を計算したいと思いませんでしたか？一番楽な方法は高さだけをキー入力すれば，後は自動的に計算するプログラムを作って利用することです。

プログラムの作成　プログラムは上でキー操作したのとほとんど同じです。プログラムの始めの部分にラベル（目印，符号）を付けると使い易くなりますし，終りの部分には [RTN] を付けます。計算ごとに新しいデータ（数値）を入れることを前提にしてプログラムを作ります。

プログラムの入力　上の問題のプログラムの入力方法は次のキー操作の通りです。（キーを押すと種々の表示が見られますが，それは後で説明するので，ここでは無視して結構です。）

キー操作	表　示	
[g] [P/R]	**000-**	HP-15C をプログラム入力状態にする。（表示部に **PRGM** の文字が見えるようになります。）

f CLEAR PRGM	000-		プログラム用メモリーを消す。(ここではこの操作を省略できます。)
f LBL A	001-	42,21,11	A をプログラムの始めに指定する。
2	002-	2	これは手計算のときと同じ操作。
×	003-	20	
9	004-	9	
·	005-	48	
8	006-	8	
÷	007-	10	
√x	008-	11	
g RTN	009-	43 32	RTN はプログラムの終りという意味です。
g P/R	7.8313		計算状態に戻す(PRGM の文字が消える)。

プログラム計算の実行　次のデータを入れてプログラム計算をやってみましょう。

キー操作	表　示	
300.51	300.51	エッフェル塔の高さ。
f A	7.8313	前の計算と同じ答になった。
1050 f A	14.6385	高さ 1050 m の気球から小石を落としたときの落下時間 (秒)。

このプログラムを入れてあれば，どんな高さからの落下時間も直ちに求められます。高さをキー入力して f A を押すだけです。それでは 100 m, 2 m, 275 m と 2,000 m から落としたときの時間を求めてください。

答は 4.5175 秒，0.6389 秒，7.4915 秒，20.2031 秒のはずです。

プログラムはこのように簡単です。第 2 部のプログラムの作成と利用でたくさんのことを説明します。まず第 1 部から順に計算機の重要な事項を覚えてください。

第 1 部

HP-15C の基礎知識

第 1 章

操作法の第一段階

スイッチの入り切り

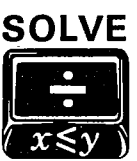
[ON] を押すと HP-15C のスイッチが入ったり切れたりします*。電池を長持ちさせるために，何もしないで約 10 分たつと自動的に切れるようにしてあります。

キ ー 操 作

第一機能と第二機能

HP-15C の大部分のキーは一つの第一機能と二つの第二機能（別のキーを併用する）の兼用になっています。どのキーも第一機能はキーの上面に文字や記号で表示してあります。第二機能はキーの向こう側に黄文字で印刷してあるのと，キーの手前側の斜面に青文字で表示してあるものです。

- キーの上面に表示している第一機能を使うにはそのキーを押すだけです。右図では [÷] です。
- 黄文字か青文字の第二機能を使うには，まず同じ色の前操作（[f] または [g]）キーを押し，次に目的のキーを押します。右図では [f] [SOLVE] や [g] [x≤y] です。



この本では第二機能のことを次のように表記することにします。機能そのものを指すときには機能の略号を枠で囲んで，[MEM] 機能のように表記します。キー操作を指すときには前操作キーも一緒に表示して，[g] [MEM] と押すように表記します。また黄文字の CLEAR と一緒に角括弧がついている 4 種の第二機能は，その前に CLEAR の文字を付け

---

* [ON] キーはうっかり押さないように外のキーよりも低くしてあります。

て，CLEAR [REG] 機能とか [f] CLEAR [REG] と押すように表記します。前操作キーの [f] または [g] を押すと，そのキー操作全体が完了するまで表示部に **f** または **g** の文字を表示します。

前操作キー

前操作キーとはある機能のために，キーを2～3回続けて押すときに最後に押す以外のキーのことです。大別して2種あって，前操作キーと数字キーとの組み合わせ，前操作キーと別のキーとの組み合わせです。参考のために前操作キーを図示しておきます。

[CF]	[ENG]	[FIX]	[GSB]	[∫]	[MATRIX]	[SCI]	[STO]
[DIM]	[f]	[g]	[HYP]	[ISG]	[RCL]	[SF]	[TEST]
[DSE]	[F?]	[GTO]	[HYP^{-1}]	[LBL]	[RESULT]	[SOLVE]	[$x\rightleftarrows$]

前操作キーを押し間違えたときは，[f] CLEAR [PREFIX] と押すと取り消しできます。この CLEAR [PREFIX] キーは表示している数値の仮数部（有効数字）10桁を表示する働きもあって，キーを押している間とキーを放してしばらくの間は仮数部を表示します。

符号の変更

[CHS] を押すと表示している数値の符号（正数または負数）を逆にします。負数をキー入力するには必要な数字キーを押し終ってから [CHS] を押してください。

指数部のキー入力

[EEX] は数値の指数部分をキー入力するときに使います。まず仮数部をキー入力し，[EEX] を押してから指数部をキー入力します。

指数部が負数のときには指数部のキー入力が終ってから [CHS] を押します*。例えばプランクの定数（6.6262×10^{-34} ジュール・秒）をキー入力して50倍するには次のようにします。

---

* [CHS] は [EEX] を押した直ぐ後に押して，それから指数部をキー入力しても同じ結果になります（仮数部のときには [CHS] を押す前に数値のキー入力が終っている必要があります）。

キー操作	表示		
6.6262	6.6262		
EEX	6.6262	00	00 はこれから指数部のキー入力が可能だという合図のようなものです。
3	6.6262	03	(6.6262×10^{3})。
4	6.6262	34	(6.6262×10^{34})。
CHS	6.6262	−34	(6.6262×10^{-34})。
ENTER	6.6262	−34	数値入力が終った。
50 ×	3.3131	−32	ジュール・秒。

注 EEX を押したときに指数部を表示できるようにするために，仮数部の数字の一部が見えなくなることがありますが，外見上見えなくなるだけで計算機の中にはその数字も残っています。

数値のキー入力違いを防ぐために，仮数部の小数点の左側（整数部分）が8桁以上になったり，仮数部が 0.0000001 よりも小さいときには EEX を押さないでください。このような数値のときには指数部の数値が大きくなる（正数でも負数でも）ようにしてください。例えば 123456789.8×10^{23} だったら 1234567.898×10^{25}，0.00000025×10^{-15} だったら 2.5×10^{-22} のようにキー入力します。

CLEAR キー

クリアとは数値のときにはそれを 0 にし，数値でないときには取り消すことです。HP-15C のクリア操作には次表のものがあります。

キー操作	その働き
g CLx	表示している X レジスタの値を 0 にする。
←	
計算状態のとき	右端の数字 1 字を取り消すか，数値全体を 0 にする。
プログラム入力状態のとき	表示している命令を削除する。
f CLEAR Σ	統計用レジスタを全部 0 にし，表示とメモリー・スタック（第3章で説明）の値も 0 にする。

キー操作	その動き
f CLEAR PRGM 計算状態のとき	プログラム用メモリーの 000 行に合わせる。
プログラム入力状態のとき	プログラム用メモリーの内容全体を削除する。
f CLEAR REG	データ記憶用レジスタ全体を 0 にする。
f CLEAR PREFIX *	キー操作途中のそれまでの前操作キーを取り消す。
*同時に表示している数値の仮数部 10 桁全部を一時的に表示する。	

表示の取り消し（CLx と ←）

HP-15C には表示の取り消し用に二つのキーがあります。CLx と ← です。

計算状態では

- CLx は表示している数値を 0 にする。
- ← は数値のキー入力途中で，ENTER やその他の関数キーなどで入力を打ち切ってなければ，最後の 1 字分だけを取り消します。この後で新しい数字をキー入力したり，あるいはまたその左の数字を取り消すこともできます。キー入力を打ち切った後では ← は CLx と同じ働きです。

キー操作	表　示	
12345	**12,345**	数値のキー入力をまだ打ち切っていない。
←	**1,234**	最後の数字だけが消えた。
9	**12,349**	
√x	**111.1261**	キー入力を打ち切った。
←	**0.0000**	全桁が 0 になった。

プログラム入力状態では

- CLx はプログラム可能で，CLx という命令としてプログラム中に入り，そのときに表示している命令を削除することはありません。
- ← はプログラム不能で，プログラム編集のときに使います。← を押すと，それまで表示していた命令だけを削除します。

計　　算

単項演算関数

単項演算関数とは表示している数値だけを使って計算する関数です。単項演算関数を使うには，必要な数値を表示させてから，目的の関数キーを押します。

キー操作	表　示
45	45
g LOG	1.6532

二項演算関数

二項演算関数はそのキーを押す前に，計算機中に二つの数値が入っていることが必要な関数です。+，−，×や÷は二項演算関数の一つです。

数値入力の打ち切り（区切り） 二つの数値を続けてキー入力するときには，第一の数値のキー入力が終ったときに計算機に第一の数値のキー入力が終ったことを知らせる必要があります。これは第二の数値をキー入力する前に ENTER を押すことで目的が達せられます。逆に言えば二つの数値の内の片方が，前の計算の答などでそれを表示していれば，もう一つの数値をキー入力する前に ENTER を押す必要がないということになります。数値入力用のキー*以外の全部の機能が，数値入力を打ち切る効果があります。

数値入力を打ち切る（例えば ENTER を押す）と，小数点の右側に一定の桁数の入力した以外の数字が付くので見分けられると思います。

長い計算 以下の操作で覚えて頂きたいことは次の通りです。

- ENTER キーは二つの数値を続けて入れるときの区切りとして使うだけです。
- 演算キーは計算に必要な数値を入れ終った後で押します。
- 計算の答は中間結果として別の計算に使うこともあります。このような中間結果は後入れ，先出しの原則で計算機中に記憶し

---

*数字キー，·，CHS，EEX と ← キーです。

たり，使ったりします。ある計算が終った後で数値をキー入力すると，それは別の数値として解釈します。

例　(9+17−4)÷4 を計算してみましょう。

キー操作	表　示	
9 [ENTER]	9.0000	数値入力を区切った。
17 [+]	26.0000	(9+17)。
4 [−]	22.0000	(9+17−4)。
4 [÷]	5.5000	(9+17−4)÷4。

中間結果を自動的に記憶や利用するので，もっと複雑な計算も同じように計算できます。括弧が幾つも重なっているときには，筆算のときと同じように内側の括弧から先に計算します。

例　(6+7)×(9−3) を計算してみましょう。

キー操作	表　示	
6 [ENTER]	6.0000	まず (6+7) の中間結果を求める。
7 [+]	13.0000	
9 [ENTER]	9.0000	次に (9−3) の中間結果を求める。
3 [−]	6.0000	
[×]	78.0000	それから中間結果同士 (13 と 6) を掛けて最後の答を求めた。

次の計算をやってみてください。[ENTER] を押す前の数値や，演算キーを押した後に表示している数値はその次の計算用として計算機中に自動的に記憶しています。

$$(16\times 38)-(13\times 11)=465.0000$$

$$4\times(17-12)\div(10-5)=4.0000$$

$$23^2-(13\times 9)+1/7=412.1429$$

$$\sqrt{[(5.4\times 0.8)\div(12.5-0.7^2)]}=0.5998$$

第 2 章

数 学 用 機 能

この章では HP–15C の数学用機能について説明します（なお統計用機能と高等数学用機能は別の章で説明します）。数学用以外の機能は別に説明します（数値入力は第 1 章，スタック操作は第 3 章，表示指定は第 5 章です）。

HP–15C の数学用機能は手でキーを押すのも，プログラム中で使うのも同じ方法です。この内の幾つか（例えば [ABS]）は初歩的なプログラムでも多く使います。

数学用機能で忘れないでもらいたいことは，数値入力用のキー以外はどのキーも自動的に数値入力を打ち切る（区切る）ことです。つまり数学用機能のキーを押すときの前後には [ENTER] を押す必要がないということです。

円 周 率 π

[g][π] を押すと円周率 π の 10 桁の近似値が計算機の X レジスタに入ります。[π] を別の数値と区切るときに [ENTER] は不要です。

数値変更用の機能

数値変更用の機能はどれも表示している X レジスタの数値に対してだけ働きます。

整数部分 [g][INT] を押すと，表示していた数値の小数部分を切り捨てて，整数部分だけにします。

小数部分 [f][FRAC] を押すと，表示していた数値の整数部分を切り捨てて，小数部分だけにします。

四捨五入 [g][RND] を押すと，X レジスタ内の数値の 10 桁以内の仮数部をそのときの [FIX]，[SCI] または [ENG] の表示指定に応じた桁数になるように四捨五入する（表示している数値と同じにする）。

絶対値 [g][ABS] を押すと，X レジスタの数値を絶対値（正負符号を取り去った値）にします。

キー操作	表　示	
123.4567 [g] [INT]	123.0000	負符号があっても整数部分は同じになる。
[g] [LSTx] [CHS] [g] [INT]	−123.0000	
[g] [LSTx] [f] [FRAC]	−0.4567	
1.23456789 [CHS]		
[g] [RND]	−1.2346	
[f] CLEAR [PREFIX]	1234600000	仮数部の全桁の一時表示。
(放す)	−1.2346	
[g] [ABS]	1.2346	

単項演算関数

HP−15C の数学用単項演算関数も表示している X レジスタの数値だけを使って計算します。

一般用関数

逆数　[1/x] を押すと，X レジスタの逆数 (1 をそれまでの数値で割った数) を計算します。

階乗とガンマ関数　[f] [x!] を押すと，X レジスタの数値 x が $0 \leqq x \leqq 69$ の整数のときはその数の階乗 (1 から x までの整数を全部掛け合わせた数) を計算します。ただし数学上の約束で $0! = 1$ です。

[x!] を使うと高等数学や統計で使っているガンマ関数 $\Gamma(x)$ も計算できます。[f] [x!] と押すと，ガンマ関数 $\Gamma(x+1)$ を計算するので，ある数のガンマ関数を求めたいときにはその数から 1 を引いたものを X レジスタに入れてから [f] [x!] を押してください。なお x が負の整数のときにはガンマ関数の定義がないので計算できません。

平方根　[√x] を押すと X レジスタの数値の平方根を計算します。

二乗　[g] [x²] を押すと X レジスタの数値の二乗 (自乗) を計算します。

キー操作	表　示	
25 [1/x]	0.0400	
8 [f] [x!]	40,320.0000	8 の階乗と $\Gamma(9)$ の答。
3.9 [√x]	1.9748	
12.3 [g] [x²]	151.2900	

三角関数関係

角度の単位 三角関数はそのときの角度の単位で計算します。角度の単位を指定しないと，それまでの角度の単位（度，ラジアンまたはグラード）のままで計算するので気を付けください。

$$1\text{直角} = 90\text{度} = \pi/2\text{ラジアン} = 100\text{グラード}$$

[g] [DEG] と押すと度単位に切り替わります。表示部分には文字が現れません。度未満の数値は必ず10進数として扱うことになっていて，度分秒のような60進数ではありません。

[g] [RAD] と押すとラジアン単位に切り替わります。表示部に **RAD** の文字が見えます。複素数計算状態では，三角関数の角度単位の表示には無関係に，[→P] と [→R] 以外の全機能はどれもラジアン単位の数値として計算しています。

[g] [GRD] と押すとグラード単位に切り替わります。表示部に **GRAD** の文字が見えます。

不揮発性メモリーは最後に指定した角度単位だけを記憶しています。最初に使うとき（工場出荷時または不揮発性メモリーの内容を消したとき）には度単位になっています。

三角関数 X レジスタの x を使って次の計算をします。

キー操作	計　　算
[SIN]	x のサイン
[g] [SIN^{-1}]	x のアーク・サイン
[COS]	x のコサイン
[g] [COS^{-1}]	x のアーク・コサイン
[TAN]	x のタンジェント
[g] [TAN^{-1}]	x のアーク・タンジェント

三角関数の計算の前に希望する角度の単位（度，ラジアンまたはグラード）を指定するようにすると良いでしょう。

三角関数の計算のときに使う角度の数値や，答の角度の数値は必ず10進法の数値で，60進法の数値ではありません。

角度と時間の換算

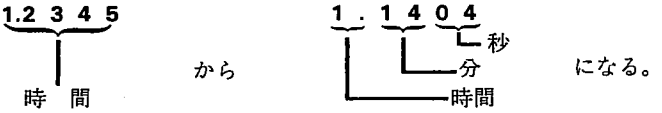
HP-15C は時間（単位は時）と角度（単位は度）の数値を10進法（単

位未満を 10 進の小数で表す方式）と 60 進法（単位未満を分と秒で表す方式）の相互に換算することができます。

時・小数の時 (H. h)	⟷ 時・分・秒・小数の秒 (H. MMSSs)
度・小数の度 (D. d)	⟷ 度・分・秒・小数の秒 (D. MMSSs)

60 進法への換算　[f] [→H.MS] と押すとそれまで X レジスタに入っていた数値を 10 進法の時間や角度と見なしてそれを 60 進法の時間や角度に換算します（上表の右向きの矢印です）。

例えば 1.2345 時間のときに [f] [→H.MS] を押すと



[f] CLEAR [PREFIX] を押すと小数点以下の全桁が見えます。

1 1 4 0 4 2 0 0 0 0

└ 10 万分の 1 秒の単位です。

10 進法への換算　[g] [→H] を押すとそれまで X レジスタに入っていた数値を 60 進法の時間や角度と解釈してそれを 10 進法の時間や角度に換算します（上表の左向きの矢印です）。

度とラジアンの相互換算

[→DEG] と [→RAD] は角度の度とラジアンの相互換算に使います (D. d ↔ R. r)。度は 10 進法の数値で，60 進法の数値ではありません。

キー操作	表　示	
40.5 [f] [→RAD]	0.7069	ラジアン。
[g] [→DEG]	40.5000	40.5 度（小数点以下も 10 進法です）。

対数関数

自然対数 [g] [LN] を押すとそれまでXレジスタに入っていた数値の自然対数（つまり底がeの対数，$\log_e x$ ただしeの近似値として2.718281828を使用）を計算します。

逆自然対数 [e^x] を押すとそれまでXレジスタに入っていた数値の逆自然対数（つまり e^x，e は上と同じ）を計算します。

常用対数 [g] [LOG] を押すとそれまでXレジスタに入っていた数値の常用対数（つまり底が10の対数，$\log_{10} x$）を計算します。

逆常用対数 [10^x] を押すとそれまでXレジスタに入っていた数値の逆常用対数（つまり 10^x）を計算します。

キー操作	表　示	
45 [g] [LN]	**3.8067**	45の自然対数。
3.4012 [e^x]	**30.0001**	3.4012の逆自然対数。
12.4578 [g] [LOG]	**1.0954**	12.4578の常用対数。
3.1354 [10^x]	**1,365.8405**	3.1354の逆常用対数。

双曲線関数

それまでXレジスタに入っていた数値 x を使って次の計算をします。

キー操作	計　算
[f] [HYP] [SIN]	x のハイパーボリック・サイン
[g] [HYP^{-1}] [SIN]	x の逆ハイパーボリック・サイン
[f] [HYP] [COS]	x のハイパーボリック・コサイン
[g] [HYP^{-1}] [COS]	x の逆ハイパーボリック・コサイン
[f] [HYP] [TAN]	x のハイパーボリック・タンジェント
[g] [HYP^{-1}] [TAN]	x の逆ハイパーボリック・タンジェント

二項演算関数

HP-15C の二項演算関数は X と Y レジスタに入っている二つの数値を使って計算します。二つの数値を続けてキー入力するときには，始めの数値をキー入力し終ったときに [ENTER] または機能のどれか（例えば [g] [INT] や [1/x] など）で数値入力を区切るのを忘れないでください。

二項演算関数では始めに入れた（1 番目の）数値が Y レジスタに入っているので y と表記します。後から入れた 2 番目の数値は X レジスタに入ったまま表示しているので x と表記します。

加減乗除の四則演算用の [+], [−], [×] と [÷] は代表的な二項演算関数です。これ以外に次のようなものもあります。

累　乗（指数乗，べき乗）

[y^x] を押すと y の x 乗を計算します。基数の y は指数の x よりも先に入れた数値です。（なお [y^x] は x や y の値が大きいと答が不正確になることがあります。79～80 ページのホーナー法を見てください。）

計　算	キー操作	表　示
$2^{1.4}$	2 [ENTER] 1.4 [y^x]	2.6390
$2^{-1.4}$	2 [ENTER] 1.4 [CHS] [y^x]	0.3789
$(-2)^3$	2 [CHS] [ENTER] 3 [y^x]	−8.0000
$\sqrt[3]{2}$ または $2^{1/3}$	2 [ENTER] 3 [1/x] [y^x]	1.2599

パーセント計算

パーセント計算用として [%] と [Δ%] の 2 種があり，どちらも計算に使う基準数が計算後にも残っています。それで次例のように割増や割引計算などのときに，基準数を改めて入れ直す必要はありません。

パーセント [%] は基準数 y の x パーセントの値を計算します。

例えば15.76ドルの品物に3%の取引税が掛かっているときの税込み価格を計算してみましょう。

キー操作	表 示	
15.76 [ENTER]	15.7600	基準数（価格）を入れる。
3 [g] [%]	0.4728	15.76ドルの3パーセント（税金）を計算。
[+]	16.2328	税込み価格(15.76+0.47)。

増減率 [Δ%] は数値が y から x に変化したときの増減率をパーセントで計算します。x が y よりも大きいときは増加なので答もプラスになり，x が y より小さいときは減少なのでマイナスになります。

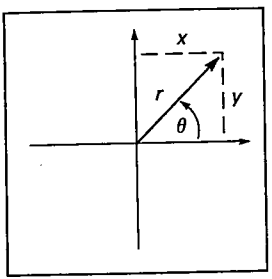
例えば15.76ドルの品物が去年は14.12ドルだったとすると，去年の値段は今年よりも何%安かったを計算してみましょう。

キー操作	表 示	
15.76 [ENTER]	15.7600	今年の値段(これが基準数)。
14.12 [g] [Δ%]	−10.4061	去年は今年よりも10.41%安かった。

極座標系と直交座標系の座標変換

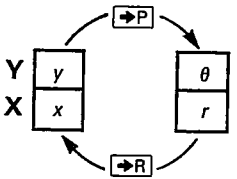
HP−15Cの [→P] と [→R] は極座標系と直交座標系の座標変換用です。

角度 θ はそのときの角度単位で，度（度未満は必ず10進法で，60進法ではありません）やラジアンまたはグラードのどれでも使えます。θ は図のように正の x 軸からの角度です。



極座標系への変換　[g] [→P] を押すと，直交座標系の x と y から極座標系の座標値（半径 r と偏角 θ）を計算します。y を先に入れ，x を後から入れてください。[g] [→P] と押したときに表示するのが r です。[x≷y] を押すと，Y レジスタに入っていた θ を表示します。θ は $-180°$～$180°$，$-\pi$～π ラジアンまたは -200～200 グラードの範囲内になります。

直交座標系への変換　[f] [→R] を押すと，極座標（半径 r と偏角 θ）系から直交座標系の x と y を計算します。θ は r よりも先に入力してください。[f] [→R] と押したときに表示するのが x で，[x≷y] を押したときに表示するのが y です。



キー操作	表　示	
[g] [DEG]		度単位にする（角度単位の文字を表示しない）。
5 [ENTER]	5.0000	y の値。
10	10	x の値。
[g] [→P]	11.1803	r。
[x≷y]	26.5651	θ，直交座標系から極座標系に変換した。
30 [ENTER]	30.0000	θ。
12	12	r。
[f] [→R]	10.3923	x の値。
[x≷y]	6.0000	y の値，極座標系から直交座標系に変換した。

第 3 章

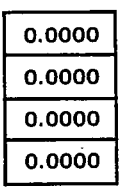
自動メモリー・スタック，ラスト x，数値の記憶

自動メモリー・スタックとスタック操作

HP の計算機はどれも，ポーランド出身の数学者 Jan Łukasiewicz (1878–1956) が考案したポーランド記法を改良した計算方式を採用しています。普通の計算記法は数値や代数の間に演算記号を付けています。ポーランド記法は数値や代数の前に演算記号を付ける方法です。HP では計算機の使用法を効率化するために数値を先に入れてから演算キーを押す方式を採用しました。そこでこれを逆ポーランド法 (Reverse Polish Notation, RPN) と呼んでいます。この考え方は日本語の計算表現に対応しています。

HP–15C も複雑な計算を括弧や等号を使わずに順々に解くために逆ポーランド法を使っています。そのために計算の中間結果を自動的に記憶していて，必要なときに使えるように工夫してあります。そしてキーの操作回数を最小限にするために，自動メモリー・スタック（自動的に数値を記憶するメモリーが 4 段重なっているもの）と [ENTER] キーを利用しています。

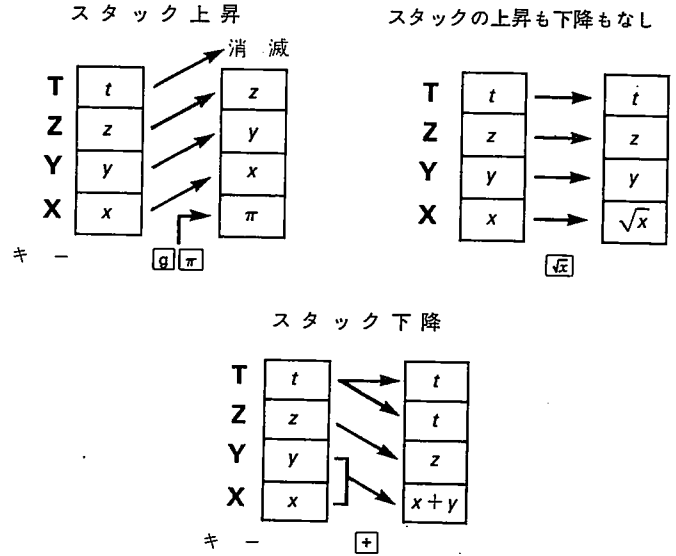
自動メモリー・スタック



いつもこれを表示。

HP–15C が計算状態（**PRGM** の文字を表示していない）ときはいつも X レジスタ内の数値を表示します。

キー入力した数値や計算の答は必ずXレジスタに入るので見られます。それはその寸前の操作やそのまた前の操作によってスタック上昇をしたり，スタック移動をしなかったり，スタック下降した数値です。スタック内の数値は後入れ先出しの順序で記憶しています。スタック内の数値の計算などによる移動は下図の3種です。図中の x，y，z や t はスタック内に入っていた数値です。

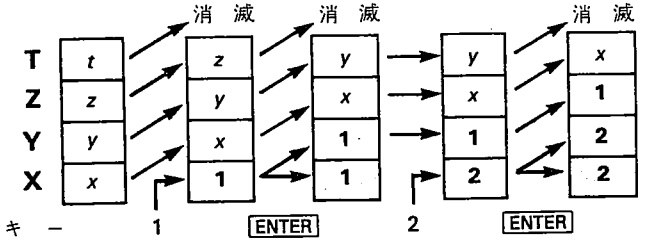


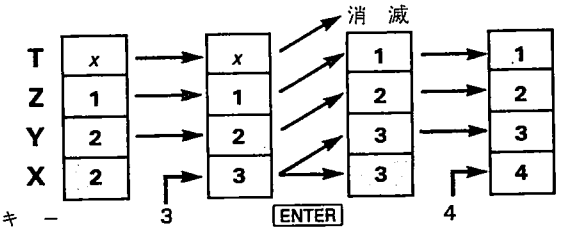
スタック下降のときにTレジスタに入っていた数値はそのまま残っているので，この数値を定数計算に使うことができます。

スタック操作機能

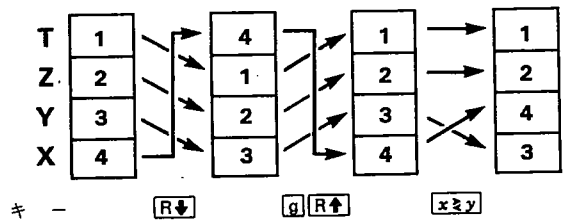
ENTER　これを押すとその前にキー入力した数値とこの後にキー入力する数値とを区切ることが出来ます。押した寸前にXレジスタにあって表示していた数値がスタック上昇してYレジスタに入ります。押した後に数値をキー入力すると，Xレジスタに入っていた数値を書き替えるだけでスタック上昇はありません。次図はスタックに 1，2，

3，4 を入れるとどうなるかを説明したものです。(網点がかかっているのは次に数値をキー入力したり，リコールすると書き替わる数値の意味です。)



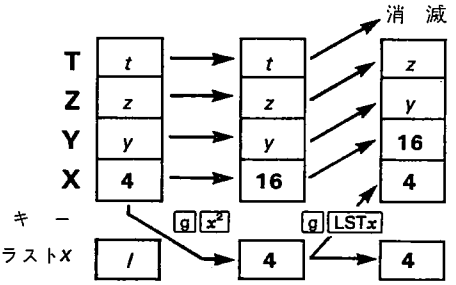


R↓，R↑と x≷y　R↓とR↑はスタック内の数値が一つずつ下または上に回転移動します (X と T レジスタ間で数値が動きます)。このときには数値の消滅はありません。x≷y は X と Y レジスタに入っている数値を交換します。スタック内に下図の左端のように 1，2，3，4 が入っているときに R↓，R↑，x≷y と押すと下図のように数値が移動します。



ラスト x レジスタと [LSTx]

ラスト x レジスタはスタックとは別で，その前の数字機能などを実行する直前に表示していたX レジスタの数値を記憶しておくレジスタです。* [g] [LSTx] を押すとラスト x レジスタに入っている数値をX レジスタにコピーして表示します。例えば次の通りです。



この [LSTx] を活用すると，前の計算に使った数値を定数計算（39 ページ）のためにもう一度入れ直したりする必要が無くなります。それだけでなく，計算途中で数値のキー入力を間違えたとか，違うキーを押してしまったなどの計算ミスも訂正できます。

一例として割算の数値（除数）を間違えたときの訂正の方法を見てください。長い計算途中には特に便利です。

キー操作	表　示	
287 [ENTER]	287.0000	
12.9 [÷]	22.2481	あっ数字を間違えてしまった。
[g] [LSTx]	12.9000	[÷] を押す前にX レジスタに入っていた数値をラスト x レジスタから呼び出す。

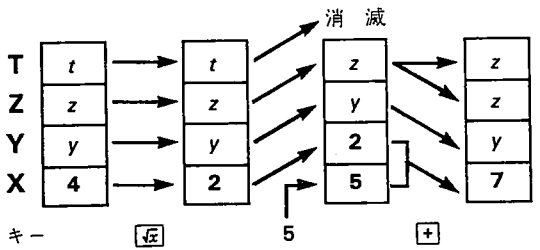
*[Σ+]，[s] それに [LR.] のときだけは統計用レジスタ（R_2－R_7）の数値だけを使って計算するために，キーを押す直前のX レジスタの数値をラスト x レジスタに記憶しません。付録B にラスト x レジスタに記憶する操作全部の表があります。

キー操作	表　示	
[×]	**287.0000**	割算前の正しい数値に戻った。
13.9 [÷]	**20.6475**	正しい答。

関数計算とスタック

数値を二つ入れるときには，一つ入れ終ったら [ENTER] を押してからもう一つの数値を入れました。しかし計算後（それと [R↓] のようなスタック操作後）に別の数値を入れるときには [ENTER] を押す必要はありません。それは HP-15C の大部分の関数や機能は次のような効果があるからです。

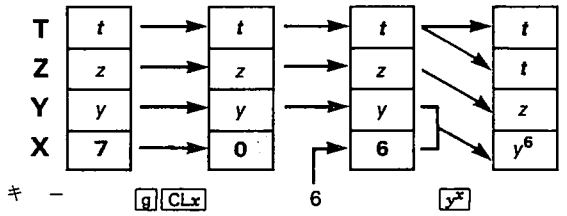
- 自動メモリー・スタックを上昇可能にする。つまりその次に数値をキー入力したり，呼び出したりすると，自動的にスタックが上昇する。
- それまでの数値入力を打ち切るので，それ以降のキー入力は別の数値として解釈する。



[ENTER]，[CLx]，[Σ+] と [Σ-] の4種だけはスタック上昇可能ではありません。* この操作の後で新しい数値をキー入力したり，呼び出したりしてもスタック内で上昇しません。この4種のどれも操作後に新しい数値を入れても表示内容が変わるだけで Y～T 内の数値は移動しません。（なお [ENTER] はその操作をしたそのときにスタック上昇があ

＊数値入力を区切った後の [←] は [CLx] とまったく同じ働きをするので，やはりスタック上昇可能ではありません。数値入力途中ではその前の数字1字分を取り消すだけですから，スタック上昇可能には無関係です。詳しくは付録 B を見てください。

りますが，その後で別の数値を入れてもスタック上昇はしません。この [ENTER] 操作の様子は 34 ページの図の通りです。）普通の場合はこのようなスタック移動に頭を使わなくとも計算が簡単にできます。

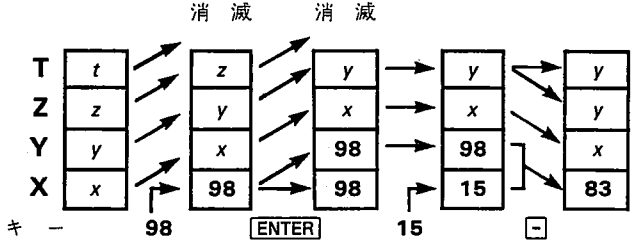


数値の入力順序と [ENTER] キー

二項演算関数で一番大切なことはスタック内の数値の位置です。加減乗除の計算をするときのスタック内の数値は筆算のときの数値の位置と同じです。

98	98	98	98
−15	+15	×15	15

このように前の数（被演算数）が Y レジスタに入り，後の数（演算数）が X レジスタに入っている必要があります。計算を実行するとスタック下降が起り，答が X レジスタに入ります。ここでは引き算をしたときの状態を図に書きました。



98 に 15 を足したり，98 に 15 を掛けたり，98 を 15 で割るときもスタック内に同じように 98 と 15 が入っている必要があります。

複雑な計算

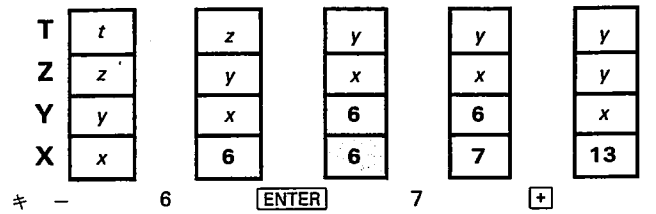
スタックの自動上昇とスタック下降を利用することによって，括弧や中間結果の記憶操作を利用しないで複雑な計算でも計算可能です。複雑な計算といっても一つ一つに分解すれば単項演算や二項演算の組み合わせです。

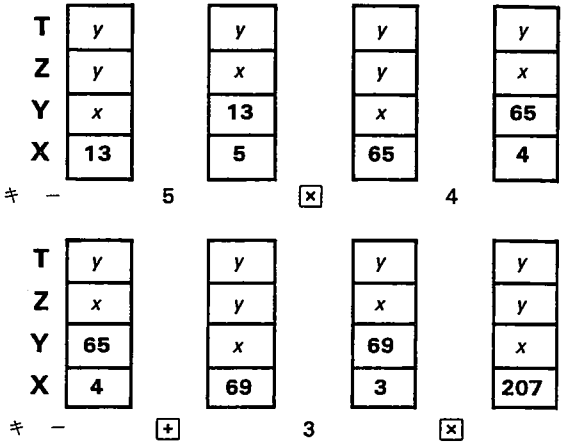
スタックの四つのレジスタを使えばかなり複雑な計算でも可能です。一番上手な方法は筆算のときと同じように，一番内側の括弧内から計算して順に外へ進む方法です。こうしないと中間結果を一時的に記憶する操作が必要になります。次のような計算で検討してみましょう。

$$3[4+5(6+7)]$$

キー操作	表　示	
6 [ENTER] 7 [+]	**13.0000**	一番内側の括弧内の答。
5 [×]	**65.0000**	5(6+7) の答。
4 [+]	**69.0000**	[4+5(6+7)] の答。
3 [×]	**207.0000**	3[4+5(6+7)] の答。

下の説明図はこの計算のときにスタック内がどうなったかを説明する図です。二つの数値を使って計算するとスタック下降し，その後で別の数値をキー入力するとスタックが上昇します。(この図以降はスタック内の移動を説明する矢印を省略しました。)





定数計算

一定数を加減乗除する定数計算は目的に応じて次の3種類の方法があります。(この外に数値記憶用レジスタを使う方法もあります。)

1. ラスト x レジスタを使う。
2. スタック全体に定数を入れておいてから計算する。(計算したら一たんXレジスタを0にしてから，別の数をキー入力して計算する。)
3. 累計や，一定数の掛け算の繰り返しのときもスタック全体に定数を入れておいてから計算する。(こちらでは計算ごとにXレジスタを0にしません。)

ラスト x　計算するときにXレジスタに入っていた数（後から入れた数）はラスト x レジスタにコピーして残っています。[g] [LSTx] を押すと定数がXレジスタに入ってそれを表示します。この方法を繰り返します。

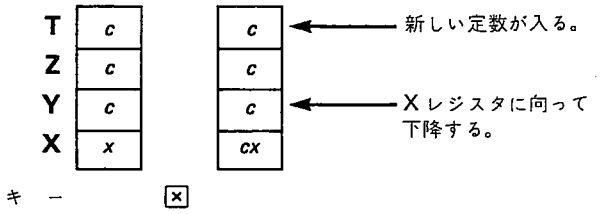
例 地球に近い恒星で明るいのはケンタウルス座のリジル（α 星，4.3 光年）とおおいぬ座のシリウス（8.7 光年）です。1 光年は約 9.5×10^{15}m です。二つの星までの距離を計算してください。（図を簡単にするために小数点以下 1 桁に四捨五入した値を記入しました。）

T	t		z		y		y
Z	z		y		x		x
Y	y		x		4.3		4.3
X	x		4.3		4.3		9.5 15
キー		4.3		ENTER		9.5 EEX	15
ラスト x	/		/		/		/

T	y		y		y		x
Z	x		y		x		4.1 16
Y	4.3		x		4.1 16		8.7
X	9.5 15		4.1 16		8.7		9.5 15
キー		×		8.7		g	LSTx
ラスト x	/		9.5 15		9.5 15		9.5 15

T	x		x	
Z	4.1 16		x	
Y	8.7		4.1 16	← リジルは 4.1×10^{16}m 離れている。
X	9.5 15		8.3 16	← シリウスは 8.3×10^{16}m 離れている。
キー		×		
ラスト x	9.5 15		9.5 15	

スタック全体に定数を入れて計算する方法　Tレジスタに入れた数値は計算でスタック下降するたびにそれまでの値のコピーがTレジスタに入る（必要なだけ増える）ので，定数計算に使うことができます。

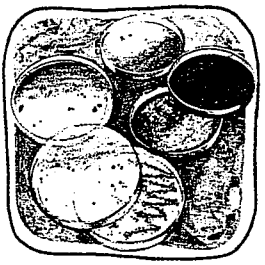


キー入力して表示している定数は ENTER を3回押すとスタック全体に入ります。それから最初の計算に使う数値をキー入力します。計算でスタック下降するごとに定数がYレジスタまで下がり，Tレジスタには定数のコピーが入ります。

前ページの例のように計算したい数値が変わるときには，新しい数値をキー入力する前にXレジスタの表示をゼロにする必要があります。こうするとスタック上昇が起きないで，Xレジスタの数値が変わるだけなので，Y～Tの定数はいつでも使えます。

定数計算で異なる数値をキー入力するのではなく，計算した答を次の計算に使うのなら，表示している答を消さずに演算キーを繰り返して押すだけです。

例　ある細菌学者が1日に15％ずつ増加する（増加比が1.15）細菌の培養試験を計画しています。1000個からスタートしたときの4日後までの細菌数を計算してください。



キー操作	表　示	
1.15	1.15	増加比。
ENTER ENTER ENTER	1.1500	スタック全体に入れる。
1000	1,000	スタートのときの細菌数。

キー操作	表　示	
[×]	1,150.0000	1日後の細菌数。
[×]	1,322.5000	2日後。
[×]	1,520.8750	3日後。
[×]	1,749.0063	4日後。

記憶レジスタの操作

数値の記憶（記録）と呼び出しは，表示するXレジスタと指定した記憶レジスタ間でコピーします。初めてスイッチを入れたとき（買ったばかりや，不揮発性メモリー内容を消したとき）には，HP-15Cでは直接操作できる記憶レジスタは，R_0～R_9，$R_{.0}$～$R_{.9}$とIレジスタの21個です。この内R_2～R_7の6個は統計計算にも使います。

数値記憶用レジスタの個数は増やしたり，減らしたりすることが可能です。付録Cのメモリー配分で説明するように [DIM] を使うとレジスタ個数を変えられます。番号の小さいレジスタの方が後までも残っているので，レジスタを使うときには番号の一番小さいレジスタから使うようにするのが一番利口な方法です。

数値の記憶と呼び出し

[STO]　このキーの後にレジスタ番号（0～9または.0～.9）の数字キーを押すと，Xレジスタに入って表示している数値をその番号のレジスタにコピーします。するとそれまでそのレジスタに入っていた数値は消えます。

[RCL]　このキーの後にレジスタ番号の数字キーを押すと，そのレジスタに入っていた数値をXレジスタにコピーして表示します。スタック上昇が可能でない状態のとき以外はスタック上昇が発生します。レジスタに入っていた数値には変化がありません。

[x≷]　この後に 0～.9 の数字キーを押すと，* Xレジスタと指定

*Iレジスタと [I] または [(i)] を利用するとどの記憶レジスタでもその内容を見ることが出来ます。これについては第10章のIレジスタと，第12章の行列演算で説明します。

したレジスタ内の数値が入れ替わります。これはY～Tレジスタ内の数値には影響を与えずに記憶レジスタ内の数値を見るのに便利です。

以上3種はスタック上昇が可能な操作なので，Xレジスタに入って表示している数値は別の数値を入れても消えないので次の計算に使えます。上記の操作で実際には存在しないレジスタを指定すると **Error 3** を表示します。

例　春になってクロッカスの花が咲き出しました。昨年花壇に植えた24個の球根から花が幾つ咲いたか調べてみたいと思います。第1日目の花の数を計算機中に記憶させておいて，翌日に新しく咲いた数を足してみましょう。

キー操作	表　示	
3 STO 0	**3.0000**	初日に咲いた数を R_0 に記憶。

一たん計算機のスイッチを切ります。翌日にスイッチを入れてから

RCL 0	**3.0000**	昨日咲いた花の数を呼び出す。
5 +	**8.0000**	今日咲いた数を足すと花の数の合計になる。

記憶レジスタ内容の消去

f CLEAR REG と押すと全記憶レジスタの内容が消えて0になります。(しかしスタック内とラスト x レジスタ内の数値には影響がありません。)一つの記憶レジスタだけの内容を消すには，そのレジスタに0を記憶します。不揮発性メモリーの内容を消すと，全記憶レジスタとスタックやラスト x レジスタ内の数値も消えてしまいます。

レジスタとの直接四則演算

直接演算記憶　表示している数値それ自体は記憶しておく必要はないけれでも，その表示値とレジスタ内容との演算結果だけを記憶しておきたいと考えたことはありませんでしたか。HP-15Cなら，次のようにすると RCL を使わずに操作できます。

1. 2番目の演算数（1番目はレジスタ中）を表示（別の計算の中間結果や，呼び出したり，キー入力などで）します。
2. STO を押す。

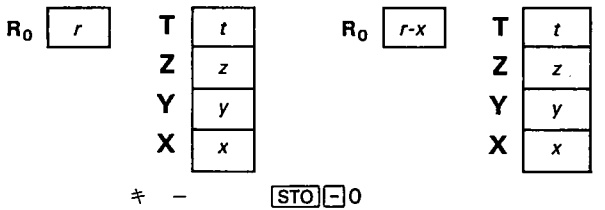
3．計算目的に応じて [+]，[−]，[×]，[÷] のどれかを押す。

4．レジスタ番号 (0～9，.0～.9) のキーを押す。(Iレジスタでも同じように出来ますが，第10章で説明します。)

レジスタ中の新しい数値は次のようになります。

直接演算記憶のとき

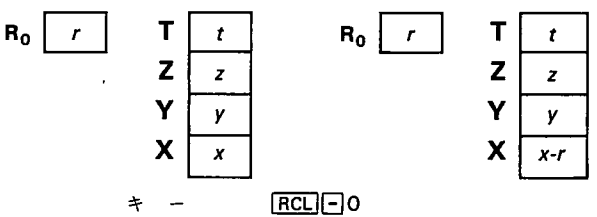
新しいレジスタ内容 ＝ 演算前のレジスタ内容 $\left\{\begin{matrix}+\\-\\\times\\\div\end{matrix}\right\}$ 演算前の表示値



直接呼び出し演算　この演算は表示している数値とレジスタ内の数値を使って，スタック上昇なしで計算できます。つまり Y～T レジスタ内の数値が消えることはありません。キー操作の順序は直接演算記憶のときの [STO] が [RCL] に変わる以外は同じです。

直接呼び出し演算のとき

新しい表示値 ＝ 演算前の表示値 $\left\{\begin{matrix}+\\-\\\times\\\div\end{matrix}\right\}$ 演算前のレジスタ内容



例　前例のクロッカスの開花数をもう2日続けて調べてみましょう。

キー操作	表　示	
8 STO 0	8.0000	2日目までの開花数合計を R_0 に入れる。
4 STO + 0	4.0000	3日目に別の四つが咲いた。
3 STO + 0	3.0000	4日目にまた三つ咲いた。
24 RCL − 0	9.0000	植えた球根数（24）から R_0 に入っている咲いた数の合計（15）を引いたら，まだ咲いてないのが9個ある。
RCL 0	15.0000	R_0 に入っている数は変わっていない。

桁あふれと下位桁あふれ

直接演算記憶や直接呼び出し演算をすると，記憶レジスタが桁あふれ（オーバーフロー，数値が入りきれない）することがあります。そのときには表示が点滅し，レジスタ内は $\pm 9.999999999 \times 10^{99}$ に書き替わります。点滅（エラー状態）を止めるには ←，ON または g CF 9 を押します。

下位桁あふれ（アンダーフロー，計算機で扱える最小値より小さくなる）のときにはレジスタ内容が0になります（表示は点滅しません）。桁あふれと下位桁あふれは61ページで説明します。

練習問題

1．次の式の x を計算しなさい。

$$x = \sqrt{\frac{8.33\,(4-5.2) \div [(8.33-7.46)\,0.32]}{4.3\,(3.15-2.75)-(1.71)\,(2.01)}}$$

答　4.5782

キー操作の一例

4 ENTER 5.2 − 8.33 × g LSTx 7.46 − 0.32 × ÷ 3.15 ENTER 2.75 − 4.3 × 1.71 ENTER 2.01 × − ÷ √x

2. 定数計算を利用して，月利1%（0.01）で1000ドル借りて毎月100ドルずつ6か月返したときの借入残高を計算してみましょう。（FIX 2で計算。）

方法 スタック中に（1+i）を入れ，次に借入総額をキー入力します。（iは利率です。）毎回返済後の借入残高は次式で計算します。

借入残高＝（（前回の借入残高）×（1+i））－返済額

最初のキー操作は次の通りです。

1.01 ENTER ENTER ENTER 1000

毎回の返済ごとに次の操作を繰り返します。

× 100 −

6か月返済後の借入残高は446.32ドル。

3. R_5 に100を入れてから，次の操作を番号順にしなさい。

1. R_5 の内容を25で割る。
2. R_5 の内容から2引く。
3. R_5 の内容に0.75を掛ける。
4. R_5 の内容に1.75を足す。
5. R_5 の内容を呼び出す。

答 3.2500。

第4章

統 計 関 数

ここで説明する統計関数は第3章のスタック操作を十分に理解してから使ってください。統計計算でキー入力の順序がどんなに大切か分かると思います。

確率計算

順列と組み合わせで入力する数値は負でない整数に限られます。yを先に，xを後から入れます。この両関数は二項演算関数のときのようにスタック下降して，答がXレジスタに入ります。

順列 f Py,x と押すとy種のものを一時にx個だけ並べるときの並べ方の種類の数を計算します。何回かに分けて並べたり，同種のものがあるときにはこのままでは計算できません。計算式は次の通りです。

$$ {}_yP_x = P_{y,x} = \frac{y!}{(y-x)!} $$

組み合わせ g Cy,x と押すとy種のものから一時にx個だけ取り出すときの組み合わせの種類の数を計算します。同種のものがあったり，取り出す順序を区別するようなときにはこのままでは計算できません。計算式は次の通りです。

$$ {}_yC_x = C_{y,x} = \frac{y!}{x!(y-x)!} $$

例　5枚の絵の内から3枚選んで壁に掛けたいと思います。並べ方が全部で何通りあるでしょうか。

キー操作	表　示	
5 ENTER 3	3	5枚(y)の絵から3枚(x)を取り出す。
f Py,x	60.0000	60通りの並べ方がある。

トランプのジョーカーを除いた52枚から4枚を取り出すとどんな組み合わせ方があるでしょうか。

キー操作	表　示	
52 [ENTER] 4	**4**	52枚(y)のカードから4枚(x)を取り出す。
[g] [Cy,x]	**270,725.0000**	手にする可能性がある組み合わせの総数。

この関数は入力するxやyが大きくなるほど計算に時間がかかります。この計算中には running の文字が点滅します。xやyの最大数は9,999,999,999です。

乱　　数

[f] [RAN#] を押すと$0 \leq r < 1$の範囲の乱数を1個（一様分布の擬似乱数列中の1個）を表示します*。

始めてスイッチを入れたとき（不揮発性メモリーの内容を消したときを含む）には，HP-15Cは初回の乱数の種（核とも言います）として0を使います。しかし一たん乱数を作ると，それが次の乱数の種になります。全然違った乱数列を作りたいときには乱数の種として異なる種を使ってください。（同じ乱数の種を使うと同じ乱数列になります。）

[STO] [f] [RAN#] はXレジスタ中の数値（$0 \leq r < 1$）を乱数の種として記憶するときに使います。（rがこの範囲の数値でないときには自動的にこの範囲内になるように内部で変換します。）

[RCL] [f] [RAN#] はそのときの乱数の種を呼び出したいときに使います。

---

*この乱数は D.Knuth の Seminumerical Algorithms（邦訳はサイエンス社の渋谷正昭訳　準数値算法／乱数）のスペクトル検定に合格しています。

キー操作	表　示	
.5764	0.5764	乱数の種として0.5764を記憶する。(このとき f キーを押すのを省略できる。)
STO f RAN#	0.5764	
f RAN#	0.3422	上記の種のときの乱数列。
f RAN#	0.2809	
←	0.0000	
RCL f RAN#	0.2809	最後の乱数 (これが次回の乱数の種) を呼び出す。(このとき f キーを押すのを省略できる。)

統計用データの集計

HP-15Cは1変数と2変数の統計計算が出来ます。データをまずYとXレジスタに入れます。それから Σ+ 機能を使うとYとX中のデータを自動的に計算し，記憶レジスタのR_2～R_7に集計します。それでこのレジスタを統計用レジスタとも呼びます。

新しい統計データの集計を始める前に，まず f CLEAR Σ を押して統計用レジスタとスタックをクリアします。(もし統計用レジスタの一部をプログラム用メモリーに配分してあるときに，CLEAR Σ，Σ+ または Σ- を使おうとすると **Error 3** の表示になります。詳しくは付録Cのメモリー配分を見てください。)

1変数の統計データのときは，各データ (x値) をXレジスタにキー入力して Σ+ を押すのを繰り返します。

2変数の統計データのときは，各データ (x値とy値) を次のように入れます。

1. yを先にキー入力して表示する。
2. ENTER を押す。(表示していたyをYレジスタにコピーする。)
3. xをキー入力して表示する。
4. Σ+ を押す。キーを押した後に表示する数値はデータの番号のn (つまり Σ+ を押した回数) です。キーを押す前のxはラストxレジスタに入り，yはYレジスタに残ります。Σ+ はスタック上昇が可能でない操作なので，次の統計データをキー入力してもスタック上昇はありません。

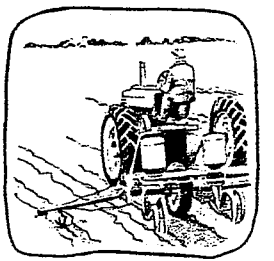
x や y の集計のときに x 同士や y 同士の差が x や y に比べて小さいときに，標準偏差や相関係数，直線の当てはめや y の推定（いずれもこの後で説明）を計算しようとすると **Error2** を表示することがあります。このようなことはめったにありませんがもしそうなったら元の個々の統計データからおよその平均を引いたものをデータとしてキー入力してください。求められた平均や y の推定，y 切片の答にキー入力時に差し引いておいたおよその平均を足すと正しい答になります。例えば x が 665999，666000，666001 のときにはデータとして −1，0，1 をキー入力し，後で平均などの答に 666000 を足します。

統計データは次のようにレジスタに集計しています。

レジスタ	内容	
R_2	n	集計したデータ数（Σ+ を押した回数，この n はキーを押した後に表示するのと同じ。
R_3	Σx	x の累計。
R_4	Σx^2	x の二乗の累計。
R_5	Σy	y の累計。
R_6	Σy^2	y の二乗の累計。
R_7	Σxy	x と y の積の累計。

この集計したデータを見るには RCL を押してから目的のレジスタの番号を押します。RCL Σ+ と押すと，R_3 と R_5 に入っている Σx と Σy がそれぞれ X と Y レジスタに入ります。（この RCL Σ+ 操作はスタック上昇が可能なときには2回，スタック上昇が可能でないときには1回のスタック上昇があります。）

例　ある篤農家が高収穫の稲を見付けたので，肥料と収穫量を調べたら次表の通りでした。Σ+ を使って窒素肥料の施肥量（x）と稲の収穫量（y）を集計し，Σx，Σx^2，Σy，Σy^2，Σxy を求めてください。



X	窒素肥料施肥量 (kg／ha), x	0.00	20.00	40.00	60.00	80.00
Y	もみの収穫量 (トン／ha), y	4.63	4.78	6.61	7.21	7.78
ha はヘクタール（＝100アール＝10,000 m²）の略号で約3,025坪に相当します。						

キー操作	表　示	
f CLEAR Σ	0.0000	統計用レジスタ(R_2～R_7とスタック）をクリア。
f FIX 2	0.00	データのように小数点以下2桁表示にする。
4.63 ENTER	4.63	
0 Σ+	1.00	1番目のデータ。
4.78 ENTER	4.78	
20 Σ+	2.00	2番目のデータ。
6.61 ENTER	6.61	
40 Σ+	3.00	3番目のデータ。
7.21 ENTER	7.21	
60 Σ+	4.00	4番目のデータ。
7.78 ENTER	7.78	
80 Σ+	5.00	5番目のデータ。
RCL 3	200.00	xの合計，$\sum x$（肥料の合計kg）。
RCL 4	12,000.00	xの二乗の合計，$\sum x^2$。
RCL 5	31.01	yの合計，$\sum y$（もみの合計）。
RCL 6	200.49	yの二乗の合計，$\sum y^2$。
RCL 7	1,415.00	xとyの積の合計，$\sum xy$。

集計したデータの訂正

キー入力したデータがもし間違えていたのに気が付いたら，それは簡単に訂正できます。xとyの内の片方だけ間違えたときでも，その両方をキー入力し直して削除します。

1. 間違えたデータをYとXレジスタにキー入力する。
2. [g][Σ-]を押して間違えたデータ分を削除する。
3. xとyの正しい値をキー入力する。
4. [Σ+]を押す。

もし間違えたデータが直前に[Σ+]を押したときのであれば，[g][LSTx][g][Σ-]と押すだけで間違えたデータ分を削除できるので，*上記の3～4の操作をしてください。

例　前例のデータを入れ終った後で，ノートのデータを読み間違えたことに気が付きました。2番目のyが4.78でなくて5.78だったのです。正しいデータに直しましょう。

キー操作	表　示	
4.78[ENTER]	**4.78**	間違えたデータ分を削除するために，間違えたデータをもう一度キー入力する。[Σ-]を押すとnが1減つて4になる。
20[g][Σ-]	**4.00**	
5.78[ENTER]	**5.78**	正しいデータをキー入力し直す。
20[Σ+]	**5.00**	nの値が元に戻る。

この集計したデータはこの後の操作例で使います。

---

*上記の方法で集計したデータの削除をしても，誤データ集計前と誤データを集計して削除した後での統計レジスタ中の数値が一致しないことがあります。これは集計するデータの有効数字が6桁以上だと内部演算途中で丸め誤差が発生するからです。そのようなときには始めからやり直してください。

平　　均

[x̄] は付録Aに記載した計算式の通りに統計レジスタ中の集計したデータを使って，x と y の平均（算術平均，相加平均）を計算します。[g][x̄] を押すとスタック上昇します(スタック上昇が可能なときには二つ，可能でないときは一つです。) x の平均 $\bar{x}$ はXレジスタに，y の平均 $\bar{y}$ はYレジスタに同時に入ります。[x≷y] を押すと $\bar{y}$ が見られます。

例　前例のデータ訂正をしてあれば，窒素肥料の施肥量の平均 $\bar{x}$ と，収穫量の平均 $\bar{y}$ を求めてみましょう。

キー操作	表　示	
[g][x̄]	40.00	$\bar{x}$, 全体の窒素肥料の施肥量の平均。
[x≷y]	6.40	$\bar{y}$, 全体の米の収穫量の平均。

標準偏差

[g][s] を押すと集計したデータの標準偏差を求められます。付録Aに記載した計算式通りに統計レジスタ中のデータを使って x の標準偏差 s_x と，y の標準偏差 s_y を計算します。

この標準偏差は集計したサンプルのデータ（抜き取りデータ）からデータ全体（集計しなかった分を含めて）のばらつき具合を知るために使うので，正しくはサンプルの標準偏差と呼びます。* [g][s] を押すとスタック上昇します（スタック上昇が可能なときには二つ，可能でないときは一つです)。s_x はXレジスタに，s_y はYレジスタに同時に入ります。[x≷y] を押すと s_y が見られます。

---

*データが全体からサンプリングしたデータでなくて，全体のデータそのものときには母集団の標準偏差（記号は σ）が必要になります。これは [s] で求めたサンプルの標準偏差に $\sqrt{(n-1)/n}$ を掛けたものです。n が大きければ両者の違いは無視できます。次の簡便法でも母集団の標準偏差を求められます。[g][x̄][Σ+] と押してから [g][s] と押すと求める値になります。（ただしこの後で別の統計計算をするときには，追加した平均値を削除する必要があります。）

例 前例の標準偏差を求めましょう。

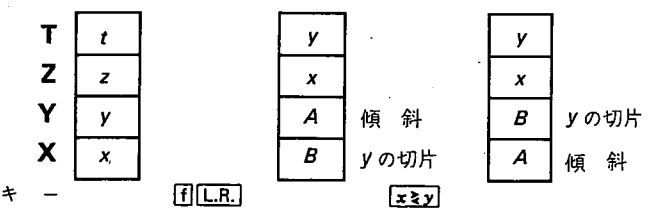
キー操作	表 示	
[g] [s]	31.62	s_x, 窒素肥料の施肥量の標準偏差。
[x≷y]	1.24	s_y, 米の収穫量の標準偏差。

直線の当てはめ

直線の当てはめ（一次回帰，直線回帰）とは2組以上のデータ（x と y）の関係を知るために，データにうまく当てはまる1本の直線の係数を求めることです。[f] [L.R.] を押すと下式の直線の傾斜 A と，y 軸の切片 B を計算します。

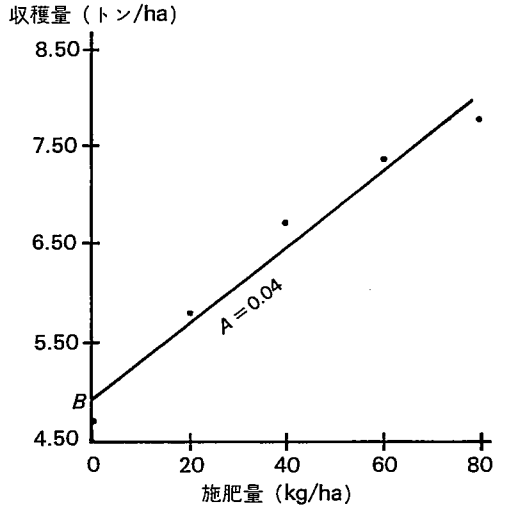
$$y = Ax + B$$

1. [Σ+] を使って統計データを集計する。
2. [f] [L.R.] を押す。傾斜 A がYレジスタに，y 軸の切片 A がXレジスタ（表示）に同時に入ります。
3. [x≷y] を押すと傾斜 A が見られます。（[x̄] や [s] のときと同じように，[L.R.] もスタック上昇可能のときは二つ，可能でないときは一つスタック内容が上昇します。）



この直線の傾斜と y 軸の切片は付録Aに記載の計算式のように統計レジスタ中のデータを使って最小二乗法で求めています。

例　前例の施肥量と収穫量に当てはまる直線を求め，下図のデータと比較してください。



キー操作	表　示	
f L.R.	**4.86**	直線の y 軸の切片。
x≷y	**0.04**	直線の傾斜。

直線推定と相関係数

f ŷ,r を押すと y の直線推定値 $\hat{y}$ が X レジスタに，相関係数 r が Y レジスタに同時に入ります。r を見るには x≷y を押します。

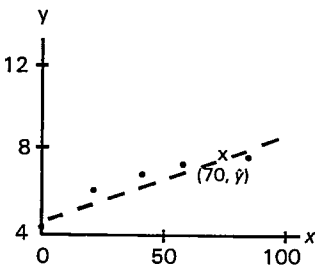
直線推定　f ŷ,r を押す前にキー入力した x と，統計レジスタ中のデータを使って付録 A に記載の計算式の通りに，x のときの y の推定値 $\hat{y}$ を求めます。

相関係数　直線の当てはめと直線推定は x と y の間に或る程度の直線関係があるものとして計算しています。相関係数 r はその x と y のデータがどの程度まで直線に当てはまっているかを示すものです。r の範囲は $-1 \leqq r \leqq 1$ で，-1 は完全に負の相関があり $+1$ は完全に正

の相関で，どちらも集計に使った x と y の全部のデータが求めた直線上に乗っている状態です。0 に近いほど x と y の間は無関係です。

[f] [ŷ,r] を押す前に x をキー入力しなかった（X レジスタ中に以前から入っていた数値を使った）ときの答は $\hat{y}$ としては意味のない数値なので気を付けてください。

例　前例で窒素肥料を 1 ヘクタール当たり 70 kg 使うと米が何トン収穫出来るでしょうか。集計したデータを使って予想してみましょう。この計算と同時に相関係数も一緒に求めてあるので，推定値を見てから集計に使ったデータに直線がどの程度当てはまっているかも見るために [x≷y] を押すことにします。



キー操作	表　示	
70 [f] [ŷ,r]	7.56	予想収穫量（トン/ha）。
[x≷y]	0.99	集計に使ったデータは直線にかなり近い。

その他の応用

補間　熱力学や統計数値表のような数表で，表中に載っていない部分を求めるために補間を使いますが，HP-15C で [ŷ,r] を使うと簡単に直線補間が求められます。直間補間も直線推定と同じように，数表の求める部分の両側の 2 点を結ぶ直線上に求める点があると仮定しているからです。

ベクトル計算　統計データの集計法は二次元ベクトルの足し算と引き算にも使えます。極座標系のベクトルのときは直交座標系に変換してから集計します（θ, [ENTER], r, [→R], [Σ+]）。ベクトル計算の答は $R_3(\sum x)$ と $R_5(\sum y)$ に入っているので [RCL] [Σ+] で見られ，必要があればそれから極座標系に戻すと良いでしょう。

ベクトル計算の第2項以降をキー入力するたびに，足し算なら[Σ+]，引き算なら[g] [Σ-]を押してください。

第5章

表示と不揮発性メモリー

表示の指定

HP-15C は3種の表示法 ([FIX], [SCI], [ENG]) があって,その後に 0~9 の数を指定します。下図は3方法で4と指定したときに, 123,456 をどう表示するか説明するためのものです。

[f][FIX] 4 : **123,456.0000**
[f][SCI] 4 : **1.2346 05**
[f][ENG] 4 : **123.46 03**

不揮発性メモリーを使っているので, 指定した表示形式は次に指定を変えるか不揮発性メモリーの内容を消すまで変わりません。

また表示形式は数値入力を区切った後で有効なので, 数値のキー入力途中は全部の数字 (10桁まで) が見られます。

小数点以下の桁数固定表示

[FIX] は小数点以下の表示桁数を指定できます(0~9で, 実際には整数部分の桁数によって制限があります)。数値が非常に小さくなったり, 大きくなると次に説明する指数部付き表示になりますが, 小数点以下の桁数固定で表示出来るようになると自動的に元の表示指定に戻ります。指定省略状態では HP-15C は [FIX] 4 になっています。キー操作の順序は [f][FIX] n です。

キー操作	表　示	
123.4567895	**123.4567895**	
[f][FIX] 4	**123.4568**	
[f][FIX] 6	**123.456790**	小数点以下7桁目で四捨五入して表示。(計算機内部では全10桁を記憶している。)
[f][FIX] 4	**123.4568**	普通の [FIX] 4 に戻す (小数点以下5桁目で四捨五入して表示。)

指数部付き表示

[SCI] 形式は科学的表示とも呼び, どんな数値も指数部付きで表示し

ます。キー操作は [f] [SCI] n で，小数点以下の桁数を指定しますが，実際には指数表示部分で3桁分使うため6桁目までしか表示出来ませんが，表示末尾の次の桁で四捨五入して表示します。7以上を指定する ([SCI] 7, 8, 9) と表示出来ない7～9桁目の次の桁で四捨五入をして表示します。*

前の数値を表示していたら表示指定を変えてみましょう。

キー操作	表　示		
[f] [SCI] 6	1.234568	02	小数点以下7桁目で四捨五入して表示。
[f] [SCI] 8	1.234567	02	小数点以下9桁目で四捨五入したけれども，表示は6桁目まで。

工学用指数部付き表示

[ENG] は技術関係の表示に向いている表示形式で，下記以外は [SCI] と同じです。

- 工学用指数部付き表示は有効数字の1桁目を必ず表示します。[f] [ENG] の後に指定する数は2桁目以降に表示したい有効数字の桁数です。
- 工学用指数部付き表示では指数部が3の倍数になります。(技術関係では k, M, G, m, μ, n のように3桁ごとに単位の接頭語が変わります。)

キー操作	表　示		
.012345	0.012345		
[f] [ENG] 1	12.	−03	頭から3桁目で四捨五入して表示。
[f] [ENG] 3	12.35	−03	指数部が3の倍数になるように小数点が移動。
10 [×]	123.5	−03	
[f] [FIX] 4	0.1235		普通の [FIX] 4表示。

---

* [SCI] 7, 8, 9のときには四捨五入する桁の左の桁が9のときだけそのまた左の桁が1多くなる以外は見掛け上の表示の変化がありません。

有効数字の表示

HP–15C は表示指定に関係なく内部では有効数字 10 桁と，10 の指数 2 桁を記憶しています。例えば π は表示指定がどうなっていても内部では 3.141592654×10^{00} です。

X レジスタ中の数値の有効数字 10 桁全部を見るには [f] CLEAR [PREFIX] と押します。[PREFIX] を押し続けている間は有効数字を表示します。

キー操作	表　示
[g] [π]	**3.1416**
[f] CLEAR [PREFIX] (押したまま)	**3141592654**

丸め誤差

上記のように HP–15C はどんな数値も有効数字 10 桁を記憶します。中間計算結果などの有効数字が 11 桁以上になると，自動的に有効数字が 10 桁になるように四捨五入しています。計算機は π や 2/3 (0.666...) のような無限数を四捨五入しているので，わずかですが丸め誤差が発生します。この誤差は長い計算のときには大きくなりますが，普通の計算では気にする必要は無いでしょう。数値解析の手法を使えばこの誤差も調べられますが，この本の目的とページ数では無理です。別途販売の英文の HP–15C Advanced Functions Handbook にはその一部を解説しています。

特別な表示

状態表示

HP–15C の表示部分にはそのときの状態を表すために 8 種の状態表示の記号があります。その意味と説明しているページは下記の通りです。

*****	低電圧表示，62 ページ。
USER	利用者優先状態，79 と 144 ページ。
f か **g**	第二機能の前操作表示，18～19 ページ。
RAD か **GRAD**	三角関数の角度単位表示，26 ページ。
C	複素数計算状態，121 ページ。
PRGM	プログラム入力状態，66 ページ。

桁区切り記号

HP-15Cのスイッチを始めて入れたときには，数値の整数部分と小数部分の区切り用に点（小数点），整数部分の3桁ごとの区切りにコンマが入ります。フランスのように点とコンマが逆の国もあるので，逆にすることも出来ます。それにはまず記算機のスイッチを切ります。それから [ON] を押したまま，[.] も押し，[ON] を先に放してから [.] も放します（[ON]／[.] と略記します）。この操作を繰り返すと記号が交互に反転します。

キー操作	表　示
12345.67	**12,345.67**
[ON]/[.]	**12.345,6700**
[ON]/[.]	**12,345.6700**

エラー表示

正しくない操作(例えば0で割る)をしようとするとエラー表示(**Error** の後に数字が続いたもの）になります。このエラー表示とその原因については付録Aにまとめました。

Error を消して，エラー発生直前に戻すにはどのキーを押しても結構です。エラー発生原因を訂正すれば正しい計算が続けられます。

桁あふれと下位桁あふれ

桁あふれ　計算途中でどれかのレジスタ中の絶対値が 9.999999999×10^{99} よりも大きくなると，そのレジスタには $\pm9.999999999\times10^{99}$ が入り，桁あふれ（オーバーフロー）フラグのフラグ9をセットします*。フラグ9は表示の点滅に関係があります。プログラム計算中に桁あふれが発生しても，プログラムの終りまで計算を続行し，答の表示が点滅します。

[←]，[ON]，または [g] [CF] 9 を押してフラグ9をクリアすれば，表示の点滅が止まります。

下位桁あふれ　計算途中にどれかのレジスタ中の絶対値が $1.000000000\times10^{-99}$ よりも小さくなると，その代わりに0が入ります。この下位桁あふれ（アンダーフロー）はこれ以外には一切の影響はありません。

＊これを表示しても有効数字の末尾3桁分は見えません。

低電圧表示

電池の電圧が下がってくると，表示部の左下隅に星印（＊）が点滅し始めて，電池交換の時期が近づいたことを知らせます。星印の点滅が始まっても，計算が連続するプログラムで10分以上，普通の手操作では1時間以上は計算を続けられます。電池交換については付録F（259ページ）を見てください。

不揮発性メモリー

計算機の状態

HP-15Cは不揮発性メモリーを採用したので，スイッチを切っても計算機の下記の状態はそのまま記憶し続けます。

- 計算機中の全データ（数値）。
- 計算機中の全プログラム。
- プログラム用メモリー中の位置。
- 表示形式の指定。
- 三角関数の角度の単位（度，ラジアン，グラード）。
- リターン条件付きのサブルーチン情報。
- フラグのセット状態（ただしフラグ9だけはスイッチを切るとクリアします）。
- 利用者優先状態になっているかどうか。
- 複素数演算状態になっているかどうか。

HP-15Cのスイッチを入れると，必ず計算状態になります。

計算機のスイッチを切ってあれば，電池交換のために電池を外しても不揮発性メモリーの内容は消えません。電池を入れると元の通りにプログラムやデータが使えます。電池交換については付録Fを見てください。

不揮発性メモリー内容の消去

HP-15Cの不揮発性メモリー内容の消去（リセット）は次のようにします。

1. 計算機のスイッチを切る。
2. [ON] を押したまま [−] も押す。
3. [ON] を先に放してから [−] も放す。（これと [ON] ／ [−] と略記します。）

不揮発性メモリーの内容を消すと **Pr Error** の表示になります。任意のキーを押すと 0.0000 の表示になります。

> 注　計算機を落とすなどの不適当な操作をすると，不揮発性メモリーの内容が消えてしまうことがあります。

第2部

プログラムの作成と利用

第 6 章

プログラム作成の基礎知識

第 6～10 章は HP–15C のプログラムの作成と利用について説明します。どの章も仕組みの部分で基本的な事柄を説明し，例の部分でその応用例を説明し，詳細説明でもっと詳しく説明します。HP–15C の利用程度によって必要な部分を読んでください。

仕　組　み

プログラムの作成

HP–15C のプログラムは手計算のときと同じキー操作通りに記録すれば良いので簡単です。(これをキー操作通りのプログラムと呼びます。) プログラムをまとめるときにはそれ以外に二つの操作が必要です。それは，(1)データをいつどのように入れるかを決めることと，(2)プログラムを計算機中に入れて記憶しておくことです。計算機はプログラム中の条件判断や無条件ジャンプ，条件ジャンプで繰り返し計算を始めたり，終りにすることも出来ます。

ここでは本書の始めの HP–15C 操作法入門のプログラム (15 ページ) を順番に説明することにします。

プログラムの入力

プログラム入力状態　[g] [P/R] を押すと計算機はプログラム入力状態 (**PRGM** の文字を表示します) に切り替わります。プログラム入力状態でキーを押してもその機能を実行しないで，それを命令として記憶するだけです。

キー操作	表　示	
[g] [P/R]	000–	プログラム入力状態に切り替えると，**PRGM** の文字が見え，行番号の 000 を表示します。

プログラム用メモリー中の位置　プログラム用メモリーとそのメモリー中の計算機の位置は行番号で表します。000行はプログラム用メモリーの始めの部分で，そこには命令を記憶しておくことは出来ません。最初の命令を記憶するのは001行です。000以外のプログラム行は命令を入れるまでは存在しません。

既にあるプログラム行（nnnと表します）のどこからでもプログラムを入れ始めることが出来ますが，サブルーチンでなくて別のプログラムを入れるときにはプログラム用メモリーの始めの部分から入れるのが一番簡単で安全な方法です。既にプログラムを入れてあっても，新しいプログラムを入れるとそれまでのプログラムは自動的に位置がずれます。

プログラム入力状態か計算状態のどちらでも [GTO] [CHS] 000と押すと，[GTO] 命令を記録しないで000行に合わせることが出来ます。計算状態で [f] CLEAR [PRGM] と押せば，プログラム用メモリーを消さずに000行に合わせられます。

逆にプログラム用メモリーを全部消して000行に合わせることも出来ます。それにはプログラム入力状態で [f] CLEAR [PRGM] と押します。

プログラムの開始点　あるプログラムやルーチン（手順）の始めを区別するためにラベル命令（[f] [LBL] の後に [A] から [E] の文字か，0～9または.0～.9の番号を続けたもの）を使います。このラベル（目印）を使うと目的のプログラムやルーチンの開始点を素早く探して計算を開始出来るようになります。

キー操作	表　示	
[f] CLEAR [PRGM]	000–	プログラム用メモリーを消して000行（プログラム用メモリーの始め）に合わせる。
[f] [LBL] [A]	001-42,21,11	

プログラムの入力　どのキー（演算記号や数字も含めて）を押しても，それをプログラム命令としてメモリー中に記憶します*。

＊81ページに記載したプログラム不能の機能は例外です。

キー操作	表示		
2	002–	2	002～008行で，Xレジスタ中のhを使って $\sqrt{\frac{2h}{9.8}}$ を計算する。
[×]	003–	20	
9	004–	9	
[·]	005–	48	
8	006–	8	
[÷]	007–	10	
[√x]	008–	11	

プログラムの終点　プログラムの終点には次の3通りがあります。

- [g][RTN]をプログラムの終点に使うと，000行に戻ってそこで止まる。
- [R/S]はプログラムを止めるだけで，000行には戻さない。
- プログラム用メモリーの最後には自動の[RTN]がある。

キー操作	表示		
[g][RTN]	009–	43 32	これ以降のプログラム用メモリー中に何も無ければ，この操作は不要です。

プログラム途中の停止

[f][PSE]　をプログラム命令として使うと，プログラムを一時的に止めて中間結果を表示することが出来ます。(一つの [PSE] だけでは表示時間が短いときは二つ以上続けてください。)

[R/S] 命令を使うとプログラムを完全に止めてしまいます。プログラムはその行に止まったままなので，計算状態で[R/S]を押すと（つまりキー操作をすると），止まっている行からプログラムを続行することが出来ます。

プログラムの走行

計算状態　プログラムを入れ終ったら[g][P/R]を押して計算状態に戻します。計算状態でないとプログラムの走行（実行とも言います）を始められません。

キー操作	表　示	
g P/R		計算状態，**PRGM** の文字が消える。(表示内容は以前の計算の答によって異なる。)

この状態切り替えではプログラム用メモリー中の位置は変わりません。計算機のスイッチを一たん切ると，次にスイッチを入れれば必ず計算状態になります。

プログラムの走行　計算状態で f の後にプログラム中のラベルの文字（A ～ E）を押すか，GSB の後にラベル番号（または文字）を押します。こうするとプログラム中の目的のラベルから実行を始めます。このとき running（走行中）の文字を表示します。

キー操作	表　示	
300.51	**300.51**	X レジスタに h をキー入力。
f A	**7.8313**	ラベル A のプログラムの答。(これは 300.51m の高所から物を落したときの地面までの所要秒数です。)

プログラムの続行　R/S 命令で止まったところからプログラムを続行するには R/S を押します。

利用者優先状態　利用者優先状態にすると，プログラム実行時のラベル名指定のキー操作を一つ減らすことが出来ます。f USER と押すと利用者優先状態になって（USER の文字を表示），A ～ E の前に f を押す必要がなくなり，その代わりに第一機能の前に f を押すようになります。こうすると A ～ E のキーを一つ押すだけで目的のプログラムを実行出来ます（f や GSB を押すのは不要です）。

データ入力法

どんなプログラムも計算に必要なデータをいつ，どのようにして入れるか検討する必要があります。これは計算状態でプログラム走行前にデータを入れておくか，プログラム走行を中断してデータを入れるかということです。

1. **計算開始前に入力**　一つのデータをプログラムの 1～2 行目で使うのであれば，プログラムの開始前に X レジスタに入れるようにします。データをもっと後の方で使うときは STO を使って記憶レジスタに入れておき，プログラム途中に RCL で呼び出すようにします。

前例ではプログラム開始前にXレジスタへhを入れました。プログラム開始時（ここでは f A ）にデータ入力を区切って，スタック上昇が可能になるので ENTER 命令は不要です。このプログラムではXレジスタの内容（h）を2倍しています。

スタックが有るのでプログラム開始前に二つ以上のデータを入れておくことが出来ます。プログラム中のスタック内の操作（例えば x≷y ）でスタックの内容がどう動くかということさえ気を付ければ，X～Tレジスタへ4個のデータを入れることも可能です。

2. 直接入力　プログラム途中でデータを入れる方法です。プログラム中にデータ入力が必要な個所に R/S 命令を入れて，プログラムがそこで止まるようにします。データを入れてから R/S を押すとプログラム走行を続けることが出来ます。

プログラム命令中にそのときによって変わる数値をキー入力しては駄目です。このような数値はプログラム実行のたびにキー入力するようにします。

プログラム用メモリー

不揮発性メモリーの内容を消したときには，HP-15Cはプログラム用メモリーとして322バイト，記憶レジスタが21個です。大部分のプログラム命令は各1バイトですが，2バイトのものもあります。メモリー配分は付録Cで説明するように変えることが出来ます。プログラム用メモリーの最大は448バイト（このときは専用記憶レジスタの $R_{.1}$，$R_{.0}$，R_1 だけが使えます），記憶レジスタの最大数は67個（このときはプログラム用メモリーが0）です。

例

ある缶詰会社で3缶1組の即席スパゲッティ用ソースの缶詰を計画しています。スパゲッティ用ソース，粉チーズ，肉だんごを別々の缶に入れておくのだそうです。この会社では個々の缶の底面積，体積，表面積とそれぞれの合計を求めたいそうです。

プログラム計算に必要な計算式とデータは次の通りです。

底面積＝円周率×半径の2乗＝πr^2

体　積＝底面積×高さ＝$\pi r^2 h$

表面積＝底面積の2倍＋側面積＝$2\pi r^2+2\pi rh$

半　径	高　さ	底面積	体　積	表面積
2.5cm	8.0cm	?	?	?
4.0	10.5	?	?	?
4.5	4.0	?	?	?
	合　計	?	?	?

計算方針

1. 半経 r を計算機に入力したら，後の計算用に一たん記憶する。底面積を求めたらこれも後の計算用に記憶し，底面積の合計用に別のレジスタに加算する。
2. 高さ h を計算機に入力して体積を計算する。体積の合計用にこれも別のレジスタに加算する。
3. r を呼び出す。体積を r で割って2倍すると側面積になる。底面積を呼び出して2倍し，側面積に加えると表面積になる。これも別のレジスタに加算する。

プログラムを書き出しているときには実際のデータを使いません。実際のデータは計算に必要なときに入れます。計算開始前に入れるか，計算途中に入れるかによって入れる順序を変えることがあります。

上の問題を解くために次のプログラムをキー入力してください。表示は行番号とキーコード（そのキーの上下左右位置を示すもの）で，これについては詳細説明の部分で説明します。これはプログラムの一例で，これ以外の方法もあります。

キー操作	表　示	
g P/R	000–	プログラム入力状態にする（**PRGM** の文字が見える）。
f CLEAR PRGM	000–	プログラム用メモリーを消して 000 行に合わせる。

キー操作	表　示	
[f] [LBL] [A]	001-42,21,11	このプログラムのラベルをAにする。
[STO] 0	002- 44 0	Xレジスタの内容をR_0に入れて記憶。プログラム計算を始める前にXレジスタにrを入れておく必要がある。
[g] [x^2]	003- 43 11	Xレジスタの内容（つまりr）を2乗。
[g] [π]	004- 43 26	
[×]	005- 20	πr^2，缶の底面積。
[STO] 4	006- 44 4	底面積をR_4に入れて記憶。
[STO] [+] 1	007-44,40, 1	底面積の合計用としてR_1に加算。
[R/S]	008- 31	底面積を表示するためと，hを入れられるようにするために一たん止める。
[×]	009- 20	底面積にhを掛けて体積を求める。
[f] [PSE]	010- 42 31	体積表示のために一時休止。
[STO] [+] 2	011-44,40, 2	体積の合計用としてR_2に加算。
[RCL] 0	012- 45 0	rを呼び出す。
[÷]	013- 10	体積をrで割る。
2	014- 2	
[×]	015- 20	$2\pi rh$，缶の側面積。
[RCL] 4	016- 45 4	底面積を呼び出す。
2	017- 2	底面積を2倍する（上側と下側）。
[×]	018- 20	

キー操作	表　示	
[+]	019-　40	側面積＋両底面積＝表面積。
[STO][+]3	020-44,40, 3	表面積の合計用に R_3 に加算。
[g][RTN]	021-　43 32	プログラムの終りで，プログラム用メモリーの000行に戻す。

それではこのプログラムで計算してみましょう。

キー操作	表　示	
[g][P/R]		計算状態に戻す。(**PRGM** の文字が消える。)
[f]CLEAR[REG]		全記憶レジスタの内容を0にする。表示は変わらない。
2.5	2.5	1番目の缶の r を入力。
[f][A] (または[GSB][A])	19.6350	ラベルAのプログラムの走行開始。1番目の缶の底面積。(計算中には **running** の文字を点滅表示。)
8	8	1番目の缶の h を入力。それから計算を再開する。
[R/S]	157.0796	1番目の缶の体積。
	164.9336	1番目の缶の表面積。
4	4	2番目の缶の r を入力。
[R/S]	50.2655	2番目の缶の底面積。
10.5	10.5	2番目の缶の h を入力。
[R/S]	527.7876	2番目の缶の体積。
	364.4247	2番目の缶の表面積。
4.5	4.5	3番目の缶の r を入力。
[R/S]	63.6173	3番目の缶の底面積。

4	**4**	3番目の缶の h を入力。
R/S	**254.4690**	3番目の缶の体積。
	240.3318	3番目の缶の表面積。
RCL 1	**133.5177**	底面積の合計。
RCL 2	**939.3362**	体積の合計。
RCL 3	**769.6902**	表面積の合計。

このプログラム中でプログラム作成の基本的な知識を説明しました。またプログラム入力状態と計算状態でデータの入力，記憶，呼び出し用に STO と RCL を使い，レジスタの直接演算とプログラム途中の停止も使っています。

詳細説明

プログラム命令

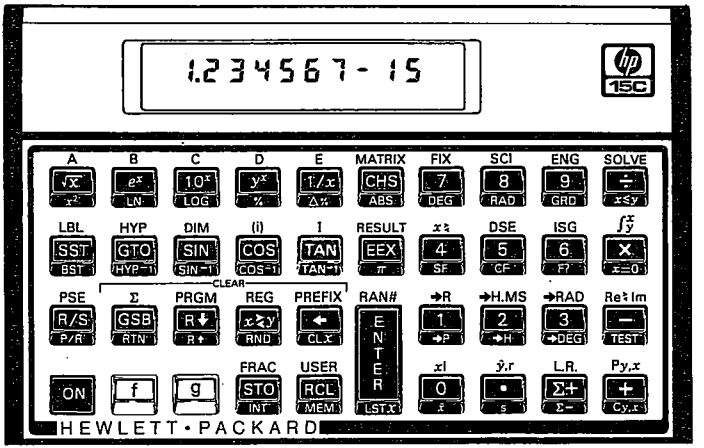
数字，小数点と各機能はどれも一つごとに一つの命令として解釈し，プログラム用メモリー中にそれぞれを1行分として記憶します。前操作キー（例えば f，STO，GTO，LBL）を併用する機能は全体で1行分です。大部分の命令はプログラム用メモリーの1バイトを使いますが，2バイトを使うのもあります。2バイトを使う命令の一覧表は付録Cにあります。

命令の読み方

プログラム入力状態ではHP-15Cの各キー（ただし数字の0～9を除きます）ごとに別々の2桁のキーコード（キーの位置に対応）が付いていて，これでキーを区別します。

命令	キーコード	
STO + 1	**006-44,40, 1**	6番目のプログラム行。
f DSE I	**XXX-42, 5,25**	DSE は5のキー。

2桁のキーコードの左側はキーの上下位置（上から順に1～4），右側は左右位置（左から順に1，2，…9，0）です。数字キーだけはその数字通りの1桁のキーコードです。



キーコードの25は上から2段目で左から5番目のキー

メモリー配分

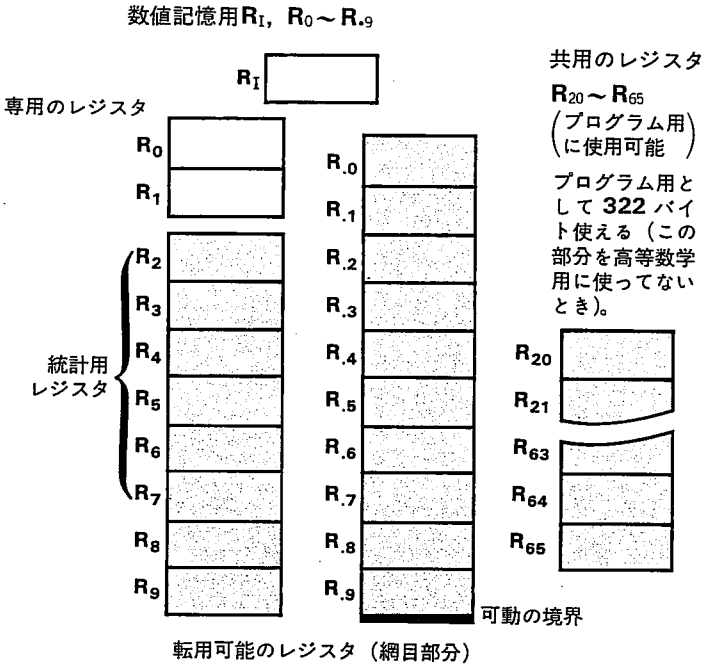
HP–15Cのメモリー配分を知っておくことはそれほど重要ではありません。大切なことは，プログラム用メモリーを効率的に使う方法を覚えることです。プログラムをたくさん作ればそれだけ利口になります。メモリー配分については付録Cメモリー配分で説明します。

もし**Error 10**を表示したらそれはHP–15Cの限度を超えています。メモリー配分を知っていればHP–15Cにデータやプログラムを入れるときの知識が豊富になります。

HP–15Cのメモリーはレジスタが67個（R_0～R_{65}とR_I）で，これを数値の記憶とプログラム用，高等数学用に分割して使います。最初の配分は次の通りです。

- プログラム用と高等数学（SOLVE，f，虚数用スタック，MATRIX 演算）用の兼用分としてレジスタ46個。レジスタ1個が7バイトなのでこれを高等数学用に使わなければ，プログラム用として322バイト分あります。
- 数値記憶用としてレジスタ21個（R_0～R_9，$R_{.0}$～$R_{.9}$，R_I）。

最初のメモリー配分



メモリー配分指定とは数値記憶用レジスタとして使う予定の最大のレジスタ番号を計算機に教えることで，数値記憶用以外のレジスタをプログラム用と高等数学用として使うことになります。

キー操作	表　示	
60 [f] [DIM] [(i)] *	60.0000	R_{60}以下を数値記憶用に使う，残りの5個(R_{61}～R_{65})はプログラム用に使える。

---

* 前操作キーの後には [f] キーを押すのを省略できます。78ページのキー操作の省略参照。

キー操作	表　示	
1 f DIM (i)	1.0000	R_1 と R_0 を数値記憶用に指定し，R_2～R_{65} がプログラム用と高等数学用に使える。
19 f DIM (i)	19.0000	最初の配分で，R_{19}($R_{.9}$) 以下が数値記憶用，R_{20}～R_{65} がプログラムと高等数学用*。
RCL DIM (i)	19.0000	そのときの最大のレジスタ番号を表示。

DIM と MEM (メモリーの状態確認) については付録 C で説明します。

これまでのメモリー配分について次のエラー表示を覚えておいてください。

1. 最初のメモリー配分のままで R_{19} よりも大きな番号のレジスタを数値の記憶などに使おうとすると **Error 3** になる。
2. 最初のメモリー配分のままで 322 バイトのプログラムを入れ，更にもっと入れようとすると **Error 4** になる。
3. メモリー配分が不適当なままで高等数学を手操作またはプログラム中で実行しようとすると **Error 10** になる。

プログラムの境界

終点　それぞれのプログラムの終点に RTN や R/S 命令が必要という訳ではありません。プログラム用メモリー中の最後のプログラムの後には自動的な RTN 命令があるので，RTN をわざわざ入れる必要はありません。またあるプログラムから別のルーチン（部分）に GTO (第7章で説明) を使ってジャンプするときは，そこがプログラムの終点になります。

ラベル　プログラムやサブルーチンなどのラベルはその開始点の目印です。f ラベルや GSB ラベル命令（キー操作を含む）があると，計算機はプログラム用メモリー中を下の方に（行番号が多い方に）それと同じラベルを探し始めます。プログラム用メモリーの最後まで行っても見付からないと，引き続き 000 行から探します。同じラベルが見

---

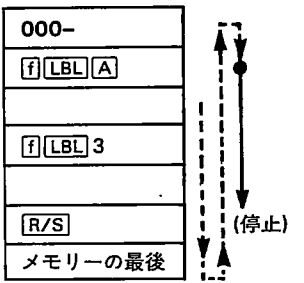
*メモリー配分やレジスタの間接指定では $R_{.0}$～$R_{.9}$ は R_{10}～R_{19} と同じに考えてください。

付かると探すのをやめて，そこから実行を始めます。

プログラム実行中にラベルがあっても，何の効果もなくただそのまま次に進むだけです。ですからラベルはあるプログラム中のルーチンの目印と考えてください(サブルーチンについては第9章で説明します)。

計算機はそのときの位置からいつも下方に向ってラベルを探すので，同じラベルを複数使うことが出来ます（あまり気のきいた方法ではありませんが)。プログラムの実行は最初に出合った指定のラベルから始めます。

ここでラベルAを探すために f A と押すと，
まず下方に進み，次に000行に戻ってからラベルAのところで探すのをやめる。それから実行を開始して（途中のラベルは無視する）停止命令まで実行を続ける。



予定外のプログラムの停止

どれかのキーを押した　プログラム走行中にどれかのキーを押すと停止します。複雑な命令を実行中のときはすぐに止まらないで，そのときの命令の実行が終ると止まります。

エラーの発生　プログラム中で正しくない操作をしようとすると直ちに走行を停止して，**Error** を表示します。

エラー原因の行でプログラムが止まっているので，その行番号とキーコードを見るには任意のキーを一つ押して **Error** 表示を消し，それからプログラム入力状態に切り替えます。

プログラムが停止したときに表示が点滅していたら，桁あふれ（61ページ）が発生したためです。 f ， ON ，または g CF 9 を押すと点滅が止まります。

キー操作の省略

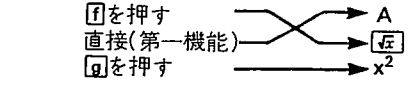
場合によっては前操作キーの f を押すのを省略することが出来ます。

これをキー操作の省略と呼び，前操作キーの後には [f] キーを押すのを省略出来ます。(前操作キーの一覧表は 19 ページにあります。)

例えば [f][LBL][f][A] は [f][LBL][A]，[f][DIM][f][(i)] は [f][DIM][(i)]，[STO][f][RAN#] は [STO][RAN#] になります。このようなときの [f] キー操作の省略は慣れれば簡単です。このようなときに [f] キーを押しても，キーコード中には [f] キーの分が入りません。

利用者優先状態

利用者優先状態とはプログラムを実行するときにキー操作を簡単にするための状態です。[f][USER] と押して利用者優先状態 (**USER** の文字を表示) にすると，[A] ～ [E] には [f] を押す必要が無くなり，代わりに第一機能の方のときに [f] を押すことが必要になります。これを図示すると次の通りです。



この状態のときに [f][USER] を押すと USER の表示が消えて普通の状態に戻ります。

多項式とホーナー法

多項式の中には同じ数値を何回も使う計算式があります。例えば次式では同じ x を 4 回も使っています。

$$f(x)=Ax^4+Bx^3+Cx^2+Dx+E$$

これをプログラムで解くには x を一たんレジスタに記憶しておき，必要なときに呼び出して使っても良い訳です。しかしもっと利口な方法はスタック内に全部入れておくことです (41 ページのスタックを利用した定数計算参照)。

このような計算式のときに計算の操作と時間を短縮するために多項式の組み合わせを変えるホーナー法というのがあります。特に [SOLVE] や [∫f] の答を求めるときに，サブルーチン中で多項式の値を求めるときに効果を発揮します。

その方法とは，次のように括弧を使って多項式中の累乗計算を減らすことです。([y^x] を使うと x や y が大きいときには答が不正確になる

ことがあります。)

$$Ax^4+Bx^3+Cx^2+Dx+E$$
$$(Ax^3+Bx^2+Cx+D)x+E$$
$$((Ax^2+Bx+C)x+D)x+E$$
$$(((Ax+B)x+C)x+D)x+E$$

例 $5x^4+2x^3$ を $(((5x+2)x)x)x$ の形で計算するプログラムを作り，$x=7$ のときの値を求めてください。

キー操作	表　示		
[g] [P/R]	000-		プログラム用メモリー中の000行にあるものとします。前に入れたプログラムを消さなくても結構です。
[f] [LBL] [B]	001-42,21,12		
5	002-	5	
[×]	003-	20	$5x$.
2	004-	2	
[+]	005-	40	$5x+2$.
[×]	006-	20	$(5x+2)x$.
[×]	007-	20	$(5x+2)x^2$.
[×]	008-	20	$(5x+2)x^3$.
[g] [RTN]	009-	43 32	
[g] [P/R]			計算状態に戻す。表示するのは前の計算結果。
7 [ENTER] [ENTER] [ENTER]	7.0000		スタック（X～Tレジスタ）に7を入れる。
[f] [B]	12,691.0000		

ここでは7をキー入力してから [ENTER] を3回押し，それからプログラムの実行を開始しましたが，上のプログラムの001～002の間に [ENTER] 命令を3個続けて入れておくと，x をキー入力するたびに [ENTER] を3回押す操作が不要になります。また利用者優先状態にすれば [B] の前に [f] を押す操作も不要になります。

プログラム不能の機能

プログラム入力状態のときに大部分の機能はプログラム命令として入れることができます。次の機能だけはプログラム命令として入れることが出来ません（これをプログラム不能と呼びます)。

f CLEAR PREFIX	g BST	SST
f CLEAR PRGM	g MEM	←
f (i)	g P/R	ON / ·
f USER	GTO CHS *nnn*	ON / −

練習問題

1. ある村では正午の合図に消防の見張り塔のサイレンを鳴らしています。サイレンから 3.2 m 離れた見張り塔のドアでは 138 デシベル（音の大きさの単位，dBと略記）です。サイレンからの距離を入れて音の大きさを求めるプログラムを作ってください。

   計算式は $L = L_0 - 20\log(r/r_0)$ で，L_0 は音源のそばの音の強さ（138 dB），r_0 は音源からの距離（ここでは 3.2 m），L は求めたい位置の音の大きさ，r は求めたい位置の音源からの距離です。

   r が 3km の位置の音の強さは何 dBでしょうか？

   上の式のプログラム入力のキー操作の一例は次の通りです。
   g P/R f LBL C 3.2 ÷ g LOG 20 × CHS 138 + g RTN g P/R で 15 プログラム行，メモリーを 15 バイト使います。この問題を一般的に解くには，(1)L_0 と r_0 を計算前にレジスタに記憶させておいてプログラム中で必要なときに呼び出す，(2)計算前に L_0 ENTER r ENTER r_0 のようにスタックに入れてから計算する，という 2 通りの方法があります。

   （答　r=3km のとき，L=78.5606dB）

2. 1個 200g のある標準的なトマトには 188g の水分（94%）があるそうです。あるトマト栽培家が水分の少ないトマトを作ろうと研究しています。別のトマトの重さとその水分の重さを入れて，標準的な含水率と比較するプログラムを作ってください。計算用の数値はプログラム途中で止まったときに入れるようにすることにします。

   230g のトマトで水分が 205g では標準的な含水率に比較してどうでしょうか。

プログラム入力のキー操作の一例は次の通りです。
[f] [LBL] [D] .94 [ENTER] [R/S] (別のトマトの水分の重さを入れる)
[ENTER] [R/S] (そのトマトの重さを入れる) [÷] [g] [Δ%] [g]
[RTN] で11プログラム行，11バイトを使います。

(答 この230gのトマトの含水率の変化は −5.1804% です。)

第7章

プログラムの編集

計算機中に入れたプログラムを種々の理由で変更したいことがあると思います。例えば [STO]，[PSE] や [R/S] のような命令を追加や削除してみたいとか，エラーが発生したなどです。HP-15C にはこのプログラムの変更用に種々の機能を用意しました。

仕組み

プログラムの変更にはどんな場合でも二つの操作が必要で，まず目的の行（変更が必要なところ）に移動し，命令を削除したり追加することです。

プログラム用メモリー中の行の移動

[GTO] 命令　計算状態でもプログラム入力状態（**PRGM** の文字を表示）でも [GTO] [CHS] nnn と押すと nnn 行に移動することが出来ます。この操作はプログラム不能だからで，プログラム用メモリー中の目的の行へ手操作で移動します。行番号は $000 \leqq nnn \leqq 448$ でしかもそのときに実際にある番号を必ず3桁で指定します（行番号が1～2桁だったらその前に0を付けて3桁になるようにします）。

[SST] 命令　プログラム用メモリー中を下方に1行ずつ動くときに [SST] を使います。これもプログラム不能です。

プログラム入力状態では [SST] で1行進んでそこの命令を表示します。命令は実行しません。このキーを押したままにすると，プログラム用メモリー中を順々に移動します。

計算状態で [SST] を押したままにするとそのプログラム行の行番号とキーコードを表示します。キーを放すとその命令を実行してその結果を表示し，その次の行に進みます。

[BST] 命令 プログラム用メモリー中を逆方向（行番号の小さな方）に一つ戻るにはプログラム入力状態または計算状態で [BST] を押します。プログラム入力状態で [BST] を押し続けると連続して戻れます。どちらもプログラム命令を実行しません。これもプログラム不能です。

プログラム行の削除

プログラム命令を削除するにはプログラム入力状態で [←] キーを押します。まず削除したい行まで移動してから [←] キーを押します。後に残っているプログラム行は番号が続くように順に繰り上がります。

計算状態で [←] キーを押してもプログラム用メモリーには関係ありません。これは表示を取り消すのに使います。(21 ページ参照。)

プログラム行の挿入

プログラムを追加するにはまず挿入したい位置の直前の行に移動します。ここでプログラム命令をキー入力すると，それが表示していた行の次に追加して入ります。命令を変更するときには，まずその命令を削除し，続いて新しい命令を追加するようにします。

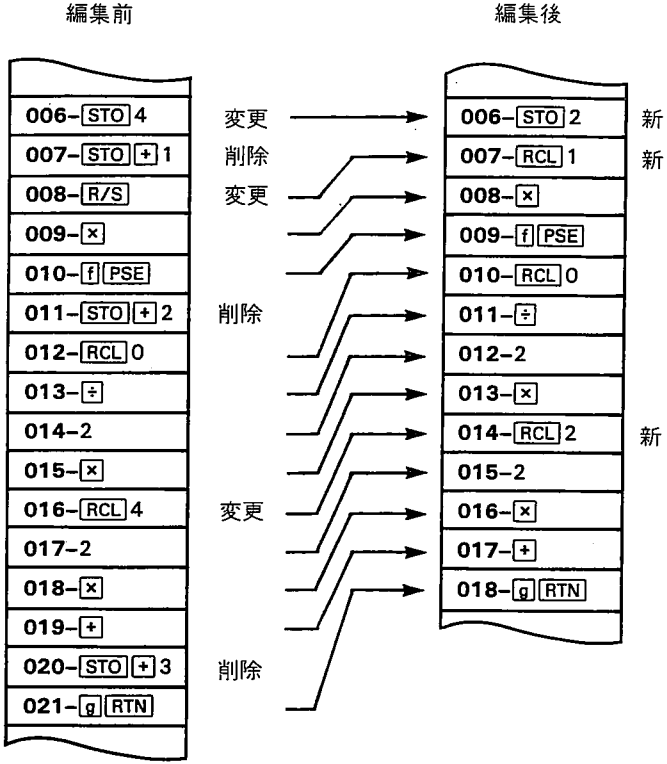
例

第6章，71 ページの缶の体積のプログラムをもう一度取り上げて，命令を少し変更してみましょう。(次ページのプログラムは 001 行から始まっているものの一部です。)

削除するもの 底面積，体積，表面積の合計を計算する必要がないときは記憶用レジスタの加算（行 007，001，020）を削除します。

変更するもの 高さ (h) を入力するためにプログラムを停止する必要がなくなったので [R/S] 命令を [RCL] 1 命令に修正して（底面積の合計を入れる R_1 はもう使いません），h の値はプログラムの実行前に R_1 に記憶しておきます。R_2 と R_3 ももう使わないのでプログラムをすっきりするために [STO] 4 (行 006) を [STO] 2 に，[RCL] 4 (修正前の行 016) を [RCL] 2 に変更します。

この編集の過程は次ページの図の通りです。

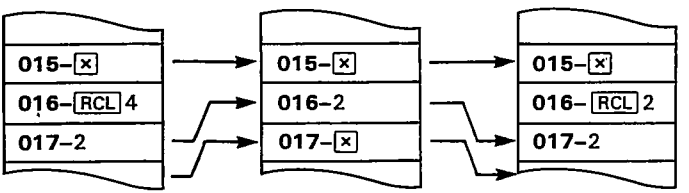


編集はプログラムの終点から始めて逆向きに進めます。こうすると命令を削除してもそれより若い行番号が変わらないからです。

キー操作	表　示	
g P/R	000–	プログラム入力状態。(最初は行000にあると思います。)
GTO CHS 020	020–44,40, 3	行020(命令は STO + 3)に移動。
(または SST を使って)		

キー操作	表　示	
←	019–	40 行 020 を削除。
g BST （押し続ける）	016–	45 4 次に編集する行は 016（RCL 4）。
←	015–	20 行 016 を削除。
RCL 2	016–	45 2 行 016 を RCL 2 に修正した。
GTO CHS 011 （または BST を押し続ける。）	011–44,40,	2 行 011（STO + 2）に移動。
←	010–	42 31 行 011 を削除。
g BST （押し続ける）	008–	31 ここです！（行 008 の R/S まで 1 行ずつ移動しても良い。）
←	007–44,40,	1 R/S を削除。
RCL 1	008–	45 1 行 008 を RCL 1 に修正。
g BST	007–44,40,	1 行 007 に戻る。
←	006–	44 4 行 007（STO + 1）を削除。
←	005–	20 行 006（STO 4）を削除。
STO 2	006–	44 2 STO 2 に変更。

プログラム行の入れ替えは次のようになります。



詳細説明

単一行ずつの操作

プログラムの 1 行ずつの実行　プログラムの内容や命令の場所を確めるときには，プログラム入力状態でプログラム中を 1 行ずつ進められ

ます。もしプログラムの走行中にエラーが発生したり，プログラムの一部が間違っていると思われるときには，プログラムを1行ずつ実行して確めるようにします。それには計算状態で [SST] キーを押します。

キー操作	表　示	
[g][P/R]		計算状態に戻す。
[f]CLEAR[REG]		記憶レジスタをクリア。
[GTO][A]		プログラムAの最初の行に移動。
8[STO]1	8.0000	缶の高さを R_1 に記憶。
2.5	2.5	缶の半径を入力。
[SST]（押したまま）	001-42,21,11	行001のキーコード。
（放す）	2.5000	行001の実行結果。
[SST]	002-　44　0	[STO]0。
	2.5000	結果。
[SST]	003-　43 11	[g][x^2]。
	6.2500	結果。
[SST]	004-　43 26	[g][π]。
	3.1416	結果。
[SST]	005-　20	[×]。
	19.6350	結果　缶の底面積。

000への戻り　[SST] でプログラムを1行ずつ進みますが，最後のプログラム行まで来て更に [SST] を押すと行000まで自動的に戻ります。（計算状態では，プログラム用メモリーの最後に [RTN]，[GTO]，[GSB] などの命令があるとそれを実行します。）

行の位置

計算機のスイッチを切っても，プログラム入力状態と計算状態を切り換え（[P/R]）ても，プログラムの行位置は変わりません。プログラム入力状態に戻しても，それまでの行位置に止まっています。（[RTN] を使ってプログラムの実行が終ったときには行位置が000です。）計算機のスイッチを入れたままで自動的にスイッチが切れたときには，スイッチを入れて，プログラム入力状態に切り換えるだけで，それまでの行に戻ります。（この計算機のスイッチを入れたときは必ず計算状態になります。）

挿入と削除

命令を挿入すると，その命令を表示します。削除すると，削除した行の一つ前の行を表示します。

プログラム用メモリーとして使える領域を全部使いきってしまうとプログラム命令の挿入はできなくて **Error 4** の表示になります。

計算機の状態の初期化

プログラム中で記憶用レジスタを使ったり，計算機の状態がプログラムに関係する場合には；レジスタの内容やそのときの状態によってプログラムの実行結果が変わります。計算機の状態がプログラムの走行に不適当なままプログラムを走らせると，正しくない結果になります。そこで，プログラムの走行直前か，プログラムの始めの部分でレジスタをクリアしたり，適切な状態に切り換えるのが利口な方法です。プログラムの中に初期化の命令を組み込んでおけば確実に間違いを防止できますが，その代りにプログラム行を余計に使います。

計算機を初期化する機能は次の通りです。[f] CLEAR [Σ], [f] [PRGM], [f] CLEAR [REG], [g] [DEG], [g] [RAD], [g] [GRD], [g] [SF], それに [g] [CF].

練習問題

同じプログラムのラベルを何回も使わないのが上手なプログラム技法です。(HP-15C には 25 種のラベルがあるので簡単だと思います。) 確実にラベルの重複を避けるために最初にプログラム用メモリーをクリアしてください。

1. ある金融機関では預金口座の元利合計を計算するために次のプログラムを使っています。計算式は $FV=PV(1+i)^n$ で，FV は元利合計，PV は元金，i は利率，そして n は期間数です。計算を開始する前に最初に PV を（Y レジスタに），次に n を（X レジスタに）入力してください。この例では年利率が 7.5% (i=0.075) の 1 年複利です。

キー操作	表　示	
f LBL . 1	001-42,21, .1	
f FIX 2	002-42, 7, 2	
1	003- 1	
.	004- 48	
0	005- 0	1+i。
7	006- 7	
5	007- 5	
x≷y	008- 34	
y^x	009- 14	$(1+i)^n$。
×	010- 20	$PV(1+i)^n$。
g RTN	011- 43 32	

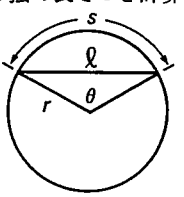
プログラムを入力し，1,000 ドルを 5 年間預金したとき，2,300 ドルを 4 年間預金したときの元利合計を求めてください。数字ラベルの付いたプログラムを走行するときには GSB を使うことを忘れないでください。(答　1,435.63 ドル，3,071.58 ドル)

年利率が 8.0% になるようにプログラムを変更してください。

変更したプログラムを使って，500 ドルを 4 年間預金したとき，2,000 ドルを 10 年間預金したときの元利合計を求めてください。(答　680.24 ドル，4,317.85 ドル)

2. 半径 r で円の角度が θ (度) の円弧の弦の長さ l を計算してください。計算式は次の通りです。

$$\ell = 2r \sin \frac{\theta}{2}.$$



$\theta = 30°$，r=25 のときの l を求めてください。

(答　12.9410。この計算のプログラムの一例は次の通りです。f LBL A　g DEG　f FIX 4　2 × x≷y 2 ÷ SIN × g RTN)。(プログラムの走行前に Y レジスタに θ，X レジスタに r が入れてあるものとします。)

上のプログラムを適当に修正して θ（ラジアン）の円弧の長さ s も計算し，表示するようにしてください。計算式は次の通りです。

$$s = r\theta。$$

次の表の？を求めてください。

θ	r	l	s
45°	50	?	?
90°	100	?	?
270°	100	?	?

（答　38.2683 と 39.2699，141.4214 と 157.0796，141.4214 と 471.2389。修正したプログラム入力のキー操作の一例は次の通りです。[f][LBL][A] [g][DEG] [f][FIX]4 [STO]0 2[×] [x≷y] [STO]1 2[÷] [SIN] [×] [f][PSE] [f][PSE] [RCL]0 [RCL]1 [f][→RAD] [×] [g][RTN]）。

第8章

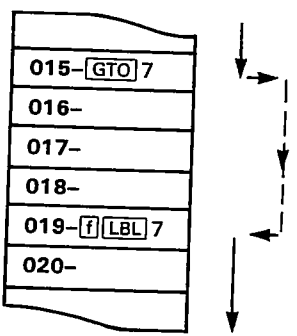
プログラムのジャンプと制御

プログラム中の命令は普通行順に実行しますが，時には次の行でないプログラムの他の部分に実行を移動させたいことがあります。HP-15C では単純なジャンプも，ある条件によってジャンプすることも可能です。元の方の行にジャンプするようにすると，プログラムの一部を何回も繰り返して実行することができ，この方法をループと言います。

仕組み

ジャンプ

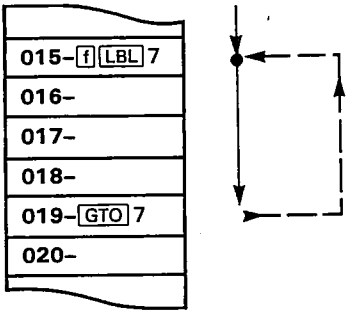
[GTO] 命令　単純なジャンプは無条件ジャンプとも呼び，[GTO] ラベル命令でジャンプします。プログラムの走行中に [GTO] で実行が指定したラベル名（行番号ではありません）のプログラムまたはルーチンに移ります。



計算機はプログラム用メモリーの前方に向ってラベルを探し，必要に応じ見付からなければ行 000 に戻ってそこからまた探し，最初に見付かった指定ラベルの所から実行を再開します。

ループ　[GTO] 命令でより若い番号の行（つまり元の方の行）を指定すると，[GTO] とラベル間の一連の命令を反復して——多分無限回

——実行します。このループの連続は条件ジャンプやループ中に入れた R/S 命令，あるいはプログラム実行中に任意のキーを押す（こうするとプログラムが停止します）だけで制御できます。



条件判断

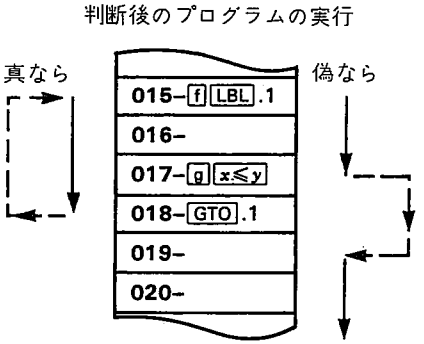
プログラムの実行の順序を変えるもう一つの方法は条件判断です。これはXレジスタの数値を0かYレジスタの数値と比較する真／偽判断です。HP-15Cには12種の条件判断を準備してあって，その内2種はキー操作だけで直接に，残りの10種は g TEST n 命令* を使って実行可能です。

1. 直接　g $x \leq y$ と g $x=0$ 。
2. 間接　g TEST n。

n	判断	*n*	判断
0	$x \neq 0$	5	$x = y$
1	$x > 0$	6	$x \neq y$
2	$x < 0$	7	$x > y$
3	$x \geq 0$	8	$x < y$
4	$x \leq 0$	9	$x \geq y$

*条件判断のうち4種は複素数にも使えますが，これは11章(132ページ)で説明します。

条件判断に出会うと，プログラムの実行は〝真なら次へ〟の原則に従います。つまり，条件が真（イエス）であれば次に進み，偽（ノー）であれば次の行だけを飛び越します。条件判断の次に GTO 命令を組み合わせると，これが条件ジャンプになります。つまり，判断条件が真のときだけジャンプする訳です。



フラグ

プログラムに使うもう一つの条件判断はフラグの判断です。フラグ（旗）はセットしてある（＝真）かクリアしてある（＝偽）かのどちらかの状態の目印です。フラグの判断でも実行は〝真なら次へ〟の原則通りで，フラグがセットのときは次に進み，フラグがクリアの場合は1行飛び越します。

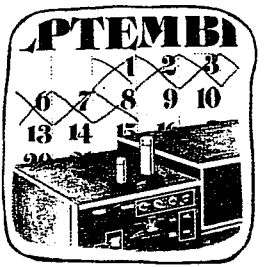
HP-15C には0から7番までの8種の利用者用フラグと8番(複素数計算状態)，9番（桁あふれの状態）の2種の機械状態フラグがあります。機械状態フラグは本章の最後で説明します。どのフラグも次のようにして，セットやクリアと判断ができます。

- g SF n　n番（0から9）のフラグをセットする。
- g CF n　n番のフラグをクリアする。
- g F? n　フラグnがセットかどうかを判断する。

一たんセットしたフラグnは CF n 命令か，または不揮発性メモリー内容を消す（リセット）までセットのままです。

例

ジャンプとループの例



ある放射線生物学研究室でラジオアイソトープ（放射性同位元素）の ^{131}I の放射能の減衰状態を調べたいと思っています。一定の限界値に到達するまで3日ごとに放射能の量を計算するプログラムを作ってみましょう。t日後の残留放射能 N_t を計算する式は

$$N_t = N_0(2^{-t/k})$$

で，k=8日（^{131}I の半減期），N_0 は開始時の量です。

次のプログラムはループを使って，理論的な残留放射能のミリキューリー（mCi）値を3日ごとに計算します。放射能が一定値（計算終了値）まで減衰したかどうかを調べてプログラムを停止するための条件判断も使います。

このプログラムは最初の測定日の t_1 を R_0 に，アイソトープの開始時の量を R_1 に，放射能の計算終了値を R_2 に記憶してから計算するものとして計画しました。

キー操作	表　示		
[g] [P/R]	000–		プログラム入力状態。
[f] CLEAR [PRGM]	000–		（任意の操作。）
[f] [LBL] [A]	001–	42,21,11	ループはこの行に戻る。
[RCL] 0	002–	45 0	そのときのtの値を呼び出す。これはループのたびに変わる。
[f] [PSE]	003–	42 31	tの表示用に一時停止。
8	004–	8	k。
[÷]	005–	10	
[CHS]	006–	16	$-t/k$。
2	007–	2	
[x≷y]	008–	34	
[y^x]	009–	14	$2^{-t/k}$。

キー操作	表　示	
[RCL] [×] 1	010-45,20, 1	R_t の内容 (N_0) を呼び出して掛け算。t日後の残留放射能 N_t を求める。
[f] [PSE]	011-　42 31	N_t の表示用に一時停止。
[RCL] 2	012-　45　2	計算終了値をXレジスタに呼び出す。
[g] [TEST] 9	013-43,30, 9	x≧y? 計算終了値（Xレジスタ）が N_t（Yレジスタ）以上かどうかの判断。
[g] [RTN]	014-　43 32	以上ならプログラムを終了。
3	015-　　3	未満ならプログラムを継続。
[STO] [+] 0	016-44,40, 0	R_0 にあるtに3日を加算。
[GTO] [A]	017-　22 11	"A" に行って，新しいtを使って新しい N_t を求めるのを繰り返す。

行012から014がないと，ループは（キーを押して止めるまで）無限に走るようになります。

それでは，$t_1=2$ 日，$N_0=100$ mCi，計算終了値を N_0 の半分（50mCi）としてプログラムを実行してみましょう。

キー操作	表　示	
[g] [P/R]		計算状態（表示は前回の答）。
2 [STO] 0	2.0000	t_{1_0}
100 [STO] 1	100.0000	N_{0_0}
50 [STO] 2	50.0000	N_t の計算終了値。
[f] [A]	2.0000	t_{1_0}
	84.0896	N_{1_0}
	5.0000	t_{2_0}
	64.8420	N_{2_0}
	8.0000	t_{3_0}
	50.0000	N_{3_0}
	50.0000	N_t の終了値，プログラム終了。

フラグの例

貸借や投資の計算には，利息前払い（計算期間の始めに払う期首払）と利息後払い（計算期間の終りに払う期末払）の2通りの方法があります。ある利率で一定期間ごとに利息と元金の一部の合計を同額ずつ支払う貸借や投資の当初価格（現価）を計算するプログラムを作るときにフラグを使うとプログラムに利息が前払いか後払いかを指定できます。

いまお子様の将来の大学経費支払いを検討中としましょう。1年間の学費が3,000ドル，つまり1か月当たり250ドルとします。年利6％(月利0.5％)の月利複利計算で4年間の在学期間中に毎月支払いを受け取るには，大学入学時に預けておく金額は幾らになるでしょうか。

計算の公式は下記の通りです。

$$V=-P\left[\frac{1-(1+i)^{-n}}{i}\right](1+i)$$ 利息が毎月前払いのとき，

$$V=-P\left[\frac{1-(1+i)^{-n}}{i}\right]$$ 利息が毎月後払いのとき，

Vは最初に預金する総額，

Pは期ごとに口座から引き出す金額，

iは毎期の利率（ここでは，利息が月利複利計算ですから“期”は月です），

nは期数（月数）です。

次のプログラムは上記のどちらの利息支払い方式にも使えます。PをZレジスタに，nをYレジスタに，iをXレジスタに入れてからプログラムを走らせるものとします。

キー操作	表　示	
g P/R	000-	プログラム入力状態。
f LBL B	001-42,21,12	利息前払いのときは"B"から出発。(B は begin— 始めの頭文字で覚えやすい。)
g CF 0	002-43, 5, 0	フラグ0をクリア (偽), 前払いの意味用。
GTO 1	003- 22 1	計算ルーチン1へ。
f LBL E	004-42,21,15	後払いのときは"E"から出発。(E は end— 終りの頭文字。)
g SF 0	005-43, 4, 0	フラグ0をセット (真), 後払いの意味用。
f LBL 1	006-42,21, 1	ルーチン1 (計算ルーチン)。
STO 1	007- 44 1	iを記憶 (X レジスタから)。
1	008- 1	
+	009- 40	$(1+i)$
x≷y	010- 34	nを X に, $(1+i)$ を Y に入れる。
CHS	011- 16	$-n$
y^x	012- 14	$(1+i)^{-n}$
CHS	013- 16	$-(1+i)^{-n}$
1	014- 1	
+	015- 40	$1-(1+i)^{-n}$
RCL ÷ 1	016-45,10, 1	$R_1(i)$を呼び出して割る, $[1-(1+i)^{-n}]/i$ になる。
×	017- 20	Pとの掛け算。
g F? 0	018-43, 6, 0	フラグ0はセットか？
g RTN	019- 43 32	フラグ0がセットなら計算の終り (後払い)。
RCL 1	020- 45 1	iを呼び出す。
1	021- 1	
+	022- 40	$(1+i)$
×	023- 20	前の答に $(1+i)$ を掛ける。
g RTN	024- 43 32	フラグ0がクリアのときの計算の終り。

それではプログラムを走らせて48か月間毎月250ドル受け取るのに必要な最初の預金額を計算しましょう。利率（ここでは月利率）を小数の形で0.005と入力します。まず毎月始めに引き出す（前払い）方，その次に毎月末に引き出す（後払い）方の必要金額を計算します。

キー操作	表　示	
[g] [P/R]		計算状態に戻す。
250 [ENTER]	250.0000	毎月の受け取り額。
48 [ENTER]	48.0000	受け取り期間（4年×12月）。
.005	0.005	月利率を小数で。
[f] [B]	10,698.3049	前払いで引き出すのに必要な預金額。
（繰り返してスタックに入力してから。）		
[f] [E]	10,645.0795	後払いで引き出すのに必要な預金額。（この預金額と全学費（$12,000）との差額は預金の利息です。）

詳細説明

GTO

プログラム不能の操作 [GTO] [CHS] nnn と違って，プログラム可能な操作の [GTO] ラベルは行番号へのジャンプは不可能ですがプログラムのラベル（[f] [LBL] ラベルの行）にジャンプします。* 実行は新しいラベル部分から継続して別の [GTO] 命令がない限り元のルーチンには戻れません。

[GTO] ラベルは計算状態（つまりキー操作）でもプログラム用メモリー中のそのラベルの付いた位置に移動するのにも使えます。移動するだけでその先には進みません。

---

*間接指定を使うとプログラム走行中に特定の行番号にジャンプできますが，これは第10章で説明します。

ループ

ループとは一種のジャンプの応用で，[GTO] 命令を使ってプログラムの一部を反復（繰り返して）実行することです。ループは無限回のものと，条件付きのものがあります。ループは異なる値で計算を反復するときによく使います。またループの反復数を調べるために，ループ1回ごとに1ずつ増えるカウンター（回数計）を使うことがあります。このカウンターを使うと条件判断でループからの脱出時期を決められます。（この例は113ページにあります。）

条件ジャンプ

一般的に条件ジャンプの応用法が二つあります。その第一は上に説明したようなループの制御です。条件判断である計算値やループ回数をチェックします。

もう一つの主な用途は複数の内の一つを選んでそれを実行するのです。例えば，セールスマンの手数料が販売額によって異なるとき，販売額と規準値を比較して，販売額が規準値より多いか，少ないかによって一定の手数料を計算するプログラムなどに使います。

判断　条件判断はXレジスタ内の数値 (x) と0を比較（[x=0]など）するかYレジスタ内の数値 (y) との比較（[x≤y]など）です。従って，x と y の比較をするには前もってXとYレジスタにそれぞれ x と y を入れておく必要があります。それには規準値をレジスタに入れておいてそれをXレジスタに呼び出します。または規準値もスタッグに入れておいて必要に応じて [x≷y]，[R↓] か [R↑] で移動することもあります。

複素数と行列の記号を使った判断　条件判断の内4種 [x=0]，[TEST] 0 ($x\neq 0$)，[TEST] 5 ($x=y$) と [TEST] 6 ($x\neq y$) は複素数と行列の記号にも使えます。詳しい説明は第11，12章でします。

フラグ

条件判断はプログラム中で2数を比較して真偽を決めましたが，フラグはそれ以前の状態で真偽を決めるのに使います。普通は目的の条件

や状態に応じてプログラムの出発点を決め(別々のラベルを使う)それに応じてまずフラグをセットまたはクリアします（この例は96ページで使いました)。

こうすると，二つの異なる入力状態（例えば，度とラジアン）でもその状態に対応した正しいプログラム計算が出来ます。例えば，変換が必要ならフラグをセットし，変換が不要ならフラグをクリアしておきます。

温度を，ケルビンで入力する式を使っていて，ときには温度を摂氏で入れたいこともあるとします。フラグを使えばケルビンでも摂氏でも入力できるプログラムが作れます。プログラムのこの部分は次のようにします。

f LBL C	摂氏のときプログラムは“C”から始める。
g CF 7	フラグをクリア（偽)。
GTO 1	
f LBL D	ケルビンのときプログラムは“D”から始める。
g SF 7	フラグをセット（真)。
f LBL 1	(温度はXレジスタに入れてあるとします。)
g F? 7	フラグのチェック（入力が摂氏かケルビンかの判断)。
GTO 2	フラグがセット（ケルビン入力）なら次の数行を飛んで後のルーチンへ。
2	フラグがクリア（摂氏入力）なら °K=°C+273
7	なのでXレジスタの値に273を足す。
3	
+	
f LBL 2	両方の状態ともここから計算を続ける。

機械状態フラグ　フラグ8と9

フラグ8　フラグ8をセットすると複素数計算状態（第11章で説明）になって，表示部に**C**の文字が見えます。別の方法で複素数計算状態に切り換えても，自動的にフラグ8がセットになります。複素数計算状態でなくする方法はフラグ8のクリアだけです。フラグ8は他のフラグと同じ方法でクリアできます。

フラグ9　桁あふれ（61ページで説明しました）が発生すると，自動的にフラグ9がセットになります。フラグ9をセットすると表示が点滅しますが，プログラム走行中には走行が終ってから点滅を始めます。

フラグ9は次の3方法のどれかでクリアできます。

- [g][CF] 9を押す（フラグをクリアする一般的な操作）。
- [←] を押す。フラグ9だけをクリアして点滅を止めますが，表示はクリア（数値を0に）しません。
- 計算機のスイッチを切る。（計算機のスイッチが自動的に切れたときにはフラグ9をクリアしません。）

フラグ9を手動でセット（[SF]9）すると，計算機の桁あふれ状態に関係なく表示が点滅します。普通はプログラムの実行が表示を点滅しないで完了します。そこで，ある条件を満足したことを視覚的に表示するために，フラグ9をプログラム用の道具として使うことが可能です。

第 9 章

サブルーチン

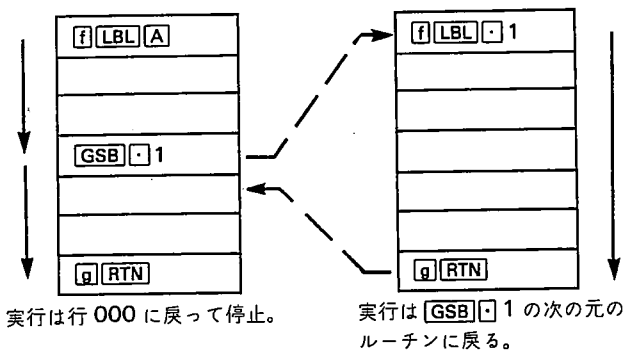
プログラム中で，同じ一連の命令を何回か繰り返して使うとき，このような一連の命令を一つのサブルーチン（下請けみたいなもの）にまとめるとメモリーの使用量が少なくてすみます。

仕　組　み

GSB と GTO

GSB 命令は GTO ジャンプ命令と同じように実行しますが，ただ一つの重要な相違点はリターン（戻る）条件付きということです。GSB ラベルは GTO ラベルと同様にプログラムの実行を対応するラベル（A から E まで，0 から 9 まで，または .0 から .9 まで）の行に移します。しかし，実行を継続して最初の RTN 命令に出会うと，実行が最後の GSB 命令の次の行に戻って，そこからさらに実行を継続します。

サブルーチンの実行

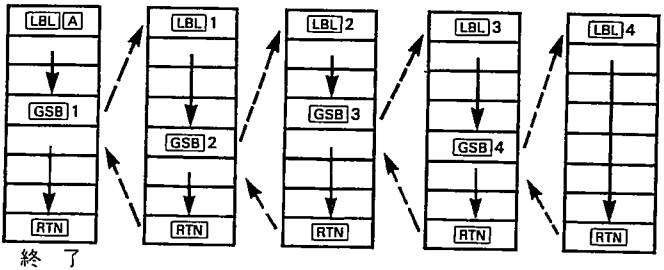


＊ GSB または GTO に文字ラベルを使っていると，78 ページで説明したように f キーを押すのは省略できます。

サブルーチンの制約

あるサブルーチンが別のサブルーチンに移って，そのサブルーチンからまた別のサブルーチンに移ることができます。このサブルーチンの中から別のサブルーチンを実行することを，内側のサブルーチンと呼び，HP-15C では内側のサブルーチンは7重（主プログラムを数えないで）以内です。下図は4重のサブルーチンの実行順序の解説図です。

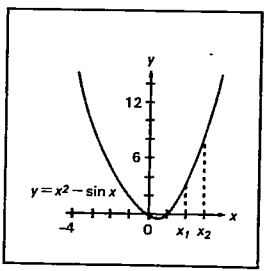
主プログラム



終　了

例

例　右のグラフ（$y = x^2 - \sin x$, xはラジアン単位）の2点 (x_1, y_1) と (x_2, y_2) を結ぶ割線の傾斜を計算するプログラムを作りましょう。



割線の傾斜は次式の通りです。

$$\frac{y_2 - y_1}{x_2 - x_1}\text{，または}\frac{(x_2{}^2 - \sin x_2) - (x_1{}^2 - \sin x_1)}{x_2 - x_1}$$

この答を求めるには，入力した x_1 と x_2 について y の値を2回（1回は y_1，もう一回は y_2）計算する必要があります。違った値について同じ計算を繰り返すので，y の計算にサブルーチンを使うとプログラムの長さが短くなります。

次のプログラムは，x_1 を Y レジスタに，x_2 を X レジスタに入れてから始めるものとします。

主プログラム

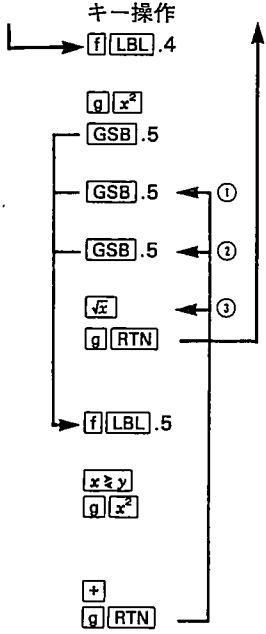
[g] [P/R]	
[f] CLEAR [PRGM]	(プログラム不能。)
000-	
001- [f] [LBL] 9	主プログラムの始め。
002- [g] [RAD]	ラジアン状態。
003- [STO] 0	x_2 を R_0 に記憶。
004- [x≷y]	x_1 を X レジスタ, x_2 を Y レジスタに。
005- [STO] [−] 0	(x_2-x_1) を R_0 に。
006- [GSB] .3	x_1 を X レジスタに入れたままサブルーチン ".3" に移る。
	サブルーチン ".3" から戻る場所。
007- [CHS]	$-y_1$。
008- [x≷y]	x_2 を X レジスタに。
009- [GSB] .3	サブルーチン ".3" に移る。
	サブルーチン ".3" から戻る場所。
010- [+]	y_2-y_1。
011- [RCL] [÷] 0	R_0 から (x_2-x_1) を呼び出して $(y_2-y_1)/(x_2-x_1)$ を計算する。
012- [g] [RTN]	プログラムの終り (行 000 に戻る)。
SUBROUTINE	
013- [f] [LBL] .3	サブルーチン .3 の始め。
014- [g] [x^2]	x_2。
015- [g] [LSTx]	x を呼び出す。
016- [SIN]	$\sin x$。
017- [−]	$x^2-\sin x$, y の値。
018- [g] [RTN]	主プログラムの [GSB] の次の行に戻る。

次の x_1 と x_2 について傾斜を計算してください。0.52 と 1.25, −1 と 1, 0.81 と 0.98。数字ラベルの付いたルーチンを使うときには [f] 9 でなくて [GSB] 9 と押すのを忘れないでください。
答 1.1507, −0.8415, 1.1652

内側のサブルーチンの例 次のラベル".4"からのサブルーチンは大きなプログラムの一部として式 $\sqrt{x^2+y^2+z^2+t^2}$ の値を計算するものです。このサブルーチンはそれぞれの二乗を計算するためにラベル".5"

の付いた別のサブルーチン（内側のサブルーチン）を使っています。

このプログラムは数値 t, z, y, x をそれぞれ T, Z, Y と X レジスタに入れてから実行します。



主サブルーチンの始め。

x^2。

y^2 と x^2+y^2 を計算する。

z^2 と $x^2+y^2+z^2$ を計算する。

t^2 と $x^2+y^2+z^2+t^2$ を計算する。

$\sqrt{x^2+y^2+z^2+t^2}$。
主サブルーチンの終りで主プログラムに戻る。

内側のサブルーチンの始め。

二乗を計算してそれまでの二乗和に足す。

内側のサブルーチンの終りで主サブルーチンに戻る。

$x=4.3$, $y=7.9$, $z=1.3$, $t=8.0$ として，このサブルーチンを（内側のサブルーチンも一緒に）単独に実行するために GSB .4 を押すと答は 12.1074 になります。

詳細説明

サブルーチンのリターン（RTN）

リターン条件付きというのは，GSB 命令の後に出会う RTN 命令で行 000 に戻るのでなくて，その GSB の次の行に戻るという意味です。それでサブルーチンはプログラム中のどの位置にでもおけて便利に使えるのです。そしていつでもジャンプした行の次に実行が戻ります。GSB ジャンプと GTO ジャンプのただ一つの違いは RTN のあとの実行が移る場所です。

多重のサブルーチン

8 重以上の内側のサブルーチンを使うと，8 重目の GSB 命令に出会ったときに計算機が実行を停止して Error 5 を表示します。

内側のないサブルーチンや内側のあるサブルーチンの組数には制限がありません（メモリーの大きさの制限は別です）。

第 10 章

I レジスタとループ制御

I レジスタ（R_I）は HP–15C の高級なプログラム手法用の有力な道具で，I は index（指標とか目印の意）の頭文字です。I レジスタはデータの記憶と呼び出しのほかに，レジスタ内の数値を使って次のことが出来ます。

- ループの回数を数えてループを制御する。
- $R_{.9}$(R_{19}) より後のレジスタも含めて記憶用レジスタを間接的に指定する。
- プログラムのラベルと同様に行番号にも間接的にジャンプする。
- 表示形式を間接的に指定する。
- フラグ操作を間接的に制御する。

[I] と [(i)] キー

I レジスタを使った直接と間接的なデータ記憶

I レジスタはデータの記憶レジスタですが，[I] キーを使って直接的にも，[(i)] キーを使って間接的にも取り扱えます*。その違いは重要なので注意してください。

[I]

[I] 機能は I レジスタ内の数値を使う。

[(i)]

[(i)] 機能は他の記憶レジスタの番号を指定するために I レジスタに入っている数値の整数部分の絶対値を使う。これを間接指定といいます。

---

*行列演算と複素数計算も [I] と [(i)] キーを使いますが，その目的が違います。こちらについては第 11，12 章を見てください。

Iレジスタを使ったプログラムの間接制御

[I] キーはレジスタの間接指定よりもそれ以外のプログラムの間接制御に使います。プログラムの間接ジャンプ，表示の間接指定やフラグ間接制御には [(i)] キーでなく [I] キーを使います。

プログラムのループの制御

HP-15C はどの記憶レジスタ（R_0 から R_9 まで，$R_{.0}$ から $R_{.9}$ まで，またはIレジスタ R_I）を使っても，プログラムのループの計数と制御ができます。ループは [(i)] を使って間接に制御することも可能です。

仕 組 み

[I] と [(i)] は別の前操作キーを使うときには直前の [f] キー操作が省略できます（78ページで説明しました）。

Iレジスタの記憶と呼び出し

直接 [STO] [I] と [RCL] [I]。Xレジスタと Iレジスタ間の数値の記憶と呼び出しは他の記憶レジスタのときと同様に操作します(42ページ)。

間接 [STO]（または [RCL]）[(i)] はIレジスタに入れてある数値（0から65まで）の整数部分で指定する記憶レジスタに記憶（または呼び出し）をします。下表（次ページに続く）の通りです。

間接指定

R_I の内容	[(i)] で指定するレジスタ	[GTO] [I] または [GSB] [I] で下記ラベルにジャンプ*
± 0	R_0	[f] [LBL] 0
⋮	⋮	⋮
9	R_9	[f] [LBL] 9
10	$R_{.0}$	〃 〃 .0
11	$R_{.1}$	〃 〃 .1
⋮	⋮	⋮
19	$R_{.9}$	[f] [LBL] .9
20	R_{20}	〃 〃 [A]

* $R_I \geq 0$ に限る。

次ページに続く。

間接指定

R_I の内容	(i) で指定するレジスタ	GTO I または GSB I で下記ラベルにジャンプ*
21	R_{21}	f LBL B
22	R_{22}	〃 〃 C
23	R_{23}	〃 〃 D
24	R_{24}	〃 〃 E
⋮	⋮	—
65	R_{65}	—

*$R_I \geq 0$ に限る。

Iレジスタの四則演算

直接 STO または RCL { +, −, ×, ÷ } I Iレジスタの直接演算記憶や直接呼び出し演算は他の記憶レジスタと同様に実行できます（43ページ）。

間接 STO または RCL { +, −, ×, ÷ } (i) Iレジスタ内の数値（0から65まで）の整数部分で指定する記憶レジスタの内容との直接演算記憶や直接呼び出し演算をします。上表を見てください。

Xレジスタとの交換

直接 f x≷ I はXレジスタとIレジスタ間でその内容を交換します。（レジスタ0から.9までの x≷ n と同様です。）

間接 f x≷ (i) はIレジスタに内の数値（0から65まで）の整数部分で指定する記憶レジスタとXレジスタ間でその内容を交換します。上表を見てください。

I を使った間接ジャンプ

I キー（(i)キーではありません）は間接ジャンプ（GTO I）や間接サブルーチンジャンプ（GSB I）に使えます。（R_I 内の数値の整数部分だけを使います。）（(i)は記憶レジスタの間接指定だけに使います。）

ラベルへ R_I の値が正なら GTO I と GSB I はIレジスタ内の数値に相当するラベルにジャンプできます（前表を見てください）。

例えば，Iレジスタに 20.00500 を入れてあると，GTO I 命令でプログラムの実行が f LBL A に移ります。108ページの表を見てください。

行番号へ R_I の値が負なら GTO I は R_I の値の整数部分の絶対値に相当する行番号にジャンプします。

例えば，R_I に -20.00500 を入れてあると，GTO I 命令でプログラムの実行が行 020 に移ります。

注 R_I の値が負だと GSB I は使えません。

I を使ったフラグの間接制御

SF I，CF I または F? I は R_I 内の数値の整数部分で指定したフラグ（0から9まで）のセット，クリアまたは判断をします。

I を使った表示の間接指定

f FIX I，f SCI I と f ENG I は0から9までの数 n として R_I に入れた数（整数部だけ）を使って普通と同じように表示の形式を指定できます。*

カウンタを使ったループ制御 ISG と DSE

ISG と DSE 機能は指定のレジスタ内のループ制御数を使ってそれを変えながらループ計算を制御します。プログラムの実行（次の1行を飛び越すかどうか）はその数によって決まります。

キー操作は f {ISG, DSE} レジスタ番号です。この番号は0から9，.0から.9，Iまたは (i) のどれかです。

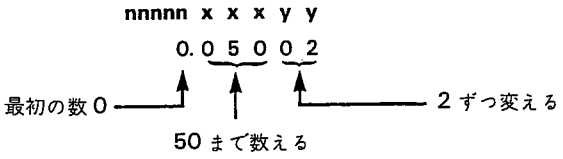
ループ制御数 ループ制御数の形式は次の通りです。

	±nnnnn	はカウンタのそのときの値，
nnnnn.xxxyy，ここで	xxx	は規準（目標）値，
	yy	は増分または減分値。

---

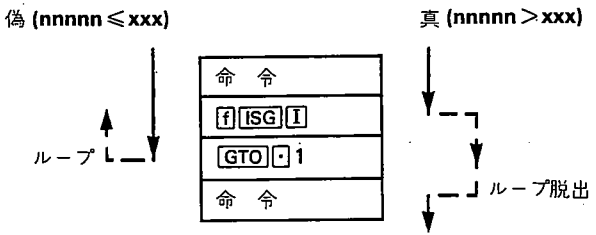
* f (第14章) を使うときだけは別です。

たとえば，ある記憶レジスタ内の数値0.05002を次のように解釈します。

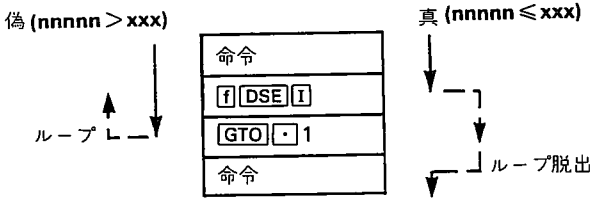


ISG と DSE 操作　プログラム走行中に ISG か DSE 命令に出会うたびに nnnnn (ループ制御数の整数部分) を増やすか減らして，ループ回数を調べながら繰り返します。この命令は nnnnn と前もって指定した目標値 xxx を比較して，ループ用の数 (nnnnn) が規準値 (xxx) より大きい (ISG のとき) かまたは以下 (DSE のとき) なら次の行だけを飛び越してループから脱出します。nnnnn の増加数または減少数は yy で指定します。

この機能を使うときの原則は普通の条件判断と逆に“真なら飛び越し”です。



ISG の実行ごとにそれまでの nnnnn.xxxyy は，nnnnn が nnnnn＋yy に増加し，これと xxx と比較して，増加後の整数部分が nnnnn＞xxx なら次のプログラム行だけを飛び越す。このようにして nnnnn が xxx より大きくなったときにループから脱出します。



DSE の実行ごとにそれまでの nnnnn.xxxyy は，nnnnn が nnnnn−yy に減少し，これと xxx と比較して，減少後の整数部分が nnnnn≦xxx なら次のプログラム行だけを飛び越す。このようにして nnnnn が xxx 以下になったときにループから脱出します。

例えば，ループを繰り返すごとに制御数が下表のように変わります。

繰り返し回数

操 作	0	1	2	3	4
ISG	0.00602	2.00602	4.00602	6.00602	8.00602 (次の行を飛び越す)
DSE	6.00002	4.00002	2.00002	0.00002 (次の行を飛びこす)	

例

レジスタ操作の例

記憶と呼び出し

キー操作	表 示	
f CLEAR REG		記憶レジスタ全部をクリア。
12.3456	12.3456	
STO I	12.3456	R_I に記憶。
7 √x	2.6458	
STO (i)	2.6458	R_I＝12.3456 なので，間接指定で R.₂ に記憶。
RCL I	12.3456	R_I の内容を呼び出す。

キー操作	表　示	
RCL (i)	2.6458	$R_{.2}$の内容を間接に呼び出す。
f x≷ .2	2.6458	点検，同じ内容かどうか$R_{.2}$を直接指定して呼び出す。

Xレジスタとの交換

キー操作	表　示	
f x≷ I	12.3456	R_IとXレジスタの内容の交換。
RCL I	2.6458	現在のR_Iの内容。
f x≷ (i)	0.0000	R_2の内容（これは0）とXとの交換。
RCL (i)	2.6458	
f x≷ 2	2.6458	点検，R_2を直接呼び出す。

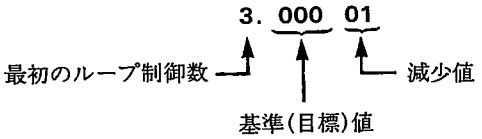
記憶レジスタの直接四則演算

キー操作	表　示	
10 STO + I	10.0000	10をR_Iに足す。
RCL I	12.6458	R_Iの新しい内容（＝旧値＋10）。
g π STO ÷ (i)	3.1416	$R_{.2}$の内容をπで割る。
RCL (i)	0.8422	$R_{.2}$の新しい内容。
f x≷ .2	0.8422	点検，$R_{.2}$を直接呼び出す。

DSE を使ったループ制御の例

残留放射能を計算するためにループを使った第8章のプログラム（94ページ）を思い出してください。このプログラムは計算結果が計算終了値（50）を越えたらループを脱出するために条件判断（$x \geqq y$?）を使いました。本章でもう一つのループ制御（ループ用のレジスタを決めてISG か DSE 機能を使って制御する）方法を説明しました。

そこで元の残留放射能のプログラムを改訂してみましょう。今度は計算終了値を求める代りにループを3回実行するようにします。R_2にループ制御数 3.00001 を入れて [DSE] を使います。



プログラムを次のように変更します（プログラムがメモリー中に残っているものとします)。ループ用のレジスタをR_2に，行番号をIレジスタに記憶することにします。

キー操作	表　示	
[g] [P/R]	000–	プログラム入力状態。
[GTO] [CHS] 013	013–43,30, 9	ループの判断の行。
[←] [←]	011–　42 31	行013と012を削除。
[f] [DSE] 2	012–42, 5, 2	ループ制御機能を追加（レジスタはR_2に記憶)。
[GTO] [I]	013–　22 25	目的の行番号(015)へ。

これでR_2が0になると，行013を飛び越して014の [RTN] 命令に行くと [RTN] 命令なので，プログラムが終了します。ループ制御数が0まで減少しないときは，行013を実行します。ここで行015にジャンプしてプログラムのループを続けます。

プログラム走行のために，t_1(最初の日)をR_0に，N_0(放射能の初期量)をR_1に，ループ制御数をR_2に，そしてジャンプ先の行番号をIレジスタに入れます。

キー操作	表　示	
[g] [P/R]		計算状態。
2 [STO] 0	2.0000	t_{1_0}。
100 [STO] 1	100.0000	N_0。
3.00001 [STO] 2	3.0000	ループ制御数。（この命令はプログラム中に入れることもできます。)

キー操作	表　示	
15 CHS STO I	-15.0000	ジャンプ先の行番号。
f A	2.0000	プログラム走行開始，ループ制御数 =3。
	84.0896	
	5.0000	ループ制御数 =2。
	64.8420	
	8.0000	ループ制御数 =1。
	50.0000	
	50.0000	ループ制御数 =0，プログラム終了。

表示指定の例

次のプログラムは一時停止のときに FIX 表式で小数点以下の桁数を変えて表示する例です。ここでは桁数指定を自動的に変えるために DSE 命令を使いました。

キー操作	
g P/R	
f CLEAR PRGM	
f LBL B	
9	nnnnn=9。ここでは xxx=0，指定を省略すると yy=0 でなくて yy=1。
STO I	
f LBL 0	
f FIX I	
RCL I	
f PSE	nnnnn のそのときの値を表示。
f DSE I	R_I の数を減らして判断する。nnnnn≦目標値なら次の 1 行を飛び越す。
GTO 0	nnnnn＞目標値 (0) ならループを継続。
g TEST 1	表示の値が 0 より大きいかを判断する，nnnnn が 0 になっても表示はまだ 1.0 なのでループを続ける。
GTO 0	
g RTN	

HP-15Cで可能な小数点以下の桁数指定全部を表示してみましょう。

キー操作	表　示	
[g][P/R]		計算状態。
[f][B]	9.000000000	
	8.00000000	
	7.0000000	
	6.000000	
	5.00000	
	4.0000	
	3.000	
	2.00	
	1.0	
	0.	[f][PSE] 命令での表示。
	0.	プログラムが終了したときの表示。

詳細説明

Iレジスタの内容

Iレジスタに記憶している数値は次の3種の方法で利用できます。

- [I] を別の記憶レジスタと同様に使う。R_I の数値はそのまま記憶，呼び出し，交換，加算などの操作ができる。
- [I] を制御数として使う。R_I に入れた数の整数部分の絶対値を小数部分と無関係に使えます。[I]を使う間接ジャンプ，フラグ制御，表示指定にはこの値だけを使います。ループ制御には小数部分も使いますが，整数部分とは別に扱います*
- [(i)] を使って他の記憶レジスタの内容を利用する。[(i)] キーは108，109ページの表に示したように間接指定に使います。(もちろん上に説明した方法で，第2のレジスタをループ制御数として使うこともできます。)

*間接ループ制御に使うI以外の記憶レジスタの数値についても同じです。

ISG と DSE

ループ制御の目的で使うときには記憶した制御数の整数部分（カウンタ値）は5桁まで可能です(nnnnn. xxxyy)。カウンタ値(nnnnn)部分を指定しないときは0になります。

制御数の小数部分の xxx は3桁の数として指定します。(例えば5なら必ず 005 とします。) xxx 部分を指定しなければ0です。ISG または DSE 命令に出会うと，15C の内部で nnnnn は増加または減少の最終値 xxx と比較します。

yy は2桁の数として指定します。yy が0では不都合なので，指定しないか 00 を指定すると，yy の値が自動的に 01 になります。nnnnn の値はループが ISG または DSE を通過するたびに yy の値だけ増加または減少します。yy も xxx も参照用の値なので，ループの実行中に変化することはありません。

表示の間接指定

Iレジスタを使って手動で（つまりキー操作だけで）表示形式の指定もできますが，この機能はふつうプログラム作成のときに活用します。f 機能を使うときにはこの性能が非常に役に立ち，表示の桁数を指定することによって計算の正確さが決まるからです（第14章で説明します)。

表示には一定の制限があることに注意してください。表示指定機能は表示を丸める小数点以下の桁数を変えるだけの働きだと覚えてください。計算機はいつでもメモリー中に10桁の仮数部と2桁の指数部を持った数値表現で数を記憶しています。

Iレジスタの数の整数部で表示の四捨五入をする小数点以下の桁数を指定します。0より小さい数は0（FIX 形式のときは小数点以下の表示桁数が0)，また9を越えるときは9（FIX 形式のときは小数点以下9桁表示）と見なします*。

---

* SCI と ENG 表示方式では最高で仮数部7桁，指数部2桁を表示します。しかし，表示指定数が8~9のときは，四捨五入する小数点以下の桁位置が変わります。(58—59 ページを見てください)。

例外は [f] 機能のときで，R_I に入れる表示指定用の数の範囲は −6〜+9 です。(これは 247 ページの脚注で説明します。) 負数は表示形式に関係ありませんが，この [f] 機能のときの正確さに関係します。

第3部

高等数学用機能

第 11 章

複素数計算

HP–15C は複素数の計算も可能です。普通は次のように表記します。

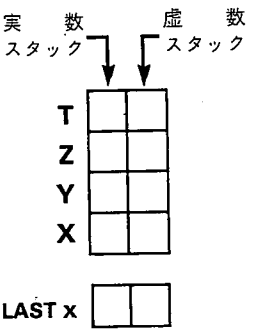
$$a+ib \text{ (または } a+bi\text{)}$$

ここで　a は複素数の実数部,
b は複素数の虚数部,
$i=\sqrt{-1}$ です。ただし，電気関係では i の代わりに j を使っています。

HP–15C の複素数計算の素晴らしさは，一たん複素数をキー入力すれば，大部分の計算が実数と同様に実行できることです。

複素数スタックと複素数計算状態

複素数の計算は二つの平行な 4 段スタック(それに二つの LAST x レジスタ)を使って実行します。平行なスタックの片方を実数スタックと呼び，計算する複素数の実数部用です。(これは普通の計算のときのスタックと同じです。)もう一方のスタックを虚数スタックと呼び，計算する複素数の虚数部用です。



複素数スタックの作成

複素数計算状態にすると（付録 C で説明するように 5 個の記憶用レジスタを転用して）虚数スタックを自動的に作り，計算機が複素数計算状態でないときには虚数スタックはありません。

次のどちらかで複素数計算状態になります。

1) [f][I] または [f][Re≷Im] を実行する。

2) 複素数計算状態フラグのフラグ 8 をセットする。

計算機が複素数計算状態のときには，表示に **C** の文字が見えます。この **C** は英語の虚数の頭文字です。この表示でフラグ 8 のセットと複素数スタックがあることが分かります。計算機が複素数計算状態かどうかに関係なく，表示している数値は実数の X レジスタ内の数値です。

> 注　複素数計算状態（**C** の文字が見える）のときには，HP-15C はどの三角関数もラジアン単位で計算します。三角関数の角度単位の表示（**RAD**, **GRAD**, 度のときは無表示）は [→R] と [→P] の二つの機能にだけ有効です（134 ページで説明します）。

複素数計算状態の解除

複素数計算状態ではメモリー用レジスタ 5 個をスタックに転用しているので，プログラム用にたくさんのメモリーが必要なときや別の高等数学用機能を実数範囲内だけで計算するときには複素数計算状態を解除すると，その分のメモリーが自由に使えます。

複素数計算状態を解除するには，フラグ 8 をクリアします（キー操作は [g][CF] 8）。表示から **C** の文字が消えます。

複素数計算状態は不揮発性メモリーの内容を消去（63 ページで説明しました）しても解除できます。どちらにしても，複素数計算状態を解除すると虚数スタックとそこに記憶していた虚数は消滅します。

複素数とスタック

複素数の入力

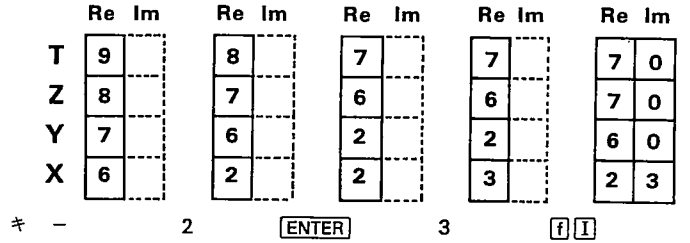
複素数の入力は次のようにします。

1. 実数部をキー入力して表示する。
2. [ENTER] を押す。
3. 虚数部をキー入力して表示する。
4. [f][I] を押す。（まだ複素数計算状態になっていなければ，この操作で虚数スタックを作って **C** の文字が見えるようになります。）

例 2+3i に 4+5i を足してください。(この操作はキー操作手順の後でスタックの図を使って説明します。)

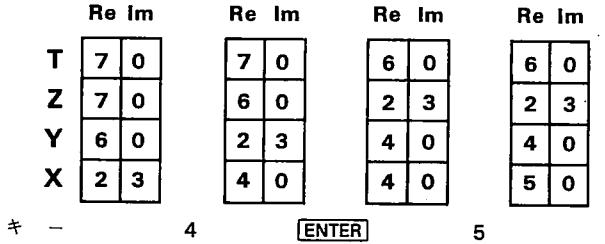
キー操作	表 示	
f FIX 4		
2 ENTER	2.0000	第1の数の実数部を(実数)Yレジスタに入れる。
3	3	第1の数の虚数部を(実数)Xレジスタに入れる。
f I	2.0000	複素数スタックを作って,3を虚数Xレジスタに移し,2を実数Xレジスタに下げる。
4 ENTER	4.0000	第2の数の実数部を(実数)Yレジスタに入れる。
5	5	第2の数の虚数部を(実数)Xレジスタに入れる。
f I	4.0000	5を実数Xレジスタから虚数Xレジスタにコピーし,4を実数Yレジスタから実数Xレジスタにコピーして,スタックを下げる。
+	6.0000	和の実数部。
f (i) (押したまま)	8.0000	(i)キーを押している間は和の虚数部を表示。(これは数値入力の区切りにもなります。)
(放す)	6.0000	

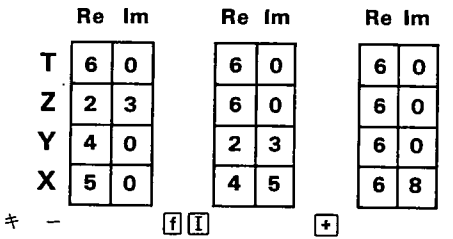
この操作のときの実数と虚数スタックの移動を次図で説明します。(スタック・レジスタには前の計算結果として図示の数が入っているものと仮定します。)次図で実数スタックの右側に示した虚数スタックは f I を押すまでは無いものとします。(またスタックの網点の部分は,次の数値をキー入力したり,レジスタから呼び出すと書き替わる数の意味です。)



f I を実行すると全スタックが下降して，Tレジスタにコピーが残り，実数Xレジスタの内容は虚数Xレジスタに移ります。

第2の複素数を入力すると，スタックは下図のように移動します。ENTER で両方のスタックが上昇するので気を付けてください。





複素数を入力する別の方法は虚数部を先に入力して，次に Re≷Im と ← を使うことです。この方法は127ページの ← を使う複素数の入力の部分で説明します。

複素数計算状態のときのスタック上昇

虚数スタックのスタック上昇は実数スタックの場合と同じです（実数スタックは複素数計算状態かどうかに関係なく同じように移動します）。実数スタック上昇が可能，可能でないまたは無関係な機能は虚数スタックの上昇でも同じように可能，可能でないまたは無関係です。（スタック移動は第3章と付録Bで説明しました。）

更に，[←]と[CLx]以外のスタック上昇に関係ある機能は次の数を入力したとき虚数のXレジスタを0にします。つまりこれらの機能を使った後で次の数をキー入力するか呼び出すと虚数のXレジスタに0が入ります。前ページのスタック移動図を見てください。この性能があるので複素数計算状態でないときのキー操作と同じように計算機が操作できる訳です。*

実数と虚数スタックの操作

[Re≷Im]（実数と虚数の入れ換え） [f][Re≷Im]を押すと実数と虚数のXレジスタの内容を交換して，数の実数部と虚数部が入れ替わります。Y，Z，Tレジスタは関係ありません。[f][Re≷Im]を2回押すと元の数の状態に戻ります。

複素数計算状態になっていないときには[Re≷Im]で複素数計算状態になります。

虚数Xレジスタの一時的表示 [f][(i)]を押すと実際に実数部と虚数部を切り換えないでXレジスタにある虚数部を一時的に表示します。キーを押している間は表示し続けます。

符号の変更

複素数計算状態では[CHS]機能は実数Xレジスタの数値の符号だけを変えて虚数Xレジスタの数値の符号は変えません。それで実数部と虚数部の符号を別々に変えられます。負の実数部または虚数部をキー入力するのには，その部の数値を入れてから符号を変えます。

既にXレジスタに入れてある複素数の実・虚数部とも符号を変えたいときには，虚数計算状態ではないときのように[CHS]を押すだけではだめです。その代りに次のどちらかの操作をします。

---

*本章（134ページ）で説明するように[→P]と[→R]機能は例外です。

- −1 を掛ける。
- スタックの X レジスタ以外を変えたくないときは [CHS] [f] [Re≷Im] [CHS] [f] [Re≷Im] と押す。

複素数の実・虚数部のどちらか片方だけの符号を変えるには次のようにします。

- 実数部だけの符号を変えるときは [CHS] を押す。
- 虚数部の符号だけを変えて共役複素数を作るときは [f] [Re≷Im] [CHS] [f] [Re≷Im] を押す。

複素数のクリア

その内に複素数をクリアする必要があると思います。片方ずつクリアして，実・虚数部両方の数値を書き変えることも可能です（[←] と [CLx] はスタック上昇可能でない機能だからです）。

実数 X レジスタのクリア　計算機が複素数計算状態のときに [←]（または [g] [CLx]）を押すと実数 X レジスタの数値だけをクリアしますが，虚数 X レジスタの数値はクリアしません。

例　6+8i を 7+8i に変えて先に Y レジスタに入れてある数値から引いてみます。(X レジスタの虚数部を見るときは [f] [Re≷Im] か [f] [(i)] を使います。) a, b, c, d は別の複素数の数値です。

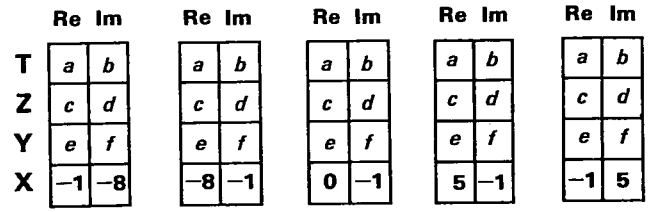
	Re	Im	Re	Im	Re	Im	Re	Im
T	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*
Z	*c*	*d*	*c*	*d*	*c*	*d*	*a*	*b*
Y	6	0	6	0	6	0	*c*	*d*
X	6	8	0	8	7	8	−1	−8
キー			[←]		7		[−]（または別の操作）	

クリア操作はスタック上昇が可能でない(上で説明しました)操作なので，入力した次の数でクリアした数値を書き換えます。実数部だけを 0 に変えたいときには，クリアした後で [ENTER] か任意の別の機能で数字入力を区切ってください（こうしないと次に入力する数値で 0 を書き換えます)。虚数部は変化しません。その後は任意の計算操作

を続けられます。

虚数Xレジスタのクリア 虚数のXレジスタにある数をクリアするには [f] [Re≷Im] を押してから [←] を押します。その0や新しくキー入力した数を虚数Xレジスタに戻すにはもう一度 [f] [Re≷Im] を押します。

例 $-1-8i$ を $-1+5i$ に書き換える。

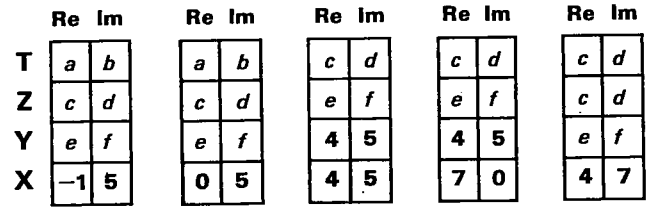


キー [Re≷Im] [←] 5 [Re≷Im]

(任意の操作を続けられる)

実数と虚数Xレジスタのクリア Xレジスタにある実・虚数部の両方をクリアしたり書き換えるには [←] を押すだけで,そのときスタックは上昇可能でないので新しい数をキー入力します。(Xレジスタを0にしたいときは0をキー入力します。)もう一つの方法は,新しい数が純実数 (0+0iも含む) なら, [R↓] を押してから0または新しい実数をキー入力するだけで, 旧複素数を簡単にクリアしたり書き換えできます。

例 $-1+5i$ を $4+7i$ に書き換える。



キー [←] 4 [ENTER] 7 [f] [I]

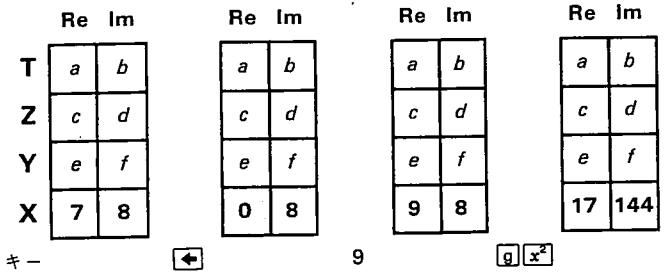
(任意の操作を続けられる)

[←]を使った複素数の入力　複素数の入力（またはクリア）の別の方法として，クリア機能の [←] か [CLx] を [f] [Re≷Im] と組み合せて使えます。この方法を使うと，Y，Z，Tスタックを変えないで，Xレジスタだけに複素数を入れられます。（[←] と [CLx] はスタック上昇可能でないのでこれが可能です。）虚数スタックがまだ無くても，[Re≷Im] を押すと同時に虚数スタックを作ります。

例　スタックを動かさないで9+8iを入れてその二乗を求めてください。

キー操作	表　示	
([←])	(0.0000)	次の数字 (8) をキー入力したときのスタック上昇を防止する。もしXレジスタの数値は保存し，Tレジスタの数値が消えてもよいときはこの操作を省略できます。
8	8	虚数部を先にキー入力する。
[f] [Re≷Im]	7.0000	実数部を表示して，複素数計算状態になる。
[←]	0.0000	スタック上昇可能でなくなる。(この操作をしないと [Re≷Im] の次なのでスタック上昇可能になる。)
9	9	実数部の入力（数字入力を区切っていない）。
[g] [x^2]	17.0000	実数部 (9^2-8^2)。
[f] [(i)] (押したまま)	144.0000	虚数部 ($2\times9\times8$)。
(放す)	17.0000	

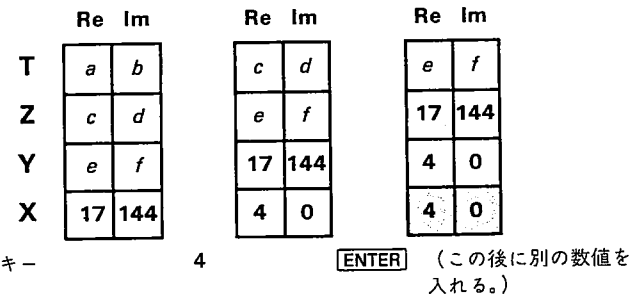
	Re	Im	Re	Im	Re	Im	Re	Im
T	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*
Z	*c*	*d*	*c*	*d*	*c*	*d*	*c*	*d*
Y	*e*	*f*	*e*	*f*	*e*	*f*	*e*	*f*
X	4	7	0	7	8	7	7	8
キー			[←]		8		[f] [Re≷Im]	



実数の入力

これまでに複素数を入力する二つの方法を説明しました。実数を入力するにはもっと簡単な方法があって，計算機が複素数計算状態でないときのようにただ目的の数をキー入力して（または呼び出して）表示します。こうすれば，虚数Xレジスタに0が入ります（124ページで説明したように，直前の操作が [←] か [CLx] でないときに限ります）。

この操作中の実・虚数スタックの移動は下図の通りです。（直前に押したキーが [←] か [CLx] でなくてスタックの内容は前例のままと仮定します。）



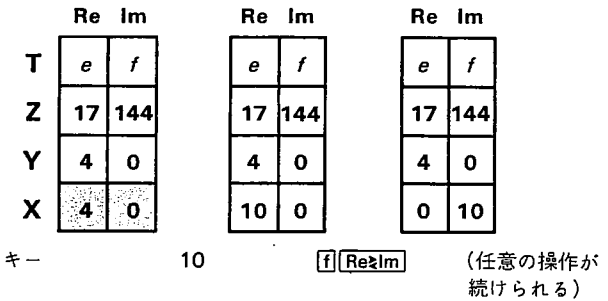
純虚数の入力

既に複素数計算状態になっていればX レジスタに純虚数を入れる簡便法があります。虚数をキー入力して [f] [Re≷Im] を押すだけです。

例　0+10i を入力してください（最後に実行した機能が [←] か [CLx] でないとします）。

キー操作	表　示	
10	10	実数 X レジスタの数値を表示しているときに 10 をキー入力すると虚数 X レジスタに 0 が入る。
[f] [Re≷Im]	0.0000	実数と虚数 X レジスタの数を交換する。表示している実数 X レジスタの数が 0 ですが，純虚数だからこれで良い。

この操作中の実・虚数スタックの移動は下図の通りです。（スタック・レジスタには前例の計算結果が残っているものとします。）



[f][Re≷Im] を押すと実・虚数Xレジスタの数値が入れ換わるだけでそれ以外のスタック・レジスタの数値は変わりません。

複素数の記憶と呼び出し

[STO] と [RCL] 機能は実数Xレジスタだけに働くだけなので，複素数の虚数部は別個に記憶したり呼び出す必要があります。このキー操作をプログラムの一部として組み込めば自動的に実行することも可能です。*

Xレジスタの複素数 a+ib を R_1 と R_2 に記憶するには次のようなキー操作をします。

[STO] 1 [f][Re≷Im] [STO]

もしスタックを元の状態に戻したければその後に [f][Re≷Im] を押します。a+ib を R_1 と R_2 から呼び出すときには次のキー操作をします。

[RCL] 1 [RCL] 2 [f][I]

Y, Z, Tレジスタを乱したくなければ次のキー操作で呼び出します。

[RCL] 2 [f][Re≷Im] [←] [RCL] 1.

(プログラム入力状態では [←] の代わりに [g][CLx] を使います。)

複素数の計算

実数を取り扱うほとんど全部の機能は複素数計算状態であってもなくても†，実数だけなら同じ答えになります。言い換えると，複素数計算状態でも実数計算は普通のときと同じです。

本章（複素数計算）のこれ以降で取り上げない機能はどれも虚数スタックを無視します。

---

*第12章で説明する HP-15C の行列の機能を使うと，複素数の記憶や呼び出しがもっと簡単にできます。行列の大きさ指定を n×2 にすると，n 個の複素数を行列の行として記憶できます。(この方法は HP-15C Advanced Functions Handbook の第3章の Applications の部分で説明します。)

†例外は [→P] と [→R] で，これは複素数計算状態では複素数の直交座標系と極座標系との変換 (134ページ) のために別の使い方をします。

単項演算関数

下記の関数はXレジスタにある数の実数部と虚数部の両方を使って複素数計算をして，結果の実数部と虚数部をそれぞれのXレジスタに入れます。

[√x] [x²] [LN] [LOG] [1/x] [10ˣ] [eˣ] [ABS] [→P] [→R]

三角関数と双曲線関数とその逆関数はどれもこの同類です*。

[ABS] 関数はXレジスタにある数の絶対値（実数部と虚数部の二乗和の平方根）を求め，絶対値の虚数部は0になります。

[→P] は極座標系に，[→R] は直交座標系に複素数を変換し，これについては本章 134 ページで説明します。

三角関数の計算では，そのときの角度単位に関係なく，実・虚数Xレジスタの数値はラジアン単位で表示しているものとして取り扱います。数値を度単位で入れてあるときには三角関数の計算前に [→RAD] を使って数値をラジアンに変換してください。

二項演算関数

下記の関数はXとYレジスタの実・虚数部にある数値を複素数として計算し，その結果をXレジスタの実・虚数部に入れます。複素数計算状態でないときに二項演算関数の実行でスタック下降するのと同様に実・虚数部両方のスタックが下降します。

[+] [−] [×] [÷] [yˣ]

スタック操作機能

下記の機能は計算機が複素数計算状態でないときにスタック操作するのと同様に複素数計算状態のときにも実・虚数部のスタックを同時に

---

*複素三角関数の定義や複素数計算状態で計算する方法の詳しい説明は HP–15C Advanced Functions Handbook を見てください。

操作します。例えば [x≷y] 機能はXとYレジスタにある数の実・虚数部の両方を交換します。

[x≷y] [R↓] [R↑] [ENTER] [LSTx]

条件判断

下記の4種の条件判断は複素数についても同じに働くのでプログラム中にも使えます。[x=0] と [TEST] 0 は(実数と虚数の) Xレジスタにある複素数を 0+0i と比較し, [TEST] 5 と [TEST] 6 は (実数と虚数の) XとYレジスタにある数を比較します。下記の4種以外の全条件判断は虚数部を無視します。

[x=0] [TEST] 0 ($x \neq 0$) [TEST] 5 ($x = y$) [TEST] 6 ($x \neq y$)

複素数演算の例 はしご型電気回路 (ラダー・ネットワーク) の特性インピーダンスは次の式で計算します。

$$Z_0 = \sqrt{\frac{A}{B}}$$

ここでAとBは複素数です。$A = 1.2 + 4.7i$, $B = 2.7 + 3.2i$ のときの Z_0 を計算してください。

キー操作	表　示	
1.2 [ENTER] 4.7 [f] [I]	1.2000	Aを Xレジスタの実・虚数部に入力。
2.7 [ENTER] 3.2 [f] [I]	2.7000	Bを Xレジスタの実・虚数部に入力するとAは Yレジスタの実・虚数部に移動。
[÷]	1.0428	A/B の計算。
[√x]	1.0491	Z_0 を計算して実数部を表示。
[f] [(i)] (押したまま)	0.2406	[(i)] を押している間は Z_0 の虚数部を表示。
(放す)	1.0491	Z_0 の実数部をまた表示。

実数演算結果の複素数

前例では複素数の入力によって（自動的に）複素数計算状態になりました。しかし，$\sqrt{-5}$ のように実数にある種の演算をするときには複素数計算状態になっていることが必要です。（複素数計算状態になっていないときにこのような操作をすると **Error 0**（数学的に不適当な計算）を表示します。）このような関数を実行する前にフラグ8をセットすれば，いつでもスタック内容を乱さないで複素数計算状態になります*。

例　2.404 のアークサイン($\sin^{-1}$)は普通の方法では **Error 0** になります。X レジスタに 2.404 が入れてあるとすれば，複素数の $\sin^{-1}2.404$ は次のように計算します。

キー操作	表　示	
[g] [SF] 8		複素数計算状態にする。
[g] [SIN⁻¹]	**1.5708**	$\sin^{-1}2.404$ の実数部。
[f] [(i)]（押したまま）	**−1.5239**	$\sin^{-1}2.404$ の虚数部。
（放す）	**1.5708**	[(i)] を放すと実数部をまた表示。

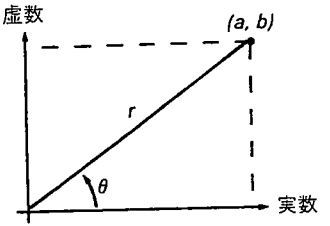
極座標系と直交座標系の座標変換

応用分野によっては複素数を極座標系で表示したり，時には位相記号を使うこともあります。しかし HP−15C では複素数は直交座標系になっているものとして取扱います。そこで，極座標系や位相表示の複素数は複素数計算状態ではその関数計算前に直交座標系に変換する必要があります。

---

* [f] [Re≥Im] を 2 回押しても同じことです。[f] [I] 操作は X と Y レジスタにある 2 数を組み合せて一つの複素数を作るから使えません。

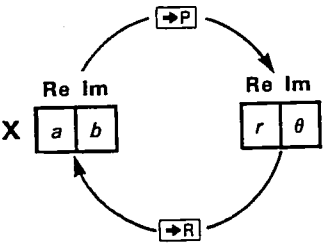
$$a+ib=\begin{cases} r(\cos\theta+i\sin\theta)=re^{i\theta} & \text{極座標系} \\ r\angle\theta & \text{位相表示} \end{cases}$$



[→R] と [→P] は複素数の直交座標系と極座標系の相互変換に使います。複素数計算状態でのこの操作は下記の通りです。

[f][→R] は複素数の極座標（または位相）系を直交座標系に変換し，実数Xレジスタにある絶対値rがaに，虚数Xレジスタにある角度θがbに置き換わる。

[g][→P] は複素数の直交座標系を極座標（または位相）系に変換し，実数Xレジスタにある実数部aがrに，虚数Xレジスタにある虚数部bがθに置き換わる。



そのときの角度単位の影響を受ける複素数計算状態の機能はこの2種類だけです。θの計算に使う角度の単位は表示部の文字（度のときは無表示）通りの角度単位と一致している必要があります。

例　$2(\cos 65^\circ + i\sin 65^\circ) + 3(\cos 40^\circ + i\sin 40^\circ)$ を計算して答を極座標系で表してください。(位相表示の $2\angle 65^\circ + 3\angle 40^\circ$ の計算。)

キー操作	表　示	
g DEG		極座標と直交座標変換の角度単位を度にする。
2 ENTER	2.0000	
65 f I	2.0000	表示部に **C** の文字を表示して，複素数計算状態になった。
f →R	0.8452	極座標系から直交座標系に変換して，実数部 (a) を表示。
3 ENTER	3.0000	
40 f I	3.0000	
f →R	2.2981	極座標系から直交座標系に変換して，実数部 (a) を表示。
+	3.1434	
g →P	4.8863	直交座標系から極座標系に変換して，r を表示。
f (i) (押したまま)	49.9612	θ (度)。
(放す)	4.8863	

練習問題

次の練習問題を解いてみると，HP-15C の複素数計算は実数計算と同様に易しいことが分かると思います。一たん数値を入力すれば，大部分の数学演算は実数のときとすっかり同じ操作手順です。では次の問題をやってみてください。

1. 次式を計算してください。

$$\frac{2i(-8+6i)^3}{(4-2\sqrt{5}\,i)(2-4\sqrt{5}\,i)}$$

キー操作	表 示	
2 [f] [Re≷Im]	0.0000	$2i$。表示は実数部。
8 [CHS] [ENTER]	−8.0000	
6 [f] [I]	−8.0000	$-8+6i$
3 [y^x]	352.0000	$(-8+6i)^3$
[×]	−1,872.0000	$2i(-8+6i)^3$
4 [ENTER]	4.0000	
5 [$\sqrt{x}$]	2.2361	
2 [CHS] [×]	−4.4721	$-2\sqrt{5}$
[f] [I]	4.0000	$4-2\sqrt{5}\,i$
[÷]	−295.4551	$\dfrac{2i(-8+6i)^3}{4-2\sqrt{5}}$
2 [ENTER] 5 [$\sqrt{x}$]	2.2361	
4 [CHS] [×]	−8.9443	
[f] [I]	2.0000	$2-4\sqrt{5}\,i$
[÷]	9.3982	答の実数部。
[f] [(i)]	−35.1344 9.3982	答は $9.3982-35.1344i$。

2. 種々の z のときの関数 $\omega=\dfrac{2z+1}{5z+3}$ を計算するプログラムを作ってください。(ω は等角写像の一種の1次変換です。) $z=1+2i$ のときの ω を計算してください。

   (答 $0.3902+0.0122i$。キー操作の一例は次の通りです。 [f] [LBL] [A] [ENTER] [ENTER] 2 [×] 1 [+] [$x\gtrless y$] 5 [×] 3 [+] [÷] [R/S] [f] [Re≷Im] [g] [RTN]。)

3. 80ページの例を少し改良して，複素数の多項式を計算できるようにしてください。z がある複素数のとき $p(z)=5z^4+2z^3$ の計算にこのプログラムが使えます。

   スタックに $z=7+0i$ を入れ実数のときと同じ結果になることを確かめてください。

   (答 12,691.0000+0.0000i)

   次に $z=1+i$ としてプログラムを実行してください。

   (答 −24.0000+4.0000i)

詳細な解説の紹介

HP–15C Advanced Functions Handbook には HP–15C の各種の関数に複素数を使うときの詳細な説明と技法を紹介してあります。プログラムもあります。主な項目は次の通りです。

- 精度の考察。
- 多価関数の主値。
- 複素線積分。
- 複素ポテンシアル。
- 行列を使った複素数の記憶と呼び出し。
- 複素数の n 乗根の計算。
- 方程式の複素根の求根。
- 複素数計算状態のときの [SOLVE] と [∫] の使用法。

第 12 章

行 列 演 算

HP–15C に行列（マトリクスとも言います）演算機能を組み込んだので，複雑な問題も簡単に解けます。この計算機は 5 個までの行列を A から E までのキーを使って取り扱えるので，A から E までの名前を付けて区別することにします。HP–15C を利用すると個々の行列の大きさを指定し，行列要素の記憶や呼び出し，行列演算（要素が実数でも複素数でも）が可能です。（本章の最後に行列用機能の要点を列記しました。）

一番多く使う行列演算は連立 1 次方程式を解くことです。たとえば次の方程式の根 x_1 と x_2 を求めてみましょう。

$$3.8x_1 + 7.2x_2 = \ \ 16.5$$

$$1.3x_1 - 0.9x_2 = -22.1$$

この一組の方程式は行列で表すと $\mathbf{AX} = \mathbf{B}$ となり，次の通りです。

$$\mathbf{A} = \begin{bmatrix} 3.8 & 7.2 \\ 1.3 & -0.9 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 16.5 \\ -22.1 \end{bmatrix}$$

次のキー操作をやってみると，HP–15C を使うと行列の問題がとても簡単に解けることが分かると思います。（この例で使った行列の操作は後で順々に説明します。）

まず，二つの既知の行列 A と B の大きさを指定し，次にその要素の値を第 1 行から最後の行まで，各行の左列から右への順に入力します。また，行列演算の結果を記憶する行列として行列 C を指名します（C = X）。

キー操作	表　示	
[g] [CF] 8		複素数計算状態の解除。
2 [ENTER] [f] [DIM] [A]	2.0000	行列 A の大きさを 2×2 にする。
[f] [MATRIX] 1	2.0000	利用者優先状態で行列の要素を自動入力する準備。
[f] [USER]	2.0000	(表示部に **USER** の文字が見えるようにする。)
3.8 [STO] [A]	A 1,1	行列 A の第 1 行，第 1 列の表示。(各要素を入力するために文字キーを押すとこのように一時的に表示する。)
	3.8000	a_{11} を記憶。
7.2 [STO] [A]	7.2000	a_{12} を記憶。
1.3 [STO] [A]	1.3000	a_{21} を記憶。
.9 [CHS] [STO] [A]	−0.9000	a_{22} を記憶。
2 [ENTER] 1 [f] [DIM] [B]	1.0000	行列 B の大きさを 2×1 にする。
16.5 [STO] [B]	16.5000	b_{11} を記憶。
22.1 [CHS] [STO] [B]	−22.1000	b_{21} を記憶。
[f] [RESULT] [C]	−22.1000	結果を記憶する行列として C を指名。

行列記号を使うと行列方程式 $\mathbf{AX}=\mathbf{B}$ の解は次式になります。

$$\mathbf{X}=\mathbf{A}^{-1}\mathbf{B}$$

ここで $\mathbf{A}^{-1}$ は A の逆行列です。この演算は行列 B と A の記号を Y と X レジスタに入れて [÷] を押すと実行できます。(記号は行列の名称と大きさを指すものです。) A と B を数値のように扱って答を計算できるのに注意してください。

キー操作	表 示			
RCL MATRIX B	b	2	1	2×1の定数行列Bの記号を呼び出す。
RCL MATRIX A	A	2	2	2×2の係数行列Aの記号をXレジスタに呼び出すと，Bの記号がYレジスタに移動。
÷	running			$A^{-1}B$を計算して行列Cに記憶するまでこれを表示。
	C	2	1	結果行列C（これは2×1行列）の記号。

これで行列Cの要素を呼び出すと行列方程式の解です。（ついでに計算機を利用者優先状態でなくして全行列をクリアします。）

キー操作	表 示	
RCL C	C 1,1	行列Cの第1行第1列と明示。
	−11.2887	$C_{11}(x_1)$の値。
RCL C	8.2496	$C_{21}(x_2)$の値。
f USER	8.2496	利用者優先状態の解除。
f MATRIX 0	8.2496	全行列をクリア。

上の連立方程式の解は $x_1=-11.2887$ と $x_2=8.2496$ になりました。

注 本章の行列演算の説明は皆様が既に行列の理論や演算を知っているものと仮定して進めます。

行列の大きさ

64個までの行列の要素をメモリーに記憶することが可能です。64個の要素全部を一つの行列に使うことも，5個までの行列に分けて使う

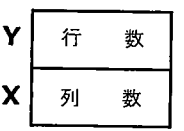
こともできます。例えば，逆行列の計算なら要素が実数だと 8×8 の行列（後で説明するように要素が複素数だと 4×4 の行列*）まで実行できます。

メモリー使用量を節約するために最初は全部の行列が 0×0 になっています。行列の大きさ指定や大きさの変更をすると，それに応じた個数のレジスタを自動的にメモリー中に確保します。行列の大きさを指定したり，ある種の行列演算前に，行列用メモリーに配分するレジスタ数を増加する必要があるかも知れません。付録 C でメモリーの構成，そのときに行列の要素の記憶に使用可能なレジスタ数を知る方法やその個数の増減法を説明します。

行列の大きさの指定

y 行 *x* 列の行列の大きさを指定するには，まずこの数を Y と X レジスタに入れ，次に [f] [DIM] を押し次に行列を指定する文字キーを押します。

1. 行数 (*y*) を表示に入れ，[ENTER] を押してこれを Y レジスタに入れる。
2. 列数 (*x*) を X レジスタにキー入力する。
3. [f] [DIM] を押し次に行列の名前を表す [A] から [E] までの文字キーのどれかを押す。†



---

*本章で説明する行列の機能は実数の行列だけに使えます。(複素数計算状態にしても，行列演算中は虚数スタックを無視します。) しかし 160～173 ページで説明するように，複素数の行列でも実数表現を使って計算可能な 4 種の行列用機能があります。

†文字キーの前に [f] を押す必要はありません。(78 ページのキー操作の省略を見てください。)

例 行列Aの大きさを2×3に設定しましょう。

キー操作	表 示	
2 [ENTER]	2.0000	行数をYレジスタにキー入力。
3	3	列数をXレジスタにキー入力。
[f] [DIM] [A]	3.0000	行列Aの大きさを2×3にする。

行列の大きさの表示

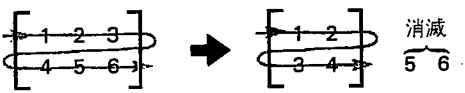
行列の大きさの表示法は2種類あります。

- [RCL] [MATRIX] を押してから行列を表す文字キーを押す。すると左側に行列の名前，右側に行数と列数を表示します。
- [RCL] [DIM] を押してから行列を表す文字キーを押す。するとYレジスタに行数，Xレジスタに列数が入ります。

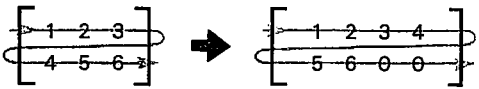
キー操作	表 示	
[RCL] [MATRIX] [B]	b 0 0	行列Bはまだ大きさを指定してないので，0行0列です。
[RCL] [DIM] [A]	3.0000	Aの列数。
[x⇄y]	2.0000	Aの行数。

行列の大きさの変更

行列の要素の値は第1行から最終行まで，各行の左列から右列への順でメモリーに記憶します。行列の大きさが小さくなるように変更すると，必要な値は新しい大きさに配分しなおされ，余った数値は消滅します。例えば，下図左側の2×3行列を2×2行列に大きさを変更すると次のようになります。



行列の大きさが大きくなるように大きさを変更すると，新しい大きさに必要な 0 の値が要素に入ります。例えば，2×3 行列を 2×4 行列に大きさを変更すると次図のようになります。



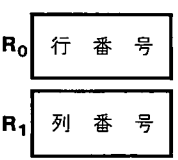
行列演算終了後は行列の要素の記憶に使ったレジスタをプログラム行や別の高等数学用機能に使うために，多分 5 個の行列全部の大きさを 0×0 に戻す必要があると思います。[f] [MATRIX] 0 を押すと，5 個の行列全部を一度に 0×0 に戻せます。(0 [f] [DIM] {[A]～[E]}を押すと，1 個の行列だけを 0×0 に戻せます。)

行列の要素の記憶と呼び出し

HP-15C では行列の要素の記憶と呼び出し方法が 2 種あります。第一は全要素を順々に記憶する（または呼び出す）方法です。第二は要素を別々に取り扱う方法です。

全要素の順次記憶と呼び出し

HP-15C は行列の要素の行番号と列番号用に記憶レジスタ R_0 と R_1 を使います。計算機を利用者優先状態にすると，第 1 行から最後の行まで各行の左列から右列への順に行列の要素を順々に記憶や呼び出すために行番号と列番号が自動的に増加します。



R_0 と R_1 の行と列の番号を第 1 行，第 1 列に合わせるのには [f] [MATRIX] 1 と押します。

行列の要素を順々に記憶または呼び出すには次のように操作します。

1. 行列の大きさ指定が正しいかどうかを確かめる。

2. f MATRIX 1を押す。この操作でR_0とR_1の両方に1が入って第1行，第1列から順々に全要素を取り扱えるようになる。

3. f USERを押して利用者優先状態にする。計算機が利用者優先状態になっていると，次例で分かるように，各要素の記憶や呼び出しごとにR_0の行番号またはR_1の列番号が自動的に1ずつ増加する。

4. 要素の記憶のときは，第1行第1列に記憶する要素の数値をキー入力する。

5. STOまたはRCLを押してから行列を表す文字キーを押す。

6. 行列の全要素について手順4と5を繰り返す。行と列の番号は指定してある行列の大きさ通りに増加します。

STOまたはRCLを押してから行列を表す文字キーを押し続けると，計算機は行列の名前と，これから記憶または呼び出す要素の行番号を左側に列番号を右側に表示します。文字キーを約3秒以上押し続けると，計算機が**null**を表示して，その要素の数値の記憶や呼び出しを実行しないで行や列番号も増加しません。(スタック・レジスタも変わりません。)

行列の最後の要素を扱うと行と列番号は1に戻ります。

例 前に大きさ指定した行列**A**の要素に下記の値を入れてください。(行列**A**が2×3になっていることを確かめてください。)

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}=\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

キー操作	表　示	
f MATRIX 1		R_0 と R_1 の行と列の番号を1にする。(表示は前の結果によって異なります。)
f USER		利用者優先状態にする。
1 STO A	A 1,1	A の第1行，第1列。(A キーを押している間一時的に表示。)
	1.0000	a_{11} の値。
2 STO A	2.0000	a_{12} の値。
3 STO A	3.0000	a_{13} の値。
4 STO A	4.0000	a_{21} の値。
5 STO A	5.0000	a_{22} の値。
6 STO A	6.0000	a_{23} の値。
RCL A	A 1,1	第1行，第1列の要素の呼び出し。(R_0 と R_1 は前の操作で1に戻っている。)
	1.0000	a_{11} の値。
RCL A	2.0000	a_{12} の値。
RCL A	3.0000	a_{13} の値。
RCL A	4.0000	a_{21} の値。
RCL A	5.0000	a_{22} の値。
RCL A	6.0000	a_{23} の値。
f USER	6.0000	利用者優先状態の解除。

行列の個々の要素の確認と変更

HP-15C では行列の特定の要素の確認（呼び出し）と変更（記憶）法が2種類あります。第一の方法は前に説明したのと同様に記憶レジスタ R_0 と R_1 を使いますが，違う点は利用者優先状態になっていなければ行と列の番号が自動的に変化しないことです。第二はスタックを使って行と列の番号を指定する方法です。

R_0 と R_1 を使う方法 行列の特定の要素を処理するために，行番号を R_0 に，列番号を R_1 に記憶します。この番号は(利用者優先状態にしていなければ) 自動的に変化しません。

- 要素の数値を呼び出すときは，(行と列の番号を記憶した後で) [RCL] を押してから行列を表す文字キーを押す。

- その要素に数値を記憶するときは，(行と列の番号を記憶した後で) 数値を X レジスタに入れ [STO] を押してから行列を表す文字キーを押します。

例 前例の行列 A の第2行，第3列の要素に数値9を記憶してください。

キー操作	表 示	
2 [STO] 0	2.0000	行番号を R_0 に記憶。
3 [STO] 1	3.0000	列番号を R_1 に記憶。
9	9	新しい要素の数値を X レジスタにキー入力。
[STO] [A]	A 2,3	A の第2行，第3列。
	9.0000	a_{23} の値。

スタックを使う方法 行列の特定の要素を指定するのにスタック・レジスタを使うことができます。この方法を使うと R_0 と R_1 の数を変える必要はありません。

- 要素の値を呼び出すときは，行番号と列番号を（この順に）スタックに入力します。次に [RCL] [g] を押してから行列を表す文字キーを押します。(行と列の番号はスタックから消えます。)

- 要素に値を記憶するときは，最初に数値，続いて行番号と列番号をスタックに入れます。次に [STO] [g] を押してから行列を表す文字キーを押します。(行と列の番号はスタックから消えて，要素の値が X レジスタに戻ります。)

黄色文字キーの直前に青色の [g] キーを押すのはこの二つの操作のときだけなので注意してください。

例　前例の行列Aの第2行，第1列の要素を呼び出してください。スタック・レジスタを使ってみましょう。

キー操作	表　示	
2 [ENTER] 1	1	行番号をYレジスタに，列番号をXレジスタに入れる。
[RCL] [g] [A]	4.0000	a_{21} の値。

一つの数値の行列の全要素への記憶

一つの数値を一つの行列の全要素に記憶するときは,その数値を表示に入れ，次に [STO] [MATRIX] を押してから行列を表す文字キーを押すだけでよいのです。

行列演算

多くの点で行列演算は数値計算に似ています。数値計算では使用する数値を指定し，また大ていの場合結果を記憶するレジスタも指定します。行列演算のときも同様に，使用する一つまたは二つの行列を指定する必要があります。特定の行列を表すために行列記号を使います。多くの計算では，結果を記憶する行列も指定する必要があります。これを結果行列と言います。

行列演算中は多数の個別計算を繰り返すので，大部分の行列演算の実行中はrunningの文字を点滅表示します。

行列記号

本章の前の方（140ページ）で [RCL] [MATRIX] を押してから行列を表す文字キーを押すと，表示部の左側に行列の名前が,右側に行列の行数その右に列数が見えたと思います。この行列の名前を行列記号と呼ぶことにします。行列記号は数値と全く同様に(つまり，[STO]，[RCL]，[ENTER] などを使って）スタック内やデータ記憶用レジスタとの間で移動できます。行列記号をXレジスタに表示すると，必ずその行列のそのときの大きさも一緒に表示します。

どの行列演算でもどの行列を使うのかを表すために行列記号を使います。本章でこれから説明する行列演算はXレジスタにある記号と

(一部の演算では) Y レジスタにある記号の行列を使って実行します。

2種の行列演算 (行列式の計算と行列方程式 $\mathbf{AX}=\mathbf{B}$ の解法) は X レジスタにある行列の *LU* 分解 (*LU* 因子分解とも言います) の計算も含みます。* *LU* 分解した行列はその記号を表示すると行列名の右に横線が二つ並びます。(*LU* 型の行列の利用法は 160 ページを見てください。)

結果行列

本章で説明する大部分の演算では演算結果を記憶する行列を指名する必要があります。この行列を結果行列と呼びます。

行列演算の中には結果行列を使わないのや影響しないのがあります。(演算の説明のときに明記します。) このような演算は元の行列を演算結果で置き換える (結果が転置行列のように一つの行列のとき) か X レジスタに数値を入れます (結果が行ノルムのように一つの数値のとき)。

結果行列が必要な演算を実行する前には, 結果行列の指名が必要です。指名は [f] [RESULT] を押してから行列を表す文字キーを押します。(予定している結果行列の記号が既に X レジスタに入っていたら, この代わりに [STO] [RESULT] と押すだけでよい。) 指名した行列は別の行列として指名するまでは結果行列のままです。† 結果行列の記号を見るには [RCL] [RESULT] と押します。

結果行列に影響がある演算を実行すると, その行列が正しい大きさになるように自動的に大きさが変わります。もしこの大きさの変更で行列用メモリーに使用可能な要素数より余分の要素が必要なときは (全部で5個の行列に使えるメモリー数は最大64個です), 演算が実行不

---

*ある行列 A の *LU* 分解とは, 左下側の三角形行列 L と右上側の三角形行列 U からなる行列で, その積 *LU* が行列 A に等しいものです (このとき幾つかの行の交換が必要なことがあります)。HP-15C Advanced Functions Handbook で *LU* 分解を詳しく説明します。

†不揮発性メモリーの内容を消すと自動的に行列 A を結果行列に指定します。

能です。演算用の行列の一つを結果行列に指定するとこの制限を免れることが可能です。(しかし，結果行列が演算用行列のどちらかと同じでは不可能な演算もあって，これについてはその演算の説明のときに明記します。)

演算結果を結果行列に記憶するための行列演算のキーを押し続けると，結果行列の記号を表示します。そのキーを約 3 秒以内に放すと，その演算を実行して，結果行列の記号が X レジスタに入ります。演算用のキーをもっと押し続けていると，演算を実行しないで **null** を表示します。

行列のコピー

ある行列の要素を別の行列の対応する要素にコピーするときには，[STO] [MATRIX] 操作を使います。

1. [RCL] [MATRIX] を押してからコピー源の行列を表す文字キーを押す。こうすると行列の記号が表示に入ります。
2. [STO] [MATRIX] を押してからコピー先の行列を表す文字キーを押す。

[RCL] の後に指定した行列と [STO] の後に指定した行列の大きさが同じでないときには，第二の行列の大きさが第一の行列の大きさと一致するように自動的に大きさが変わります。[STO] の後に指定する行列は前もって大きさの指定が必要ありません。

例　前例の行列 **A** を行列 **B** にコピーしましょう。

キー操作	表示			
[RCL] [MATRIX] [A]	**A**	**2**	**3**	コピー源の行列の記号を表示。
[STO] [MATRIX] [B]	**A**	**2**	**3**	行列 **B** の大きさを変えてから **A** を **B** にコピー。
[RCL] [MATRIX] [B]	**b**	**2**	**3**	新しい行列 **B** の記号を表示。

単一行列の演算

次表は X レジスタにある行列だけを演算する機能の一覧表です。一つ

の行列と別のスタック・レジスタ内の数値間の演算はスカラー演算(151ページ)で説明します。

キー操作	Xレジスタの結果	Xレジスタで指定した行列への影響	結果行列への影響
CHS	変化なし。	全要素の符号が変わる。	なし。‡
1/x	結果行列の記号。	なし。‡	指定した行列の逆行列。§
f MATRIX 4	転置行列の記号	転置行列に置き換わる。	なし。‡
f MATRIX 7	指定した行列の行ノルム。*	なし。	なし。
f MATRIX 8	指定した行列のフロベニウスまたはユークリッドのノルム。†	なし。	なし。
f MATRIX 9	指定した行列の行列式。	なし。‡	指定した行列の*LU*分解。§

*行ノルムとは指定した行列の各行ごとの要素の絶対値の和の最大値。

†フロベニウスのノルムまたはユークリッドのノルムとは指定した行列の全要素の二乗和の平方根。

‡結果行列がXレジスタで指定した行列と同じでないときだけです。

§指定した行列が特異行列(つまり,逆行列が存在しない行列)のときは,丸め誤差のために多少*LU*の値が違うことがあります。1/xでは元の特異行列に近似の行列の逆行列を計算します。(更に詳しい説明はHP-15C Advanced Functions Handbookを見てください。)

例　行列**B**の転置行列を計算してください。行列**B**は前例で次のようになっています。

$$\mathbf{B}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 9\end{bmatrix}$$

キー操作	表　示			
RCL MATRIX B	b	2	3	2×3の行列**B**の記号を表示。
f MATRIX 4	b	3	2	3×2の転置行列の記号。

行列**B**（利用者優先状態でRCL Bを押すと見られます）は次のようになりました。

$$\mathbf{B}=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 9\end{bmatrix}$$

スカラー演算

スカラー演算とはスカラー（つまり，数値）と行列の各要素との間の四則演算です。スカラーと行列記号の両方をXとYレジスタに入れておくことが必要です。（レジスタに入れる順序によって−と÷の結果が違うので注意してください。）演算結果は結果行列の対応する要素に記憶します。

実行可能な演算は次表の通りです。

演　算	結果行列の要素*	
	Yレジスタに行列 Xレジスタにスカラー	Yレジスタにスカラー Xレジスタに行列
[+]	スカラー値を行列の各要素に足す。	
[×]	行列の各要素にスカラー値を掛ける。	
[−]	行列の各要素からスカラー値を引く。	スカラー値から行列の各要素を引く。
[÷]	行列の各要素をスカラー値で割る。	行列の逆行列を計算してその各要素にスカラー値を掛ける。

*結果行列は演算用に指名した行列でもよい。

例　行列 $\mathbf{B} = 2\mathbf{A}$ を計算し，次に**B**の各要素から1を引いてください。前例で**A**は次の通りになっています。

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 9 \end{bmatrix}$$

キー操作	表　示	
[f] [RESULT] [B]		行列**B**を結果行列に指名。
[RCL] [MATRIX] [A]	A 2 3	行列**A**の記号を表示。
2 [×]	b 2 3	行列**B**を**A**と同じ大きさに合わせ，**A**の各要素に2を掛け，その値を**B**の対応する要素に記憶し，結果行列の記号を表示する。

キー操作	表　示			
1 −	b	2	3	行列Bの各要素から1を引いて，その値をBの同じ要素に記憶する。

結果（利用者優先状態で RCL B を押すと見られます）は次の通りになります。

$$\mathbf{B}=\begin{bmatrix}1 & 3 & 5\\ 7 & 9 & 17\end{bmatrix}.$$

行列の和と差

二つの行列記号をXとYレジスタの両方に入れて，＋ または − を押すと行列の和または差を計算します。

キー操作	計算*
＋	Y＋X
−	Y−X
*結果を結果行列に記憶する。結果行列はXまたはYでもよい。	

例　C＝B−A を計算してください。ここでAとBは前例から次の通りになっています。

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 9\end{bmatrix} \quad \text{で} \quad \mathbf{B}=\begin{bmatrix}1 & 3 & 5\\ 7 & 9 & 17\end{bmatrix}$$

キー操作	表　示			
f RESULT C				Cを結果行例に指名。
RCL MATRIX B	b	2	3	行例Bの記号を呼び出す。（記号が既にXレジスタにあれば，この操作を省略してもよい。）
RCL MATRIX A	A	2	3	行例Aの記号をXレジスタに呼び出すと，行列Bの記号はYレジスタに移動する。

キー操作	表示	
−	C 2 3	B−A を計算し，大きさを合わせた行列 C に数値を記憶。

結果は右のようになります。 $\mathbf{C}=\begin{bmatrix}0 & 1 & 2\\ 3 & 4 & 8\end{bmatrix}$

行列の積

二つの行列記号を X と Y レジスタの両方に入れると，3 種類の異なった行列の積を計算できます。下表は X レジスタに入れた行列 **X** と Y レジスタに入れた行列 **Y** 間の 3 種の演算結果です。$\mathbf{X}^{-1}$ は **X** の逆行列。また $\mathbf{Y}^T$ は **Y** の転置行列です。

キー操作	演 算*
×	**YX**
f MATRIX 5	$\mathbf{Y}^T\mathbf{X}$
÷	$\mathbf{X}^{-1}\mathbf{Y}$
*結果を結果行列に記憶する。÷のときは，結果行列は **Y** でも良いが **X** では駄目。それ以外の演算は，結果行列は **X** でも **Y** でも駄目。	

注 式 $\mathbf{A}^{-1}\mathbf{B}$ の計算のために ÷ を使うときには，行列記号は式中の順序でなくて **B**，**A** の順に入れてください。*

結果行列の各要素に記憶する数値は普通の行列の積の規則通りです。

MATRIX 5 では，演算に使うのは Y レジスタに入れた行列の転置行列ですが，この操作をしても Y レジスタの行列は変わりません。この結果は MATRIX 4（転置）と × を使ったときの結果と同じです。

---

*これは $a^{-1}b = b/a$ を計算するために b と a をスタックに入れるときと同じ順序です。

[÷] では,X レジスタに入れた行列はその LU 分解に置き換わります。[÷] は [1/x] と [×] を使うよりも直接的に $\mathbf{X}^{-1}\mathbf{Y}$ を計算するので，もっと速くて正確な結果になります。

例　前例の行列 **A** と **B** を使って $\mathbf{C}=\mathbf{A}^T\mathbf{B}$ を計算してください。

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 9\end{bmatrix} \quad \text{で} \quad \mathbf{B}=\begin{bmatrix}1 & 3 & 5\\ 7 & 9 & 17\end{bmatrix}$$

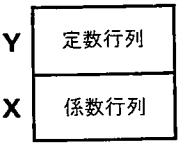
キー操作	表　示		
[RCL] [MATRIX] [A]	**A**	**2**	**3**	行列 **A** の記号を呼び出す。
[RCL] [MATRIX] [B]	**b**	**2**	**3**	行列 **B** の記号を X レジスタに呼び出し，行列 **A** の記号を Y レジスタに移す。
[f] [RESULT] [C]	**b**	**2**	**3**	行列 **C** を結果行列に指名。
[f] [MATRIX] 5	**C**	**3**	**3**	$\mathbf{A}^T\mathbf{B}$ を計算して 3×3 に大きさを変更した行列 **C** に結果を記憶。

結果を入れた行列 **C** は次の通りです。

$$\mathbf{C}=\begin{bmatrix}29 & 39 & 73\\ 37 & 51 & 95\\ 66 & 90 & 168\end{bmatrix}$$

方程式 $\mathbf{AX} = \mathbf{B}$ の解

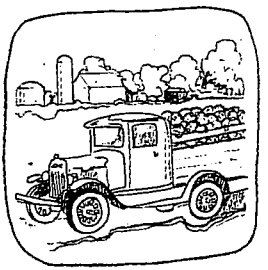
[÷] は $\mathbf{AX} = \mathbf{B}$ の形の行列方程式を解くのに便利です。ここで $\mathbf{A}$ は係数の行列，$\mathbf{B}$ は定数の行列，$\mathbf{X}$ は解の行列です。定数の行列 $\mathbf{B}$ の記号は Y レジスタに，係数の行列 $\mathbf{A}$ の記号は X レジスタに入れる必要があります。 [÷] を押すと $\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$ の解を計算します。*



[÷] は係数の行列をその *LU* 分解で置き換えることとこの行列を結果行列に指定できないことを忘れないでください。なお [1/x] と [×] の代りに [÷] を使うともっと速くてしかもより正確な解が求められます。

本章の始めで，定数行列と解行列が1列のときの連立1次方程式の解を求めました。次の例では HP-15C を使って，定数が2組以上ある（つまり，定数行列と解行列の列数が2以上の行列）ときでも解を求めています。

例 農家のサイラス・ファーマーさんはこの3週間に出荷したキャベツとブロッコリー（西洋野菜の一種）の受領書を見て次のように表にまとめました。



* もし $\mathbf{A}$ が特異行列（つまり，逆行列が存在しない行列）のときは，丸め誤差のために多少 *LU* の値が違うことがあります。計算した解は元の特異行列に近似の特異でない係数行列に対応するものです。

	週		
	1	2	3
重量合計 (kg)	274	233	331
価格合計	$120.32	$112.96	$151.36

出荷相場はキャベツ1 kgについて0.24ドル，ブロッコリー1 kgについて0.86ドルです。行列演算を使ってサイラスさんが各週に出荷したキャベツとブロッコリーの重量を計算してください。

解法　各週の出荷量は二つの未知数(キャベツとブロッコリーの重量)を含む重量と価格の連立1次方程式で表せます。行列方程式を使えば3週分を同時に計算できます。

$$\begin{bmatrix} 1 & 1 \\ 0.24 & 0.86 \end{bmatrix} \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \end{bmatrix} = \begin{bmatrix} 274 & 233 & 331 \\ 120.32 & 112.96 & 151.36 \end{bmatrix}$$

または

$$\mathbf{AD} = \mathbf{B}$$

ここで行列 **D** の第1行は3週間のキャベツの重量，第2行はブロッコリーの重量です。

キー操作	表　示	
2 [ENTER] [f] [DIM] [A]	2.0000	**A** を 2×2 の大きさにする。
[f] [MATRIX] 1	2.0000	R_0 と R_1 の行と列番号を1にする。
[f] [USER]	2.0000	利用者優先状態にする。
1 [STO] [A]	1.0000	a_{11} を記憶。
[STO] [A]	1.0000	a_{12} を記憶。
.24 [STO] [A]	0.2400	a_{21} を記憶。
.86 [STO] [A]	0.8600	a_{22} を記憶。
2 [ENTER] 3 [f] [DIM] [B]	3.0000	**B** を 2×3 の大きさにする。

274 STO B	274.0000			b_{11} を記憶。*
233 STO B	233.0000			b_{12} を記憶。
331 STO B	331.0000			b_{13} を記憶。
120.32 STO B	120.3200			b_{21} を記憶。
112.96 STO B	112.9600			b_{22} を記憶。
151.36 STO B	151.3600			b_{23} を記憶。
f RESULT D	151.3600			行列 **D** を結果行列に指名。
RCL MATRIX B	b	2	3	定数行列の記号を呼び出す。
RCL MATRIX A	A	2	2	係数行列 **A** の記号を X レジスタに呼び出し，定数行列 **B** の記号を Y レジスタに移動。
÷	d	2	3	$\mathbf{A}^{-1}\mathbf{B}$ を計算して結果を行列 **D** に記憶。
RCL D	186.0000			d_{11}，第1週のキャベツの重量を呼び出す。
RCL D	141.0000			d_{12}，第2週のキャベツの重量を呼び出す。
RCL D	215.0000			d_{13} を呼び出す。
RCL D	88.0000			d_{21} を呼び出す。
RCL D	92.0000			d_{22} を呼び出す。
RCL D	116.0000			d_{23} を呼び出す。
f USER	116.0000			利用者優先状態を解除。

*行列 **B** の要素の記憶を始める前に f MATRIX 1 を押す必要はありません。それは行列 **A** の最後の要素を記憶した後では R_0 と R_1 にある行と列の番号が自動的に1に戻るからです。

サイラスさんの出荷量は次の通りです。

	週		
	1	2	3
キャベツ (kg)	186	141	215
ブロッコリー (kg)	88	92	116

残差の計算

HP-15C は残差，つまり次式の答の行列も計算できます。

$$\text{残差} = \mathbf{R} - \mathbf{YX}$$

ここで R は結果行列，X と Y はそれぞれ X と Y レジスタに入れた行列です。

この機能は非常に便利で，例えば，連立方程式の解を反復して精密化するとか，線形回帰に使います。例えば，C が $\mathbf{AX} = \mathbf{B}$ の一つの解とすると，$\mathbf{B} - \mathbf{AC}$ はこの解が方程式をどの程度満足するかを表します。(反復法と線形回帰については HP-15C Advanced Functions Handbook などを見てください。)

残差機能(MATRIX 6) は結果行列と X，Y レジスタに入れた行列のそのときの値を使って上式の残差を計算します。残差はそれまでの結果行列の代わりに結果行列に入ります。X と Y レジスタに入れた行列を結果行列には指名できません。

× と − よりも MATRIX 6 を使うと，特に引かれる行列に比較して残差が小さいときにより正確な答になります。

残差を計算するには次のように操作します。

1. Y 行列の記号を Y レジスタに入れる。
2. X 行列の記号を X レジスタに入れる。
3. R 行列を結果行列に指名する。
4. f MATRIX 6 を押す。それまでの結果行列 (R) が残差に置き換えられる。結果行列の記号が X レジスタに入る。

*LU*形行列の利用法

前に注記したように，2種の行列演算（行列式の計算と行列方程式 **AX** = **B** の解法）では X レジスタに入れた行列の *LU* 分解を作りました。*LU* 分解した行列の記号は行列名の右に二つの横線が付きます。*LU* 形の行列の要素は元の行列の要素とは変化しています。

しかし，*LU* 形の行列の記号は，逆行列を含む演算や行列式の演算には元の行列の記号の代りに使えます。つまり，次の演算には元の行列でも，その *LU* 分解のどちらでも使えます。

[1/x]
X レジスタに入れた行列で [÷]
[MATRIX] 9

この3種の機能では，*LU* 形の行列を使った逆行列は，元の行列を使ったときと同じ答になります。

例えば，行列方程式 **AX** = **B** を解くと，行列 **A** はその *LU* 形に変わっています。もし **B** 行列を替えて，もう一度方程式を解くのなら，行列 **A** を元の行列に変えなくても計算でき，*LU* 行列で正しい解が求まります。

上記以外の全行列演算では，計算機は *LU* 分解した行列を元の行列の代理とは認めません。そして，*LU* 行列の要素は行列用メモリーにある通りに使うので，その結果は元の行列を使った結果とは一致しません。

複素行列の計算

HP–15C は複素行列（要素が複素数である行列）の積と逆行列の計算や複素連立方程式（係数と変数が複素数の方程式）を解くことができます。

しかし，HP–15C は実数の行列だけの記憶や演算だけです。複素行列の計算機能は前章で説明した複素数計算機能とは関係がありません。

つまり，複素行列の計算のときには複素数計算状態にする必要はありません。

その代わりに，複素行列の計算は，元の複素行列から次に説明する方法で作る実数行列を使って計算し，普通の行列演算以外にある種の変換が必要です。この変換は計算機の 4 種の機能を使えば可能です。本章でこれからこの計算方法を説明します。(HP–15C Advanced Functions Handbook に複素行列の計算例があります。)

複素行列の要素の記憶

$m\times n$ の複素行列 $\mathbf{Z}=\mathbf{X}+i\mathbf{Y}$ を考えてみましょう。ここで $\mathbf{X}$ と $\mathbf{Y}$ は実数の $m\times n$ 行列です。この行列は計算機中で下のように $2m\times n$ の分割した行列として表せます。

$$\mathbf{Z}^P=\begin{bmatrix}\mathbf{X}\\ \mathbf{Y}\end{bmatrix}\begin{matrix}\}\text{ 実数部}\\ \}\text{ 虚数部}\end{matrix}$$

上付き文字の P は複素行列を分割行列で表している意味に使います。

$\mathbf{Z}^P$ の要素は全部実数で，上側の半分は実数部（行列 $\mathbf{X}$）の要素を表し，下側の半分は虚数部（行列 $\mathbf{Y}$）の要素を表します。$\mathbf{Z}^P$ の要素は本章の始めに説明したように，5 個の行列のどれか（例えば $\mathbf{A}$）に普通の方法で記憶できます。

例えば $\mathbf{Z}=\mathbf{X}+i\mathbf{Y}$ が次の通りだとすると

$$\mathbf{X}=\begin{bmatrix}x_{11} & x_{12}\\ x_{21} & x_{22}\end{bmatrix}\text{ そして }\mathbf{Y}=\begin{bmatrix}y_{11} & y_{12}\\ y_{21} & y_{22}\end{bmatrix},$$

Z は計算機中で次のようになります。

$$\mathbf{A}=\mathbf{Z}^{P}=\begin{bmatrix}\mathbf{X}\\ \mathbf{Y}\end{bmatrix}=\left[\begin{array}{cc}x_{11} & x_{12}\\ x_{21} & x_{22}\\ \hline y_{11} & y_{12}\\ y_{21} & y_{22}\end{array}\right]$$

複素行列を上の **Z** のように実数行列と虚数行列を別々に書かないで，行列の各要素を次式のように普通の複素数で書いた形で計算したいことがあると思います。

$$\mathbf{Z}=\begin{bmatrix}x_{11}+iy_{11} & x_{12}+iy_{12}\\ x_{21}+iy_{21} & x_{22}+iy_{22}\end{bmatrix}$$

この行列をこれとよく似た形(i と ＋ 符号を省略しただけ)の実数行列として計算機中に入れることが可能です。例えば，上の 2×2 の行列 **Z** は計算機中で次の複素数形の 2×4 行列として表せます。

$$\mathbf{A}=\mathbf{Z}^{C}=\begin{bmatrix}x_{11} & y_{11} & x_{12} & y_{12}\\ x_{21} & y_{21} & x_{22} & y_{22}\end{bmatrix}$$

上付き文字の C は複素行列を複素数形で表している意味に使います。

複素行列は最初から計算機に $\mathbf{Z}^{C}$ の形の行列として入れられますが，複素行列の積や逆行列計算では行列が $\mathbf{Z}^{P}$ の形の行列であるものとして演算します。HP-15C に $\mathbf{Z}^{C}$ と $\mathbf{Z}^{P}$ の間で複素行列を変換する 2 種の機能も入れてあります。

キー操作	操作前	変換後
f Py,x	$\mathbf{Z}^{C}$	$\mathbf{Z}^{P}$
g Cy,x	$\mathbf{Z}^{P}$	$\mathbf{Z}^{C}$

この変換のどちらかを実行するには，$\mathbf{Z}^{C}$ または $\mathbf{Z}^{P}$ の記号を表示に呼

び出し，前表のキーを押します。この変換は指定した行列内で実行するので，結果行列は関係がありません。

例　下記の複素行列を $\mathbf{Z}^C$ 形に記憶してください。次にそれを $\mathbf{Z}^P$ 形に変換してください。

$$\mathbf{Z}=\begin{bmatrix}4+3i & 7-2i\\ 1+5i & 3+8i\end{bmatrix}$$

まず $\mathbf{Z}^C$ の要素を行列 **A** に記憶してから [Py,x] 機能を使います。

$$\mathbf{A}=\mathbf{Z}^C=\begin{bmatrix}4 & 3 & 7 & -2\\ 1 & 5 & 3 & 8\end{bmatrix}.$$

キー操作	表　示			
f MATRIX 0				全部の行列をクリア。
2 ENTER 4 f DIM A	4.0000			行列 **A** の大きさを 2×4 にする。
f MATRIX 1	4.0000			R_0 と R_1 の行と列番号を 1 にする。
f USER	4.0000			利用者優先状態にする。
4 STO A	4.0000			a_{11} を記憶。
3 STO A	3.0000			a_{12} を記憶。
7 STO A	7.0000			a_{13} を記憶。
2 CHS STO A	-2.0000			a_{14} を記憶。
1 STO A	1.0000			a_{21} を記憶。
5 STO A	5.0000			a_{22} を記憶。
3 STO A	3.0000			a_{23} を記憶。
8 STO A	8.0000			a_{24} を記憶。
f USER	8.0000			利用者優先状態を解除。
RCL MATRIX A	A	2	4	行列 **A** の記号を表示。
f Py,x	A	4	2	$\mathbf{Z}^C$ を $\mathbf{Z}^P$ に変換し，行列 **A** の大きさをそれに合わせる。

これで複素行列 $\mathbf{Z}$ は $\mathbf{Z}^P$ の形で行列 $\mathbf{A}$ に入っています。

$$\mathbf{A}=\mathbf{Z}^P=\left[\begin{array}{cc}4 & 7\\ 1 & 3\\ \hdashline 3 & -2\\ 5 & 8\end{array}\right]\begin{array}{l}\}\ \text{実数部}\\ \\ \}\ \text{虚数部}\end{array}$$

複素変換

二つの複素行列の積を計算するときにはもう一つの変換が必要ですし，複素行列の逆行列の計算では更にもう一つ変換が必要です。それは $m\times n$ 複素行列の $\mathbf{Z}^P$ 表現と次の形の $2m\times 2n$ 分割行列との間の変換です。

$$\tilde{\mathbf{Z}}=\left[\begin{array}{cc}\mathbf{X} & -\mathbf{Y}\\ \mathbf{Y} & \mathbf{X}\end{array}\right]$$

MATRIX 2 変換で作った行列 $\mathbf{Z}$ は $\mathbf{Z}^P$ の2倍の要素があります。

二つの行列 $\mathbf{Z}$ と $\mathbf{Z}^P$ の関係は下図の通りです。

$$\mathbf{Z}^P=\left[\begin{array}{cc}1 & -6\\ \hdashline -4 & 5\end{array}\right]\quad\leftrightarrow\quad\tilde{\mathbf{Z}}=\left[\begin{array}{cc:cc}1 & -6 & 4 & -5\\ \hdashline -4 & 5 & 1 & -6\end{array}\right]$$

$\mathbf{Z}^P$ と $\mathbf{Z}$ 間の複素行列の表現変換は下表の通りです。

キー操作	変換前	変換後
f MATRIX 2	$\mathbf{Z}^P$	$\tilde{\mathbf{Z}}$
f MATRIX 3	$\tilde{\mathbf{Z}}$	$\mathbf{Z}^P$

この変換を実行するのには，$\mathbf{Z}^P$ または $\tilde{\mathbf{Z}}$ の記号を表示に呼び出してから，上表のようにキーを押します。変換は指定した行列だけが対象で，結果行列には関係ありません。

複素行列の逆行列の計算

$(\tilde{\mathbf{Z}})^{-1} = (\widetilde{\mathbf{Z}^{-1}})$ を利用して複素行列の逆行列の計算ができます。

複素行列 $\mathbf{Z}$ の逆行列 $\mathbf{Z}^{-1}$ の計算は次のように操作します。

1. $\mathbf{Z}$ の要素を $\mathbf{Z}^P$ または $\mathbf{Z}^C$ の形のどちらかでメモリーに記憶する。
2. $\mathbf{Z}$ を入れた行列の記号を表示に呼び出す。
3. $\mathbf{Z}$ の要素を $\mathbf{Z}^C$ の形で入力したときは，[f][$P_{y,x}$] を押して $\mathbf{Z}^C$ を $\mathbf{Z}^P$ 形に変換する。
4. [f][MATRIX] 2 を押して $\mathbf{Z}^P$ を $\tilde{\mathbf{Z}}$ に変換する。
5. 結果行列を一つ指名する。この行列は $\tilde{\mathbf{Z}}$ の記憶に使った行列と同じでもよい。
6. [1/x] を押す。これで $(\tilde{\mathbf{Z}})^{-1}$ を計算しますが，これは $(\widetilde{\mathbf{Z}^{-1}})$ と同じです。この行列の要素の値が結果行列に入り，結果行列の記号が X レジスタに入ります。
7. [f][MATRIX] 3 を押して $(\widetilde{\mathbf{Z}^{-1}})$ を $(\widetilde{\mathbf{Z}^{-1}})^P$ に変換する。
8. 逆行列を $(\mathbf{Z}^{-1})^C$ 形にしたいときは [g][$C_{y,x}$] を押す。

$\mathbf{Z}^P$ または $\mathbf{Z}^C$ の要素を呼び出して前で説明した通りに組み合わせると，$\mathbf{Z}^{-1}$ の複素要素が求められます。

例　前例の複素行列 $\mathbf{Z}$ の逆行列を計算してください。

$$\mathbf{A} = \mathbf{Z}^P = \left[\begin{array}{cc} 4 & 7 \\ 1 & 3 \\ \hdashline 3 & -2 \\ 5 & 8 \end{array}\right]$$

キー操作	表　示			
[RCL][MATRIX][A]	A	4	2	行列 A の記号を呼び出す。
[f][MATRIX] 2	A	4	4	$\mathbf{Z}^P$ を $\tilde{\mathbf{Z}}$ に変換し，行列 A の大きさを変更する。

キー操作	表示			
f RESULT B	A	4	4	**B** を結果行列に指名する。
1/x	b	4	4	$(\widetilde{\mathbf{Z}})^{-1} = (\widetilde{\mathbf{Z}^{-1}})$ を計算してそれを行列 **B** に記憶する。
f MATRIX 3	b	4	2	$(\widetilde{\mathbf{Z}^{-1}})$ を $(\mathbf{Z}^{-1})^P$ に変換する。

行列 **B** に分割形の $\mathbf{Z}^{-1}$ を次のように記憶しています。

$$\mathbf{B} = \left[\begin{array}{cc} -0.0254 & 0.2420 \\ -0.0122 & -0.1017 \\ \hdashline -0.2829 & -0.0022 \\ 0.1691 & -0.1315 \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\b\end{array}}\right\} \text{実数部} \\ \left.\vphantom{\begin{array}{c}a\\b\end{array}}\right\} \text{虚数部} \end{array}$$

複素行列の積

$(\mathbf{YX})^P = \widetilde{\mathbf{Y}}\mathbf{X}^P$ を利用して二つの複素行列の積を計算できます。

Y と **X** が複素行列のとき **YX** を計算するには次のように操作します。

1. **Y** と **X** の要素を $\mathbf{Z}^P$ または $\mathbf{Z}^C$ のどちらかの形でメモリーに記憶する。
2. **Y** の行列の記号を表示に呼び出す。
3. **Y** の要素を $\mathbf{Y}^C$ の形で入力したときには，f Py,x を押して $\mathbf{Y}^C$ を $\mathbf{Y}^P$ に変換する。
4. f MATRIX 2 を押して $\mathbf{Y}^P$ を $\widetilde{\mathbf{Y}}$ に変換する。
5. **X** の行列の記号を表示に呼び出す。
6. **X** の要素を $\mathbf{X}^C$ 形で入力したときは，f Py,x を押して $\mathbf{X}^C$ を $\mathbf{X}^P$ 形に変換する。
7. 結果行列を指名する。これは上の二つの行列のどちらかと同じには出来ない。

8. [×] を押して $\widetilde{\mathbf{Y}}\mathbf{X}^P = (\mathbf{YX})^P$ を計算する。この行列の要素の値が結果行列に入り，結果行列の記号が X レジスタに入る。

9. 積を $(\mathbf{YX})^C$ の形にしたいときには，[g][Cy,x] を押す。

$\mathbf{X}^P$ は $\widetilde{\mathbf{X}}$ に変換しないので注意してください。

$(\mathbf{YX})^P$ または $(\mathbf{YX})^P$ の要素を呼び出して前に説明した通りに組み合わせると行列の積 $\mathbf{YX}$ の複素要素が求められます。

例　$\mathbf{Z}$ が前例の複素行列のときの積 $\mathbf{ZZ}^{-1}$ を計算してください。

両方の行列の要素は既に記憶している（$\widetilde{\mathbf{Z}}$ は $\mathbf{A}$ に，$(\mathbf{Z}^{-1})^P$ は $\mathbf{B}$ に）ので，手順の1，3，4 と 6 を省略します。

キー操作	表　示		
[RCL][MATRIX][A]	A	4 4	行列 A の記号を表示。
[RCL][MATRIX][B]	b	4 2	行列B の記号を表示。
[f][RESULT][C]	b	4 2	C を結果行列に指名。
[×]	C	4 2	$\widetilde{\mathbf{Z}}(\mathbf{Z}^{-1})^P = (\mathbf{ZZ}^{-1})^P$ を計算。
[f][USER]	C	4 2	利用者優先状態にする。
[RCL][C]	C 1,1		行列 C の第 1 行，第 1 列。（最後のキーを押している間一時的に表示。
	1.0000		c_{11} の値。
[RCL][C]	−2.8500	−10	c_{12} の値。
[RCL][C]	−4.0000	−11	c_{21} の値。
[RCL][C]	1.0000		c_{22} の値。
[RCL][C]	1.0000	−11	c_{31} の値。
[RCL][C]	3.8000	−10	c_{32} の値。
[RCL][C]	1.0000	−11	c_{41} の値。
[RCL][C]	−1.0500	−10	c_{42} の値。
[f][USER]	−1.0500	−10	利用者優先状態を解除。

Cの要素を書き出すと次の通りです。

$$\mathbf{C}=\left[\begin{array}{cc} 1.0000 & -2.8500\times10^{-10} \\ -4.0000\times10^{-11} & 1.0000 \\ 1.0000\times10^{-11} & 3.8000\times10^{-10} \\ 1.0000\times10^{-11} & -1.0500\times10^{-10} \end{array}\right]=(\mathbf{ZZ}^{-1})^{P},$$

ここで行列**C**の上半分は$\mathbf{ZZ}^{-1}$の実数部，下半分は虚数部です。そこで，行列**C**を並べ変えると次のようになります。

$$\mathbf{ZZ}^{-1}=\left[\begin{array}{cc} 1.0000 & -2.8500\times10^{-10} \\ -4.0000\times10^{-11} & 1.0000 \end{array}\right]$$

$$+i\left[\begin{array}{cc} 1.0000\times10^{-11} & 3.8000\times10^{-11} \\ 1.0000\times10^{-11} & -1.0500\times10^{-10} \end{array}\right]$$

予想通りに下式のようになりました。

$$\mathbf{ZZ}^{-1}\cong\left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right]+i\left[\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array}\right]$$

複素行列方程式 $\mathbf{AX}=\mathbf{B}$ の解

複素行列方程式 $\mathbf{AX}=\mathbf{B}$ は $\mathbf{X}=\mathbf{A}^{-1}\mathbf{B}$ を計算して解けます。そのために $\mathbf{X}^{P}=(\widetilde{\mathbf{A}})^{-1}\mathbf{B}^{P}$ を計算します。

A, **X**と**B**が複素行列のときに方程式 $\mathbf{AX}=\mathbf{B}$ を解くには，次のように操作します。

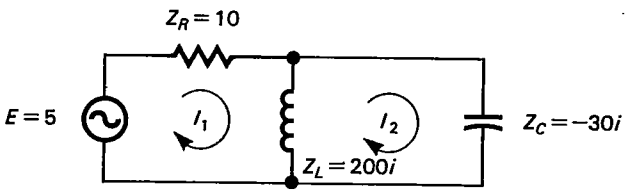
1. **A**と**B**の要素を$\mathbf{Z}^{P}$または$\mathbf{Z}^{C}$のどちらかの形でメモリーに記憶する。
2. **B**の行列の記号を表示に呼び出す。
3. **B**の要素を$\mathbf{B}^{C}$の形で入力したときには，[f][Py,x]を押して$\mathbf{B}^{C}$を$\mathbf{B}^{P}$に変換する。

4. **A** の行列の記号を表示に呼び出す。
5. **A** の要素を $\mathbf{A}^C$ の形で入力したときには，[f][Py,x] を押して $\mathbf{A}^C$ を $\mathbf{A}^P$ に変換する。
6. [f][MATRIX] 2 を押して $\mathbf{A}^P$ を $\widetilde{\mathbf{A}}$ に変換する。
7. 結果行列を指名する。これは **A** を入れた行列と同じではいけない。
8. [÷] を押すと $\mathbf{X}^P$ を計算する。この行列の要素の値が結果行列に入り，結果行列の記号が X レジスタに入る。
9. 解を $\mathbf{X}^C$ の形にしたいときには，[g][Cy,x] を押す。

$\mathbf{B}^P$ は $\widetilde{\mathbf{B}}$ に変換しないので注意してください。

$\mathbf{X}^P$ または $\mathbf{X}^C$ の要素を呼び出して前で説明したように組み合わせると，解 **X** の複素数要素が求められます。

例　大学工学部の学生 A.C. ディマー君は下図の電気回路を解折したいと思っています。部品のインピーダンスは複素数です。電流 I_1 と I_2 の複素数を計算してください。



この回路は次の複素行列方程式で表せます。

$$\begin{bmatrix} 10+200i & -200i \\ -200i & (200-30)i \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

別の形では次式になります。

$$\mathbf{AX} = \mathbf{B}$$

この **A** と **B** は分割形では次のようになります。

$$\mathbf{A}=\left[\begin{array}{cc} 10 & 0 \\ 0 & 0 \\ \hline 200 & -200 \\ -200 & 170 \end{array}\right] \text{ で } \mathbf{B}=\left[\begin{array}{c} 5 \\ 0 \\ \hline 0 \\ 0 \end{array}\right],$$

ここで 0 の要素は実数部または虚数部の値が 0 の部分です。

キー操作	表　示			
4 [ENTER] 2 [f] [DIM] [A]	2.0000			行列 **A** の大きさを 4×2 にする。
[f] [MATRIX] 1	2.0000			R_0 と R_1 の行と列番号を 1 にする。
[f] [USER]	2.0000			利用者優先状態にする。
10 [STO] [A]	10.0000			a_{11} を記憶。
0 [STO] [A]	0.0000			a_{12} を記憶。
[STO] [A]	0.0000			a_{21} を記憶。
[STO] [A]	0.0000			a_{22} を記憶。
200 [STO] [A]	200.0000			a_{31} を記憶。
[CHS] [STO] [A]	−200.0000			a_{32} を記憶。
[STO] [A]	−200.0000			a_{41} を記憶。
170 [STO] [A]	170.0000			a_{42} を記憶。
4 [ENTER] 1 [f] [DIM] [B]	1.0000			行列 **B** の大きさを 4×1 にする。
0 [STO] [MATRIX] [B]	0.0000			**B** の全要素に 0 を記憶。
5 [ENTER] 1 [ENTER]	1.0000			第 1 行，第 1 列に数値 5 を指定。
[STO] [g] [B]	5.0000			数値 5 を b_{11} に記憶。
[RCL] [MATRIX] [B]	b	4	1	行列 **B** の記号を呼び出す。
[RCL] [MATRIX] [A]	A	4	2	行列 **A** の記号を X レジスタに入れて，行列 **B** の記号を Y レジスタに移す。

キー操作	表　示			
f MATRIX 2	A	4	4	$\mathbf{A}^P$ を $\tilde{\mathbf{A}}$ に変換。
f RESULT C	A	4	4	行列 **C** を結果行列に指名。
÷	C	4	1	$\mathbf{X}^P$ を計算し，**C** に記憶。
g Cy,x	C	2	2	$\mathbf{X}^P$ を $\mathbf{X}^C$ に変換。
RCL C	0.0372			c_{11} を呼び出す。
RCL C	0.1311			c_{12} を呼び出す。
RCL C	0.0437			c_{21} を呼び出す。
RCL C	0.1543			c_{22} を呼び出す。
f USER	0.1543			利用者優先状態を解除。
f MATRIX 0	0.1543			全行列の大きさを 0×0 に戻す。

行列 **C** から電流は次の複素行列 **X** になります。

$$\mathbf{X} = \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0.0372 + 0.1311i \\ 0.0437 + 0.1543i \end{bmatrix}.$$

上例では行列方程式を解くために行列用メモリーのレジスタが24個——4×4 行列 **A** に16個(これは最初 2×2 複素行列用として 4×2 行列に入力しました)，行列 **B** と **C** のそれぞれに4個(どちらも 2×1 複素行列用) 必要でした。(しかし，結果行列として行列 **B** を指名すればレジスタ数は4個少くて済みました。) **X** と **B** はベクトル (つまり，1列の行列) とは限らないので，**X** と **B** にもっと多くのメモリーを必要とする場合もあります。

HP-15C には **A** が 2×2 であれば **X** と **B** は6列まで，また **A** が 3×3 であれば **X** と **B** が2列までの複素行列方程式 $\mathbf{AX} = \mathbf{B}$ を上記の方法で解くのに十分なメモリーがあります。*(定数行列 **B** を結果行列として使えば，計算可能の列数が2倍になります。) **X** と **B** の列数がも

*利用可能なメモリー全体を共通メモリー領域に変えた場合です (MEM : 1 64 0-0)。付録 C のメモリー配分を見てください。

っと多かったり，$\mathbf{A}$ が 4×4 のときには，この方程式を次の別方法で解けます。この方法の違いは個々の逆行列の計算と行列の積を求めるので，レジスタの使用数が前の方法よりも少くて済む点です。

1. $\mathbf{A}$ の要素を $\mathbf{A}^P$ または $\mathbf{A}^C$ の形のどちらかで記憶する。
2. 行列 $\mathbf{A}$ の記号を表示に呼び出す。
3. $\mathbf{A}$ の要素を $\mathbf{A}^C$ 形で入力したときには，[f][Py,x] を押して $\mathbf{A}^C$ を $\mathbf{A}^P$ に変換する。
4. [f][MATRIX] 2 を押して $\mathbf{A}^P$ を $\tilde{\mathbf{A}}$ に変換する。
5. [STO][RESULT] を押して $\mathbf{A}$ を入れた行列を結果行列に指名する。
6. [1/x] を押して $(\tilde{\mathbf{A}})^{-1}$ を計算する。
7. 前の手順後に表示する記号の行数を半分の行数にするために $\mathbf{A}$ の大きさを変える。
8. $\mathbf{B}$ の要素を $\mathbf{B}^P$ かまたは $\mathbf{B}^C$ のどちらかの形でメモリーに記憶する。
9. $\mathbf{A}$ を入れた行列の記号を表示に呼び出す。
10. $\mathbf{B}$ を入れた行列の記号を表示に呼び出す。
11. $\mathbf{B}$ の要素を $\mathbf{B}^C$ の形に入力したときには，[f][Py,x] を押して $\mathbf{B}^C$ を $\mathbf{B}^P$ に変換する。
12. [f][MATRIX] 2 を押して $\mathbf{B}^P$ を $\tilde{\mathbf{B}}$ に変換する。
13. 結果行列を指名する。これは上記二つの行列のどちらかと同じではいけません。
14. [×] を押す。
15. [f][MATRIX] 4 を押して結果行列を転置する。
16. [f][MATRIX] 2 を押す。
17. 前の手順後に表示する記号の行数を半分の行数にするために結果行列の大きさを変える。

18. [RCL] [RESULT] を押して結果行列の記号を呼び出す。

19. [f] [MATRIX] 4 を押して $\mathbf{X}^P$ を計算する。

20. 解を $\mathbf{X}^C$ の形で求めたいときには，[g] [Cy,x] を押す。

この方法を使うときの問題点は HP–15C Advanced Functions Handbook の〝大きな連立複素方程式の解法〟に説明があります。

行列を使った各種の操作

行列の要素を使ったレジスタ操作

次の機能キーに続いて行列を指定する文字キーを押すと，データ記憶用レジスタのときと同様に R_0 と R_1 内の行番号と列番号で指定した行列の要素を操作できます。

[STO]*
[STO] {[+], [−], [×], [÷]}
[DSE]
[x≷]

[RCL]*
[RCL] {[+], [−], [×], [÷]}
[ISG]

I レジスタ内の行列記号の利用法

利用分野によっては，行列 **A** から **E** のどれかを使った行列演算をプログラム中に組み込みたいことがあると思います。このようなときに，I レジスタ (R_I) に行列の記号を入れて行列演算に使います。

I レジスタ内に行列記号を記憶しているときには

- 上記のどれかの機能の後に [(i)] を押すと，R_I で指定した行列の R_0 と R_1 で指定した行と列番号の要素を使って演算を実行する。
- [STO] [g] または [RCL] [g] の後に [(i)] を押すと，R_I で指定した行列の X と Y レジスタで指定した行と列番号の要素を使って演算を実行する。

---

*利用者優先状態のときにはこの操作で R_0 と R_1 内の行番号と列番号が指定した行列の大きさの範囲内で変わります。

- f DIM I を押すと，R_I で指定した行列の大きさをXとYレジスタにある大きさに変える。
- RCL DIM I を押すと，R_I で指定した行列の大きさをXとYレジスタに呼び出す。
- GSB I または GTO I を押すと，GSB または GTO の後に R_I 内と同じ行列の文字キーを押したのと同じ結果になる。（これは実際の行列演算ではなくて，単に行列の記号の文字を使うだけのことです。）

行列記号の条件判断

4種の条件判断（x=0，TEST 0 ($x \neq 0$)，TEST 5 ($x = y$) と TEST 6 ($x \neq y$)）はXとYレジスタ内の行列記号についても使えます。条件判断は第8章で説明したように，プログラムの実行の制御に使います。

行列の記号がXレジスタにあると，x=0 の結果は偽となり，TEST 0 の結果は真となります（行列の要素の値に関係なく）。

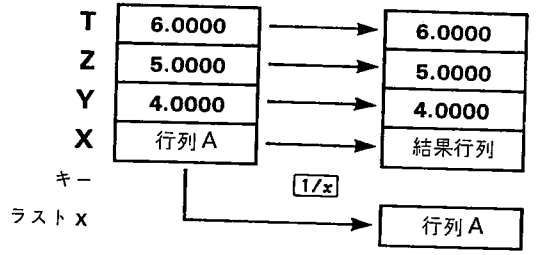
行列の記号がXとYレジスタにあるときに，TEST 5 または TEST 6 の条件判断を実行すると，XとYレジスタに同じ記号があれば x と y は等しく，そうでないときは等しくないと見なします。比較するのは行列の記号そのものであって，行列の要素間ではありません。

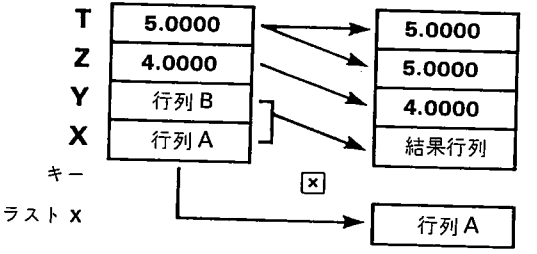
これ以外の条件判断は行列の記号には使えません。

行列演算とスタック操作

行列演算の実行中は，スタック・レジスタの内容は数値計算のときとよく似た移動をします。

行列演算によっては結果を結果行列に記憶します。引き数（Xレジスタまたは XとYレジスタ内の一つまたは二つの記号か数値）は演算によって組み合わされて，結果行列の記号がXレジスタに入ります。（演算前にXレジスタにあった引き数は LAST x レジスタに入ります）。



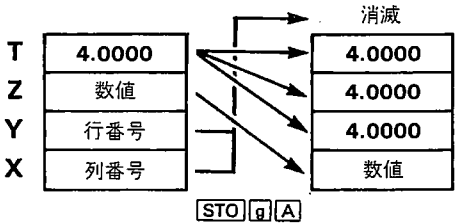


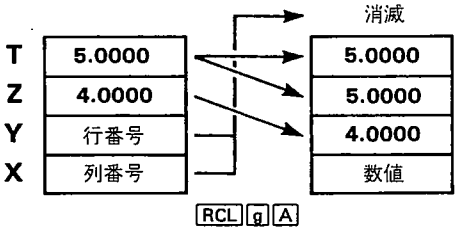
行列機能によってはXレジスタに指定した行列だけについて演算をして，その結果を同じ行列に記憶します。このような演算をしても，スタック（LAST x レジスタも含む）の内容は移動しません。しかし表示は必要に応じて新しい大きさを表示するために変わることがあります。

[MATRIX] 7，[MATRIX] 8 と [MATRIX] 9 の 3 種は，Xレジスタに指定した行列の記号がLAST x レジスタに入り，ノルムまたは行列式（[MATRIX] 9 の場合）がXレジスタに入ります。Y，Z と Tレジスタは変わりません。

スタック上昇が可能のときに行列の記号または要素をXレジスタに呼び出すと，既にスタックに入っている別の記号や数値がスタック上昇して，Tレジスタの内容が消滅します。(LAST x レジスタは変わりません。) 記号または行列の要素を記憶するときには，スタック（とLAST x レジスタ）内容は変わりません。

[STO] [g] と [RCL] [g] 機能は上記と違って，LAST x レジスタには関係なくて次図のように変わります。

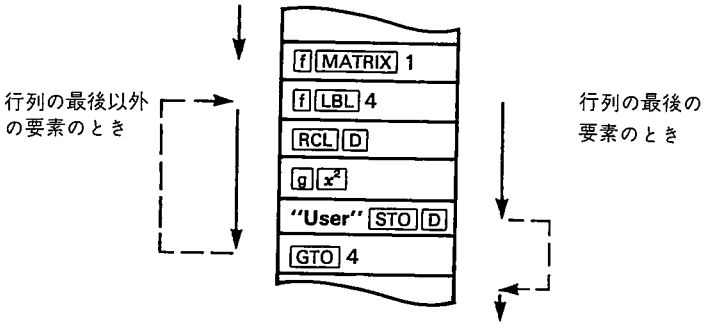




プログラム中での行列演算

プログラム入力時に利用者優先状態にしてあって行列の要素の記憶や呼び出しに [STO] または [RCL] { [A] ～ [E] か [(i)] } 命令を入力すると，普通の行番号の右に表示する横線の代わりに **u** を表示します。計算状態でこの行を実行すると，利用者優先状態にしてあるときと同じように働きます。つまり，R_0 と R_1 内の行と列番号が指定した行列の大きさに従って自働的に増加します。この働きがあるので行列の要素を順々に扱えます。(表示部の **USER** の文字はプログラム走行時には何の効果もありません。)

更に **u** 付きの [STO] か [RCL] 命令で最後の要素を処理して R_0 と R_1 が 1 に戻るとプログラムが次の行を飛び越します。これは，行列の各要素の記憶や呼び出しにループを使って，それが終ったらプログラムの先を実行するようなプログラムを作るときに便利です。例えば，次の図は行列 **D** の全要素を二乗するプログラムの一部です。



MATRIX　7（行ノルム）と MATRIX 8（フロベニウスのノルム）もプログラム中で条件ジャンプ命令として使えます。X レジスタ内が行列の記号だと，この機能が普通の方法でノルムを計算してから次のプログラム行に行きます。X レジスタ内が数値だと，プログラムはこの機能の次の行を飛び越します。どちらの場合も，X レジスタの元の内容は LAST x レジスタに入ります。これはプログラム中で X レジスタに行列の記号があるかどうかの判断に便利です。

行列機能の要点

キー操作	結　果
g Cy,x	$\mathbf{Z}^P$ を $\mathbf{Z}^C$ に変換する。
CHS	X レジスタに指定した行列の全要素の符号を変える。
f DIM {A ~ E, I}	指定した行列の大きさを変える。
f MATRIX 0	行列全部の大きさを 0×0 にする。
f MATRIX 1	R_0 と R_1 内の行と列番号を 1 にする。
f MATRIX 2	$\mathbf{Z}^P$ を $\tilde{\mathbf{Z}}$ に変換する。
f MATRIX 3	$\tilde{\mathbf{Z}}$ を $\mathbf{Z}^P$ に変換する。
f MATRIX 4	X レジスタ中の行列の転置行列を計算する。
f MATRIX 5	Y レジスタ中の行列の転置行列と X レジスタ中の行列を掛けて，結果行列に記憶する。

キー操作	結　果
f MATRIX 6	結果行列の残差を計算する。
f MATRIX 7	X レジスタ中の行列の行ノルムを計算する。
f MATRIX 8	X レジスタ中の行列のフロベニウスまたはユークリッドのノルムを計算する。
f MATRIX 9	X レジスタ中の行列の行列式を計算する。結果行列に LU 分解が入る。
f Py,x	$\mathbf{Z}^C$ を $\mathbf{Z}^P$ に変換する。
RCL {A ~ E, (i)}	R_0 と R_1 にある行と列番号を使って指定した行列の要素の値を呼び出す。
RCL g {A ~ E, (i)}	Y と X レジスタにある行と列番号を使って指定した行列の要素の値を呼び出す。
RCL DIM {A ~ E, I}	指定した行列の大きさを X と Y レジスタに呼び出す。
RCL MATRIX {A ~ E}	指定した行列の記号を表示する。
RCL RESULT	結果行列の記号を表示する。
f RESULT {A ~ E}	指定した行列を結果行列に指名する。
STO {A ~ E, (i)}	R_0 と R_1 にある行と列番号を使って表示している数値を指定した行列の要素に記憶する。
STO g {A ~ E, (i)}	Y と X レジスタにある行と列番号を使って表示している数値を指定した行列の要素に記憶する。
STO MATRIX {A ~ E}	行列の記号を表示しているときは，その行列の全要素を指定した行列の対応する要素にコピーする。数値を表示しているときは，その値を指定した行列の全要素に記憶する。

キー操作	結　果
[STO] [RESULT]	X レジスタ中の行列を結果行列に指名する。
[f] [USER]	[STO] か [RCL] {[A] ～ [E], [(i)]} を押すたびに R_0 と R_1 にある行と列の番号が自動的に増加する。
[1/x]	X レジスタ中の行列の逆行列を作って，結果行列に記憶する。
[+], [−]	行列の記号が X と Y レジスタの両方に入れてあるときは，指定した行列の対応する要素同士の足し算または引き算をする。行列の記号が X か Y レジスタのどちらか片方に入れてあるときは，指定した行列の全部の要素ともう一方のレジスタ内のスカラー間で足し算または引き算をして，結果行列に記憶する。
[×]	行列の記号が X と Y レジスタの両方に入れてあれば，指定した行列の積（**YX** の形で）を計算する。行列の記号が X と Y レジスタの片方だけに入れてあるときは，指定した行列の個々の要素ともう一方のレジスタ内のスカラーを掛けて，結果行列に記憶する。
[÷]	行列の記号が X と Y レジスタの両方に入れてあるときは，X レジスタに入れた行列の逆行列と Y レジスタに入れた行列との積を求める。Y レジスタにだけ行列の記号が入れてあるときは，指定した行列の個々の要素の全体を X レジスタ内のスカラーで割る。X レジスタにだけ行列の記号を入れてあるときは，指定した行列の逆行列の個々の要素と Y レジスタにあるスカラーを掛けて結果行列に記憶する。

詳細な解説の紹介

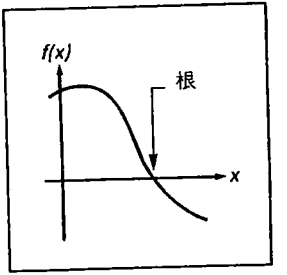
HP–15C Advanced Functions Handbook には HP–15C の行列機能のもっと詳しい解説や技法とプログラム例があります。主な内容は，最小二乗法の計算，非線形方程式の解法，条件の悪い行列と特異行列，正確さの考察，反復法の改良，単位行列の作成などです。

第13章

方程式の求根

計算目的によっては次式を解くことが必要になります。

$$f(x)=0\ *$$



これはこの式が成り立つようなxを見付けることです。このxの値を方程式$f(x)=0$の根とか関数$f(x)$をゼロにする値と呼びます。根が実数のときには実根と言います。大部分の場合は方程式の根を代数学的に解けますが，その方法では解けない場合もかなりあります。代数的解法が使えないときには数値計算によって近似根を推定します。HP-15Cの [SOLVE] キーを使うと，内部の高級な数値計算技法を利用して式の広い範囲から効果的にしかも簡単に実根を求められます†

[SOLVE] の使用法

根の計算では，$f(x)$を計算するサブルーチンを書き込んでおいて，[SOLVE] 機能で何度もこのサブルーチンを繰り返して使います。

[SOLVE] を使うときの基本的な原則は次の通りです。

＊一つの変数だけを含む方程式ならどんな方程式でもこれと同じように変形できます。例えば，$f(x)=a$ は $f(x)-a=0$ に，$f(x)=g(x)$ は $f(x)-g(x)=0$ と同じです。

† [SOLVE] 機能は虚数スタックを使いません。複素根については HP-15C Advanced Functions Handbook を見てください。

1. プログラム入力状態で，関数 $f(x)$ の値を計算するサブルーチンのプログラムをキー入力する。このサブルーチンは必ずラベル命令（ f LBL ラベル）で始まって最後に X レジスタに $f(x)$ の値を残すことが必要です。

計算状態にしてから

2. 求める根の二つの初期推定値を ENTER で区切って X と Y レジスタに入力します。この推定値は最初に $f(x)=0$ の根を捜すときの x の大体の範囲を計算機に指示するだけのものなので正確でなくても結構です。

3. f SOLVE と押してからサブルーチンのラベルのキーを押します。計算機は関数 $f(x)$ をゼロにする値を捜してそれを表示します。関数をゼロにする x の値が二つ以上あってもそのどれか一つを見付けると計算を停止します。別の x の値を求めたいときには，最初の推定値を変えてもう一度 SOLVE を使います。

SOLVE で指定したサブルーチンを呼び出す直前に X，Y，Z と T レジスタに x の値を HP-15C が自動的に入れます。次にこの値を使ってそのサブルーチンで $f(x)$ の値を計算します。スタック全体に x の値が入っているので，この数値をサブルーチンで何回でも繰り返して使えます。（この手法は 41 ページで説明しました）。

例　SOLVE を使って次の方程式を解いてください

$$f(x)=x^2-3x-10=0$$

ホーナー法（79 ページを見てください）を使って $f(x)$ を次のように変形するとプログラムが簡単になります。

$$f(x)=(x-3)x-10$$

プログラム入力状態にして，$f(x)$ を計算する次のサブルーチンをキー入力してください。

キー操作	表　示	
g P/R	000-	プログラム入力状態。
f CLEAR PRGM	000-	プログラム用メモリー内容を消す。

キー操作	表 示		
f LBL 0	001-	42,21, 0	プログラムのラベルを指定。スタック全体に x の値が入っているものとしてサブルーチンを始める。
3	002-	3	
−	003-	30	$x-3$ の計算。
×	004-	20	$(x-3)x$ の計算。
1	005-	1	
0	006-	0	
−	007-	30	$(x-3)x-10$ の計算。
g RTN	008-	43 32	

計算状態に戻して，二つの初期推定値をXとYレジスタに入れましょう。正の根を捜すために初期推定値として0と10を入力してみます。

キー操作	表 示*	
g P/R		計算状態。
0 ENTER	0.0000	初期推定値。
10	10	

これで f SOLVE 0 と押せば目的の根が求められます。こうしても直ちに答を表示する訳ではありません。HP-15Cの根の推定計算に反復法の計算手順を使っているからです。この内部計算手順はその関数を何回も，多分10回以上も繰り返して計算しています。そのたびに入れたサブルーチンを実行します。根が求まるまでに普通30秒から2分位かかりますが，場合によってはもっと時間がかかることもあります。

f SOLVE 0 を押してからHP-15Cがその素晴らしい性能を発揮するまでちょっと待ってください。 SOLVE の計算中は **running** の文字を点滅表示します。

＊これと同じ表示にしたいときには f FIX 4 を押してください。表示指定は SOLVE の動作と関係ありません。

キー操作	表　示	
f SOLVE 0	5.0000	求める根。

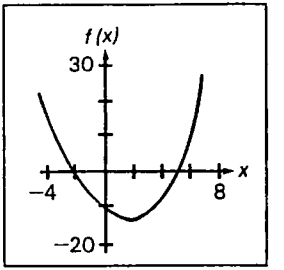
根を見付けて表示したら，スタックを調べると表示の数値が $f(x) = 0$ の正しい根であるかどうかを確認することも可能です。表示（X レジスタ）には求めた根の値が入っています。Y レジスタには一つ前の根の推定値が入っていますが，これは表示の根の値に非常に近いはずです。Z レジスタには表示している根から計算した関数の値が入っています。

キー操作	表　示	
R↓	5.0000	一つ前の根の推定値。
R↓	0.0000	関数の値で,求めた根のときに $f(x) = 0$ になることを意味します。

上で解いた 2 次方程式には根が二つあります。別の初期推定値を二つ指定して,もう一つの根も求められます。負の根用に今度は 0 と −10 を入力してみます。

キー操作	表　示	
0 ENTER	0.0000	} 初期推定値
10 CHS	−10	
f SOLVE 0	−2.0000	第 2 の根。
R↓	−2.0000	一つ前の推定値。
R↓	0.0000	第 2 の根のときの $f(x)$ の値。

これで $f(x) = 0$ の二つの根が求まりました。この2次方程式は代数的に解くことも可能で SOLVE を使って求めた根と同じ根になります。



$f(x)$ のグラフ

SOLVE キーの便利さと性能は代数的に解けない方程式を解いてみるともっとはっきり分かると思います。

例 リジット投げのチヤンピオンのチヤック・ファー君は毎秒 50 m の速さでリジットを上向きに投げられます。リジットの高さを次式のように表すと，地面に落ちるまでに何秒かかるでしょうか。(このリジット投げとは空想の競技です。数値にこだわらないで，計算法そのものを練習してください。)

$$h = 5000(1 - e^{-t/20}) - 200t$$

この式で h は高さで単位はメートル，t は時間で単位は秒です。

解法 求める答は $h = 0$ となるときの t の正の値です。

高さを計算するため次のサブルーチンを使います。

キー操作	表　示		
g P/R	000-		
f LBL A	001-42,21,11		始めのラベル。
2	002-	2	このサブルーチンでは t が X と Y レジスタに入っているものとします。
0	003-	0	
÷	004-	10	

キー操作	表　示		
[CHS]	005–	16	$-t/20$
[e^x]	006–	12	
[CHS]	007–	16	$-e^{-t/20}$
1	008–	1	
[+]	009–	40	$1-e^{-t/20}$
5	010–	5	
0	011–	0	
0	012–	0	
0	013–	0	
[×]	014–	20	$5000(1-e^{-t/20})$
[x≷y]	015–	34	もう一つの t をXレジスタに入れる。
2	016–	2	
0	017–	0	
0	018–	0	
[×]	019–	20	$200t$
[−]	020–	30	$5000(1-e^{-t/20})-200t$
[g][RTN]	021–	43 32	

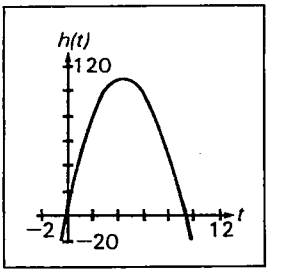
計算状態に戻して，時間の初期推定値を二つ（例えば5と6秒）をキー入力してから [SOLVE] を押します。

キー操作	表　示	
[g][P/R]		計算状態。
5 [ENTER]	5.0000	初期推定値
6	6	初期推定値
[f][SOLVE][A]	9.2843	求める根。

YとZレジスタを見て根が正しいかどうか調べてみましょう。

キー操作	表　示	
[R↓]	9.2843	一つ前の根の推定値。
[R↓]	0.0000	求めた根のときの関数値でこれなら $h=0$ になる。

彼のリジットは投げてから 9.2843 秒後に地上に落下するので,ものすごい投げ方だと言えます。



h と *t* のグラフ

実根を持たない場合

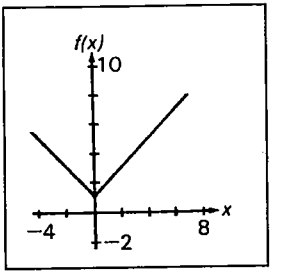
SOLVE キーが $f(x)=0$ の形の方程式の根を推定して表示する様子が分かったと思います。しかし，方程式の中には実根を持たない（つまり，この等式を満足するような x の実数値がない）こともあります。もちろん，このような場合には計算機で根を求めるのは期待できません。その代わりに **Error 8** を表示します。

例 次の方程式を考えてみましょう。

$$|x|=-1$$

絶対値関数は決して負にならないのでこの方程式の根はありません。この方程式を次のように変形して SOLVE で解いてみましょう。

$$|x|+1=0$$



f(x)＝|x|＋1 のグラフ

キー操作	表　示	
g P/R	000-	プログラム入力状態。
f LBL 1	001-42,21, 1	
g ABS	002- 43 16	
1	003- 1	
+	004- 40	
g RTN	005- 43 32	

絶対値関数は引き数が 0 の付近で極小になるので，最初の推定値はその付近，例えば 1 と −1 にします。それでは解いてみましょう。

キー操作	表　示	
g P/R		計算状態
1 ENTER	1.0000	初期推定値
1 CHS	−1	初期推定値
f SOLVE 1	Error 8	この表示は根が見付からなかったことを表す。
←	0.0000	エラー表示を消す。

このように，HP–15C は（少くとも最初に指定した範囲内では）$f(x) = 0$ の根がないと判断すると求根計算を中止します。**Error 8** の表示は正しくない操作を知らせるのではなくて，単に（指定した初期推定値からは）SOLVE で根が見付からなかったことを表すだけです。

HP–15C が求根計算を中止してエラーを表示するのは，次の三つのどれかです。

- 繰り返し計算を行ってもその関数値が 0 でない一定数になる場合に，実行を中止して **Error 8** を表示する。
- 何回か関数値を計算しても推定範囲内で 0 でない絶対値の極小値であると判断すると，実行を中止して **Error 8** を表示する。
- サブルーチン内で不適当な引き数を使って正しくない数学演算をすると，実行を中止して **Error 0** を表示する。

関数値が一定になる場合には，内部計算手順で関数値が 0 に向う傾向を持たないことを読み取ります。これは有効数字の 10 桁目までが一定になる（グラフでは 0 でない水平の漸近線に接近する）ときとか，あるいは推定した x の範囲に比べて比較的幅の広い平らな領域があるときに現れます。

関数の絶対値が 0 でない極小値に達したとは，内部計算手順で関数の

大きさが逐次小さくなるような一連の x の値を論理的に追求しても，関数のグラフが x 軸に接するか交差する x の値が見付からないときです。

第3の場合は，根を求める内部計算手順の限界というよりも，サブルーチン内の欠陥と言えます。初期推定値を適切に指定することによって，数学的に不合理な計算を避けることが可能な場合もときにはあります。しかし [SOLVE] の内部計算手順は非常に積極的に，広い範囲にわたって関数の計算をするのでエラーになることもあります。計算を始める前に，サブルーチンを点検して，不合理な引き数を避けるように（例えば [√‾] の前に [ABS] を置く）修正することをお勧めします。数値を変換して，大きな数を扱うのを避けることも役に立つでしょう。

[SOLVE] の内部計算手順を使ってうまく根が見付かるかどうかは，主として関数の性質と初期推定値次第です。単に根の存在が分かっているだけでは，不用意に [SOLVE] を使っても必ず根が見付かるとは限りません。関数 $f(x)$ がゼロでない水平漸近線を持つか，あるいは0でない絶対値の極小値があっても，初期推定値がこれらの無効な領域に集中していなければ，そして，当然のことですが，根が実際に存在すれば，$f(x) = 0$ の根を内部計算手順が見付けることを期待してよいでしょう。

初期推定値の選び方

[SOLVE] キーを使って方程式を解く場合，ルーチンが求根計算を開始する x の値は入力した二つの初期推定値から決まります。一般に，関数の性質がよく分かっているほど，捜したい特定の根が見付かる確率が大きくなります。現実に即した合理的な推定値を使うと，求根計算は非常に容易になります。

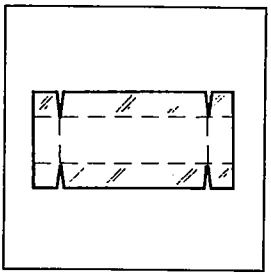
初期推定値を選ぶ基準は幾つかあります。

解として意味があると考えられる x の範囲が限定されているときには，この範囲内で初期推定値を選ぶのが適当です。実際の問題に適用した方程式は希望する解のほかに，物理的に無意味な根も含むことがよくあります。これは，解こうとしている方程式が x の一定の限界内でしか意味を持たないときに発生します。この制約を理解してそれに応じ

て結果を解釈することが大切です。

x の値が変化したときの関数 $f(x)$ の傾向について予備知識があると，関数のゼロ値の近くに初期推定しやすくなります。また，関数値が比較的に変わらなかったり，関数の大きさが極小になるような厄介な x の範囲を避けることもできます。

例　4 dm (＝40 cm) と 8 dm の長方形の鉄板を使って，容積が 7.5 リットルの上方に開いた箱を作りたいと思います。鉄板の折り曲げ寸法を幾らにしたらよいでしょうか。(箱の高さは低いのよりも高い方が良いのです)。



解法　答として求めたいのは指定の容積の箱の高さ（つまり鉄板の四隅の折り曲げ寸法）です。箱の高さ（折り曲げ寸法）を x とすると，長さが $(8-2x)$ で幅が $(4-2x)$ になります。容積 V は次式の通りです。

$$V=(8-2x)(4-2x)x$$

この式を展開してホーナー法 (79 ページ) を使って書き直すと下式になります。

$$V=4((x-6)x+8)x$$

$V=7.5$ から次式を満足する x の値を求めることになります。

$$f(x)=4((x-6)x+8)x-7.5=0$$

$f(x)$ を計算するサブルーチンは次の通りです。

キー操作	表　示	
g P/R	000-	プログラム入力状態。
f LBL 3	001-42,21, 3	ラベル。
6	002- 6	スタックに x が入れてあると考えます。

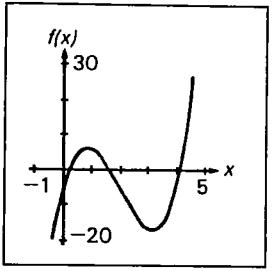
キー操作	表 示		
[−]	003−	30	
[×]	004−	20	$(x-6)x$
8	005−	8	
[+]	006−	40	
[×]	007−	20	$\{(x-6)x+8\}x$
4	008−	4	
[×]	009−	20	$4\{(x-6)x+8\}x$
7	010−	7	
[·]	011−	48	
5	012−	5	
[−]	013−	30	
[g] [RTN]	014−	43 32	

高さが高くて底面積が狭いか，高さが低くて底面積が広いのでも希望の容積の箱が作れそうです。ここでは高い方を希望しているので，高さの初期推定値は大きい方がよいでしょう。しかし物理的に 2 dm より高くは出来ません（鉄板の幅が 4 dm しかないからです）。そこで初期推定値は 1 と 2 dm が適当でしょう。

キー操作	表 示		
[g] [P/R]			計算状態。
1 [ENTER]	1.0000	}	初期推定値。
2	2		
[f] [SOLVE] 3	1.5000		求める高さ。
[R↓]	1.5000		一つ前の推定値。
[R↓]	0.0000		根のときの $f(x)$ の値。

高さを 1.5 dm とすると，5.0×1.0×1.5 dm の箱が求めるものです。

高さの上限を無視して初期推定値を 3 と 4 dm（これでも幅よりは小さい）にすると，高さが 4.2026 dm という物理的に意味のない答になります。初期推定値をもっと小さくして 0 と 1 dm にすると，高さが 0.2026 dm となって，こんな箱は浅過ぎて使いものになりません。



$f(x)$ のグラフ

関数の性質を調べる手助けとして，プログラム用メモリーに入っているサブルーチンを使っていろいろな x のときの関数値を簡単に計算できます。それにはまずスタック全体に x を入れてから，サブルーチンを実行すると関数値が計算できます（ f 文字ラベルか GSB 数字ラベルを押します）。

計算した数値をグラフ用紙に記入すると関数のグラフが作れます。この操作は関数の性質が分からないときに使うととても便利です。一見単純そうな関数でも，予期しなかったような極端な変化のグラフになることがあります。関数が部分的に大きく変化する領域に根があるときには，初期推定値を根の近くに指定しないと根を見付けることは困難でしょう。

関数の性質や根の位置について予備知識がないとか直観的に理解できないときには，手探り法で根を捜すようになります。根がうまく見付かるかどうかは関数自体の性質にもよります。手探り法は成功することもありますが，いつでも成功するとは限りません。

- 適当な大きさの正か負の二つの推定値を入力すれば，関数のグラフに水平の漸近線がないときには（三角関数のように関数が何回も振動しなければ），計算機は正か負のなるべく遠くにある根を捜し出します。
- すでに根が一つ求まっていれば，その根から比較的離れた推定値を使うと，別の根が求められます。

- 関数の中には引き数が0に接近すると特異な性質があるものがあります。問題の関数に種々の引き数を代入すればその関数の値を0にする x の値が分かるので，その値かそれに近い値を推定値にします。

[SOLVE] を使うときには普通二つの異なる初期推定値を入れますが，[SOLVE] はXとYレジスタに同じ推定値を入れても計算可能です。XとYレジスタの推定値が同じときは計算機中で第2の推定値を自動的に作ります。もし入力した推定値が同じで0でないときは，第2の推定値は第1推定値の有効数字の7けた目に1を加えたものになります。また入力した推定値が同じで0のときは，第2の推定値として 1×10^{-7} を使います。

プログラム中での[SOLVE]

プログラムの一部として[SOLVE]も使えます。プログラム中で[SOLVE]演算に入る直前に必ず初期推定値をXとYレジスタに入れるようにします。[SOLVE]の内部計算手順はXレジスタに x の値を，Zレジスタにその関数の値を入れて終了します。この x が根であればプログラムは次の行に進みます。x が根でなければ次の行を飛び越します。(根についてももっと詳しい説明は226ページの結果の解釈を見てください。)実質的には，[SOLVE] 命令で x の値が根かどうかを判断し，続いて〝真なら次へ〟の原則通りに働きます。この性能があるので，プログラムで根を求められないときは，新しい初期推定を選ぶか，あるいは関数の引き数を変えるなどして，計算を続けることができます。

プログラム命令として [SOLVE] を使うと，計算機の7重までのサブルーチンの内の1重分を使います。[SOLVE] で呼び出すサブルーチンでもう1重分を使うので，利用できるサブルーチンは5重までが限度になります。一方キー操作だけで実行すると，[SOLVE] それ自体はサブルーチンではないので，[SOLVE] で呼び出すサブルーチンは6重までが限度になります。もし7重まで重ねていて，更にも一つサブルーチンを呼び出そうとすると，**Error 5** を表示します。(106ページ参照。)

SOLVE 使用上の制約

SOLVE の使用上の一つの制限は SOLVE を再帰的に使えないことです。つまり，SOLVE の実行中に呼び出すサブルーチン内では SOLVE が使えません。こうすると実行を停止して **Error 7** を表示します。しかし，SOLVE と f を組み合せて使うことはできます。

メモリーの必要量

SOLVE はその演算に共通部分のメモリーレジスタ 5 個分を使います。(メモリーのレジスタの自動的配分については付録 C で説明します。)もし未使用のレジスタが 5 個利用できないときには，SOLVE を実行しないで **Error 10** を表示します。

SOLVE と f を組み合せた計算ではレジスタ 23 個分が必要です。

詳細な解説の紹介

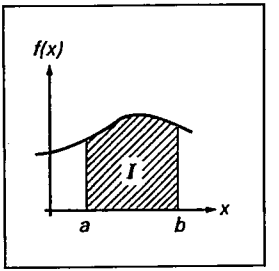
付録 D で SOLVE の高度な利用法や SOLVE を使うときの技法や説明をします。その主な内容は次の通りです。

- SOLVE の仕組み。
- 根の正確さ。
- 結果の解釈。
- 多根の求め方。
- 計算時間の短縮。

第14章

数値積分

数学や物理，工学の分野ではある関数の定積分を求める問題がかなりあります。関数を $f(x)$，積分区間を a から b までとすると，積分は数学的に次式のように表記します。



$$I=\int_a^b f(x)\,dx$$

I は図形的には曲線 $f(x)$，x 軸，区間の $x=a$ と $x=b$ で囲んだ部分の面積になります。*

積分の計算が解析的に困難か不可能なときには，数値積分でしか計算できません。この数値積分はコンピュータ用のかなり複雑なプログラムを使って計算するのが普通です。しかし，HP-15C の [∫f]（積分）キーを使うと，数値積分を簡単に計算できます。†

[∫f] の使用法

[∫f] の使用法の基本的原則は次の通りです。

1. プログラム入力状態で，積分したい関数 $f(x)$ の値を計算するサブルーチンをキー入力する。このサブルーチンは必ずラベル命令（[f][LBL] ラベル）で始めて，最後に X レジスタに $f(x)$ の値を残すようにします。

---

* ただし積分区間内で $f(x)$ が負の値にはならないものとします。

† [∫f] 機能は虚数スタックを使いません。複素数計算状態での [∫f] の使用法は HP-15C Advanced Functions Handbook を見てください。

計算状態で，

2．積分の下限値 (a) をXレジスタに入れて，[ENTER] を押してYレジスタに移す。

3．積分の上限値 (b) をXレジスタに入れる。

4．[f][∫] キーを押してから目的のサブルーチンのラベルのキーを押す。

例　物理や工学でベッセル関数を使います。0次の第1種ベッセル関数は次式の通りです。

$$J_0(x)=\frac{1}{\pi}\int_0^{\pi}\cos(x\sin\theta)\,d\theta$$

そこで $x=1$ のときの値を計算してみましょう。

$$J_0(1)=\frac{1}{\pi}\int_0^{\pi}\cos(\sin\theta)\,d\theta$$

プログラム入力状態にして関数 $f(x)=\cos(\sin\theta)$ を計算する次のサブルーチンをキー入力します。

キー操作	表　示		
[g][P/R]	000–		プログラム入力状態。
[f]CLEAR[PRGM]	000–		プログラム用メモリーを消す。
[f][LBL]0	001–42,21, 0		サブルーチンを [LBL] 命令で始める。サブルーチンはXレジスタに θ の値が入っているものとする。
[SIN]	002–	23	$\sin\theta$ の計算。
[COS]	003–	24	$\cos(\sin\theta)$ の計算。
[g][RTN]	004–	43 32	

計算状態に戻して積分の下限値をYレジスタに上限値をXレジスタに入れます。更にこの問題では三角関数の計算に角度をラジアン単位にする必要があります。

キー操作	表　示	
[g] [P/R]		計算状態。
0 [ENTER]	0.0000	下限値の0をYレジスタに入れる。
[g] [π]	3.1416	上限値のπをXレジスタに入れる。
[g] [RAD]	3.1416	角度をラジアン単位に指定。

これで [f] [∫] 0 と押せば積分を計算できます。こうしても，[SOLVE] のときと同様に直ぐには答を表示しません。HP–15C はうまい反復計算手順を使って積分を計算します。簡単に言えば，この内部計算手順は積分区間内の多数の x 点で積分する関数の $f(x)$ を計算します。その度に計算機が関数値の計算用に書き込んだサブルーチンを呼び出します。[SOLVE] のときと同様に，サブルーチンを何回も実行するので答が直ちに求まることは期待できません。大部分の積分は30秒～2分位かかりますが，積分によってはもっと時間がかかることがあります。計算時間の短縮法は後で説明しますが，ここでは [f] [∫] 0 を押して，HP–15C が面倒な計算をしている間少し休んで（または先を読んで）ください。

キー操作	表　示	
[f] [∫] 0	2.4040	$=\int_0^{\pi} \cos(\sin\theta)\, d\theta$

一般に積分の外部に定数があれば，それを加減乗除することを忘れないでください。この例では $J_0(1)$ にするために積分に $1/\pi$ を掛ける必要があります。

キー操作	表　示	
[g] [π]	3.1416	
[÷]	0.7652	$J_0(1)$

HP–15C が $f(x)$ を計算するサブルーチンを呼び出す前に，[∫] の内部計算手順は（[SOLVE] の内部計算手順と同様に）x の値を X，Y，Z，T の各レジスタに自動的に入れます。x の値がスタック全体に入っているので，サブルーチン中で x の値を記憶用レジスタから呼び出す必要はありません。次の 2 例のサブルーチンはこの性能の利点を使っています。（スタック・レジスタ全体に x の値を入れておいて多項式の値を計算する方法は 79 ページで説明しました。）

注　計算機がスタック・レジスタ全体に x の値を入れるので，それまでスタック内にあった数値が消えてしまいます。そこで，積分計算後必要な中間値がスタックにあるときには計算前にそれを一たん記憶用レジスタに入れておいて後でそれを呼び出すようにしてください。

場合によっては [∫] 計算用に書き込んだサブルーチンを $f(x)$ の計算だけに使いたいことがあると思います。その関数が x の値をスタックから 2 回以上使うときには，そのサブルーチン実行前に [ENTER] [ENTER] [ENTER] を押してスタック全体に手動で x の値を入れるようにします。

例　1 次の第 1 種ベッセル関数は次式の通りです。

$$J_1(x)=\frac{1}{\pi}\int_0^{\pi}\cos(\theta-x\sin\theta)\,d\theta$$

次式の $J_1(1)$ を計算してください。

$$J_1(1)=\frac{1}{\pi}\int_0^{\pi}\cos(\theta-\sin\theta)\,d\theta$$

まず関数 $f(\theta)=\cos(\theta-\sin\theta)$ を計算する次のサブルーチンを入れます。

キー操作	表　示	
[g] [P/R]	000–	プログラム入力状態。
[f] [LBL] 1	001–42,21, 1	サブルーチンをラベルで始める。

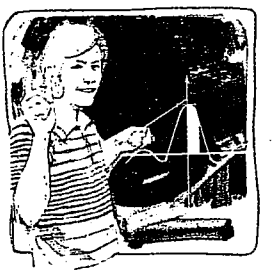
キー操作	表 示		
[SIN]	002-	23	$\sin\theta$の計算。
[-]	003-	30	[∫f] の内部計算手順がこのサブルーチン実行前にθの値をYレジスタにも入れるので，この [-] は（$\theta-\sin\theta$）の計算。
[COS]	004-	24	$\cos(\theta-\sin\theta)$ の計算。
[g][RTN]	005-	43 32	

計算状態に戻して積分区間の両端をXとYレジスタに入れます。角度がラジアン単位になっていることを確認してから [f][∫f] 1 を押すと積分を計算します。最後に$1/\pi$を掛けて$J_1(1)$を求めます。

キー操作	表 示	
[g][P/R]		計算状態。
0 [ENTER]	0.0000	下限値をYレジスタに入力。
[g][π]	3.1416	上限値をXレジスタに入力。
[g][RAD]	3.1416	（まだラジアン単位になっていないときだけ。）
[f][∫f] 1	1.3825	$=\int_0^{\pi}\cos(\theta-\sin\theta)\,d\theta$
[g][π][÷]	0.4401	$J_1(1)$

例 通信理論の問題（例えば，理想的回路網を使ったパルス伝送）では次の積分（正弦積分と呼びます）の計算が必要です。

$$Si(t)=\int_0^t\frac{\sin x}{x}\,dx$$



$Si(2)$ を計算してください。

$f(x) = (\sin x)/x$ を計算する次のサブルーチンをキー入力します*。

キー操作	表　示		
[g] [P/R]	000–		プログラム入力状態。
[f] [LBL] .2	001–42,21,.2		[LBL] 命令からサブルーチンを始める。
[SIN]	002–	23	$\sin x$ の計算。
[x≷y]	003–	34	[∫f] の内部計算手順はこのサブルーチンの実行直前に，x の値を Y レジスタにも入れるので，[x≷y] 操作によって X レジスタに x の値が入り $\sin x$ が Y レジスタに入る。
[÷]	004–	10	$\sin x/x$
[g] [RTN]	005–	43 32	

計算状態に戻して積分区分の両端を X と Y レジスタに入れます。ラジアン単位で [f] [∫f] .2 を押すと積分を計算します。

キー操作	表　示	
0 [ENTER]	0.0000	下限を Y レジスタに入力。
2	2	上限を X レジスタに入力。
[g] [RAD]	2.0000	(まだラジアン単位になっていないときだけ。)
[f] [∫f] .2	1.6054	$Si(2)$

---

*計算機が積分の下限値 $x = 0$ の点で $f(x) = (\sin x)/x$ を計算しようとすると **Error 0** を表示して (0 で割るからです) 停止するので，積分の計算はできないはずです。しかし，[∫f] の内部計算手順では積分の上下限値では関数の値を普通は計算しないので，上下限値で不定になる関数の積分も計算できます。積分区間の両端が極めて近いときか計算する点の数が極めて多いときにだけ積分区間の限界点でも関数値の計算をします。

∫f の正確さ

ある関数の積分の正確さはその関数自体の正確さ次第です。ですから，∫f を使って計算した積分の正確さはサブルーチンで計算した関数の正確さによって制限を受けます*。関数の正確さを指定するには，関数の値†が正確だと考える桁数の以上の桁数を表示できるようにします。指定した表示桁数が少ないほど積分計算が速く終ります。‡しかし関数は表示形式で指定した桁数までしか正確でないと思ってください。計算する積分値の正確さの決めかたを説明する前に表示形式の説明を少し補足しておきます。

HP-15C では FIX, SCI と ENG の3種の表示形式が使えます。大部分の積分ではどの表式形式（ただし，桁数は関数の値を考慮して正しく指定する必要があります）を使っても結果は同じになるのでどの表示形式を使うかは使い易さで決めます。普通の積分計算では SCI 表示形式を使うのが一番便利なので，以下の積分計算例では SCI を使うことにします。

> 注　表示形式を一たん指定した後でも，第10章で説明したように，桁数をIレジスタに入れておいて，f FIX I，f SCI I または f ENG I を押せば表示形式を変えられます。これはプログラムの一部として ∫f を実行するときに便利です。

---

* ある種の特性（とがった形や非常に速い振動のような）がある関数の積分の計算は不正確になることがあります。しかしこの可能性は非常に少ないのです。問題になるような関数の一般的特性と，それを処理する方法は付録 E で説明します。

† 計算した関数の正確さには関数計算途中に使う定数の正確さや計算途中の丸め誤差が関係します。このような問題点は HP-15C Advanced Functions Handbook で詳しく説明します。

‡ この理由は付録 E で説明します。

積分の正確さは関数の正確さ（表示形式で決まる）で制限を受けるので，計算機は積分値を正確に計算するのでなくて近似値を計算するだけです。HP–15C は X レジスタに近似値を入れると同時に Y レジスタに積分の近似値の最大誤差* を入れます。近似値の正確さを知りたいときには，[x≷y] を押して最大誤差を調べてください。

例　表示形式を [SCI] 2 にして (197 ページの例の) $J_1(1)$ の式の積分を計算してみましょう。

キー操作	表　示		
0 [ENTER]	0.0000		下限を Y レジスタに入力。
[g] [π]	3.1416		上限を X レジスタに入力。
[g] [RAD]	3.1416		(まだラジアン単位にしてないときだけ。)
[f] [SCI] 2	3.14	00	表示形式を [SCI] 2 にする。
[f] [∫] 1	1.38	00	[SCI] 2 のときの積分の近似値。
[x≷y]	1.88	–03	[SCI] 2 のときの最大誤差。

積分値は 1.38±0.00188 です。最大誤差は小数点下 3 桁目までしか近似値に影響しないので，この表示した近似値の数字は全部正確であると考えられます。しかし一般には，近似値の何桁目までが誤差の影響を受けないかを予測するのが困難です。それは積分する関数や積分区間，表示形式によって変わります。

---

*普通の数値積分の計算手順では近似値と実際の積分値との正確な差を計算しません。しかし HP–15C の内部計算手順ではこの差の上限を計算し，これが最大誤差です。例えば，積分 $Si(2)$ が 1.6054±0.0001 であれば，積分の近似値は 1.6054 で，その最大誤差が 0.0001 です。これは，実際の積分値とその近似値の正確な差を知ることはできないけれども，この差が 0.0001 より大きくなることはまずないという意味です。(200 ページの最初の脚注も見てください。)

もし，近似値の最大誤差が許容値よりも大きかったら，表示形式の桁数を多くして積分を繰り返すと最大誤差が小さくなります*。

積分の近似計算を繰り返すときに積分区間の上下限値をまたXとYレジスタに入れる必要はありません。積分の計算が終ったときに，近似値をXレジスタに，最大誤差をYレジスタに入れるだけでなく，積分区間の上限値がZレジスタに，下限値がTレジスタに入ります。それで積分をまた計算するために上下限値をXとYレジスタに戻すには [R↓] [R↓] と押すだけで済みます。

例 $J_1(1)$ の式の積分を，今度は小数点下2桁でなく小数点下4桁の精度で計算しましょう。

キー操作	表示		
[f] [SCI] 4	1.8826	−03	表示形式を [SCI] 4 にする。
[R↓] [R↓]	3.1416	00	上限がXレジスタに現れるまでスタックを下向きに回転移動。
[f] [∫] 1	1.3825	00	[SCI] 4 のときの積分の近似値。
[x≷y]	1.7091	−05	[SCI] 4 のときの最大誤差。

この最大誤差からこのときの近似値は少くても小数点下4桁までは正確であることが分かります。[SCI] 4 で計算したときの最大誤差は [SCI] 2 のときの最大誤差の約 1/100 になっています。一般的に言って [∫] 計算の最大誤差は表示桁数が1桁増えるごとに約 1/10 の割合で減少します。

＊ただし表示した桁数の範囲内は $f(x)$ の値を正確に計算できると仮定します。

前例の最大誤差から計算した近似値は小数点以下 4 桁までしか正確でない可能性があることが分かります。しかし近似値の全桁の一時表示で，積分の真値(実際には，小数点以下のかなりの桁数までが正確であることが分かっている近似値)と比較すれば，実際には近似値が最大誤差の表示よりもかなり正確なことが分かると思います。

キー操作	表　示	
[x≷y]	1.3825　00	表示を近似値に戻す。
[f] CLEAR [PREFIX]	1382459676	近似値の 10 桁表示。

この積分の小数点以下 8 桁まで正確な数値は 1.38245969 です。計算機の近似値をこれと比較すると，小数点以下 4 桁どころか 7 桁まで正確です。実際に近似値の最大誤差はかなり内輪に見積って計算しているので，大部分の場合は表示している最大誤差よりも計算機の近似値の方がもっと正確です。しかし近似値がどの程度まで正確であるかを確める方法までは用意してありません。

[∫] 計算の近似値の正確さと最大誤差の詳しい説明は付録 E を見てください。

プログラム中での [∫]

[∫] をプログラムの命令として入れられますが，そのプログラムをサブルーチンとして [∫] 自体が呼び出すことは出来ません。要するに [∫] を二重に使えません。従って，重積分の計算には [∫] は使えませんし，このようにすると，計算を停止して Error 7 を表示します。しかし [∫] は [SOLVE] で呼び出されるサブルーチン中に命令として入れられます。

プログラム命令として [∫] を使うと計算機の 7 重まで可能のサブルーチンの内の 1 重分を使います。[∫] で呼び出すサブルーチンの分としてもう 1 重分を使うので，実際に利用できるサブルーチンは 5 重が限度です。なおキー操作だけで実行すると，[∫] 自体ではサブルーチン

を利用しないので，[∫] で呼び出すサブルーチンは6重が限度です。もし7重のサブルーチンを利用していて，更にもう一つサブルーチンを呼び出そうとすると Error 5 を表示するのを忘れないでください。(106ページ参照。)

メモリーの必要量

[∫] は内部演算に共通部分のメモリーをレジスタ23個分使います。(メモリーの自動的配分については付録Cで説明します。) 占有してないレジスタが22個以下だと，[∫] を実行しないで Error 10 を表示します。

[∫] と [SOLVE] を組み合せたルーチンでもレジスタの使用数は23個です。

詳細な解説の紹介

この章ではいろいろな分野のために [∫] を間違いなく使うのに必要な説明をしました。付録Eで [∫] のもっと詳しい説明をします。その主な内容は次の通りです。

- [∫] の働き。
- 正確さ，最大誤差と計算時間。
- 最大誤差と表示形式。
- 間違った結果をひき起す要因。
- 計算時間が長くなる要因。
- 計算の実行途中の近似値。

エラー発生原因

計算途中で不適当な操作(例えば 0 で割る)をすると，計算機は Error の文字と数字を表示します。そのエラー表示を消すには任意のキーを 1 個押します。こうするとエラー直前の表示に戻ります。

HP–15C のエラー表示は次の通りです。(Error 2 の説明中に統計計算用の公式も載せました。)

Error 0　数学的に不適当な計算

計算に使った数値が不適当です。

[÷]　$x = 0$ のとき。

[y^x]　次のどれかです。
- 複素数計算状態でないときに，$y < 0$ で x が整数でない。
- 複素数計算状態でないときに，$y = 0$ で $x = 0$。
- 複素数計算状態のときに，$y = 0$ で x の実数部 ≤ 0。

[$\sqrt{x}$]　複素数計算状態でないときに，$x < 0$。

[1/x]　$x = 0$ のとき。

[LOG]　次のどちらかです。
- 複素数計算状態でないときに，$x \leq 0$。
- 複素数計算状態のときに，$x = 0$。

[LN]　次のどちらかです。
- 複素数計算状態でないときに，$x \leq 0$。
- 複素数計算状態のときに，$x = 0$。

[SIN^{-1}]　複素数計算状態でないときに，$|x| > 1$。

[COS^{-1}]　複素数計算状態でないときに，$|x| > 1$。

[STO][÷]　$x = 0$ のとき。

[RCL][÷]　指定したレジスタの内容が 0 のとき。

[Δ%]　Y レジスタの値が 0 のとき。

[HYP^{-1}][COS]　複素数計算状態でないときに，$x < 1$。

[HYP^{-1}][TAN]　複素数計算状態でないときに，$|x| < 1$。

[Cy,x] または [Py,x]　次のどれかです。

- x または y が整数でない。
- $x < 0$ または $y < 0$。
- $x > y$。
- x または $y \geqq 10^{10}$。

Error 1 不適当な行列演算

行列演算でない演算を行列に実行した。つまり，関係するレジスタ (X, Y レジスタまたは記憶用レジスタ) に行列を入れて非行列演算を実行したとき。

Error 2 不適当な統計計算

[x̄]	$n = 0$ のとき
[s]	$n \leqq 1$ のとき
[ŷ,r]	$n \leqq 1$ のとき
[L.R.]	$n \leqq 1$ のとき

下記の式の計算途中に 0 で割るか平方根の中が負数になっても Error 2 を表示します。

$$\bar{x} = \frac{\Sigma x}{n} \qquad \bar{y} = \frac{\Sigma y}{n}$$

$$s_x = \sqrt{\frac{M}{n(n-1)}} \qquad s_y = \sqrt{\frac{N}{n(n-1)}} \qquad r = \frac{P}{\sqrt{M \cdot N}}$$

$$A = \frac{P}{M} \qquad B = \frac{M\Sigma y - P\Sigma x}{n \cdot M}$$

$$\hat{y} = \frac{M\Sigma y + P(n \cdot x - \Sigma x)}{n \cdot M}$$

ただし

$$M = n\Sigma x^2 - (\Sigma x)^2$$

$$N = n\Sigma y^2 - (\Sigma y)^2$$

$$P = n\Sigma xy - \Sigma x \Sigma y$$

(A と B は $y = Ax + B$ の係数で) [L.R.] を押すと，それぞれ Y と X レジスタに入ります。

Error 3 レジスタ番号または行列の要素が不適当

使おうとした記憶用レジスタが無いか，使おうとした行列の要素が無い。

Error 4 行番号またはラベルが不適当

呼び出した行番号を使っていないかあるはずがない(> 448)，または入れようとしたプログラム行用のメモリー容量が不足，または呼び出したラベルが無い。

Error 5 サブルーチンが多重過ぎる

サブルーチンが 7 重を超えた。

Error 6 フラグ番号が不適当

9 より大きいフラグ番号を使おうとした。

Error 7 再帰的な [SOLVE] または [∫] の使用

[SOLVE] で呼び出したサブルーチン中に [SOLVE] 命令があった，または [∫] で呼び出したサブルーチン中に [∫] 命令があった。

Error 8 根がない

指定した推定値からは [SOLVE] で根を見付けられない。

Error 9 修理が必要

自己診断中に回路不良を発見したか，キー動作試験中に間違ったキーを押した。付録 F を見てください。

Error 10 メモリー不足

ある演算を実行しようとしたらメモリーが不足した。

Error 11 行列の引き数が不適当

次の行列演算をしようとしたら行列の引き数が一致しないか不適当な場合。

[+] または [−] 大きさが同じでない。

[×] 次のどちらか。

- 大きさが同じでない。
- 結果行列として演算行列のどちらかを指定した。

[1/x] 正方行列でない。

スカラー／行列 [÷] 正方行列でない。

[÷] 次のどれか。

- X レジスタにある行列が正方でない。
- 大きさが同じでない。
- 結果行列として X レジスタにある行列を指定した。

[MATRIX] 2 入力がスカラーであるか，行数が奇数である。

[MATRIX] 3 入力がスカラーであるか，列数が奇数である。

[MATRIX] 4 入力がスカラーである。

[MATRIX] 5 次のどれか。

- 入力がスカラーである。
- 大きさが同じでない。
- 結果行列として演算行列を指定した。

[MATRIX] 6 次のどれか。

- 入力がスカラーである。
- 結果行列を含めて大きさが同じでない。
- 結果行列として演算行列を指定した。

[MATRIX] 9 正方行列でない。

[RCL] [DIM] [I] R_I の内容がスカラーである。

[DIM] [I] R_I の内容がスカラーである。

[STO] [RESULT] 入力がスカラーである。

[Py,x] 列数が奇数である。

[Cy,x] 行数が奇数である。

Pr Error (*Power Error*, 電源のエラー)

電池の消耗などのために不揮発性メモリーの内容が消えた。

スタック上昇とラスト *x* レジスタ

HP–15C は普通の計算手順通りに操作すればよいように設計してあります。この本で説明してきたように，大部分の計算では自動メモリー・スタックの操作を考える必要はありません。

しかし，プログラムを作るときなどに，どういう操作をするとスタック内でデータが移動するかを知っておくと便利です。以下の説明が参考になると思います。

数値入力の区切り

計算機の大部分の操作は，プログラム中の命令として実行しても，キーを押したときでも，数値入力を区切り（打ち切り）ます。つまり計算機はそれ以後にキー入力する数字は前のとは全然別の数値だと判断するわけです。

次の数値入力用キーだけは数値入力を区切らない（打ち切らない）操作です。

[0] から [9] まで　　[CHS]　　[←]
[.]　　[EEX]

スタック上昇

この計算機のキー操作（とプログラム命令の実行）はその次に X レジスタに数値を入れたときスタック上昇するかどうかによって 3 種類に分類できます。つまり，スタック上昇が可能でない操作，スタック上昇が可能な操作，それにスタック上昇に無関係な操作です。

計算機が複素数計算状態のときには，どの操作も実数スタックと虚数スタックの両方に関係します。スタック上昇の影響は両スタックとも同じです。更に，[←] と [CLx] 以外の全操作後に表示（実数 X レジスタ）へ数値をキー入力するとそれと同時に虚数 X レジスタに 0 が入ります。

スタック上昇が可能でない操作

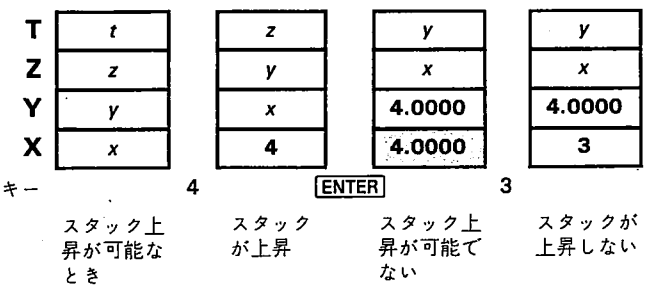
スタック上昇 HP-15C にはスタック上昇が可能でない操作が 4 種あります。*このスタック上昇が可能でない操作とは，そのキーを押した後に新しい数値を入れても，表示していた X レジスタの数値が変わるだけで，それ以外のスタック内の数値は移動しないからです。この 4 種の操作は次のキーです。

[ENTER] [CLx] [Σ+] [Σ-]

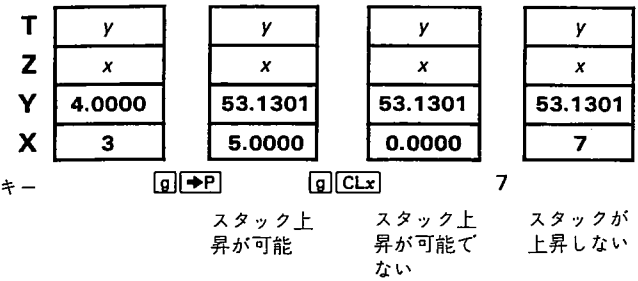
虚数 X レジスタ [ENTER]，[Σ+] または [Σ-] のキー操作後に新しい数値を表示（実数 X レジスタ）にキー入力するか呼び出すと，虚数 X レジスタに 0 が入ります。しかし [←] または [CLx] キー操作後に次の数値をキー入力するか呼び出しても虚数 X レジスタの数値は変わりません。

スタック上昇が可能な操作

スタック上昇 大部分のキー操作は，[x^2] や [×] のような単項演算や二項演算キーも含めて，スタック上昇が可能な操作です。そのキー操作後に数値をキー入力するとそれまでスタックにあった数値が一つずつスタックを上昇します（スタックが上昇可能になっているからです）。実数スタックと虚数スタックのどちらも上昇します。(網点がかかっている X レジスタは，次の数値をキー入力するか呼び出すとその内容が書き換わるという意味です。)



*36 ページの脚注も見てください。



虚数X レジスタ スタック上昇可能な全機能の後に次の数値を表示にキー入力するか呼び出すと自動的に虚数X レジスタに0 を入れます。

スタック上昇に無関係な操作

スタック上昇 FIX のようにスタック上昇に無関係な操作が少しあります。つまりその前の操作によってスタック上昇するかしないかが決まる操作です。例えばスタック上昇可能でない操作の ENTER を押した後で f FIX *n* と押し，それから新しい数値を入れてもそれまで表示していたX レジスタの値が変わるだけでスタックは上昇しません。またスタック上昇が可能な操作（例えば $\sqrt{x}$）の後で FIX *n* を押して，それから新しい数値を入れるとそれまでスタック内にあった数値が一つずつ上昇します。*

HP–15C では次の操作がスタック上昇に無関係です。

FIX	GRD	USER	R/S
SCI	GTO CHS *nnn*	CLEAR PREFIX	P/R
ENG	BST	CLEAR REG	(i)†
DEG	SST	CLEAR Σ	
RAD	MEM	PSE	

＊数値入力用キーはどれでも数値入力の途中ではスタック上昇に無関係です。数値入力終了後は CHS と EEX はスタック上昇が可能なキーになり，← はスタック上昇が可能でないキーになります。

† これは虚数X レジスタの内容を見る操作（f (i)）の意味です。

虚数 X レジスタ　上記の操作は虚数 X レジスタについても同じに働きます。

ラスト x レジスタ

次の操作はそれまで X レジスタにあった数値をラスト x レジスタに入れます。

[−]	[x^2]	[HYP^{-1}] [COS]	[%]
[+]	[SIN]	[HYP^{-1}] [TAN]	[Δ%]
[×]	[COS]	[→H.MS]	[→P]
[÷]	[TAN]	[→H]	[→R]
[ABS]	[SIN^{-1}]	[→DEG]	[$P_{y,x}$]*
[FRAC]	[COS^{-1}]	[→RAD]	[$C_{y,x}$]*
[INT]	[TAN^{-1}]	[LN]	[Σ+]
[RND]	[HYP] [SIN]	[e^x]	[Σ−]
[1/x]	[HYP] [COS]	[LOG]	[$\hat{y}$,r]
[x!]	[HYP] [TAN]	[10^x]	[MATRIX] 5 ～ 9
[$\sqrt{x}$]	[HYP^{-1}] [SIN]	[y^x]	[f]†

＊行列演算用として使うときを除きます。

† [f] は付録 E で説明するようにラスト x レジスタを特殊に使います。

メモリー配分

メモリー領域

記憶レジスタ，プログラムの行，高等数学用機能の実行*にはどれもHP-15Cの共通メモリー領域を使います。ある特定の目的にメモリーをどれだけ利用できるかは，計算機のメモリー容量だけでなくそのときのメモリー配分によって決まります。

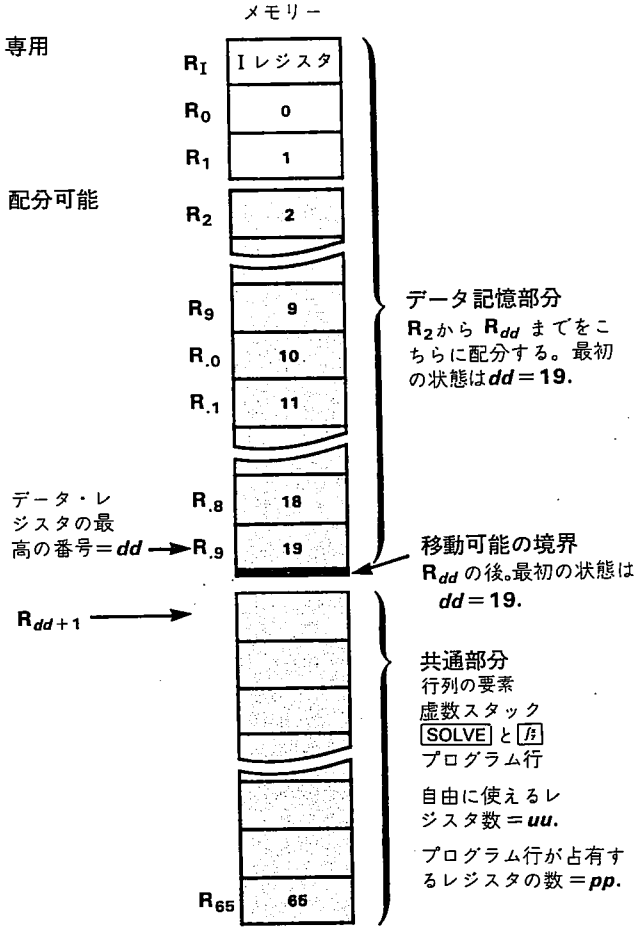
レジスタ

HP-15C のメモリー領域はレジスタ 1 個を単位にして配分します。メモリー領域は二つの部分に分かれていて，それぞれにレジスタの使い方が決まっています。この二つの部分を合計すると 67 個のレジスタになります。

- データ記憶部分のレジスタはデータ記憶専用です。最初の使用時（不揮発性メモリー内容を消した後）にはこのレジスタが全部で 21 個あります。この部分には少なくても 3 個のレジスタ（R_I，R_0 と R_1）が必ずあります。

- 共通部分のレジスタはプログラム用，行列，虚数スタックや [SOLVE] と [∫] 演算に使うためのレジスタです。最初の使用時には共通部分に 46 個のレジスタがあります。

* [SOLVE]，[∫]，複素数計算状態と行列演算用には一時的に更に余分なメモリー領域が必要で，218 ページで説明します。



配分可能なメモリー数　R_2 から R_{65} までの64個のレジスタ。〔(dd−1)＋uu＋pp＋(行列の要素)＋(虚数スタック)＋(SOLVE と ƒs)〕＝64。メモリー配分と間接指定では，データ・レジスタの $R_{.0}$ から $R_{.9}$ までを R_{10} から R_{19} と読んでください。

メモリーの状態（MEM）

そのとき計算機のメモリー配分を見るには，g MEM と押して MEM を押し続けると見られます。*表示には四つの数が次の形で現れます。

dd uu pp-b

ここで
dd = データ記憶部分にある最高のレジスタ番号(R_0 と R_1 もあるのでデータ・レジスタの全数は dd+2 になります)。
uu = 共通部分にある自由に使えるレジスタ数。
pp = プログラム命令を入れたレジスタ数。
b = uu と pp を変えずに入れられるバイト数(uu が 1 減ると 7 バイト増えて，pp も 1 増える)。

工場出荷後に HP-15C のスイッチを始めて入れたときと，メモリー内容を全部消したときには下記のようになっています。

19 46 0-0

データ記憶部分と共通部分の移動可能の境界はいつでも R_{dd} と R_{dd+1} の間にあります。

メモリーの配分指定

メモリーには 67 個のレジスタがあって，どれも 7 バイト分の容量です。この内の 64 個のレジスタ(R_2 から R_{65} まで)はデータ記憶部分と共通部分の間で入れ替え可能です。

DIM (i) 機能

プログラム用などの共通メモリー部分かデータ記憶部分を増やしたいとき（両方同時には増やせません）には DIM (i) 機能† を使ってメモリー配分を指定します。その手順は次の通りです。

---

* MEM はプログラム中に入れられません。

† DIM は A から E までか I と一緒に行列の大きさ指定に使いました。しかし，こちらはデータ記憶部分の大きさ（レジスタ数）という意味で (i) と一緒に使います。

1. 配分したいデータ記憶レジスタの最高番号 dd を表示に入れます。1 ≦dd ≦65。共通部分の自由に使える（つまりプログラム用などに使える）レジスタ数は(65−dd)になります。
2. [f] [DIM] [(i)] を押します。

メモリー配分を調べる方法は2種類あります。

- [RCL] [DIM] [(i)] を押すとデータ記憶用のレジスタの最高番号 dd がスタックに入る。(こちらはプログラム可能です。)
- [g] [MEM] を押すと更に詳しいメモリー状態(dd uu pp−b)を見られます（前ページで説明しました)。

キー操作	表示	
(プログラム用メモリーは消してあるものとします)*		
1 [f] [DIM] [(i)]	1.0000	R_1, R_0 と R_I がデータ記憶用。
[g] [MEM] (押したまま)	1 64 0-0	共通部分のレジスタは64個で，プログラム命令は一つもない。
19 [f] [DIM] [(i)]	19.0000	R_{19}(R.9)が最高番号のデータ記憶レジスタです。46個のレジスタが共通部分に残っている。
[RCL] [DIM] [(i)]	19.0000	

メモリー配分の制約

新しい [DIM] [(i)] を実行するか不揮発性メモリー内容を消すまでは，配分したメモリー状態を不揮発性メモリーが保持します。1 より少なくしようすると dd = 1 になります。65 より多くしようとすると Error 10になります。

---

*プログラム用メモリーを消さないと，レジスタをプログラム用メモリー(pp)にも使っているので，自由に使えるレジスタ数(uu)が減ります。このときは，pp > 0 で b は一定しません。

レジスタを配分するときには次の点に注意してください。

- 共通部分からデータ記憶用に変えられるのは自由に使えるレジスタだけです。例えばプログラム命令が入っているレジスタをデータ記憶用に配分しようとすると Error 10（メモリー不足）になります。
- データ記憶部分からデータが入っているレジスタを共通部分に移せますが，記憶しているデータは消滅します。消えた（つまり存在しなくなった）データ記憶レジスタを使おうとすると Error 3 になります。そこで，データはレジスタが後まで使えるように小さい番号のレジスタから先に使うのが上手な使い方です。

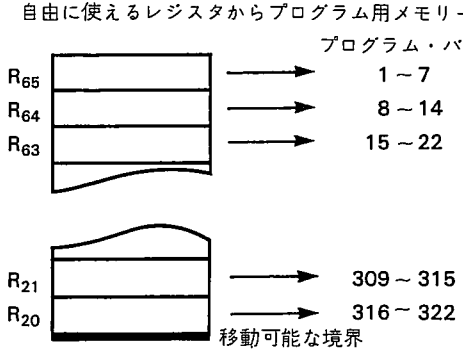
プログラム用メモリー

前に説明したように，レジスタ 1 個はメモリー 7 バイト分です。プログラム命令はメモリーを 1 バイトか 2 バイト使います。大部分のプログラム行は 1 バイトだけ使い，2 バイト使うプログラム命令は 218 ページにまとめました。

HP-15C の最大プログラム容量は 448 プログラム・バイト（転換可能のレジスタが 64 個で 1 レジスタについて 7 バイト）です。計算機の使用開始時の容量は 332 プログラム・バイト（配分したレジスタが 46 個で 1 レジスタについて 7 バイト）です。

プログラム用メモリーへの自動変換

共通レジスタ部分の中でプログラム用メモリーは必要に応じて自動的



に大きくなります。利用可能な最高番号のレジスタから一回に自由に使えるレジスタ1個がプログラム用メモリーの7バイト分に変わります。

最初のプログラム命令が R_{65} の7バイト全部を自由に使えるレジスタからプログラム用メモリーに変えます。8番目のプログラム命令が R_{64} を変えるというように，共通部分の境界までプログラム用に変えられます。データ記憶部分のレジスタ(使用開始時には R_{19} 以下)は [DIM] [(i)] を使って配分指定をしないとプログラム用メモリーとして使えません。プログラム用メモリー中のプログラム命令を消すか削除すると，7バイトごとに逆の順序で自由に使えるレジスタに戻ります。

2バイトのプログラム命令

下記の命令だけが命令一つでメモリーを2バイト使います。(これ以外の命令はどれも1バイトです。)

[f] [LBL] [·] ラベル
[GTO] [·] ラベル
[g] [CF] (*n* または [I])
[g] [SF] (*n* または [I])
[g] [F?] (*n* または [I])
[f] [FIX] (*n* または [I])
[f] [SCI] (*n* または [I])
[f] [ENG] (*n* または [I])
[f] [SOLVE]
[f] [∫]

[f] [MATRIX] {0 ～ 9}
[f] [x≷] {2 ～ 9, .0 ～ .9}
[f] [DSE] {2 ～ 9, .0 ～ .9}
[f] [ISG] {2 ～ 9, .0 ～ .9}
[STO] {[+], [−], [×], [÷]}
[RCL] {[+], [−], [×], [÷]}
[STO] [MATRIX] {[A] ～ [E]}
[STO] {[A] ～ [E], [(i)]} 利用者優先状態のとき
[RCL] {[A] ～ [E], [(i)]} 利用者優先状態のとき
[STO] [g] [(i)]
[RCL] [g] [(i)]

高等数学用機能に必要なメモリー量

4種の高等数学用機能は共通レジスタ部分から一時的に下表の個数のレジスタを使います。

機能	必要なレジスタ数
SOLVE ∫	5 23 } 同時に使えば 23
複素スタック 行列	5 行列の要素一つについて 1

SOLVE と ∫ は必要なレジスタ領域の確保と取り消しを自動的に行います。*この計算の実行中だけメモリーを占有します。

虚数スタックに使用するメモリー領域は f I, f Re≷Im または g SF 8 を押したときに確保します。CF 8 を押すと虚数スタック用の配分を取り消します。

行列の要素に使用するメモリー領域は（ DIM を使って）大きさを指定するまでは確保してありません。行列の大きさを変えると配分を変更します。MATRIX 0 で全行列の大きさが 0×0 に戻ります。

* SOLVE または ∫ の計算実行の途中にどれかのキーを押して実行を中断したら計算状態で g RTN か f CLEAR PRGM を押せばこの計算用のレジスタ配分を取り消せます。

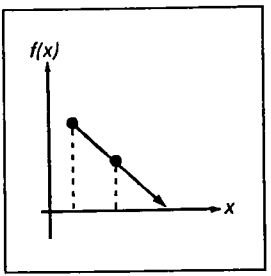
付 録 D

SOLVE の使用法の詳細

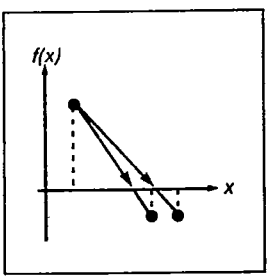
第 13 章　方程式の求根では SOLVE を効果的に使うのに必要な基本的事項を説明しました。この付録は SOLVE についてもっと詳しく説明します。

SOLVE の仕組み

SOLVE の基本的な内部計算手順を知っていると SOLVE をもっと有効に使えるようになります。



この計算手順では指定した関数のゼロ値を捜すために，前回の 2 点または 3 点の関数値を計算して関数のグラフの傾斜を近似的に推定します。次にこの傾斜を使ってグラフが x 軸と交差する点を予測します。この x を使って関数のサブルーチンを実行して新しい関数値を計算します。SOLVE の計算手順はこの過程の繰り返しです。



任意の 2 点で関数値が反対の符号になれば，この関数のグラフは 2 点間の少くも 1 点で x 軸に交差するとこの計算手順で推定します。このようにして方程式の根が見付かるまで区間を系統的に狭めます。

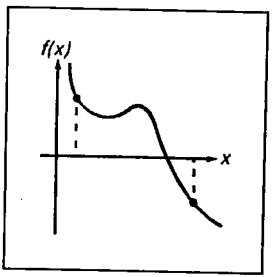
計算した関数値が 0 になるか，または二つの x の差が有効数字の 10 桁目で 1 でしかも関数値が反対符号になれば，根がうまく見つけられたことになります。そうしたら実行を停止して x の推定値を表示します。

第 13 章の 186 ページで説明しましたように，繰り返し計算の過程でこれ以外の状態になったら関数のゼロ値が明らかに存在しないことになります。それはゼロに近い関数値になりそうな新しい推定値を数学的に予測する方法がないからです。こうなると **Error 8** を表示します。

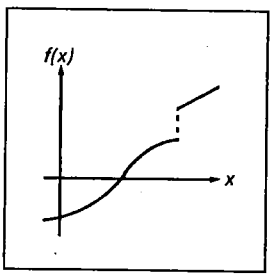
計算開始前に入力した初期推定値を x の予測に使っています。初期推定値をうまく選ぶと，予測が一段と正確になって計算時間がそれだけ短くなります。

桁あふれする限界内に根があって，次の四つの条件のどれかを満足すれば SOLVE の計算手順で必ず根を見付けます。

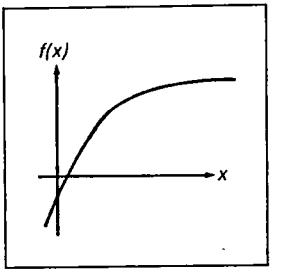
- ある二つの x の推定値で関数の符号が反対である。



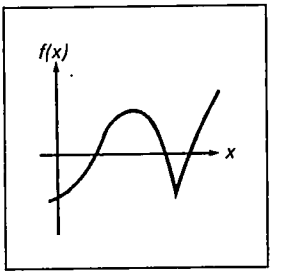
- 関数が単調関数，つまり x が増加すると関数は常に増加するか減少する。



- 関数のグラフがどこかで凸か凹である。



- 関数のグラフのゼロ値の近くに極大値か極小値のどちらかがある。



当然のことですが，誤操作などで SOLVE の計算手順が中断されないとしての話です。

根の正確さ

SOLVE キーを使って計算した方程式の根はかなり正確です。表示した根を使って計算した関数値 ($f(x)$) は正確に 0 になるか，または関数のグラフが x 軸と交差する点のそばで有効数字の 10 桁目が 1 以内になります。このような根の正確さは有効数字の 10 桁目で 2 か 3 以内です。

大部分の場合は計算で求めた根が理論的な方程式の（無限に正確な）根の正確な推定値です。しかし，条件によっては計算機内部の有限桁数の関係で理論的な期待値に一致しないような結果になることがあります。

計算して値が $1.000000000\times10^{-99}$ より小さくなると，計算機はそれを 0 として扱います。この状態を下位桁あふれと言います。関数値を計算するサブルーチンである範囲の x で下位桁あふれを生じるときには，この範囲内の根は幾らか不正確になります。例えば，次の方程式の根は $x=0$ です。

$$x^4=0$$

しかし下位桁あふれのために，初期推定値を 1 と 2にすると [SOLVE] は **1.5060 −25** という根になります。別例の次の方程式の根は無限大です。

$$1/x^2=0$$

これも下位桁あふれのために，初期推定値を 10 と 20 にすると [SOLVE] は **3.1707 49** という根になります。このどちらも，[SOLVE] の計算手順で関数値が 0 になる x の値を見付けました。下位桁あふれのことを知っていれば，このような答に納得できると思います。

計算機中では限りなく正確な数値を 10 桁に丸めているので，丸め誤差のために計算値の正確さにかなり影響することがあります。もしもサブルーチン中である範囲内の x の関数値を正しく計算できるようにしても，[SOLVE] によって求められる答が不正確になることがあります。例えば，次の方程式の根は $x=\sqrt{5}$ です。

$$|x^2-5|=0$$

10 桁の数値では厳密に $\sqrt{5}$ に等しくないので，どんな初期推定値を使っても関数値が 0 にならないし符号も変わらないので [SOLVE] で根を求めると **Error 8** になります。別例として，次の方程式は左辺がいつでも右辺より大きいので根がありません。

$$[(|x|+1)+10^{15}]^2=10^{30}$$

しかし次式の計算では初期推定値を 1 と 2 にすると桁落ちのために根として **1.0000** を見付けます。

$$f(x)=[(|x|+1)+10^{15}]^2-10^{30}$$

[SOLVE] の計算に影響するような丸め誤差が発生する原因が分かれば，それに応じて答を評価したり，丸め誤差を少くするように関数を変形することが出来ると思います。

実際の応用例では，方程式中の数値（ことによると方程式自体も）は単なる近似だけのことがあります。物理学上の数値にはそれに固有な正確さ（または不正確さ）があります。基礎となる仮定が正確でなければ，物理現象の数学的表現は単なる近似的な模型に過ぎません。このようなことや別の不正確さを知っていれば計算に役立つと思います。普通に計算したのでは非常に長時間かかる計算でも，サブルーチンのプログラムを書くときに答が実用上無視できる誤差範囲内になったら関数値が0になるようにしておくと，[SOLVE] の計算時間をかなり短縮できます。

例　チャック・ファーさんのようなリジット投げの上手な人達はリジットを高さ 105 m 以上にも投げられます。実際ファーさんはいつも高さ 107 m まで投げます。リジットの高さと時間の関係は第 13 章の 184 ページの式の通りとすると，ファーさんのリジットが 107 m まで到達するのにどれ位時間がかかるでしょうか。

解法　求める答は $h = 107$ となるときの t の値です。リジットの高さを計算する 184 ページのサブルーチンをまず入力してください。このサブルーチンを次式の計算用のサブルーチンに使うことにします。

$$f(t) = h(t) - 107.$$

$f(t)$ を計算するサブルーチンは次の通りです。

キー操作	表　示	
[g] [P/R]	000–	プログラム入力状態。
[f] [LBL] [B]	001–42,21,12	新しいラベルから始まる。
[GSB] [A]	002– 32 11	$h(t)$ の計算。

キー操作	表　示	
1	003– 1	
0	004– 0	
7	005– 7	$h(t) - 107$ の計算。
[−]	006– 30	
[g] [RTN]	007– 43 32	

第 1 回目として高さが 107 m になる時間を計算するために初期推定値を 0 と 1 秒にしてラベル [B] を使って [SOLVE] を実行します。

キー操作	表　示	
[g] [P/R]		計算状態。
0 [ENTER]	0.0000	初期推定値。
1	1	
[f] [SOLVE] [B]	4.1718	求める根。
[R↓]	4.1718	一つ前の根の推定値。
[R↓]	0.0000	根のときの $f(t)$ の値。

リジットが正確に 107 m に達するには 4.1718 秒かかるというのが答です。(この答の計算に約 1 分かかります。)

今度は関数 $h(t)$ がメートルの整数部分までしか正確でないと仮定しましょう。そうすると，計算した $f(t)$ の大きさが 0.5 m より小さくなったら $f(t) = 0$ になるようにサブルーチンを修正可能です。前回のサブルーチンを次のように修正します。

キー操作	表　示		
[g] [P/R]	000–		プログラム入力状態。
[GTO] [CHS] 006	006–	30	[RTN] 命令の直前の行。
[g] [ABS]	007–	43 16	$f(t)$ の大きさ。
.	008–	48	正確さ。
5	009–	5	
[g] [TEST] 7	010–	43,30, 7	$x > y$ かどうか調べて正確さの範囲内 ($0.5 > \|f(t)\|$) なら 0 にする。
[g] [CLx]	011–	43 35	
[g] [TEST] 0	012–	43,30, 0	$x \neq 0$ かどうか調べて x が 0 でなければ $f(t)$ に戻す。
[g] [LSTx]	013–	43 36	

もう一度 [SOLVE] を実行してみましょう。

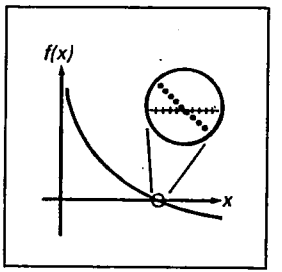
キー操作	表　示	
[g] [P/R]		計算状態。
0 [ENTER]	0.0000	初期推定値。
1	1	
[f] [SOLVE] [B]	4.0681	求める根。
[R↓]	4.0681	一つ前の根の推定値。
[R↓]	0.0000	根のときの修正した $f(t)$ の値。

4.0681 秒後にリジットは 107±0.5 m の高さです。この答は前回の答えと違っていますが，高さを求める計算の誤差を考慮すると正確な値といえます。(しかもこの計算に必要な時間は前回の半分以下です。)

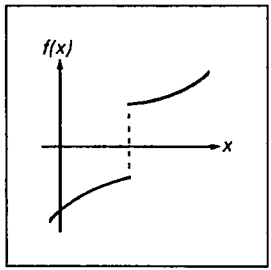
結果の解釈

[SOLVE] の計算後にある数値が X，Y，Z レジスタに入り，これが方程式の根を求めた結果を評価するのに役立ちます。*根が見付からなかったときでも，この数値には意味があります。

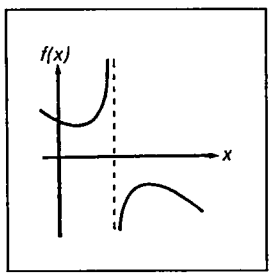
[SOLVE] で指定した方程式の根を見付けると，根と関数値が X と Z レジスタに入ります。関数値が 0 になることが期待できます。しかし，関数のグラフは計算した根からかなり近い距離内で x 軸と交差することがはっきり分かっているので，関数値が 0 でなくても容認できます。このような場合は大部分の関数値が 0 にかなり近いはずです。



T レジスタ中の数値は関数サブルーチンの最後の計算で Y レジスタに残ったものと同じです。一般にはこの数値は意味がありません。



[SOLVE] の計算で根が見付かっても関数値が 0 でないというときには特別の考慮が必要となることがあります。関数のグラフに x 軸と交差する不連続部があると，[SOLVE] は不連続部のすぐ隣の x の値を根と見なします。二つの隣接した x 値の間で関数が大きく変化するのは，連続的な移動でも関数値が非常に急激な変化をしているので納得できると思います。[SOLVE] の内部計算手順はこのようなことを区別できないので，表示した答は使う人が解釈してください。



関数が極を持つ場合には関数値が無限大に近付きます。関数値の符号が極で変わるとき，x 軸と交差する別の不連続部と同様に，その x の値を根であるかのように見なします。しかし，このような場合は根が見付かったときに Z レジスタに入る関数値は比較的大きくなります。もし極が 10 桁以内の数値で正確に表せる x 値のところにあれば，その値でサブルーチンの関数値の計算をしてみると，短時間で Error 表示になって停止するでしょう。この場合は [SOLVE] 計算は完了しません。もちろん，これはサブルーチン中で条件判断を慎重に使えば避けられます。

例　設計コンサルタントのルシー I. ビームさんは建築構造の構成部品の応力を解析して，はりのせん断応力が次式で表せることが分かりました。

$$Q=\begin{cases} 3x^3-45x^2+350 & \text{ただし } 0<x<10 \\ 1000 & \text{ただし } 0\leqslant x<14 \end{cases}$$

ここで Q はせん断応力（単位は N/m^2），x は片端からの距離（単位は m）です。任意の x のときのせん断応力を求めるサブルーチンを書い

てください。[SOLVE] を使ってせん断応力が 0 になる位置を求めてください。

解法　x が 0 と 10 の間ではせん断応力の式を効率よくプログラムに組むためにホーナー法を使って次のように書き直します。

$$Q=(3x-45)x^2+350 \quad \text{ただし} 0<x<10.$$

キー操作	表　示	
[g] [P/R]	000–	
[f] [LBL] 2	001–42,21, 2	プログラム入力状態。
1	002– 1	
0	003– 0	x の範囲の判断。
[g] [x≤y]	004– 43 10	
[GTO] 9	005– 22 9	$x \geqq 10$ のときジャンプ。
[g] [CLx]	006– 43 35	
3	007– 3	
[×]	008– 20	$3x$
4	009– 4	
5	010– 5	
[−]	011– 30	$(3x-45)$
[×]	012– 20	
[×]	013– 20	$(3x-45)x^2$
3	014– 3	
5	015– 5	
0	016– 0	
[+]	017– 40	$(3x-45)x^2+350$
[g] [RTN]	018– 43 32	サブルーチン終り。
[f] [LBL] 9	019–42,21, 9	$x \geqq 10$ のときのサブルーチン。
[EEX]	020– 26	
3	021– 3	$10^3=1000$
[g] [RTN]	022– 43 32	サブルーチン終り。

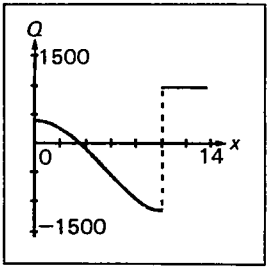
はりの外端からせん断応力が 0 の点を捜すために初期推定値 7 と 14 を使って [SOLVE] を実行します。

キー操作	表　示		
[g] [P/R]			計算状態。
7 [ENTER]	7.0000	}	初期推定値。
14	14		
[f] [SOLVE] 2	10.0000		根の可能性がある。
[R↓] [R↓]	1,000.0000		応力は 0 でない。

根の位置で応力値が大きいことは [SOLVE] で不連続部を見付けたことを意味します。これははりで応力が負から正に急激に変化する点です。はりの他端から始めて（推定値 0 と 7）もう一度 [SOLVE] を使ってみましょう。

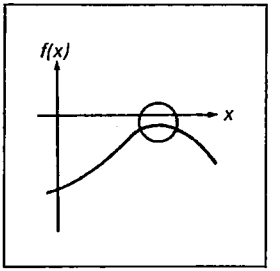
キー操作	表　示		
0 [ENTER]	0.0000	}	初期推定値。
7	7		
[f] [SOLVE] 2	3.1358		根の可能性がある。
[R↓] [R↓]	2.0000	−07	無視できる応力。

ビームさんのはりは約 3.1358 m のところで応力が 0 になり，10 m のところで応力が急激に変化します。



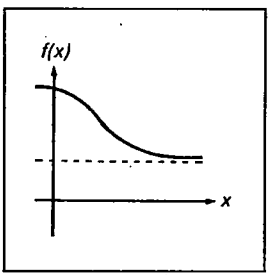
x と Q のグラフ

根が見付からなくて **Error 8** を表示したときには，[←] か任意のキーを押してエラー表示を消すと，関数値が 0 に最も近いときの x の推定値が見られます。また Y と Z レジスタの数値も調べると根の推定値の近くでの関数の性質が分かるのでこれを積極的に利用してください。



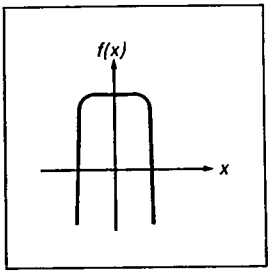
関数値の絶対値の極小付近で [SOLVE] の内部計算手順が求根計算を停止したときには **Error 8** を消してからスタックを下方に回転移動して X，Y，Z レジスタの数値を調べてください。Z レジスタに入っていた数値が 0 に比較的近いときには，多分この方程式の根が見付かって X レジスタに入っていた数値は根の理論値に非常に近い 10 桁の数値でしょう。スタックを上方に回転移動して，先の計算で求めた推定値を X と Y レジスタに戻してこれを初期推定値としてもう一度 [SOLVE] を実行すると隠れた極小付近の様子をもう少し詳しく調べられます。真の極小が見付かるとまた **Error 8** を表示し，そのとき X レジスタの数値は前回とほぼ同じになりますが，多分実際の極小の位置に一層接近しています。

関数値の絶対値の極小の位置を求めるために [SOLVE] を慎重に使うことはもちろん可能です。しかし，この場合絶対値の極小値を捜す範囲を注意して決めてください。[SOLVE] は懸命に関数の 0 値を捜そうと努力するのを忘れないでください。



水平な漸近線に到達すると（このとき広範囲の x で関数値がほぼ一定値になります）[SOLVE] の内部計算手順が求根計算を停止して **Error 8** を表示します。このとき X と Y レジスタに入っている推定値は互いにかなりかけ離れています。Z レジスタの数値は漸近線と推定される値です。X と Y レジスタに入っている数値を初期推定値としてまた [SOLVE] を実行すると，水平漸近線のためにまた **Error 8** になりますが，X と Y レジスタに入っている数値は前回と異ります。Z レジスタに入っている関数値は前回の値とほぼ同じになるでしょう。

根の捜索が関数の部分的に平らな領域に集中したために **Error 8** を表示したときには，X と Y レジスタに入っている推定値は比較的接近しているか差が極端に小さいでしょう。X と Y レジスタの数値（出来ればもう少し離れた2数値）を初期推定値として，もう一度 [SOLVE] を実行してください。関数の絶対値が極小でもなく一定でもなければ，この内部計算手順が捜索範囲を拡げてもっと意味のある答が求められるでしょう。



例　下記のサブルーチンを使って次の関数の性質を調べてください。

$$f(x)=3+e^{-|x|/10}-2e^{x^2e^{-|x|}}$$

キー操作	表　示		
[g][P/R]	000–		プログラム入力状態。
[f][LBL] .0	001–42,21,.0		
[g][ABS]	002–	43 16	
[CHS]	003–	16	
[e^x]	004–	12	$e^{-\|x\|}$
[x≷y]	005–	34	xの値をXレジスタに入れる。
[g][x^2]	006–	43 11	
[×]	007–	20	$x^2e^{-\|x\|}$
[e^x]	008–	12	
2	009–	2	
[×]	010–	20	
[CHS]	011–	16	$-2e^{x^2e^{-\|x\|}}$
[x≷y]	012–	34	xの値をXレジスタに入れる。
[g][ABS]	013–	43 16	
[CHS]	014–	16	
1	015–	1	
0	016–	0	
[÷]	017–	10	$-\|x\|/10$
[e^x]	018–	12	

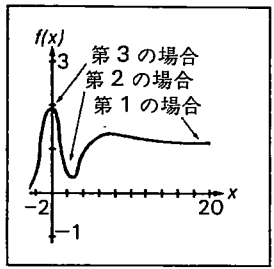
キー操作	表　示		
[+]	019–	40	$e^{-\lvert x\rvert/10}-2e^{x^2e^{-\lvert x\rvert}}$
3	020–	3	
[+]	021–	40	$3+e^{-\lvert x\rvert/10}-2e^{x^2e^{-\lvert x\rvert}}$
[g][RTN]	022–	43 32	

10, 1, 10^{-20} をそれぞれ単一の初期推定値として [SOLVE] を使います。

キー操作	表　示		
[g][P/R]			計算状態。
10 [ENTER]	10.0000		単一の初期推定値。
[f][SOLVE] .0	Error 8		
[←]	455.4335		最良の x 値。
[R↓]	48,026,721.85		一つ前の x 値。
[R↓]	1.0000		関数値。
[g][R↑][g][R↑]	455.4335		スタックを元に戻す。
[f][SOLVE] .0	Error 8		
[←]	48,026,721.85		もう一つの x 値。
[R↓][R↓]	1.0000		前回と同じ関数値（漸近線）。
1 [ENTER]	1.0000		単一の初期推定値。
[f][SOLVE] .0	Error 8		
[←]	2.1213		最良の x 値。
[R↓]	2.1471		一つ前の x 値。
[R↓]	0.3788		関数値。
[g][R↑][g][R↑]	2.1213		スタックを元に戻す。
[f][SOLVE] .0	Error 8		
[←]	2.1213		前回と同じ x 値。
[R↓][R↓]	0.3788		前回と同じ関数値（極小）。
[EEX][CHS] 20 [ENTER]	1.0000	–20	単一の初期推定値。
[f][SOLVE] .0	Error 8		
[←]	1.0000	–20	最良の x 値。
[R↓]	1.1250	–20	一つ前の x 値。
[R↓]	2.0000		関数値。

キー操作	表　示		
[g][R↑][g][R↑]	1.0000	−20	スタックを元に戻す。
[f][SOLVE].0	Error 8		前回のもう一つの x 値。
[←]	1.1250	−20	一つ前の x 値。
[R↓]	1.5626	−16	
[R↓]	2.0000		

以上三つの場合のどれも [SOLVE] はまず初期推定値の付近のグラフから予測される方向に根の捜索を始めます。初期推定値として 10 を使ったときは，[SOLVE] が水平漸近線を見付けました(値 1.0000)。1 を初期推定値としたときは，$x = 2.1213$ で極小値 0.3788 を見付けました。10^{-20} を初期推定値としたときは，関数は小範囲の x を調べたらほぼ一定値でした（関数値 2.0000)。



多根の求め方

多くの方程式は複数の根を持っています。そこでこのような複数の根を求める方法を幾つか説明します。

多根を求める一番簡単な方法は根があると思う幾つかの x の区間で根を捜す方法です。初期推定値によって最初に捜索する x の範囲が決まります。第 13 章の例では全部この方法を使いました。この方法で方程式の根が求められる場合がかなりあります。

もう一つの方法は減次法です。減次というのは根を方程式から消去することです。これは根が一つ見付かったらその根だけは根として含まないけれども残りの根はそのまま根であるように方程式を修正する方法です。

ある関数 $f(x)$ が $x = a$ で 0 になるとすると，新しい関数 $f(x)/(x-a)$ はこの領域で 0 に近付きません（a は $f(x) = 0$ の単根だとします)。このことから既知の根の消去が簡単にできます。それまでの関数サブルーチンの最後に数行追加するだけでよいのです。追加する行は x 値から既知の根(有効数字 10 桁の精度) を引いて，この差で関数を割

る命令です。多くの場合は根が単根なので，新しい関数で既知の根以外を直接 [SOLVE] できるでしょう。

これと反対に，重根の場合もあります。重根は次の性質があって，重根の位置で $f(x)$ のグラフが x 軸と交差するだけでなく，その傾斜（そして多分それより高次の導関数も）も 0 となります。既知の根が重根だったら，上のように $(x-a)$ で割るだけでは消去できません。例えば，次の方程式は $x=a$ で重根です（しかも三重根です）。

$$f(x)=x(x-a)^3=0$$

この根は $f(x)$ を $(x-a)$ で割っただけでは除けません。$(x-a)^3$ で割ったときだけ除けます。

例　減次法を使って次の方程式の根を求めてください。

$$60x^4-944x^3+3003x^2+6171x-2890=0$$

ホーナー法を使って次のように書き換えます。

$$(((60x-944)x+3003)x+6171)x-2890=0$$

多項式を計算するサブルーチンのプログラムを作ります。

キー操作	表　示	
[g] [P/R]	000–	プログラム入力状態。
[f] CLEAR [PRGM]	000–	
[f] [LBL] 2	001–42,21, 2	
6	002– 6	
0	003– 0	
[×]	004– 20	
9	005– 9	
4	006– 4	
4	007– 4	
[−]	008– 30	
[×]	009– 20	
3	010– 3	
0	011– 0	

キー操作	表　示
0	012– 0
3	013– 3
[+]	014– 40
[×]	015– 20
6	016– 6
1	017– 1
7	018– 7
1	019– 1
[+]	020– 40
[×]	021– 20
2	022– 2
8	023– 8
9	024– 9
0	025– 0
[−]	026– 30
[g] [RTN]	027– 43 32

計算状態で，二つの大きな負の初期推定値（例えば −10 と −20）をキー入力して [SOLVE] を使って負の最大の根を求めてみましょう。

キー操作	表　示	
[g] [P/R]		計算状態。
10 [CHS] [ENTER]	−10.0000	初期推定値。
20 [CHS]	−20	
[f] [SOLVE] 2	−1.6667	最初の根。
[STO] 0	−1.6667	減次のため記憶。
[R↓] [R↓]	4.0000 −06	関数値は 0 に近い。

プログラム入力状態に戻して，いま求めた根を除く命令をサブルーチンに追加します。

キー操作	表　示	
[g] [P/R]	000–	プログラム入力状態。
[g] [BST] [g] [BST]	026– 30	[RTN] の一つ前の行。
[x≷y]	027– 34	x を X レジスタに入れる。
[RCL] 0	028– 45 0	$(x-a)$ で割る。a は既知の根。
[−]	029– 30	
[÷]	030– 10	

同じ初期推定値を使って次の根を求めてみましょう。

キー操作	表　示	
[g] [P/R]	4.0000 −06	計算状態。
10 [CHS] [ENTER]	−10.0000	同じ初期推定値。
20 [CHS]	−20	
[f] [SOLVE] 2	0.4000	第二の根。
[STO] 1	0.4000	減次のため根を記憶する。
[R↓] [R↓]	0.0000	減次した関数値。

ここで第二の根を消去するようにサブルーチンを修正する。

キー操作	表　示	
[g] [P/R]	000−	プログラム入力状態。
[g] [BST] [g] [BST]	030− 10	[RTN] の一つ前の行。
[x≷y]	031− 34	x を X レジスタに入れる。
[RCL] 1	032− 45 1	第二の根の減次。
[−]	033− 30	
[÷]	034− 10	

また同じ初期推定値を使って次の根を求める。

キー操作	表　示	
[g] [P/R]	0.0000	計算状態。
10 [CHS] [ENTER]	−10.0000	同じ初期推定値。
20 [CHS]	−20	
[f] [SOLVE] 2	8.4999	第三の根。
[STO] 2	8.4999	減次のため根を記憶。
[R↓] [R↓]	−1.0929 −07	減次した関数の値は 0 に近い。

第三の根を消去するようにサブルーチンを変更する。

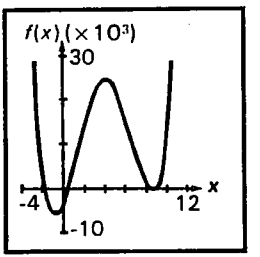
キー操作	表　示	
[g] [P/R]	000−	プログラム入力状態。
[g] [BST] [g] [BST]	034− 10	[RTN] の一つ前の行。
[x≷y]	035− 34	x を X レジスタに入れる。

キー操作	表　示	
[RCL] 2	036　45 2	} 第三の根の減次。
[−]	037−　30	
[÷]	038−　10	

第四の根を求める。

キー操作	表　示	
[g] [P/R]	−1.0929 −07	
10 [CHS] [ENTER]	−10.0000	} 同じ初期推定値。
20 [CHS]	−20	
[f] [SOLVE] 2	8.5001	第四の根。
[STO] 3	8.5001	参考のために根を記憶。
[R↓] [R↓]	−0.0009	減次した関数値は 0 に近い。

毎回同じ初期推定値を使って，4 次多項式の方程式の四つの根が求められました。しかし，最後の二つの根は互に非常に接近していて実際は一つの根（二重根）です。この根の点で 1 次分減次しても根が消えなかったのはこのためです。（丸め誤差のために x が 8.4999 と 8.5001 の間で元の関数は小さな正と負の値になります。$x = 8.5$ のとき関数が正確に 0 です。）



$f(x)$ のグラフ

消去しようと試みる根の重複度（重根の程度）は前もって分からないのが普通です。そこである根を消去して，また [SOLVE] を実行しても同じ根になったら，次のどれかで処理します。

- 別の根を捜すために減次した関数について初期推定値を変えて実行する。
- 重根を消去するためにさらに減次する。根の重複度が分からなければ，これを何回も繰り返す必要があります。

- 既知の根のそばの x 値で減次した関数の性質を調べる。関数の計算値が x 軸と滑らかに交わるときには，もっと根があるか根の重複度が大きいのです。
- 元の関数とその導関数を代数的に解析する。それで既知の根のそばの x 値での関数の性質を調べられるでしょう。(例えば，テーラーの級数展開からも根の重複度が分かります。)

計算時間の短縮

[SOLVE] を使って根を求めるときに計算時間を短縮したいこともあります。そのようなときには次のどちらかを使います。反復数を数えるか許容値を指定する方法です。

反復数を数える方法

根を探索するときに，普通は [SOLVE] で少くても 10 回は関数値を計算します。ときには 100 回以上も計算することがあります。(しかし [SOLVE] は必ず自動的に停止します。) 関数値を計算するたびにサブルーチンを一回実行するので，その回数を数えて制限することも可能です。これを簡単にするには，[DSE] か [ISG] 命令を使ってIレジスタか別の記憶レジスタで反復回数を制御するのです。

[SOLVE] を実行する前に適当な回数をそのレジスタに入れておくと，繰り返し予定数を超えたときにサブルーチンが [SOLVE] の内部計算手順から抜け出せます。

許容値を指定する方法

関数値の許容できる誤差を指定して求根の計算時間を短縮できます。計算した関数値が指定した許容値よりも小さくなったときに関数値が 0 になるようにサブルーチンを作ります。指定する許容値は実用上無視できる値か計算の正確さから決めます。この方法を使うと要求以上に正確な推定値を求めるために必要な時間が短縮できます。(224 ページの例はこの方法を使いました。)

高度な使用法の紹介

HP–15C Advanced Functions Handbook には [SOLVE] を使ったもっと高級な方法や応用を記載しました。その主な内容は次の通りです。

- 多項式に [SOLVE] を使用する方法。
- 連立方程式の解法。
- 関数の極値の求め方。
- 金融計算への [SOLVE] の使用法。
- 複素数計算状態のときの [SOLVE] の使用法。
- 方程式の複素根の解法。

付 録 E

∫ の使用法の詳細

第14章の数値積分では ∫ を使うのに必要な基本的事項を説明しました。この付録では ∫ に興味がある人のために ∫ をもう少し詳しく説明します。

∫ の仕組み

∫ の内部計算手順は積分区間内でたくさんの x の点（これをサンプル点と呼びます）での関数値の加重平均を計算して関数 $f(x)$ の積分を計算しています。このようにサンプル点を使ったときの結果の正確さはサンプル点の数によって違います。一般にサンプル点が多いほど結果が正確になります。もし無限に多くのサンプル点で $f(x)$ を計算することが可能なら（計算する関数 $f(x)$ の誤差による制約を無視すれば）内部計算手順で正確な答になります。

無限に多くのサンプル点で関数を計算すると，計算のために非常に長い時間がかかります(極端に言えば永久に終らない)。計算する積分の正確さは関数の計算値の正確さによって制限されるのでサンプル点を多くする必要はありません。有限数のサンプル点だけで計算しても，この内部計算手順は $f(x)$ の固有の最大誤差を考慮してそれよりも十分な正確さで積分の計算をします。

∫ の内部計算手順はまず少ないサンプル点だけを使って計算をして，最大誤差の近似値を求めます。この近似値の正確さが $f(x)$ 自体の正確さに及ばなければ，内部計算手順はサンプル点の数を増やして反復します(つまり計算を繰り返します)。この反復は判定回数が増えるごとに約2倍のサンプル点を使って関数値の計算を続けて，最後に積分の近似値は $f(x)$ に固有の最大誤差以内に納まる正確さに達します。

最終近似の最大誤差は関数の表示形式によって決まります*。1回分の反復を終るごとに内部計算手順は計算した近似値をその前2回の反復の近似値と比較します。この三つの近似値の内のどれか一つとそれ以外の二つとの差が最終近似で許される最大誤差よりも小さければ，計算を打ち切って，Xレジスタにその近似値，Yレジスタに最大誤差が入ります。

この三つの連続した近似値のそれぞれの誤差（つまり，真の積分値と近似値との差）が近似値自体間の差よりも大きくなる確率は非常に小さいのです。従って最終の近似値のときの誤差は最大誤差よりも小さいでしょう†。もちろん，最終の近似値のときの誤差を知ることは出来ませんが，誤差が表示する最大誤差を超えることはまずないでしょう。言い方を変えると，Yレジスタに入った最大誤差の推定値が近似値と真の積分値との差の上限になることはほとんど確実です。

正確さ，最大誤差と計算時間

[∫] の近似値の正確さは表示形式の桁数を一つ増やせば必ず変わるというものではありませんが，最大誤差は減少します。同様に，表示形式を変えると積分の計算時間が変わることも変らないこともあります。

例　4次の第1種ベッセル関数は次式の通りです。

$$J_4(x)=\frac{1}{\pi}\int_0^{\pi}\cos(4\theta-x\sin\theta)\,d\theta$$

＊表示形式と関数の最大誤差，積分の近似値の最大誤差の関係はこの付録の後の方で説明します。

†ただし $f(x)$ は急激に変化しないものとします。この点についてはこの付録の後の方でもっと詳しく説明します。

$J_4(1)$ の式中の次の積分を計算してください。

$$\int_0^{\pi} \cos(4\theta - \sin\theta)\, d\theta$$

まずプログラム入力状態に切り換えて関数 $f(\theta) = \cos(4\theta - \sin\theta)$ を計算するサブルーチンを入力します。

キー操作	表　示	
[g][P/R]	000–	プログラム入力状態。
[f]CLEAR[PRGM]	000–	
[f][LBL]0	001–42,21, 0	
4	002– 4	
[×]	003– 20	
[x≷y]	004– 34	
[SIN]	005– 23	
[−]	006– 30	
[COS]	007– 24	
[g][RTN]	008– 43 32	

これで計算状態に戻して積分区間をXとYレジスタに入力します。角度がラジアン単位になっていることを確認し，表示形式を [SCI] 2 にします。それでは [f][∫] 0 を押して積分の計算を始めましょう。

キー操作	表　示	
[g][P/R]		計算状態。
0[ENTER]	0.0000	下限をYレジスタに入力。
[g][π]	3.1416	上限をXレジスタに入力。
[g][RAD]	3.1416	角度をラジアン単位にする。
[f][SCI]2	3.14 00	表示形式を [SCI] 2 にする。
[f][∫]0	7.79 −03	[SCI] 2 のときの積分の近似値。
[x≷y]	1.45 −03	[SCI] 2 のときの最大誤差。

この最大誤差を見ると表示の近似値の数字はどの桁も正確だとは思えません。実際には，この近似値は表示した最大誤差よりもはるかに正確です。

キー操作	表　示	
[x≷y]	7.79 −03	近似値を表示に戻す。
[f] CLEAR [PREFIX] (押したまま)	7785820888	[SCI] 2 のときの近似値の 10 桁全部。

この積分の有効数字 5 桁まで正確な真の値は 7.7805×10^{-3} です。従って，この近似値の誤差は約 $(7.7858-7.7805)\times10^{-3}=5.3\times10^{-6}$ となります。この誤差は最大誤差の 1.45×10^{-3} よりかなり小さい値です。最大誤差は近似値の誤差の上限だけで，実際の誤差は一般にもっと小さいのです。

それでは [SCI] 3 で積分を計算して [SCI] 2 のときの正確さと比較してみましょう。

キー操作	表　示	
[f] [SCI] 3	7.786 −03	表示形式を [SCI] 3 に変える。
[R↓] [R↓]	3.142 00	スタックを下方に回転移動して上限を X レジスタに移動。
[f] [∫] 0	7.786 −03	[SCI] 3 のときの積分の近似値。
[x≷y]	1.448 −04	[SCI] 3 のときの最大誤差。
[x≷y]	7.786 −03	近似値を表示に戻す。
[f] CLEAR [PREFIX] (押したまま)	7785820888	[SCI] 3 のときの 10 桁全部。

[SCI] 2 と [SCI] 3 の近似値は 10 桁とも同じで，[SCI] 3 の最大誤差は [SCI] 2 の最大誤差よりも小さいのに，[SCI] 3 の近似値の正確さは [SCI] 2 の近似値の正確さよりも良くなっていません。なぜでしょうか。既に説明したように，近似の正確さは主として関数 $f(x)$ の計算をするサンプル点の数によって決まります。[f] の内部計算手順は連続した 3 回の近似値の差が表示形式によって決まる最大誤差よりも小さくなるまでサンプル点の数を増やして反復を続けます。ある特定の反復後に，三つの近似値間の差が非常に小さくなっていて最大誤差を 10 で割った値よりもっと小さいことがあります。このようなときには，表示桁数を 1 桁増やして最大誤差を小さくしても，内部計算手順はサンプル点を増やす必要がないので，近似値は最大誤差減少前と同じ結果になります。

前例の二つの近似値を実際に計算してみると，[SCI] 3 と [SCI] 2 は計算時間が変わらないことに気が付くはずです。これはある関数の積分の計算時間は，受け入れ可能の正確さになるまでに必要なサンプル点の数によって決まるからです。[SCI] 3 の近似でも，内部計算手順は [SCI] 2 の近似のときよりも多くのサンプル点を取り扱う必要がなかったので，計算時間が長くならなかったのです。

しかし，表示桁数を増やすとサンプル点を増やして計算することが必要になって，計算時間も長くなることがあります。今度は [SCI] 4 で同じ計算をしてみましょう。

キー操作	表　示		
[f][SCI]4	7.7858	-03	[SCI] 4 の表示。
[R↓][R↓]	3.1416	00	X レジスタに区間の上限が現れるまでスタックを下方に回転移動。
[f][f]0	7.7807	-03	[SCI] 4 のときの積分値。

この近似計算は [SCI] 3 や [SCI] 2 のときの約 2 倍の時間がかかります。この場合は受け入れ可能の正確さまで近似するために内部計算手順は 2 倍のサンプル点で関数の計算をする必要がありました。しかし，時間が長くかかったので，この近似値の正確さはサンプル点が半数のときよりも約 2 桁良くなっています。

前例から表示形式を変えて積分の計算をするとある場合には答の正確さが良くなるけれども，そうならないこともあることが分かりました。正確さが変わるかどうかはその関数次第で，実際に試してみるまでは分かりません。

更に一段と正確な答を求めようとすると，その代わりに計算時間が 2 倍近くかかります。より正確な答を求めるために最大誤差を減らしたいと思ったら，正確さと計算時間のどちらが大切か考えてください。

ある関数の積分計算に必要な時間は表示形式で指定する桁数だけでなく，ある程度は積分区間の幅にも関係します。もし積分計算で余りにも時間がかかるようなら，積分区間の幅（つまり限界値の差）が関数の特性に比べて広過ぎることが考えられます。しかし，大部分の問題では計算時間への積分区間の影響については考慮する必要がありません。その条件とか，その場合の処理方法については本付録の後半で説明します。

最大誤差と表示形式

$f(x)$ を計算するサブルーチンには丸めの誤差があるので $f(x)$ を正確に計算できなくて，代わりに次式を計算します。

$$\hat{f}(x)=f(x)\pm\delta_1(x),$$

ここで $\delta_1(x)$ は丸めの誤差による $f(x)$ の最大誤差です。$f(x)$ が物理量

だとすると，積分を計算したい関数は $f(x)$ でなくて次式です。

$$F(x)=f(x)\pm\delta_2(x),$$

ここで $\delta_2(x)$ は実際の物理量を $f(x)$ によって近似したために $f(x)$ と表現したときの最大誤差です。

$f(x)=f(x)\pm\delta_1(x)$ なので，積分しようとする関数は次の形になります。

$$F(x)=\hat{f}(x)\pm\delta_1(x)\pm\delta_2(x)$$

または $$F(x)=\hat{f}(x)\pm\delta(x)$$

ここで $\delta(x)$ は $f(x)$ と表現したときの正味の最大誤差です。

従って，求める積分は次式の通りです。

$$\int_a^b F(x)\,dx=\int_a^b[\hat{f}(x)\pm\delta(x)]\,dx$$

$$=\int_a^b\hat{f}(x)\,dx\pm\int_a^b\delta(x)\,dx$$

$$=I\pm\Delta$$

ここで I は $\int_a^b F(x)\,dx$ の近似値，Δx はこの近似値のときの最大誤差です。 ∫ の内部計算手順は計算終了時に I の値を X レジスタに，Δ の値を Y レジスタに入れます。

サブルーチンで計算する関数 $f(x)$ の最大誤差 $\delta(x)$ は次のようにして決めています。いま関数値が有効数字 3 桁まで正確だとして，表示形式を SCI 2 にしたとします。すると関数値の正確な部分だけを仮数部として表示します。例えば **1.23 −04** です。

X レジスタ内の数値を表示形式に従って丸めるので，関教値の暗黙の

最大誤差は $\pm 0.005\times10^{-4} = \pm 0.5\times10^{-2}\times10^{-4} = \pm 0.5\times10^{-6}$ になります。従って，表示を [SCI] n か [ENG] n に指定する（n は整数です*）と，関数値の最大誤差は次のようになります。

$$\delta(x) = 0.5\times10^{-n}\times10^{m(x)}$$

$$= 0.5\times10^{-n+m(x)}$$

この式で n は表示形式で指定した桁数，$m(x)$ は関数値を [SCI] 形式で表示したときの指数部分の値です。

最大誤差は x の関数値の大きさを表す係数の $10^{m(x)}$ に比例します。従って，[SCI] と [ENG] の表示形式では暗黙の内に関数値の大きさによって最大誤差が相対的に変わります。

同様に，関数値を [FIX] n で表示すると，表示するときの丸めから暗黙の内に関数の最大誤差は下式になります。

$$\delta(x) = 0.5\times10^{-n}$$

この最大誤差は関教値の大きさに関係がないので，[FIX] 表示形式は暗黙の内に絶対的な最大誤差になります。

[∫] の内部計算手順がある x の値のときの関数値を計算するごとに，関数の最大誤差 $\delta(x)$ も計算します。これは表示形式で指定した桁数 n を（表示形式が [SCI] か [ENG] のときには）x のときの関数値の大きさ $m(x)$ も使って計算します。数値 Δ は求めたい積分の近似値との最

---

* [SCI] 8 か 9 を指定すると [SCI] 7 と同じ表示になりますが，計算した積分の最大誤差は [SCI] 7 のときよりも小さくなります。（同じことは [ENG] 表示のときにも言えます。）負の n（これは I レジスタを使って間接に指定できます）も計算の最大誤差に影響します。最大誤差に影響する n の最小値は -6 です。R_I に -6 より小さい値（例えば -8）があっても -6 と解釈します。

大誤差で $\delta(x)$ の積分値です。

$$\Delta = \int_a^b \delta(x)\,dx$$

$$= \int_a^b [0.5 \times 10^{-n+m(x)}]\,dx$$

この積分は関数の積分の近似値が $f(x)$ のサンプルを使って計算するのと同じように $\delta(x)$ のサンプルを使って計算します。

Δ は係数 10^{-n} に比例するので，近似値の最大誤差は表示形式で指定した桁数が一つ増えるごとに約 1/10 の割合で変化します。しかし，SCI や ENG のときには，表示桁数が変わると関数を計算するサンプル点の数が変わるので，$\delta(x)$ と $10^{m(x)}$ の関係も変わるから正確には成り立ちません。

FIX で積分を近似するときには $m(x) = 0$ となるので，近似計算の最大誤差は次のようになります。

$$\Delta = 0.5 \times 10^{-n}(b-a)$$

普通は関数の近似の最大誤差を正確に知る必要はないと思います。(それには非常に複雑な解析が必要なことが多いのです。) 一般的に，関数の最大誤差を相対的に知りたいときには SCI か ENG を使う方が便利です。それに対し，関数の最大誤差を絶対的に知りたいときには FIX の方が便利です。しかし，関数値の大きさと最大誤差の両方が積分区間内で極端に小さいときには FIX の使用は不適当です（変な答になります)。同様に，関数値の大きさがその最大誤差よりもはるかに小さくなるときには SCI の使用は不適当です（この場合も変な答になります)。積分の計算結果が変だと思うときには，表示形式

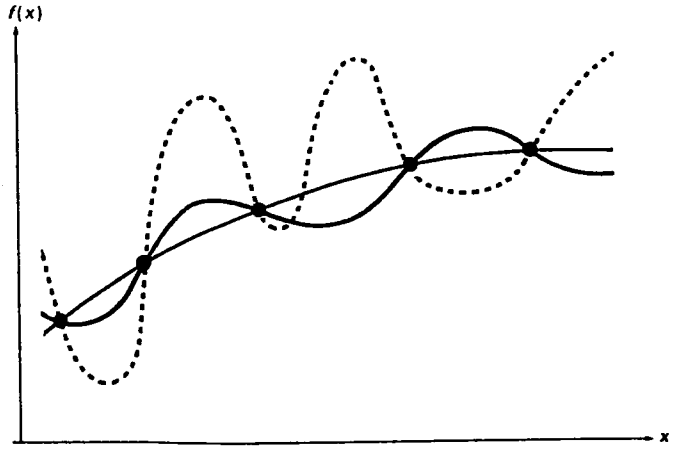
を変えて計算しなおすのが良いでしょう。

間違った答になる原因

HP-15C の [∫f] の内部計算手順はこの種の計算手順の中では最良なものの一つですが，条件によっては（大部分の数値積分の計算手順と同様に）間違った答になることがあります。この可能性は非常に少ないと言えます。[∫f] の内部計算手順は滑らかに変化する関数では正確な答を出すように設計してあります。極端な凹凸がある関数のときだけ不正確な答になる危険があります。この種の関数が現実の物理的現象に関連した問題に現れることはほとんどないと言って良いでしょうし，また現れたとしても，普通はすぐに発見して簡単に処理できます。

240 ページで説明したように，[∫f] の内部計算手順は積分区間内のたくさんの点で関数 $f(x)$ をサンプルします。このサンプル点の関数値の加重平均を計算して $f(x)$ の積分を近似します。

困ったことに，内部計算手順が $f(x)$ について知っているのはサンプル点での関数値だけなので，全部のサンプル点で $f(x)$ と同じ値になる別の関数と目的の $f(x)$ を区別できません。この事情を次ページの図で説明すると，図は有限個のサンプル点で関数値が等しくなる無限に多い関数のグラフの内，積分区間の一部について3種だけを描いたものです。



図中のサンプル点の数では，内部計算手順ではどの関数の積分計算も同じ近似値になります。実線と太い実線の実際の積分は大体同じなので，$f(x)$ がこのような関数なら近似はかなり正確です。しかし，点線の実際の積分は実線や太い実線の実際の積分と全然違うので，$f(x)$ がこのような関数ならその近似はかなり不正確になります。

[f] 内部計算手順で関数をサンプルする点数が増えるにつれて関数の形状がはっきりして来ます。ある領域での関数の変動が積分区間内の他の領域での傾向とあまり違わなくても，ある反復段階に達すると関数の変動を内部計算手順が検出します。そうすると，サンプル点を増やして反復を繰りかえして非常に急激な（関数通りの）変動も含む近似値が求められます。

一例として次の積分の近似を考えてみましょう。

$$\int_0^\infty xe^{-x}dx$$

この積分を数値計算で求めるにはこの積分区間の上端として 10^{99}（計算機にキー入力出来る実質上の最大値）をキー入力すれば良いと思うかも知れません（この場合はそう考えるのが普通ですが）。計算してどうなるか調べてみましょう。

まず関数 $f(x) = xe^{-x}$ を計算するサブルーチンをキー入力します。

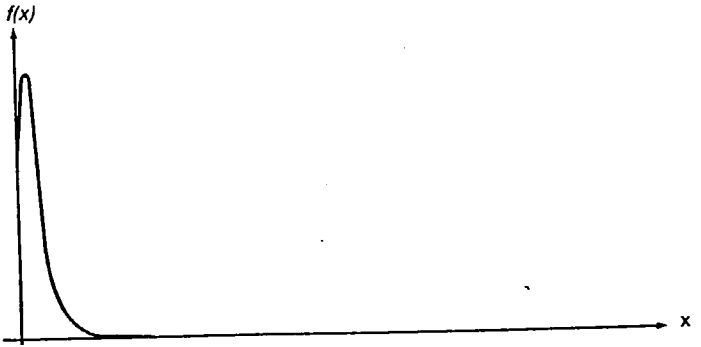
キー操作	表　示	
[g][P/R]	000-	プログラム入力状態。
[f][LBL]1	001-42,21, 1	
[CHS]	002- 16	
[e^x]	003- 12	
[×]	004- 20	
[g][RTN]	005- 43 32	

計算機を計算状態に戻します。次に表示形式を [SCI] 3 にして積分区間をXとYレジスタにキー入力します。

キー操作	表　示	
[g][P/R]		計算状態。
[f][SCI]3		表示形式を [SCI] 3 にする。
0[ENTER]	0.000 00	下端をYレジスタに入力。
[EEX]99	1 99	上端をXレジスタに入力。
[f][∫]1	0.000 00	積分の近似値。

$f(x) = xe^{-x}$ の0から∞までの真の積分値は正確に1なので，計算機が計算した答は明らかに間違っています。しかしこの関数の0から，10^{99} までの真の積分は1に非常に近いので，∞の代わりに 10^{99} を使

ったことが原因ではありません。正しくない答になった理由は積分区間内の関数のグラフを見ればはっきりすると思います。

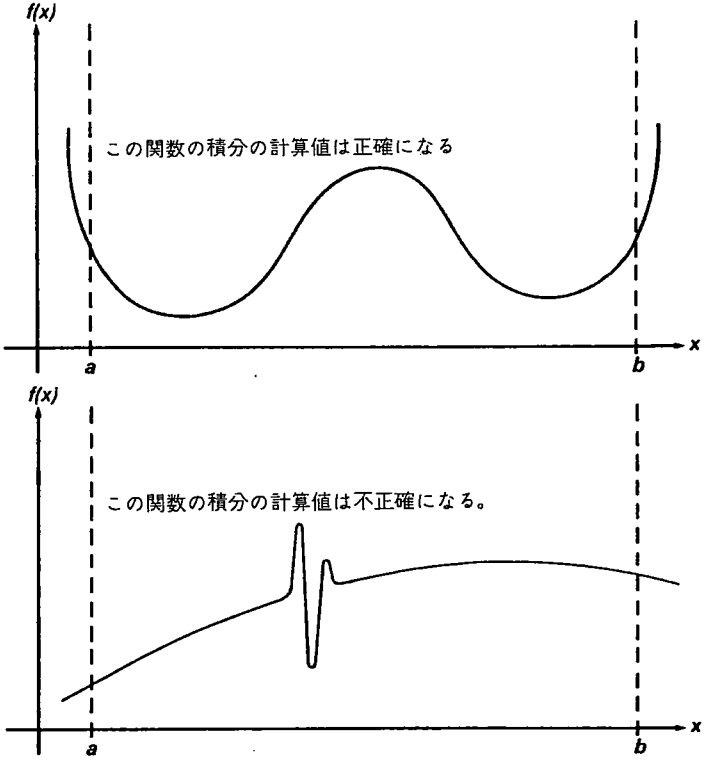


グラフは原点の近くに鋭いとがりがあります。(実際には，$f(x)$ の形を見やすくするためにとがりの幅を誇張してあります。もし積分区間を実寸通りに描くと，とがりと y 軸の線が区別できなくなります。)この場合には，どのサンプル点でもとがりを発見できなかったので，内部計算手順は積分区間全体で $f(x)$ は常に 0 であると推定しました。計算のサンプル点を増やすために [SCI] 9 にしても，この積分区間でこの関数を積分するのならサンプル点が増えないのでとがりを発見できません。(この問題の良い解決策は次の〝計算時間が長くなる原因〟で説明します。)

$f(x)$ がどこかの部分でそれ以外の区間の傾向とは全く違う変動があるときには [fx] の内部計算手順でも間違った答になる可能性があることが分かったと思います。幸いにして，このように複雑な関数はめったにありませんから，それを知らずに積分を計算することはまずないでしょう。

間違った答になる可能性がある関数は積分区間内での関数の急激な増減とその低次の導関数を調べると区別できます。基本的には，関数やその導関数の変動が早いほど，また急速に変動する導関数の次数が低いほど，[fx] の内部計算手順の終了が遅くなって，答の近似の信頼度が低くなります。

関数（またはその低次導関数）の変動の速さは積分区間の幅によって決まります。サンプル点の数が一定だとすると，変動が三つある関数 $f(x)$ でこの変動が積分区間内の狭い部分だけにあるときよりも積分区間全体に広がっているときの方が正しく計算できます。（この状況は下図を見てください。）変動を一種の振動と考えると，問題点の基準は振動の周期と積分区間の幅の比率で，この比率が大きいほど，内部計算手順が早く終って，答の近似の信頼度が高くなります。



大部分の場合は積分したい関数が分かっているので積分区間内で急速な関数値の変動があるかどうか判断できます。もし関数の性質が分からなくても，問題がありそうだと思ったら，関数値計算の目的で作ったサブルーチンを使って幾つかの点の関数値を計算してグラフに描くことができます。

積分の近似値を計算した後で，何らかの理由でその答が正しいかどうか気になったら，次のようにしてそれを確めると良いでしょう。積分区間を二つ以上に細分化して（もちろん空白や重複が無いようにします）それぞれの小区間ごとに関数を積分し，その答の近似値を合計します。この方法はサンプル点を新しく選んでサンプルするので，それまでに見付からなかったとがりが見付かるかも知れません。もし最初の近似が正しければ，それは細分化した答の合計と一致します。

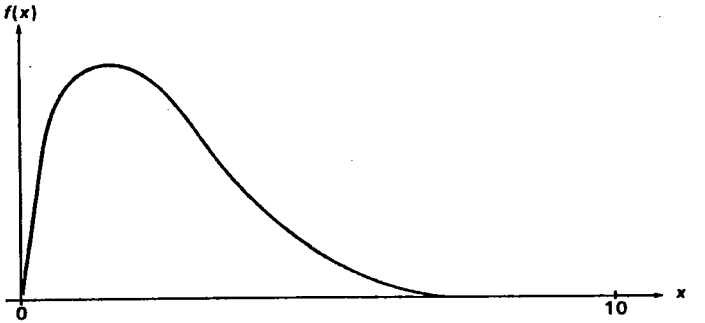
計算時間が長くなる原因

251 ページの前例では内部計算手順が関数のとがりを検出できなかったので正しくない答になりました。これは積分区間の幅に対して関数の変動が速過ぎたからです。区間の幅がもっと広ければ正しい答が求められますが，それでも積分区間がかなり広ければ非常に長い時間がかかります。

この例のような積分では，積分する関数のある特性に比較して積分区間の幅が広過ぎると計算にはなはだしく時間がかかることがあります。間違った答になるほどには区間が広くないけれども積分区間が広いためにはなはだしく計算時間がかかる積分を考えてみましょう。$f(x) = xe^{-x}$ は x が ∞ に近付くと急速に 0 に接近するので，x の大きな部分での積分の影響は無視できます。そこで積分区間の上限の ∞ の代わりに 10^{99} ほども大きくない数，例えば 10^3 を使って積分を計算してみましょう。

キー操作	表　示	
0 [ENTER]	0.000 00	下限を Y レジスタに入力。
[EEX] 3	1 03	上限を X レジスタに入力。
[∫f] [∫f] 1	1.000 00	積分の近似値。
[x≷y]	1.824 -04	近似の最大誤差。

これは正しい答ですが，しかし非常に長時間かかりました。その理由を知るために，積分区間全体の関数のグラフ（これは 252 ページのグラフと大体同じ）を $x=0$ から $x=10$ までの関数のグラフと比較してみましょう。



この二つのグラフを比較すると関数は x の小さな値のときだけが重要なことが分かります。x の値が大きい部分は関数は滑らかに次第に減少することが予想できるので，重要ではありません。

前に説明したように [∫f] の内部計算手順は引き続いて計算した近似値間の差が十分小さくなるまでサンプル点の密度を増やします。要するに，関数について十分な情報を得てサンプル点をそれ以上増やしてもほとんど変化しないような近似になるまで，内部計算手順はサンプル点の数を増やして関数値を計算します。

積分区間を (0, 10) にすると内部計算手順は重要でしかも，比較的滑かな部分だけ関数をサンプルするので，従って，数回の反復をすれば後には引き続いて計算した近似値間の差が十分小さくなるので内部計算手順が終了して希望する正確さの近似値になります。

反対に積分区間が 252 ページのグラフのようであると，大部分のサンプル点はグラフの傾斜があまり変わらないような領域で関数値を計算します。x が小さい部分の少数のサンプル点だけが 1 回の反復で関数値が大きく変わったことを検出します。そこで引き続き計算した近似値間の差が十分小さくなるまでサンプル点を増やして関数値を計算する必要があります。

狭い区間と同じ正確さで広い区間の積分を近似するためには，関数の重要な領域とサンプル点の密度を同じにすることが大切です。サンプル点の密度を同じにするためには，広い区間のときに必要なサンプル点の数が狭い区間のときに必要なサンプル点の数よりもなるかに多くなります。そのために，広い区間で同じ正確さの近似に達するまでにさらに多くの反復が必要になるので，積分の計算時間がかなり長くなります。

関数の重要な領域でいかに早くあるサンプル密度に到達するかによって計算時間が決まるので，積分区間の関数の重要でない領域の比率が大きいほど計算時間が長くなります。幸いにして，この種の積分を計算する必要があるときには，問題を変形すると計算時間をかなり短縮できます。それには積分区間を複数に分割する方法と計算式を変換する方法の 2 種があります。このどちらかを使うと積分区間か関数の計算方法が変わるので積分区間内のサンプル点を増やすよりも効果があります。(この手法は HP–15C Advanced Functions Handbook で説明します。)

計算途中の積分の近似値

積分の計算に時間がかかって待ちきれなくなったら，実行を止めて計算途中の近似値を見たくなると思います。こうすると計算途中の近似値は分かりますが，最大誤差は分かりません。

HP-15C の積分計算途中に [R/S] を押すと，走行中のプログラムの走行を中止したように計算を中止します。こうすると，関数値の計算用に入力したサブルーチンのそのときのプログラム行で計算機が停止して，その前のプログラム行の実行結果を表示します。計算を途中で止めたときには，そのときの積分の近似値は x レジスタにある数値でもないしその以外のスタック・レジスタにある数値でもありません。普通のプログラムと同様に，[R/S] を押すと停止したプログラム行からまた計算を再開します。

[∫f] の内部計算手順は毎回新しいサンプル点で関数値を計算した後でそのときの積分の近似値を更新してそれをラスト x レジスタに入れます。そこで，計算途中の近似値を見るには，普通に計算を止め，計算機が関数値の計算を終えてそれまでの近似値を更新するまでサブルーチンを（[R/S] を押して）1 ステップずつ手動操作します。サブルーチンの [RTN] 命令を実行すると近似値を更新して，ラスト x レジスタに入れるのでそれを呼び出します。

計算機が計算途中の近似値を更新しているときの表示は **running** の点滅でなく空白になります。（計算機が関数のサブルーチンを実行中は **running** を点滅表示しています。）そこで，表示が空白になったばかりのときには計算機を止めてサブルーチンを 1 ステップずつ手動操作するのを避けてください。

要約すると，計算途中の積分の近似値を見るには次の手順の操作をします。

1. 出来れば表示が空白のときに，[R/S] を押して計算機を止める。
2. 計算機が止まったら，プログラム入力状態に切り換えてそのときのプログラム行を調べる。
   - その行にサブルーチンのラベル名があれば，計算状態に戻してラスト x レジスタの内容を見ます（手順 3 へ）。

- それ以外のプログラム行だったら，計算状態に戻して [RTN] 命令（キーコードが 43 32）または 000 行（[RTN] 命令がないとき）に到達するまで 1 行ずつ（[SST] を押して）手動操作をします。（プログラム行の番号とキーコードを確認できるように [SST] を押している時間を長くしてください。）

3. [g] [LSTx] を押してそれまでの近似値を見る。最終近似まで計算を続けたいときには，[←] [+] [R/S] と押す。これでそのときの x 値がスタックに戻って計算を再開します。

高度な使用法の紹介

HP–15C Advanced Functions Handbook には [∫] の高級な手法やその応用例があります。その主な内容は次の通りです。

- 積分する関数の正確さ
- 計算時間の短縮。
- 難かしい積分の計算。
- 複素数計算状態のときの [∫] の用法。

電池，保証と修理について

電　　池

HP–15C は電池を 3 個使っています。アルカリ電池でごく普通の使用法では 6 か月以上使えます。HP–15C に付属の電池はアルカリ電池ですが，酸化銀電池（寿命が約 2 倍になります）も使用可能です。

新品のアルカリ電池ではプログラムの連続計算（電気を一番多く使っている状態です*）で 60 時間以上使えます。新品の酸化銀電池では同じ状態で 135 時間以上使えます。普通のプログラム計算ではプログラムの連続計算のときよりも電池の消耗が少くなります。数字などを表示しているだけのとき（キーを押したり，プログラム計算の途中でないとき）には電気を少ししか使わないからです。

HP–15C のスイッチを切っておくと不揮発性メモリーの内容が消えないようにほんのごく僅かに電流を流すだけなので，新品のアルカリ電池で約 1 年半，新品の酸化銀電池で約 2 年は持ちます。しかし HP–15C を長時間使わないときには電池の液漏れによる故障を防ぐために電池を外しておくことをお勧めします。

実際の電池の寿命は HP–15C をどう使うか，つまりプログラム計算の比率が多いかとか，手操作の比率が多いか，更にどんな計算に使うのかによってかなり変化します*。

HP–15C に付属の電池も，次表の交換用電池も充電できません。

---

*HP–15C の電流消費量はそのときの状態，つまりスイッチを切ってあるとき（不揮発性メモリーにだけ電流を流す），待機状態（表示しているだけ）のとき，動作中（プログラム計算や手計算中，あるいはキーを押した）のときによって変わります。スイッチを入れてあるときには待機中と動作中が混じりあった状態になります。そのため実際の電池の寿命は HP–15C がどの状態だったかによってかなり大幅に変わります。

ご 注 意

電池を充電しないでください。電池を高温の場所で保管しないでください。こうすると電池から液が漏れ出し，そのまま HP–15C 中に入れると故障の原因になります。電池を燃えている火の中に投げ込まないでください。こうすると爆発することがあります。同様に燃やすごみの中に入れないでください。

HP–15C 用の交換用電池は次の通りです。この記号は IEC（国際電気標準会議）の小型電池規格に入っているので，ヨーロッパ諸国でも同じ記号で通用します。

アルカリ電池	酸化銀電池
LR44	SR44

次の電池はアメリカ製なので上の記号と異なりますが相当品です。

Eveready A76	Eveready 357
UCAR A76	UCAR 357
RAY–O–VAC RW82	RAY–O–VAC RS76 または RW42
Varta 4276	Duracell MS76 または 10L14
	Varta 541

電圧低下の症状

電池の電圧が低下すると，表示部の左下部分に＊印が点滅します。アルカリ電池を使用のときは

- ＊印が見え始めてから連続で 1.5 時間以上使用できます*。
- スイッチを切っておけば＊印が見え始めてから約 1 か月間は不揮発性メモリーの内容が消えません。

＊これは連続してプログラム計算をしているときの最低使用可能時間です。手操作で HP–15C を使っているときは，動作と待機が組み合わさっている状態なので，＊印が見え始めてからの使用可能時間はこれよりも長くなります。

酸化銀電池を使用のときは

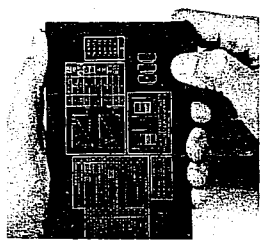
- ＊印が見え始めてから連続で10分以上使用できます。
- スイッチを切っておけば＊印が見え始めてから約１週間は不揮発性メモリーの内容が消えません。

新しい電池との交換

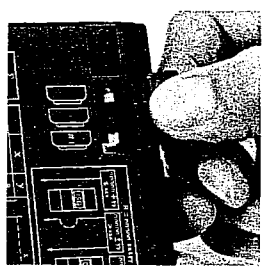
電池交換などで電池を外してもしばらくの間はHP–15Cの不揮発性メモリーの内容は消えません。（ただし電池を外す前に必ずスイッチを切ってください。）これは電池交換の間にプログラムやデータが消えないようにするためです。しかしあまり長時間電池を外したままにしておくと不揮発性メモリーの内容が消えてしまいます。

新しい電池と交換するには次のようにしてください。

1. HP–15Cのスイッチを切る。
2. HP–15Cを写真のように持って電池蓋の部分を外側に押し出して少しあけます。



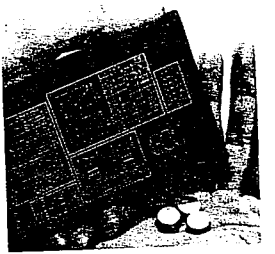
3. 電池蓋の外側をつまんで全部あけます。



ご 注 意

次の4～5で電池を外している間はキーにさわらないようにしてください。そうしないと不揮発性メモリーの内容が消えてしまったり，キーが働かなくなることがあります。

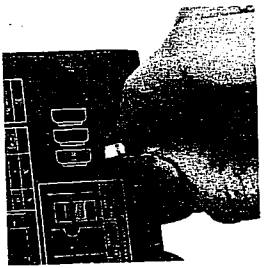
4. 電池側を下に向けて軽く振って，電池を手のひらに落してください。



ご 注 意

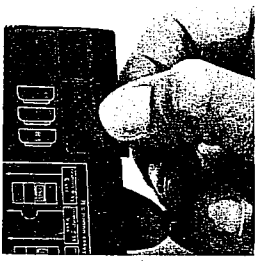
この次で入れる電池は3個とも新しい電池にしてください。1～2個だけ新しいのにすると，古い電池から液漏れしてHP-15C本体が腐食することがあります。
電池の方向（＋と－側）を間違えないでください。間違えると不揮発性メモリーの内容が消えてしまうことがあります。

5. 新しい電池を3個電池室へ入れてください。HP-15Cの裏側の図にあるように平らな面（＋マークがあります）がゴム足の方に向くようにしてください。



6. 電池蓋をつまんでHP-15C裏側の蓋のみぞに入れます。

7. 電池蓋が裏側と同じ高さになるようにしてから，電池蓋を完全にしめてください。



8. [ON]を押してスイッチを入れてください。もし不揮発性メモリーの内容が消えてしまったら **Pr Error** の表示になります。そのときはキーのどれかを押すとこの表示が消えます。それから必要なプログラムなどを入れてください。

動作の確認（自己診断）

HP-15Cのスイッチを入れても表示しないとか，正しく働かないときには次のように点検してください。

キーを押しても表示が変わらないとき

1. [ON] を押したまま[y^x]を押し [ON]を先に放します。こうするとXレジスタの内容が変わるので，後で [←] を押してください。
2. それでもまだキーを押しても変化しないときには，まず電池を外してもう一度入れ直してください。このときに電池の向きに気を付けてください。
3. それでもまだキーを押しても働かないときには電池を外してから，HP-15Cの電池側端子をゼムピンのようなものでほんのち

ょっとの間ショートしてみてください。(電池側端子の両側のプラスチックフィルムを外側に曲げないとショートできないかも知れません。)
ショートするのはほんの短時間だけです。このようにすると不揮発性メモリーの内容が消えてしまうので，電池を入れ直してから [ON] を 2 ～ 3 回押す必要があると思います。

4. それでもまだ働かなかったら，新しい電池と交換してみてください。それでも働かなければその HP-15C は修理が必要です。

キーを押すと表示が変わるとき

1. スイッチを切ってから，[ON] を押したまま [×] を押します。
2. [ON] を先に放してから [×] も放してください。これは HP-15C 内の電気回路の動作試験（自己診断）の開始です。回路が正常なら約 25 秒後（それまでは **running** の文字を点滅します）に **−8, 8, 8, 8, 8, 8, 8, 8, 8, 8,** の表示と状態表示の文字を全部表示します(ただし低電圧表示の＊は表示しません)。* このときに **Error 9** の表示だったり，何も表示しなかったり，何か変な表示のときには修理が必要です。†

注 [ON] キーを押したまま [+] キーや [÷] キーを押しても同じように試験できます。* この試験は主として製造工程や修理途中のときの方法です。

＊この試験をやると HP-15C でいつもは使わないものまで表示します。

† [ON] / [×] や [ON] / [+] 試験のあとで **Error 9** を表示しても，何とか使いたいときは 63 ページの方法で不揮発性メモリーの内容を消してください。

* [ON] / [+] 試験は [ON] / [×] 試験と大体同じですが試験終了の表示が出ません。どれかのキーを押すと約 25 秒後に上のような表示になります。 [ON] / [÷] 試験はキーと表示を組合わせた試験です。[ON] を離すと表示の一部だけが見えます。試験を始めるには各段ごとに左から右に，第 1 段目が終ったら 2 段目というように全部のキーを順番に押してください。各キーを押すごとに表示が少しずつ変わります。全部のキーを順番通り正しく押すと 15 の表示になるはずです。([ENTER] キーは 3 段目のときと 4 段目のときの両方に押してください。) 計算機が正しく働かないときや，キー順序を間違えると **Error 9** を表示します。なおこのようにキーを押す順番を間違えたときに **Error** 表示が出ますが，この場合には修理の必要はありません。試験を途中で打切るにはわざとキーの順序を間違えてください（当然のことですが **Error 9** の表示になります)。**Error 9** や **15** の表示はどれかのキーを押せば消えます。

上記 2 の操作では正しく表示するけれども，計算途中で **Error** 表示をするようでしたら，**Error** 番号に応じてこの本の該当部分を読み直してください。それでもまだ分からないときはお買い上げの販売店や YHP に，手紙または電話でお問い合わせくださるとよいでしょう。

保証について

保証の内容

HP–15C は材質上や製造工程上の不具合に対してお買求めの日から 1 年間の保証をいたします。他の方に贈物としたり，転売された場合は自動的にその人に権利が移りますが，保証期間はあくまでも初めにお買求めの日から 1 年間だけです。保証期間中の修理・改造・部品交換などの費用とお客様への返送料は当社負担ですが，お客様から当社の修理担当への送料はお客様負担でお願いいたします。

保証を適用できない場合

通常使用外での故障（床面や水中などへの落下，誤使用や誤接続などを含む）の場合や当社以外で修理や改造したものは保証の対象外です。また保証期間内でも保証書がないとやはり保証の対象にはなりません。HP–15C の電池は保証の対象外です。

修　　理

当社の修理担当の宛先は下記の通りで直接こちらへお送りください。送料はお客様，返送料は YHP 負担です。

〒 229　神奈川県相模原市矢部 1-27-15
横河・ヒューレット・パッカード㈱
相模原事業所
ベンチリペアセンター
電話　0427 59 1311（大代表）

外国へ出張の場合の修理先は製品に同封の多国語版のサービスカードをご参照ください。この場合でも保証書は国際的に通用いたします（多少の例外はあるかも知れません）のでお持ち下さい。この場合でも修

理先への送料，通関手数料，関税等はお客様の負担になります。

修理科金

前記のように１年間の保証期間がありますが，保証期間切れや期間内でも保証書の添付がないときの修理は有料で機種ごとの定額制になっています。普通の使用方法以外による故障修理は別途料金になります。この場合は修理前に見積料金を連絡してご了解を得てから修理します。

修理完了後の保証期間

有料修理したものは３か月間の保証（使用部品と修理技術について）をいたします。修理保証書をご覧ください。

修理依頼品の発送について

もし修理が必要になりましたら次の方法でご返送くださるようお願いいたします。

- サービスカードに故障状況を含めて全部記入してください。
- 保証期間内の修理の場合は保証書または修理保証書を付けてください。

輸送途中の破損を防ぐためにHP-15Cをケースに入れ，サービスカード（または故障状況とお客様の住所，お名前，連絡電話番号を書いた紙）と保証書（保証期間内の場合だけ）を適当な箱に入れて当社の修理担当に郵便またはトラック便でお送りください。輸送途中の破損や紛失は保証の対象になりませんので書留小包にしたり，輸送保険を付けることをお勧めします。

現品が保証期間内でも期間満了後でも当社の修理担当までの送料はお客様のご負担でお願いします。

修理のときに不揮発性メモリーの内容（プログラムやデータなど）は全部消えてしまいます。プログラムを作ったら，プログラムのリスト，操作手順，操作例などをノートにきちんと記録する習慣をつけることをお勧めします。

修理が完了しましたら保証期間内のものは送料YHP負担でお送りいたします。保証適用外のものは代金引換郵便にてお送りいたします(大部分の外国も同様です)。お問い合わせは前記の修理担当までお願いいたします。

温度範囲

- 使用時　0～55℃
- 保管時　−40～65℃

受信障害について

HP−15Cは弱い電波を出すのでラジオやテレビ受信障害の原因となることがあります。HP−15CはアメリカFCC文書20780の規則15条J項のクラスBコンピュータの規定に適合していますが，これは一般住宅用受信機を対象にしています。したがって特殊なラジオやテレビに対しては保証しておりません。もしHP−15Cがラジオやテレビ受信の障害原因と思われましたらHP−15Cのスイッチを入り切りして確めてください。もし障害の原因なら次の方法を組み合わせて防止してください。

- 受信アンテナやフィーダの方向を変えてみる。
- HP−15Cの使用方向を変えてみる。
- 受信機とHP−15Cの距離を離してみる。

機能の要点と索引

HP–15C 本体の裏側も見てください。

[ON]

[ON]

HP–15C の電源スイッチで表示の点滅の切り換え (18 ページ)。[−]と一緒に使うと不揮発性メモリー内容の消去 (63 ページ), [·]と一緒に使うと桁区切りの変更, [y^x], [×], [+], [÷] と一緒に使うと HP–15C の動作確認 (263–264 ページ)。

Iレジスタ

[I]

index register, インデックス・レジスタ (R_I)。次の操作用の記憶レジス

タ。プログラムの間接実行(GTO か GSB によるジャンプ, ISG か DSE によるループ), フラグの間接制御, 表示形式の間接指定 (107 ページ)。複素数の入力と複素数計算状態への切り換えにも使用 (121 ページ)。

(i)

indirect operation, 間接操作。記憶, 呼び出し, レジスタの四則演算, プログラム・ループの制御のために R_I に記憶した数値を使って別のレジスタの番号指定に使用 (107 ページ)。また, DIM と一緒に使うと記憶レジスタの配分が可能 (215 ページ)。

確率

Cy,x

combination, 組み合わせ。y 種のものから一度に x 個だけ取り出して組み合わせられる方法の数を計算してスタックが下降する (47 ページ)。(行列での使用は行列機能キー, 271 ページ参照。)

Py,x

permutation, 順列。y 種のものから一度に x 個だけ取り出して並べられる方法の数を計算して, スタックが下降する (47 ページ)。(行列での使用は行列機能キー, 271 ページ参照。)

行列機能

DIM

dimension, 大きさ。指定する名前 {A~E, I} の行列の大きさを変える (141 ページ)。

RESULT

ある行列演算の結果を入れる行列を指名する (148 ページ)。

USER

利用者優先状態。こうすると STO

	か RCL {A~E, (i)} を押すたびに R_0 と R_1 にある行か列の番号が自動的に増加する (144 ページ)。
STO か RCL {A~E, (i)}	R_0 と R_1 に入れてある行と列番号を使って行列の要素を記憶または呼び出す (144, 146 ページ)。
STO g か RCL g {A~E, (i)}	X と Y レジスタに入れてある行と列番号を使って行列の要素を記憶または呼び出す (146 ページ)。
STO か RCL MATRIX {A~E}	指定の行列の記号を記憶または呼び出す (142, 147 ページ)。
STO か RCL RESULT	結果行列の記号を記憶または呼び出す (148 ページ)。
RCL DIM {A~E, I}	指定の行列の大きさを Y (行) と X (列) レジスタに呼び出す (142 ページ)。
1/x	記号を表示している行列の逆行列を計算して，それを指定の結果行列に入れ，結果行列の記号を表示する (150 ページ)。
+, −, ×	二つの行列または一つの行列とスカラーの対応する要素間の足し算，引き算または掛け算をして，結果行列に記憶する (152–155 ページ)。
÷	二つの行列では，X にある行列の逆行列と Y にある行列の掛け算を行う。一つの行列では，行列が Y にあれば，各要素を X にあるスカラーで割る。行列が X にあれば，その行列の逆行列の各要素と Y にあるスカラーを掛ける。結果行列に記憶する (152–155 ページ)。
CHS	X レジスタにある行列の全要素の符

号を変える (150 ページ)。

[MATRIX] {0 9} 行列演算。

[MATRIX] 0 全行列を 0×0 に戻す(143 ページ)。

[MATRIX] 1 R_0 と R_1 に入れた行と列番号を 1 にする (143 ページ)。

[MATRIX] 2 複素変換。Z^P を $\tilde{Z}$ に変換する (164 ページ)。

[MATRIX] 3 逆複素変換。$\tilde{Z}$ を Z^P に変換する(164 ページ)。

[MATRIX] 4 転置。X から転置行列 X^T を作る (150 ページ)。

[MATRIX] 5 転置積。Y と X から Y^TX を計算する (154 ページ)。

[MATRIX] 6 結果行列の残差を計算する (159 ページ)。

[MATRIX] 7 X レジスタにある行列の行ノルムを計算する (150 ページ)。

[MATRIX] 8 X レジスタにある行列のフロベニウスのノルムを計算する(150 ページ)。

[MATRIX] 9 X レジスタにある行列の行列式を計算する (行列の *LU* 分解も行う) (150 ページ)。

[Cy,x] 分割形 (Z^P) で記憶した行列を複素形 (Z^C) に変換する (162 ページ)。

[Py,x] 複素形 (Z^C) で記憶した行列を分割形 (Z^P) に変換する (162 ページ)。

[x=0], [TEST] 0, [TEST] 5, [TEST] 6 X, または X と Y レジスタにある行

列記号の条件判断。[x=0] と [TEST] 0 ($x \neq 0$) はXレジスタにあるのが0かどうかを判断する。行列記号は0でないと見なす。[TEST] 5 ($x = y$) と [TEST] 6 ($x \neq y$) はXとYにある記号が同じかどうかを判断する。その結果はプログラムの実行に影響して，偽なら次の1行を飛び越す (174ページ)。

三角関数と角度

[DEG]

degree，度。三角関数の角度を度単位に切り換えて，GRADやRADの文字を消す (26ページ)。複素数の三角関数には効果がない。

[RAD]

radian，ラジアン。三角関数の角度をラジアン単位に切り換え，RADの文字を表示する (26ページ)。

[GRD]

grad，グラード。三角関数の角度をグラード単位に切り換え，GRADの文字を表示する (26ページ)。複素数の三角関数には効果がない。

[SIN], [COS], [TAN]

表示 (Xレジスタ) の数値のサイン，コサインまたはタンジェントを計算する (26ページ)。

[SIN^{-1}], [COS^{-1}], [TAN^{-1}]

表示 (Xレジスタ) の数値のアークサイン，アークコサインまたはアークタンジェントを計算する (26ページ)。

数学演算

[−], [+], [×], [÷]

四則演算用で，計算後スタックが下降する (29ページ)。

[$\sqrt{x}$]

xの平方根を計算する (25ページ)。

[x^2]

xの二乗を計算する (25ページ)。

[x!] xの階乗 ($n!$) または ($1+x$) のガンマ関数 (Γ) を計算する (25 ページ)。

[1/x] 逆数を計算する (25 ページ)。(行列での使用は行列機能, 270 ページ参照。)

[π] 円周率 (π) を表示に呼び出す (24 ページ)。

[SOLVE] 利用者がラベル付きサブルーチンに書き込んだ関数 $f(x)$ の値を計算する式を使って $f(x)$ の実根を求める (180 ページ)。

[∫f] integrate, 積分。利用者がラベル付きサブルーチンに書き込んだ関数 $f(x)$ の値を計算する式を使って $f(x)$ の定積分を計算する (194 ページ)。

数値入力

[ENTER] X レジスタ (表示) にある数値を Y レジスタにコピーする。複数の数値を入力するときの数値の区切りに使う (22, 37 ページ)。

[CHS] change sign, 符号を変える。表示の数値または 10 の指数部の符号を変える (19, 124 ページ)。

[EEX] enter exponent, 指数部の入力。[EEX] の後に入力する数値は 10 の指数部 (19 ページ)。

[0] ～ [9] digit key, 数字キー (22 ページ)。

[.] decimal point, 小数点(22 ページ)。

数値変更

[ABS] absolute value, 絶対値。表示の数

値の絶対値を計算する (24 ページ)。

[FRAC] fractional portion, 小数部分。表示 (X レジスタ) の数値の整数部分を切り捨てて小数部分だけにする (24 ページ)。

[INT] integer portion, 整数部分。表示 (X レジスタ) の数値の小数部分を切り捨てて整数部分だけにする (24 ページ)。

[RND] round, 丸める。X レジスタの数値の仮数の全有効数字 (10 桁) を表示形式通りに四捨五入する (24 ページ)。

スタック操作

[x≷y] X と Y レジスタの内容の入れ換え (34 ページ)。

[x≷] X-register exchange, X レジスタの内容と指定した記憶レジスタの内容を交換。[i],[(i)],, 番号または[.]番号と一緒に使う (42 ページ)。

[Re≷Im] real exchange imaginary, 実数と虚数の交換。実数と虚数の X レジスタの内容を交換して複素数計算状態にする (124 ページ)。

[R↓] roll down, 下方に移動。スタックの内容を一つずつ下方に回転移動 (34 ページ)。

[R↑] roll up, 上方に移動。スタックの内容を一つずつ上方に回転移動 (34 ページ)。

[CLx] clear x, x をクリア。表示 (X レジスタ) の内容を 0 にする (21 ページ)。

[←] 計算状態ではキー入力した最後の数字を消すか,(数字の入力を区切ってあると)表示を0にする(21ページ)。

双曲線関数

[HYP] [SIN], [HYP] [COS], [HYP] [TAN] それぞれ,ハイパボリック・サイン,ハイパボリック・コサイン,ハイパボリック・タンジェントを計算する(28ページ)。

[HYP^{-1}] [SIN], [HYP^{-1}] [COS], [HYP^{-1}] [TAN] それぞれ,逆ハイパボリック・サイン,逆ハイパボリック・コサイン,逆ハイパボリック・タンジェントを計算する(28ページ)。

対数関係と指数関係

[LN] natural logarithm,自然対数。表示(Xレジスタ)の数値の自然対数($\log_e x$)を計算(28ページ)。

[e^x] natural antilogarithm,自然対数の真数。表示(Xレジスタ)の数値のe^xを計算(28ページ)。

[LOG] common logarithm,常用対数。表示(Xレジスタ)の数値の常用対数を計算する(28ページ)。

[10^x] common antilogarithm,常用対数の真数。表示(Xレジスタ)の数値の10^xを計算する(28ページ)。

[y^x] Yレジスタの数値を表示(Xレジスタ)数値乗する(yを先に入れてからxを入れる)。スタックが下降する(29ページ)。

統計

[$\Sigma+$] XとYレジスタの数値を使って統計

用の $R_2 \sim R_7$ に集計する（49
ジ）。

[Σ-] [Σ+] の集計を訂正するために，
Y レジスタの数値を $R_2 \sim R_7$ カ
く（52 ページ）。

[x̄] [Σ+] で集計した x と y 値の平均
算する（53 ページ）。

[s] [Σ+] で集計した x と y 値のサン
の標準偏差を計算する（53 ペー

[ŷ,r] 直線推定と相関係数。最小二乗
使って，入力した x 値のときの
推定値（$\hat{y}$）を計算して X レジ
に入れ，集計したデータから相
数 r を計算して Y レジスタに入
（55 ページ）。

[L.R.] linear regression，直線回帰。
したデータに最も良く当てはま
線の y 切片と傾斜を計算する。
片の値が X レジスタに，傾斜の
Y レジスタに入る（54 ページ）

[RAN#] random number，乱数。[STO] [
を使って記憶した種から発生す
似乱数（48 ページ）。

CLEAR [Σ] 統計用レジスタ（$R_2 \sim R_7$）の
を消す（49 ページ）。

パーセント計算

[%] percent，パーセント。Y レジス
数値の x（表示の数値）%を計
る（29 ページ）。大部分の二項
関数と違ってスタックは下降し

[Δ%] percent difference，増減率。Y
スタの数値と X レジスタの数値

増減の百分率を計算（30 ページ）。スタックは下降しない。

表示指定

[FIX] — fixed point，小数点以下の桁数固定表示にする（58 ページ）。

[SCI] — scientific notation，指数部付き表示にする（58 ページ）。

[ENG] — engineering notation，工学用指数部付き表示にする（59 ページ）。

[f] CLEAR [PREFIX] — 押している間は X レジスタの数値の仮数部の 10 桁までの有効数字を表示する（60 ページ）。前操作キーを押したのを取り消すのにも使う（19 ページ）。

複素数機能

[Re≷Im] — real exchange imaginary，実数と虚数の交換。複素数計算状態にして（虚数スタックを作る），実数と虚数の X レジスタの内容を交換する（124 ページ）。

[I] — 複素数の入力に使う。複素数計算状態にする（虚数スタックを作る）（121 ページ）。また，[DIM] と一緒に使って行列の大きさを間接に指定する（174 ページ）。（I レジスタ機能については I レジスタ，268 ページ参照）。

[(i)] — キーを押している間，虚数 X レジスタの内容を表示する（124 ページ）。

[SF] 8 — フラグ 8 をセットして，複素数計算状態にする（121 ページ）。

[CF] 8 — フラグ 8 をクリアして，複素数計算

状態を解除する（121 ページ）。

変換

[→R]

X と Y レジスタに入れた極座標系の大きさ r と角度 θ を直交座標系の x と y 座標に交換する（31 ページ）。複素数計算状態での変換については 134 ページ参照。

[→P]

X と Y レジスタに入れた直交座標系の x と y を極座標系の大きさ r と角度 θ に変換する（31 ページ）。複素数計算状態での変換については 134 ページ参照。

[→H.MS]

10 進数の時（または度）を時，分，秒（または度，分，秒）に換算する（27 ページ）。

[→H]

時，分，秒（または度，分，秒）を 10 進数の時（または度）に換算する（27 ページ）。

[→RAD]

度単位の角度をラジアンに換算する（27 ページ）。

[→DEG]

ラジアン単位の角度を度に換算する（27 ページ）。

前操作キー

[f]

キーの向こう側の黄文字の関数や機能を使うときに，そのキーを押す前にこれを押す（18 ページ）。

[g]

キーの手前側の青文字の関数や機能を使うときに，そのキーを押す前にこれを押す（18 ページ）。

これ以外の前操作キーについては表示指定(277 ページ)，メモリーと記

憶(279 ページ), プログラム用キーの索引 (280 ページ) を参照。

CLEAR [PREFIX]

前操作キーや [f][SCI] のような押し終らない命令を取り消す (19 ページ)。また, 表示数値の仮数部の有効数字 10 桁を表示する (60 ページ)。

メモリーと記憶

[STO]

store, 記憶 (貯蔵)。X レジスタの数値を指定の {0~9, .0~.9, [I], [(i)]} レジスタに記憶する(42 ページ)。また直接演算記憶にも使って, レジスタの新しい内容=レジスタの古い内容 {[+], [−], [×], [÷]} 表示数値 (44 ページ)。

[RCL]

recall, 呼び出し。指定の {0~9, .0~.9, [I], [(i)]} 記憶レジスタから数値を表示に呼び出す(42 ページ)。また直接呼び出し演算にも使う。新しい表示=それまでの表示 {[+], [−], [×], [÷]} レジスタ内容(44 ページ)。

CLEAR [REG]

記憶レジスタ全部の内容を 0 にする (43 ページ)。

[LSTx]

その前の演算直前に X レジスタにあった数値を表示に呼び出す(35 ページ)。

プログラム用機能の要点と索引

HP-15C 本体の裏側も見てください。

[P/R]	program／run mode，プログラム入力状態／計算状態。計算機をプログラム入力状態（PRGMの文字を表示）または計算状態（PRGMの文字が消える）に切り換える(66 ページ)。
CLEAR [PRGM]	プログラム入力状態ではプログラム用メモリーを全部クリアしてプログラム用メモリーへの配分を取り消す。計算状態では計算機を 000 行に合わせるだけ（67 ページ)。
[MEM]	計算機のメモリー状態（データ記憶用レジスタ，共通部分，プログラム用メモリーに配分したレジスタ数）を表示する（215 ページ)。
[←]	back arrow，左向き矢印。プログラム入力状態では表示のプログラム命令を削除する。それ以降の全命令が 1 行ずつ繰り上がる（84 ページ)。
[LBL]	label，ラベル（目印)。下記のラベル名と一緒に使ってプログラムのルーチンの出発点を示す(67 ページ)。
[A] [B] [C] [D] [E] 0 1 2 3 4 5 6 7 8 9 .0 .1 .2 .3 .4 .5 .6 .7 .8 .9	ラベル名。 [LBL] の後に続けて，プログラムのルーチンの出発点を指定する（67 ページ)。また（ [LBL] を付けないで）指定のルーチンの実行開始に使う（69 ページ)。
[USER]	利用者優先状態の設定と解除。利用者優先状態ではキーボードの最上段

左側5個のキーの第一機能（白）と f（黄）を使う第二機能を交換する（69ページ）。利用者優先状態は行列操作の STO または RCL A～E, (i) にも関係があり，行列要素の記憶または呼び出しに使う R_0（行番号）か R_1（列番号）を自動的に増やす（144ページ）。

GTO

go to，移動。ラベル名（上記）またはIと一緒に使って計算機の位置を指名のラベルに移す。GTO がプログラム命令であればプログラムの実行を続ける。プログラム命令でなければ位置が変わるだけ（90ページ）。R_I に負数を入れておくと，GTO I は行番号への移動となる（110ページ）。

GTO CHS *nnn*

go to line number。計算機を nnn で指定した実在の行番号に移す。プログラム中に入れられない（83ページ）。

GSB

go to subroutine，サブルーチンにジャンプ。ラベル名（上記）または I と一緒に使って指定のラベル名のルーチンの実行を開始する。プログラム中でも計算状態のキー操作だけでも使える。RTN 命令で GSB の次の行に実行が戻る（102ページ）。

BST

back step，逆行。計算機をプログラム用メモリー内で1行または複数行戻す。（プログラム入力状態で押し続けるとぐるぐる逆行を続ける。）一つ前の行番号とキーコードを表示する（84ページ）。

SST

single step，順行。プログラム入力状態では計算機が1行または複数行プログラム用メモリー内を前方に進める。計算状態ではそのときのプログラム行を表示し，それを実行して

次の行に進む（83ページ）。

[PSE] pause，一時停止。Xレジスタの内容を表示するために約1秒間停止し，それから走行を続ける（68ページ）。

[R/S] run／stop，走行／停止。プログラム用メモリー中のそのときの行からプログラムの走行を開始する。プログラムが走行中なら走行を停止する（68～69ページ）。

[RTN] return，戻る。計算機を000行に戻して走行を止める（計算中のとき）（68ページ）。サブルーチン中では[GSB]の次の行に戻すだけ（102ページ）。

[SF] set flag，フラグをセット（＝真）する。指定のフラグ（0～9）をセットする。フラグ0～7は利用者用フラグ。フラグ8は複素数計算状態，フラグ9は桁あふれ用（93ページ）。

[CF] clear flag，フラグをクリア（＝偽）する。指定のフラグ（0～9）をクリアする（93ページ）。

[F?] is flag set ?，フラグをセットしてあるか。指定のフラグを調べる。セットならプログラムの走行を継続する。クリアならプログラムの走行は次の1行だけを飛び越して続ける（93ページ）。

[x≤y]，[x=0]，[TEST]{0～9} 条件判断。判断はXレジスタの数値とYレジスタの数値または0と比較する。真なら計算機はプログラム用メモリーの次の行の命令を実行する。偽なら計算機は次の1行だけを

飛び越してその次の行から命令を実行する（92 ページ）。[x=0] と [TEST] 0，5，6 は複素数と行列の記号についても有効（132，174 ページ）。

[TEST] 0	$x \neq 0$
[TEST] 1	$x > 0$
[TEST] 2	$x < 0$
[TEST] 3	$x \geqslant 0$
[TEST] 4	$x \leqslant 0$
[TEST] 5	$x = y$
[TEST] 6	$x \neq y$
[TEST] 7	$x > y$
[TEST] 8	$x < y$
[TEST] 9	$x \geqslant y$

[DSE]

decrement and skip if equal or less than，減少して以下なら飛び起し。指定のレジスタにあるカウンタ値を指定通り減らす。新しいカウンタ値が指定の判定値に等しいか小さければ次の 1 プログラム行だけ飛び越す（110 ページ）。

[ISG]

increment and skip if greater than，増加して大きければ飛び越し。指定のレジスタにあるカウンタ値を指定通り増やす。新しいカウンタ値が指定の判定値より大きければ次の 1 プログラム行だけ飛び越す（110 ページ）。

事項索引

太文字のページ番号は第一に見てもらいたい部分，細文字のページ番号はその次に見てもらいたい部分です。

アルファベット

カ行

サ行

タ行

ナ行

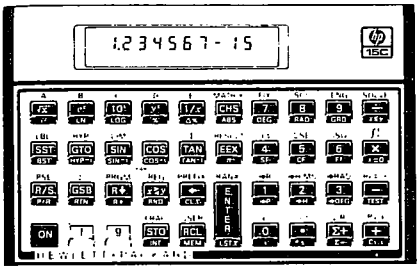
ハ行

マ行

ヤ行

ラ行

HP-15C のキー配置と不揮発性メモリー



メモリー・スタック

	実数	虚数
T		
Z		
Y		
X		

いつも実数 X レジスタの内容を表示

ラスト x

データ記憶部分

R_I

R_0	0	R_0	10
R_1	1	R_1	11
R_2	2	n	R_2 12
R_3	3	Σx	R_3 13
R_4	4	Σx^2	R_4 14
R_5	5	Σy	R_5 15
R_6	6	Σy^2	R_6 16
R_7	7	Σxy	R_7 17
R_8	8		R_8 18
R_9	9		R_9 19

最初の配分は R_0 ～ R_{19} がデータ記憶用。

配分は [DIM] [(i)] 機能を使って変えられる。

共通部分

R_{20}	行列用メモリー
	虚数スタック
	[SOLVE] と [∫]
	未使用のレジスタ
R_{65}	プログラム用メモリー レジスタ1個にプログラム命令が最大7命令入れられる

共通部分のメモリーだけがプログラム用に転用可能です。

最初の配分は R_{20} ～ R_{65} が共通部分用で、上記の高等数学用機能とプログラム用に使える。

レジスタ1個が7バイトの容量。プログラム命令ごとに1または2バイト使う。プログラム用メモリーが必要になると、番号の多い方から少ない方への順に一度に1レジスタずつ転用する。

YHP 横河・ヒューレット・パッカード株式会社

高井戸本社
営業本部：〒168　東京都杉並区高井戸東三丁目29番21号
Tel. 03 331 6111（大代表）

東京支社：〒160　東京都新宿区西新宿2 7 1（新宿第一生命ビル）
Tel. 03 348 4611（大代表）

西部支社：〒532　大阪市淀川区西中島5 4 20（中央ビル）
Tel. 06 304 6021（代表）

00015-90008 Rev. B - 3/86

Printed in Canada