マクロ計算機を使う

マクロ計算機は、データの分析や変換を行うプログラム機能が搭載された強力なツールです。KaleidaGraph でタスクの各ステップを実行する代わりに、マクロを記述して自動化することができます。例えば、マスクされたデータセルの値を消去するマクロを作成したりすることができます。

この章では、以下について説明します。

- マクロ計算機の使用方法。
- すべての計算機能の使用方法。各関数の詳しい説明といくつかの例が含まれます。
- マクロ計算機のスタートから終了までのプログラミングプロセス。サンプルマクロを使用して、プラン、入力、実行、記録およびマクロの保存方法について説明します。またマクロメニューに追加 / 既存のマクロの編集方法についても説明します。
- 利用可能なマクロコマンドの意味と構文。

1.1 マクロ計算機の 使用方法

ウィンドウメニューから**マクロ計算機**を選択するとマクロ計算機が表示されます (Figure 1-1)。この計算機には RPN (Reverse Polish Notation) プログラミング言語が組み込まれています。これは HP の関数電卓などにも見られるもので、1000 プログラムステップの能力があり、データの操作や変換を行うさまざまな方法を利用することができます。

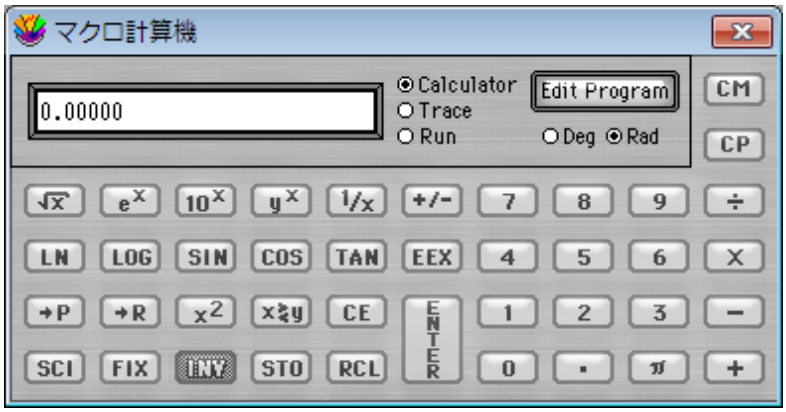

Figure 1-1 マクロ計算機

1.1.1 単数値関数

RPN 計算機は他の計算機とは異なる処理方法を使用します。RPN 計算機では数値ファンクションキーを押すと、すぐに答えが表示されます。そのため、関数が実行される前に、全てのオペランドを入力する必要があります。

単数値関数はディスプレイ上で計算されます。単数値関数には、サイン、平方根および対数などがあります。

単数値関数の使用方法は以下のとおりです。

1. 数値を入力します（ディスプレイにまだ数値がない場合)。
2. 関数ボタンをクリックします。

例

1. 4.5 を入力します（マウスかキーボードを使用します)。
2. COS キーをクリックします。マクロ計算機には 0.99692 または -.2108 と表示されます。

1.1.2 複数値関数

複数値関数では、計算の前に 2 つの値が計算機に入力されている必要があります。複数値関数には、加算、減算、乗算、除算などがあります。

複数値関数の使用方法は以下のとおりです。

1. 最初の値を入力します。
2. Enter をクリックし、最初の値と 2 つめの値を区切ります。
3. 2 つめの値を入力します。
4. 関数ボタンをクリックします。

例

計算 :	入力 :	表示 :
8 + 6	8 ENTER 6 +	14.000
8 - 6	8 ENTER 6 -	2.000
8 * 6	8 ENTER 6 ×	48.000
8 / 6	8 ENTER 6 ÷	1.333

それぞれ、Enter ボタンを押すと最初の値が入力されます。二つ目の値を入力して、関数 (function) ボタンをクリックします。ここで以下のことが起きています。

- 関数ボタンをクリックする前に、両方の数字が計算機に存在しています。
- Enter ボタンは、順に入力した 2 つの値を区切ります。前の関数による数字がすでに計算機に表示されている場合は、Enter をクリックする必要はありません。
- 関数ボタンを押すと即座に実行され、結果が表示されます。

いくつかの例を以下に示します。

計算 :	入力 :	表示 :
12 * (10 - 6)	12 ENTER 10 ENTER 6 - ×	48.000

計算:	入力:	表示:
5 * (8 - 2) / 3	8 ENTER 2 - 5 × 3 /	10.000
(9 - 3) * (3 / 9)	9 ENTER 3 - 3 ENTER 9 ÷ ×	2.000
625 / (20 + 5)	625 ENTER 20 ENTER 5 + ÷	25.000

1.1.3 自動スタック

自動スタックは、マクロ計算機が中間結果を保有しておくところです。スタックは、レジスタと呼ばれる 12 の記憶場所からなっています。スタックの底の値は、常に表示されます。入力したり、関数の結果のすべての値は、底のレジスタに配置されます。

スタックの一番下のレジスタが **X レジスタ**です。X レジスタの真上が **Y レジスタ**です。 x⇄yキーは、他のスタックに影響を与えないで、X と Y レジスタの数値を入れ換えます。Figure 1-2 に示すように、X レジスタディスプレイをクリックすると、いつでもスタックを見ることができます。

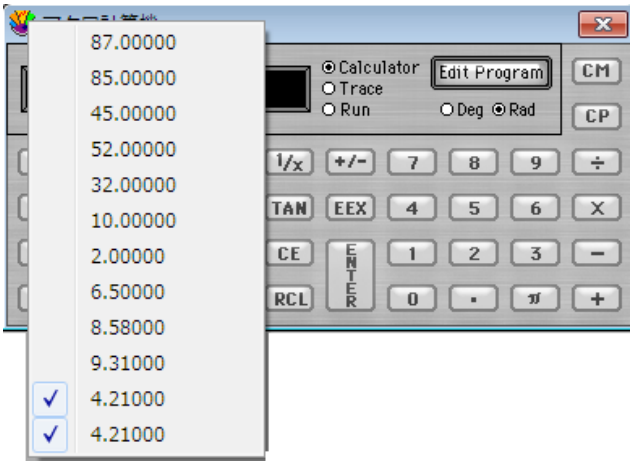

Figure 1-2 スタックの表示

単数値関数では、スタックはドロップされません。関数の結果は X レジスタに置かれ、関数で操作された値に置き換えられます。複数値関数は、実行されると常にスタックがドロップされます。関数の結果は、X レジスタに置かれます。関数の結果は X レジスタに置かれ、マクロ計算機で使用された 2 つの数値はスタックから削除されてドロップが実行されます。

1.1.4 数値のストアとリコール

マクロ計算機には、00–99 の番号が付られた 100 のメモリレジスタがあります。これらのレジスタは、STO（ストア）と RCL（リコール）コマンドを通じてアクセスします。これらのコマンドの動作には、計算機の X レジスタを使用します。

注意 : RCL コマンドはスタックをリフトしますが、STO コマンドではリフトしません。

メモリレジスタに値をストアする方法は以下の通りです。

1. 計算機にストアする値を入力します。
2. STO をクリックします。
3. 記憶場所 (00–99) を入力します。
4. 指定した場所に値がストアされます。

メモリレジスタから値をリコールする方法は以下の通りです。

1. RCL をクリックします。
2. 記憶場所 (00 – 99) を入力します。
3. 指定された記憶場所の値は、X レジスタに置かれます。

例

STO 05	X レジスタの値を記憶場所 05 にストア
STO 29	X レジスタの値を記憶場所 29 にストア
RCL 15	記憶場所 15 の値を X レジスタに配置
RCL 34	記憶場所 34 の値を X レジスタに配置

1.2 計算機コマンド

1.2.1 計算機のクリア

· **クリアエントリ** – このボタンは、前の入力に影響を与えずに表示（X レジスタ）をクリアします。

· **クリアメモリ** – このボタンは、メモリレジスタの内容を 0 にリセットします。

· **クリアプログラム** – このボタンは、プログラムテキストエディタのプログラムすべてを消去します。

1.2.2 表示の変更

· **固定 10 進表記法** – この表記法では、計算機は指定された小数点以下の桁数にまるめられた数値を表示します。小数点以下の桁数の表示を変更するには、FIX をクリックして小数点以下の桁数 (0–9) を入力します。

· **科学的記数法** – 科学的記数法 – この表記法では、計算機は数値を小数点の左に 1 桁、小数点の右に指定した桁数を表示します。小数点以下の桁数の表示を変更するには、SCI をクリックし小数点以下の桁数 (0–9) を入力します。

· **指数表記** – このボタンで数値を指数形式 ($A \times 10^B$) で入力することができます。数値の 10 進 (A) 部分を入力し、EEX をクリックして指数 (B) をタイプします。

1.2.3 単数値関数

· **記号変更** – このボタンは、X レジスタの記号を変更します。

· **円周率** – このボタンは、円周率の近似値 (3.14159...) を X レジスタに置きます。

· **逆数** – このボタンは、X レジスタの逆数を計算します。

· **平方** – このボタンは、X レジスタの二乗を計算します。

· **平方根** – このボタンは、X レジスタの平方根を計算します。

· **常用対数** – このボタンは、X レジスタの対数（底 10）を計算します。

· **自然対数** – このボタンは、X レジスタの対数（底 e）を計算します。

· **常用 Anti-log** – このボタンは、X レジスタの 10 のべき乗を計算します。

· **自然 Anti-log** – このボタンは、X レジスタの e (2.7182...) のべき乗を計算します。

1.2.4 ??? 関数

· **指数** – このボタンは、Y レジスタの値を X レジスタで指定したべき乗にします。X が整数の場合、Y の負値のみ許可されます。

· **極から直交** – このボタンは、X と Y レジスタの値（R と φ）を直交座標（X と Y）に変換します。φ は、計算機の設定により度またはラジアンで表わせます。

· **直交から極** – このボタンは、X と Y レジスタの値（R と φ）を極座標（X と Y）に変換します。φ は、計算機の設定により度またはラジアンで表わせます。

1.2.5 三角関数

· **正弦、余弦、正接と逆関数** – SIN , COS および TAN ボタンは、X レジスタの値の適切な三角関数を計算します。 Deg と Rad ボタンは、度またはラジアンで表示される値を決定します。これらの関数を計算する前に、正しいモードが選択されているか確認してください。これらの関数のいずれかを選択する前に INV をクリックすると、指定した三角関数の逆関数が計算されます。

	関数	逆関数
正弦	sin	INV sin = $\sin^{-1}$
余弦	cos	INV cos = $\cos^{-1}$
正接	tan	INV tan = $\tan^{-1}$

1.3 計算機のプログラミング

このセクションでは、マクロの書き方のはじめから終わりまでのプロセスを説明します。サンプルマクロは、マクロを書くときに関連するいくつかのステップの実例に使用します。より上級のマクロはこのセクションの最後に示します。コマンドの詳細については、セクション 1.4 を参照してください。

1.3.1 プログラムテキストエディタ

マクロ計算機の **Edit Program** ボタンをクリックすると、プログラムエディタが表示されます。プログラムテキストエディタを以下に示します (Figure 1-3)。

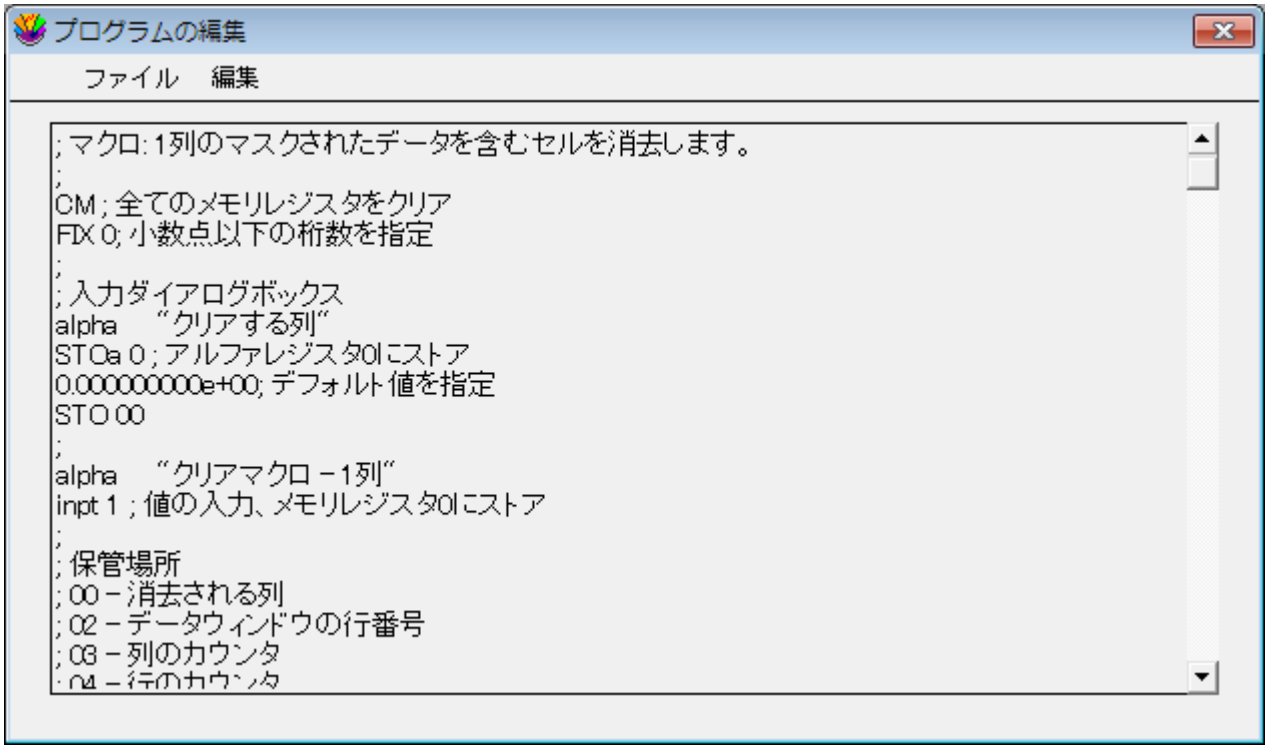

Figure 1-3 プログラムテキストエディタ

このエディタは、新しいマクロを作成したり既存のマクロを編集するために使用します。テキストの移動と配置のために、カット、コピー、ペーストおよび検索コマンドがサポートされています。編集を終了したら、**OK** をクリックするか、**ファイル > 閉じる** を選択するとマクロの構文エラーが確認されます。エラーが検索された場合は、エディタが再表示され、エラーがハイライト表示されます。他のエラーを捜す場合は、**検索**コマンドを使用してください。

1.3.2 マクロ作成の概要

マクロのプランニング

マクロ作成の最初のステップは、マクロに何を実行させるかを正確に決めることです。使用するコマンドのいくつかを書き留め、データウィンドウにアクセスする方法（間接的にまたはベクターコマンドを使用して）について考えることをお勧めします。一般的なルールとして、数式入力を使用してタスクを実行できれば、ベクターアドレッシングはマクロに対して通常通りに動作します。そうでない場合は、間接アドレッシングを使用します。

最初の例では、階乗を計算するマクロを書くことが目標です。値から継続して 1 を減算し、その結果を前の値の積によって乗算します。5 の階乗の例は 5*4*3*2*1 となります。

マクロの入力

プログラムを入力するには、プログラムエディタを表示する必要があります。プログラムのエディタを表示するには、Edit Program をクリックします。エディタにいくつかテキストが含まれている場合は、**ファイル**メニューから**新規**を選択して、エディタの内容を消去します。これで、マクロを入力する準備が整いました。サンプルマクロは次のとおりです。

階乗マクロ – 階乗のマクロを以下に示します。各コマンドをエディタに入力した後、キャリッジリターンを続けます。右側のコメントは、各コマンドが何を行うかを識別するために示しています。

```
" 数値入力 "                    ; “ 数値入力 ” をアルファレジスタにコピー
STOa 1                          ; アルファレジスタの内容をアルファメモリにストア
" 階乗 "                        ; この文字列をダイアログのタイトルとして使用します
prmt 1                          ; 階乗の値の入力を促すダイアログを表示
                                ; そして、メモリレジスタ 01 に値をストアします
1                               ; X レジスタに 1.0 を置く
STO 02                          ; メモリレジスタ 02 に 1.0 をストア
LBL 00
RCL 01                          ; X レジスタにメモリレジスタ 01 の内容をリコール
MUL 02                          ; X レジスタとレジスタ 02 の内容を掛け算
1                               ; X レジスタに 1.0 を置く
SUB 01                          ; メモリレジスタ 01 から 1.0 を引き算
RCL 01                          ; X レジスタにレジスタ 01 の内容をリコール
0.0                             ; X レジスタに 0.0 を置く
x < y                           ; 0.0 <（メモリレジスタ 01 の内容）かどうかテスト
GTO 00                          ; 真の場合 : LBL 00 へ
RCL 02                          ; 偽の場合 : X レジスタにメモリレジスタ 02 の内容をリコール
STOP                            ; プログラムの実行を停止
```

※本マニュアルの例題プログラムをコピーして実行する場合は、ペースト後「""」引用符を再入力して下さい。

マクロの実行

マクロ計算機にプログラムを入力した時点でマクロが実行可能になります。プログラムエディタを開いている場合は、OK をクリックするか、**ファイル > 閉じる** を選択して計算機に戻ります。プログラムを実行するには Run をクリックします。STOP コマンドに到達するまで、プログラムが実行されます。STOP コマンドが検出されない場合は、マクロを中止する前に他のコマンドを求めて 1000 ステップのプログラムメモリすべてが検索されます。

注意 : STOP コマンドは、マクロの最後のプログラムステップにある必要はありません。プログラム内で論理にかなっていれば、どこにでも配置することができます。

階乗のプログラムを実行すると、Figure 1-4 のダイアログが表示されます。

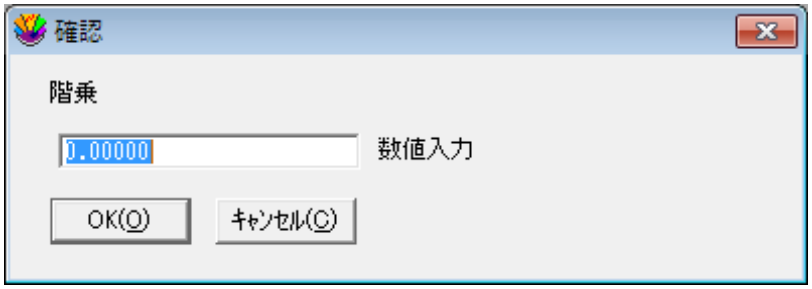

Figure 1-4 入力ダイアログ

階乗の計算を行う値を入力し、OK をクリックします。プログラムが終了して、マクロの結果は計算機のディスプレイに表示されます。

マクロのデバッグ

マクロを実行しても、その結果が期待どおりでない時もあります。そのようは場合は、細かくプログラムを調べて、問題の箇所を検索することをお勧めします。マクロ計算機には、プログラムのデバッグを支援する 2 つのボタン（SST および R/S）が用意されています。

SST

- **シングルステップ** – このボタンは、一度に 1 ステップずつプログラムを実行します。SST をクリックすることで（または**スペースバー**、**return**、**enter** キーを押すことによって）次のステップに進むことができます。プログラムの各ステップで、X レジスタの現在の値が表示されます。**スペースバー**、**return**、**enter** 以外のキーがタイプされると、プログラムは Run モードに切り替わり、プログラムの残りを実行します。

R/S

- **実行 / 停止** – このボタンは、長いループをデバッグするのに便利です。プログラム実行中に R/S コマンドが押されると、計算機は Run モードから**シングルステップ**モードに切り替わります。シングルステップモードで通常通りに継続できます。**スペースバー**、**return**、**enter** 以外のキーがタイプされると、プログラムは Run モードに戻り、プログラムの残りの実行を続けます。

もう一度階乗プログラム実行します。ただし、今度は Trace をクリックしたら、SST ボタンを使用してステップスルーを実行します。現在のステップは、計算機のウィンドウでハイライト表示され、ステップの結果はディスプレイに表示されます。

コメント行の追加

マクロが正しく実行された後は、プログラムにコメントを追加できます。コメントは、プログラムの目的や各メモリレジスタの用途を記述して、プログラムをより理解しやすくするために使用します。プログラムステップに置かれるコメントは、そのステップが何を行っているかを正確に説明する必要があります。

コメントは、マクロのどこにでもセミコロンを使用して追加できます。セミコロンの後からキャリッジ・リターンの出現までは、コメントとして扱われます。

元のプログラムの右側にあるコメントのいくつかを追加してみて下さい。コメントはコマンドと同じ行にも、別の行にも置くことができます。以下に例を示します。

```
" 数値入力 " ; この文字列 をアルファレジスタにコピー
```

マクロの保存

マクロが完成したら、将来使用するために保存する必要があります。保存ダイアログを表示するには、プログラムエディタで、**ファイル > 別名で保存**を選択します。プログラムに名前をつけたら、**保存**をクリックして、マクロの作成を完了します。

1.3.3 サンプルマクロ

このセクションでは、階乗の例題よりもさらに高度なマクロについて紹介します。このマクロでは、ループ、サブルーチンおよび間接アドレス指定の使用方法を示します。ほとんどのプログラムステップに、そのコマンドの動作に関する説明のコメントも加えてあります。

クリアマクロ－(1 列)- このマクロの目的は、指定したデータ列をステップダウンして、マスクされたデータを含むセルを消去することです。プログラムの直後に、プログラムがどのように動作するかの説明があります。

```
; マクロ : 1 列のマスクされたデータを含むセルを消去します。
;
CM                                    ; 全てのメモリレジスタをクリア
FIX 0                                 ; 小数点以下の桁数を指定
;
; 入力ダイアログボックス
alpha    " クリアする列 "
STOa 0                                ; アルファレジスタ 0 にストア
0.000000000e+00                       ; デフォルト値を指定
STO 00
;
alpha    " クリアマクロ－ 1 列 "
inpt 1                                ; 値の入力、メモリレジスタ 0 にストア
;
; 保管場所
; 00 - 消去される列
; 02 - データウィンドウの行番号
; 03 - 列のカウンタ
; 04 - 行のカウンタ
; 06 - 行と列 # 10 進形式
;
FIX 6                                 ; 小数点以下の桁数をリセット
ibase                  ;1000 列全てにアクセスするために間接アドレス指定を許可
size                                  ; データウィンドウのサイズを返す
STO 02                                ; データウィンドウにある行番号をストア
RCL 00                                ; 列番号をリコール
STO 03                                ; 列カウンタをセット
XEQ 50                  ; 10 進形式のアドレスをストア、行カウンタをセット
;
LBL 20
RCLi 06                 ; メモリレジスタ 06 で指定したセルからデータをリコール
test 2                                ; マスクされたデータセルのテスト
CLRi 06                               ; 06 で指定したセルをクリア
0.000000000e+00
DSE 04                  ; カウンタから 1 を引き、04 の内容 > 0.0  をテスト

GTO 40                                ; 真の場合 : 行番号を 1 増やす
STOP                                  ; 偽の場合 : プログラムの停止
;
LBL 40
XEQ 70                                ; 現在のアドレスを増やす
GTO 20                                ; Label 20 へ
;
LBL 50
RCL 03                                ; 現在の列をリコール
1.000000000e+03
/                                     ; 1000 で列を除算
STO 06                                ; 10 進形式で列をストア
XEQ 60                                ; 列カウンタをリセット
```

```
GTO 20
;
LBL 60
RCL 02
STO 04                                          ; 行カウンタをリセット
RTN                                    ; XEQ の後に次のステップに戻る
;
LBL 70
1.000000000e+00
ADD 06                                          ; アドレスを 1 で増やす
RTN                                    ; XEQ の後に次のステップに戻る
```

1. プログラムの最初の部分のメモリレジスタをクリアし、Figure 1-5 に示す入力ダイアログを生成します。**alpha** および **STOa** コマンドは、入力値の隣に説明を置くために使用されます。二つ目の alpha コマンドは、ダイアログのタイトルに使用されます。**inpt 1** コマンドは、入力された値が最初のメモリレジスタ (00) にストアするようプログラムに伝えます。

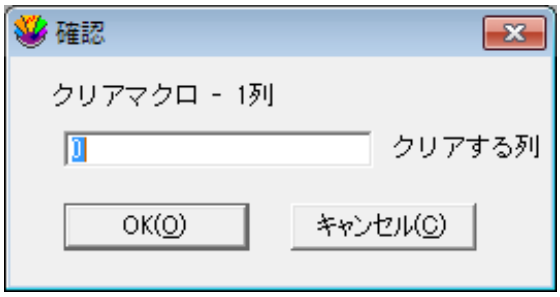

Figure 1-5 入力ダイアログ

2. プログラムの次の部分は、データウィンドウのサイズを決定して、メモリレジスタ 02 に行数をストアします。**ibase** コマンドは、間接アドレス指定を 1000 列全てにアクセスすることを許可します。**XEQ 50** コマンドは、リコールされる列番号を 1000 で割り、メモリレジスタ 06 にストアします。**LBL 60** が実行されると、行カウンタの結果はデータウィンドウの行番号に等しくなるようにセットされます。
3. **LBL 20** の部分は、プログラムのメイン部分です。このセクションは、メモリレジスタ 06 で指定されたセルからデータをリコールします。**test 2** コマンドは、データがマスクされているか確認します。データがマスクされていると、セルの内容は消去されます。データがマスクされていなければ、プログラムは次の行に移動します。このセクションは、カウンタの減少とデータウィンドウの最終行に達しているかを確認するために使用されます。最終行に到達するとプログラムはストップし、そうでなければ、プログラムは動作を継続します。

   DSE 04 は、毎回 1 ずつカウンタを減少させます。カウンタが 0 より大きければ、プログラムは **LBL 40** にジャンプし、次の行が読み込めるようにアドレスを 1 ずつ増加させます。カウンタが 0 に等しい場合、データウィンドウの最終行に達します。カウンタが 0 に等しい時は、**STOP** コマンドに到達するため、プログラムは次のステップ (**GTO 40**) をスキップし、実行を中止します。
4. 全てのサブルーチンがプログラム本体の後に置かれていることに注意してください。この形式は一般的に、構成の目的で使用されます。

Figure 1-6 は、クリアマクロが実行される前のサンプルデータセットを示しています。

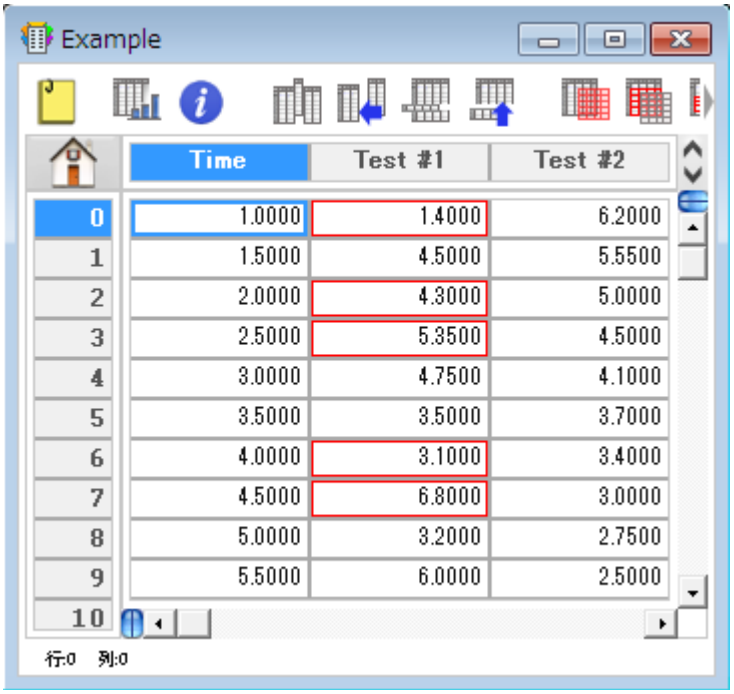

Figure 1-6 実行前のデータセット

Figure 1-7 は、マクロ実行後の結果を示しています。

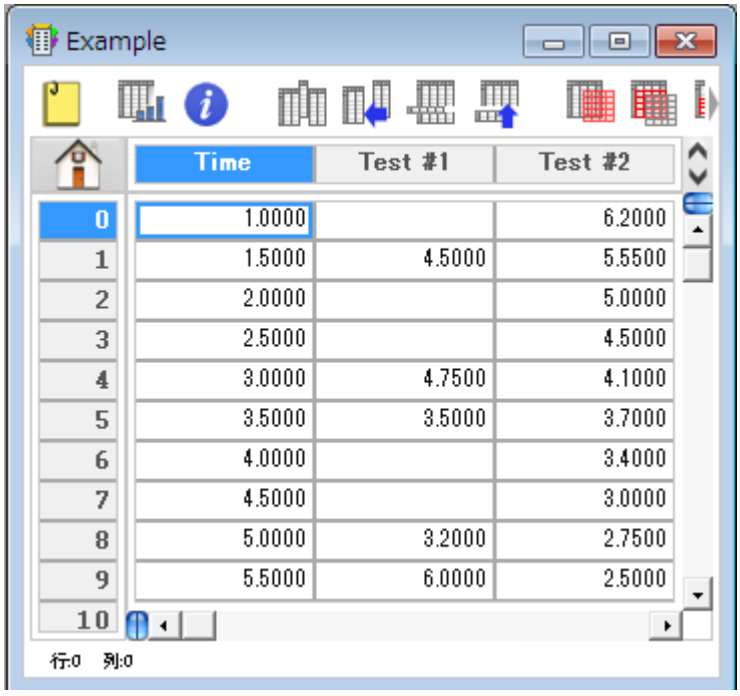

Figure 1-7 マクロ実行後の結果

1.3.4 マクロメニューにプログラムを追加

いったんプログラムを記述したら、**マクロ**メニューに追加して、後から使用することができます。最大 100 のマクロを一度にメニューにストアできます。計算機内にマクロが存在する場合、それらをマクロメニューに追加する方法は以下のとおりです。

1. **マクロ > マクロの表示**を選択します。Figure 1-8 のダイアログが表示されます。

Figure 1-8 追加をクリックする前のマクロの表示ダイアログ

2. **計算機**をクリックして、追加するマクロの現在位置を指定します。
3. ダイアログ左側のマクロリストからマクロの 1 つを選択します。追加するマクロは、このマクロの後に挿入されます。
4. **追加**をクリックします。**計算機**という名前がマクロリストに挿入されます。これはマクロに割り当てられるデフォルト名です。このステップが完了すると、ダイアログは Figure 1-9 のようになります。

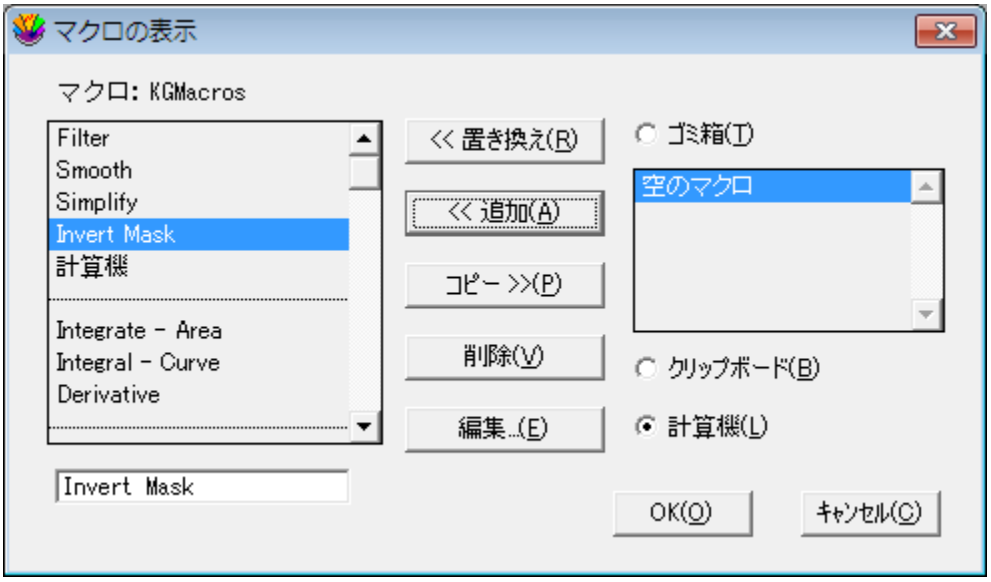

Figure 1-9 追加をクリックした後のマクロの表示ダイアログ

5. マクロのデフォルト名を選択します。名前はマクロリストの下にある名前フィールドで編集することができます。
6. 別のマクロを選択するか、OK をクリックすると名前が変更されます。

1.3.5 マクロの再編集

プログラムが**マクロ**メニューにあれば、これを読みだして編集することができます。マクロがマクロメニューに追加されていない場合は、プログラムエディタの中から開くことができます。クリアマクロの例では、列の範囲で動作できるようにプログラムを修正することをお勧めします。

マクロを編集する手順は以下のとおりです。

1. **マクロ > マクロの表示** を選択します。
2. 編集したいマクロの名前を選択して、**編集** をクリックします。テキストエディタにプログラムのソースコードが表示されます。元のマクロを修正したら、変更を保存します。OK をクリックするか、**ファイル > 閉じる** を選択すると、エディタが終了します。
3. OK をクリックして、マクロの表示ダイアログを終了します。

修正したプログラムは以下のとおりです。元のプログラムのほとんどは完全な状態で残ります。追加または変更されたものは、ボールドで表示されその前に番号がついています。変更の説明がプログラムのリストに続きます。

注意 : このサンプルマクロを使用して、列のセルを消去する以外に、さまざまな操作を簡単に実行することができます。列の範囲で実行する間接アドレス指定のための基本的なコマンドが既に用意されています。

クリアマクロ –(複数列)

```
; マクロ : 列の範囲にあるマスクされたデータを含む任意のセルを消去します
;
CM                                          ; 全てのメモリレジスタをクリア
FIX 0                                       ; 有効桁数を指定
;
; 入力ダイアログボックス
alpha    " クリアする最初の列 "
STOa 0
0.000000000e+00
STO 00
alpha    " クリアする最後の列 "
STOa 1                                      (1)
1.000000000e+00
STO 01
alpha    " クリアマクロ – 複数列 "
inpt 2
;
; Storage Locations                         (2)
; 00 – 最初の列が消去されます
; 01 – 最後の列が消去されます
; 02 – データウィンドウの行番号
; 03 – 列カウンタ
; 04 – 行カウンタ
; 06 – 10 進形式の行と列番号
;
```

```
                FIX 6                                   ; 有効桁数 のリセット
                ibase                   ; 1000 列全てにアクセスするための間接アドレス指定を許可
                size                                    ; データウィンドウのサイズを返す
                STO 02                                  ; データウィンドウの行数をストア
                RCL 00                                  ; 列番号のリコール
                STO 03                                  ; 列カウンタのセット
                XEQ 50                                  ; 小数点形式で列をストア
                ;
                LBL 10
(3)             XEQ 20
                RCL 01                                  ; 最終列の値をリコール
                ISG 03                          ; 1 で増分されるカウンタで 03 の内容 <= 01
                GTO 50                                  ; 真ならば、列を増やします
                STOP                                    ; 偽ならば、プログラムを停止します
                ;
                LBL 20
                RCLi 06                 ; メモリレジスタ 06 の指定されたセルからデータをリコール
                test 2                                  ; マスクされたデータセルかをテスト
                CLRi 06                                 ; 6 で指定されたセルをクリア
                0.000000000e+00
                DSE 04                          ; 1 で減少するカウンタで 04 の内容 > 0.0
                GTO 40                                  ; 真ならば、行番号を 1 増やす
(4)             RTN
                ;
                LBL 40
                XEQ 70                                  ; カレントアドレスを増やす
                GTO 20
                ;
                LBL 50
                RCL 03                                  ; カレント列をリコール
                1.000000000e+03
                /                                       ; 1000 で列を除算
                STO 06                                  ; 小数点形式の列をストア
                XEQ 60                                  ; 行カウンタのリセット
(5)             GTO 10
                ;
                LBL 60
                RCL 02
                STO 04                                  ; 行カウンタのリセット
                RTN
                ;
                LBL 70
                1.000000000e+00
                ADD 06                                  ; アドレスを 1 で増分
                RTN
```

変更点の説明

1. 最初の alpha コマンドは変更され、列の値を得るために別の alpha コマンドが追加されました。STOa コマンドが、余分な alpha コマンドをストアするために追加されました。Inpt コマンドは、入力ダイアログに入力された余分な値をストアするために変更されました。Figure 1-10 は、修正したマクロを実行したときに表示されるダイアログです。

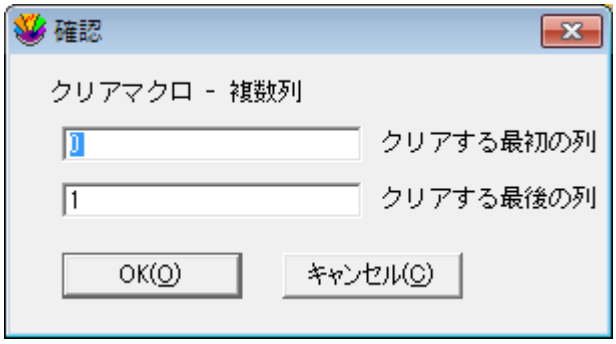

Figure 1-10 修正したマクロの入力ダイアログ

2. 保管場所 00 は、クリアされる最初の列の値をストアするために使用されます。保管場所 01 は、クリアされる最後の列の値をストアするために追加されました。
3. LBL 10 のコマンドでは、列カウンタを増分するために使用されます。プログラムは LBL 20 にジャンプし、列の最後に到達するまで継続します。プログラムは最後の列の値をリコールするために LBL 10 の内部を返し、列カウンタに対してテストします。カウンタが最後の列の値未満か等しい場合、プログラムは LBL 50 にジャンプして、次の列の新しいアドレスをストアします。カウンタが最後の列より大きい場合は、プログラムがストップします。
4. STOP コマンドが RTN (return) 文に変更されました。これにより、プログラムは列の最後の行に達したときに、LBL 10 の内部を返します。
5. ジャンプのターゲットが LBL 20 から LBL 10 に変更されました。

Figure 1-11 は、修正されたクリアマクロ が実行される前のサンプルデータセットです。

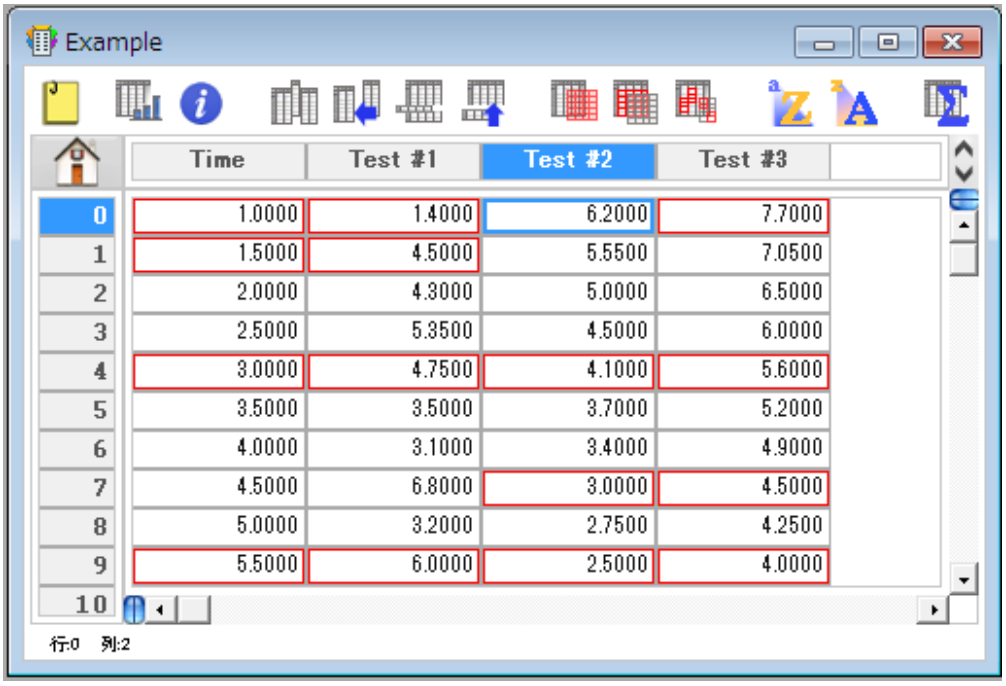

Figure 1-11 実行前のデータセット

Figure 1-12 は、クリアする最初の列を 0、最後の列を 3 に指定したクリアマクロが実行された画面です。

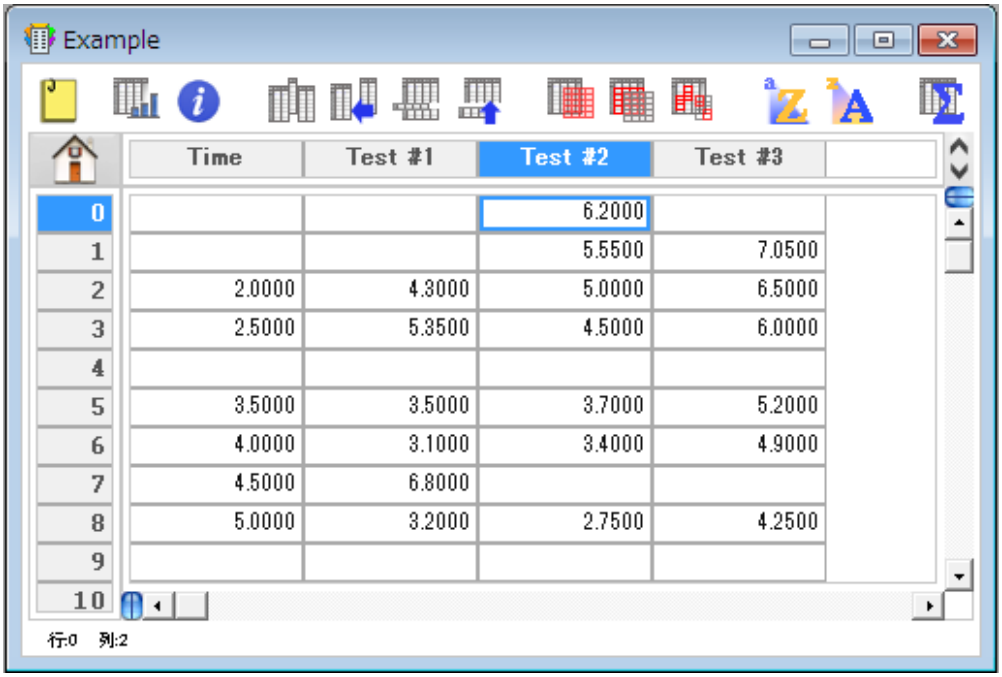

Figure 1-12 実行後のデータセット

1.3.6 マクロメニューに変更を保存

マクロメニューに表示されるマクロは、デフォルトマクロファイルから取得されます。これらのマクロが変更されたり、マクロの追加や削除が行われた場合は、次の方法の一つを使用して変更をマクロファイルに保存する必要があります。

- **ファイル > プリファレンス**を選択します。ファイル設定タブをクリックして、**マクロ**ポップアップメニューから、**確認**を選択します。KaleidaGraph を終了するとき、起動時に開かれたマクロファイルを上書きするか確認するダイアログが表示されます。
- **ファイル > プリファレンス**を選択します。ファイル設定タブをクリックして、**マクロ**ポップアップメニューから、**常に保存**を選択します。KaleidaGraph を終了するとき、起動時に開かれたマクロファイルが自動的に上書きされます。
- **ファイル > 書き出し > マクロ**を選択して、マクロファイルを保存します。ファイルは KGMacros と名前を付けて、KaleidaGraph アプリケーションと同じフォルダに保存する必要があります。

1.4 プログラムコマンド

1.4.1 制約事項

各コマンドに関連付けられた制約を指定するために、ほとんどのコマンドの後ろに n 、nn または nnn のいずれかが存在します。コマンドに n が続く場合、n の値は、0 から 9 までの範囲で指定できます。コマンドに nn が続く場合、nn の値は、0 から 99 までの範囲で指定できます (1 から 6 までの範囲で指定できる inpt コマンドを除く)。コマンドに nnn が続く場合、nnn の値は、0 から 999 までの範囲で指定できます。

間接アドレス指定の制約事項

間接アドレス指定では、計算に使用する行と列のアドレス数を指定できます。アドレス数は、メモリレジスタにストアされます。整数部分の数は、行アドレス、少数部分の数は、列アドレスです。たとえば、メモリレジスタに 9.03 がストアされていた場合、間接アドレス関数は、9 行 3 列のデータセルで動作します。

ibase コマンドがマクロにある場合、列アドレスは .000 から .999 までの範囲で指定できます。ibase コマンドがマクロにない場合は、列アドレスは .00 から .99 までの範囲指定になります。ibase コマンドの詳細については、セクション 1.4.6 を参照してください。

注意 : 全てのアドレス計算は、データ選択した先頭を基準にしています。

1.4.2 定数

数字を入力するだけで、定数をマクロ内で使用することができます。プログラム実行中に定数が検索された場合、その数字は、X レジスタに置かれます。定数は、一般的にダイアログでデフォルト値を設定するために使用します。

デフォルトでは定数は倍精度です。単精度の数字にしたい場合は、数字の前に const を置いてください。

例

```
35.958145799                    ; この定数は倍精度です
const 35.9581                   ; この定数は単精度です
```

1.4.3 プログラム制御とループ

プログラムラベル

- LBL nn – 00 から 99 まで番号付けられた 100 のプログラムラベルがあります。ラベルは、一連のプログラムステップ開始時のマーカーとして、また前方や後方へのプログラムジャンプのためのターゲットとして使用します。

例

```
LBL 40              ; このラベルは、ステップのグループの始まりを示しています
```

ジャンプ

- GTO nn – このコマンドは、プログラムの別の部分に分岐するために使用します。GoTo(GTO nn) コマンドが検出されると、プログラムは LBL nn にジャンプし、そこから実行を継続します。
- GTOi nn – このコマンドは、GoTo コマンドの別の形式です。間接ジャンプ (GTOi nn) は、指定したメモリレジスタの値を読み、ジャンプのターゲットラベルとして使用します。このコマンドは、プログラムの実行中にジャンプのターゲットを変更することができます。

例

```
GTO 20          ; プログラムを LBL20 にジャンプさせて実行を継続します。
GTOi 15         ; レジスタ 15 にある値のラベルにジャンプし、実行を続けます。
```

サブルーチンの呼び出し

- XEQ nn – 実行コマンド (XEQ nn) は、GoTo 文に似ています。GoTo コマンドと同様に、XEQ コマンドはプログラムの別の部分に分岐し、実行を継続します。しかし、リターンコマンド (RTN) が見つかると、プログラムは元の実行コマンドに続いて即座にプログラムステップに戻ります。
- XEQi nn – このコマンドは、XEQ 文の別の形式です。両者の違いは、XEQi は指定したメモリレジスタの値を読み、その値をサブルーチンのターゲットラベルとして使用することです。プログラムはリターンコマンド (RTN) を検出するまで、そのポイントから実行を続けます。それから、元の XEQi コマンドに続いて即座にプログラムステップに戻ります。この命令では、プログラムの実行中にターゲットのサブルーチンを修正できます。

例

```
XEQ 70          ; LBL 70 に分岐し、RTN が見つかるまで継続します
XEQi 20         ; メモリレジスタ 20 で指定した値のラベルに分岐します
```

注意 : RTN を見つける前に、最大 20 の XEQ/XEQi 文を入れ子にできます。

ループ

- ISG nn – increment and skip if greater (ISG nn) コマンドは、プログラムでループするのに便利です。この関数は指定したメモリレジスタ (nn) の値を 1.0 で増分し、X レジスタの値に対してテストします。メモリレジスタの値が X レジスタ以下の場合は、次のプログラムステップが実行されます。その値が X レジスタより大きい場合は、次のプログラムステップがスキップされます。
- DSE nn – decrement and skip if equal (DSE nn) コマンドは、プログラムでループするのに便利です。この関数は指定したメモリレジスタ (nn) の値を 1.0 で減少し、X レジスタの値に対してテストします。メモリレジスタの値が X レジスタより大きい場合は、次のプログラムステップが実行されます。その値が X レジスタ以下の場合は、次のプログラムステップはスキップされます。

次の例では Section 1.3.2 の階乗プログラムを示します。これは ISG と DSE コマンドを含めるよう修正されました。元のプログラムと 2 つの新しいプログラムを以下に示します。

元のプログラム	**ISG** コマンド	**DSE** コマンド
“ 数値入力 ”	“ 数値入力 ”	“ 数値入力 ”
STOa 1	STOa 1	STOa 1
“ 階乗 ”	“ 階乗 ”	“ 階乗 ”
prmt 1	prmt 1	prmt 1
1	1	1
STO 02	STO 02	STO 02
LBL 00	STO 03	LBL 00
RCL 01	LBL 00	RCL 01
MUL 02	RCL 02	MUL 02
1	MUL 03	0.0
SUB 01	RCL 01	DSE 01
RCL 01	ISG 02	GTO 00
0.0	GTO 00	RCL 02
x < y	RCL 03	STOP
GTO 00	STOP	
RCL 02		
STOP		

Entry と Exit サブルーチン

entry と exit 関数は、マクロの初期化やマクロ実行後のクリーンアップに役立ちます。これらのコマンドは、end コマンドと一緒に使用され、マクロの開始または終了時にだけ実行される特別なサブルーチンを定義します。これらの関数はマクロの終わりの STOP コマンドの後に配置する必要があります。

以下の階乗の例は entry と exit コマンドの使用方法を示します。

階乗の例題

```
LBL 00
RCL 01                          ; メモリレジスタ 01 の内容をリコール
MUL 00                          ; 00 の内容と X レジスタを掛ける
0.0
DSE 01                          ; 01 の内容 > 0.0 かどうかをテスト
GTO 00                          ; 真の場合 : LBL 00 へ
STOP                            ; 偽の場合 : プログラムの停止

entry                           ; この部分は、他より前に実行されます
" 数値入力 "                    ; この文字をアルファレジスタに置く
STOa 1                          ; 文字をアルファメモリ 1 にストア
" 階乗 "                        ; この文字をダイアログのタイトルとして使用
prmt 1                          ; ユーザーに数値入力を促す
1
STO 00                          ; メモリレジスタ 00 に 1.0 をストア
end                             ; entry 関数の終了

exit                            ; この部分は、STOP コマンドが到達後に実行されます
RCL 00                          ; マクロの結果をリコール
" 結果 ="                       ; この文字を出力ダイアログのタイトルとして使用します
view 1                          ; マクロの結果を表示 ( メモリレジスタ 00)
end                             ; xit 関数の終了
```

1.4.4 条件テスト

条件テストでは、プログラム実行中にユーザーの意思決定が可能です。基本形式を以下に示します。

```
If （条件テストが真ならば）
    （次の命令文を実行）
else
    （次のステップをスキップし、後に続く命令文にジャンプ）
```

条件テストは、次の命令文を実行するかスキップするかを決めるために使用されます。この命令文は、任意の有効なコマンドになります。条件テストの全てのリストは以下のとおりです。

- **x > y** X レジスタ > Y レジスタならば真
- **x < y** X レジスタ < Y レジスタならば真
- **x <= y** X レジスタ <= Y レジスタならば真
- **x >= y** X レジスタ >= Y レジスタならば真
- **x != y** X レジスタが Y レジスタと等しくなければ真
- **x = y** X レジスタ = Y レジスタならば真

例

```
RCL 05          ; メモリレジスタ 05 の内容をリコール
RCL 06          ; メモリレジスタ 06 の内容をリコール
x > y           ; 06 の値が 05 の値より大きいかテスト
GTO 20          ; 真 – この命令文を実行、偽 – 次のステップに進む
GTO 80
```

注意 : マクロでこれらのコマンドを使用するには、プログラムエディタに正しく入力されていなければなりません。KaleidaGraph がコマンドを認識するために、x の後、y の前にスペースを入力する必要があります。

1.4.5 ブール関数

ブールテスト

ブールテストは、X と Y レジスタの値が比較される点で条件テストに似ています。しかし、ブールは比較の結果 (0= 偽 , 0 以外 = 真) を X レジスタに置き、スタックから元の X と Y の値をドロップする点が異なります。ブールテストの全てのリストは以下のとおりです。

- **bool 0** y > x Y レジスタ > X レジスタならば真
- **bool 1** y >= x Y レジスタ >= X レジスタならば真
- **bool 2** y < x Y レジスタ < X レジスタならば真
- **bool 3** y <= x Y レジスタ <= X レジスタならば真
- **bool 4** y = x Y レジスタ = X レジスタならば真
- **bool 5** y != x Y レジスタが X レジスタと等しくなければ真
- **bool 6** y || x Y レジスタ **or** X レジスタが真ならば真
- **bool 7** y && x Y レジスタ **and** X レジスタが真ならば真

その他のブールコマンド

- **BMv nnn** (Boolean mask vector) – X レジスタのブール値が真 (0 以外) ならば、列 **nnn** のカレント行がマスクされます。
- **BUv nnn** (Boolean unmask vector) – X レジスタのブール値が真 (0 以外) ならば、列 **nnn** のカレント行がマスク解除されます。
- **ifelse** – このコマンドは、スタックの一番下の 3 つのレジスタ、ブール、Y と X を使用します。ブールが真 (0 以外) ならば、Y レジスタの値は X レジスタに置かれます。ブールが偽 (0) ならば、X レジスタの値は変わりません。他のすべてのレジスタは、スタックからドロップされます。

例

```
RCL 05          ; メモリレジスタ 05 の内容をリコール
RCL 06          ; メモリレジスタ 06 の内容をリコール
bool 4          ; 06 の値が 05 の値と等しいかどうかをテストします
BMv 109         ; 上記のブールテストが真ならば列 109 のカレント行をマスクします
```

1.4.6 データウィンドウのアドレス指定

間接アドレス指定

- ibase – デフォルト範囲の 100 列を超える列をアドレス指定するには、ibase コマンドを使用する必要があります。このコマンドを使用する場合は、間接アドレスの列の部分は .0 から .999 の範囲をとります。このコマンドはマクロのどこでも使うことができ、一度使用するだけで十分です。

データのリコール

- RCL nn – メモリレジスタ nn から値をリコールし、その値を X レジスタに置きます。
- rclr nn – relative recall (rclr nn) は、メモリレジスタ nn で指定されたターゲットメモリレジスタから値を読み込みます。この値は、X レジスタに置かれます。
- RCLv nnn – この関数は、行方向で選択したデータからデータをリコールするのに使用します。プログラムは、選択範囲の行ごとに一度だけ評価されます。このコマンドはマスクされたデータセルをスキップします。
- RCLi nn – このコマンドは、メモリレジスタ nn で指定したアドレスからデータ値をリコールします。その数の整数部分は、行アドレスとして使用され、端数部分は列番号です。例えば、メモリレジスタが 9.03 をストアしていると、**RCLi** コマンドは 9 行 3 列のデータ値を返します。

例

```
RCL 08          ; メモリレジスタ 08 の値をリコール
rclr 20         ; 20 で指定したメモリレジスタから値をリコール
RCLv 000        ; 行方向で選択した最初の列をリコール
RCLi 05         ; 05 で指定したアドレスからデータをリコール
```

- getcell – 特定のアドレスにあるセルの値を取得し、X レジスタにその値を置きます。このアドレスは、スタックの下にある 2 つのレジスタ (X と Y) で指定します。列アドレスの値は 1000 未満でなければならず、スタックの下 (X) に配置されます。行アドレスの値は、その上のスタック (Y) に置かれます。このコマンドが実行されると、スタックからアドレスが削除され、セルから読み出された値は、X レジスタに置かれます。

例

```
45              ; 最初の値は、スタックに置かれます。これは行アドレス (Y) です
110             ; 次の値は、スタックに置かれます。これは列アドレス (X) です
getcell         ; 行 45、列 110 のセルの値を X レジスタに置きます
```

注意 : マクロが存在しない行や列の番号をリコールしようとした場合、作成されていない位置を指定したというメッセージとともに中断します。

データのストア

- **STO nn** – X レジスタの値をメモリレジスタ **nn** にストアします。
- **stor nn** – relative store (**stor nn**) は X レジスタの値をメモリレジスタ **nn** で指定したターゲットのメモリレジスタに置きます。
- **STOv nnn** – このコマンドは、ベクトル計算の結果をストアするのに使用します。その結果は、列 **nnn** に行ごとにストアされます。特別なケースとして、**STOv** コマンドの直前にアルファコマンドが置かれた場合、テキスト文字は出力列に書き込まれ、与えられた列は、テキストとしてフォーマットされます。
- **STOi nn** – このコマンドは X レジスタの値をメモリレジスタ **nn** で指定したアドレスにストアします。この数の整数部分は、行アドレスとして使用され、端数部分は、列番号です。特別なケースとして、**STOi** のコマンドの直前にアルファコマンドが置かれた場合、テキスト文字は、出力列に書き込まれます。与えられた列は、テキストとしてフォーマットされます。

例

```
STO 12          ; X レジスタの値をメモリレジスタ 12 にストア
stor 22         ; 22 で指定したメモリレジスタに、X レジスタをストアします
STOv 101        ; データウィンドウの列 101 の結果をストアします
STOi 15         ; メモリレジスタ 15 で指定したアドレスに、X レジスタをストアします
″Pass″          ; この文字列はアルファレジスタ内に置かれます
STOv 10         ; 列 10 にテキスト文字 ″Pass″ を置き、与えられたこの列は、テキスト列として
                  フォーマットされます
```

- **setcell** – 特定アドレスのセルに X レジスタの値をストアします。ストアされる値は、スタックの一番下のレジスタ (X) です。このアドレスは、スタックの次の 2 つのレジスタ (Y と Z) で指定されます。列アドレスの値は、1000 未満でなければならず、Y レジスタに配置されます。行アドレスの値は、Z レジスタに置かれます。コマンドを実行すると、アドレスはスタックから取り除かれ、X レジスタは影響を受けずに残されます。

例

```
35              ; 最初の値は、スタックに置かれます。これは行アドレス (Z) です
112             ; 次の値がスタックに置かれます。これは、列アドレス (Y) です
13.98           ; ストアされる値は、X レジスタのスタックに置かれます
setcell         ; 行 35、列 112 のセルに 13.98 を置きます
```

注意 : プログラムが作成されていない行または列に数値をストアしようとすると、データウィンドウが拡張されます。ただし、これはデータウィンドウの最大列数 (1000) と利用可能なメモリ量に制限されます。

エラー条件

エラーをテストするコマンドが 3 つ用意されています。最初のテストは、プログラム実行中のエラーを検出するのに使用します。他のふたつのテストは、recall indirect (RCLi) コマンドエラーの検出に使用します。

- test 0 – プログラムの実行中にいかなるタイプのエラーも発生しない場合には、このテストは真です。一度このフラグがセットされると、プログラムが停止するまでリセットすることができません。
- test 1 – このテストは、計算の値を使用する前にデータセルが数値を含んでいるかどうか判断するために使用します。このテストは、最後の RCLi コマンドが空のセルからデータを読み込もうとした場合に真です。次の RCLi 命令が実行された時に、フラグはリセットされます。
- test 2 – このテストは、計算の値を使用する前にデータセルがマスクされているかどうかを判断するために使用します。このテストは、最後の RCLi コマンドがマスクされたセルからデータを読み込もうとした場合に真です。次の RCLi 命令が実行されると、フラグはリセットされます。

例

```
RCLi 06          ; メモリレジスタ 06 で指定したセルからデータをリコール
test 2           ; データがマスクされているかどうかを判断
GTO 30           ; データがマスクされていれば、ラベル 30 へ
XEQ 50           ; データがマスクされていなければ、ラベル 50 を実行
```

データセルのクリア

- CLRv nnn – このコマンドはベクトル計算中にカレントセルをクリアします。このコマンドを使用したマクロは、選択したデータで行ごとに繰り返し実行されます。
- CLRi nn – このコマンドはメモリレジスタ nn のアドレスで指定したセルをクリアします。この数値の整数部分は、行アドレスとして使用され、数値の端数部分は列番号です。

例

```
CLRv 202         ; 列 202 のカレントセルの内容をクリア
CLRi 08          ; メモリレジスタ 08 のアドレスで指定したセルをクリア
```

データのマスク / マスク解除

- Mv nnn – このコマンドはベクトル計算中にカレントセルをマスクします。このコマンドを使用するマクロは、選択したデータで行ごとに繰り返し実行されます。
- UMv nnn – このコマンドはベクトル計算中にカレントセルをマスク解除します。このコマンドを使用するマクロは、行ごとに繰り返し実行されます。
- Mi nn – このコマンドはメモリレジスタ nn のアドレスで指定したセルをマスクします。この数値の整数部分は、行アドレスとして使用され、数値の端数部分は列番号です。
- UMi nn – このコマンドはメモリレジスタ nn のアドレスで指定したセルをマスク解除します。この数値の整数部分は、行アドレスとして使用され、数値の端数部分は列番号です。

例

```
Mv 109           ; 列 109 のカレントセルをマスク
UMv 200          ; 列 200 のカレントセルをマスク解除
Mi 18            ; メモリレジスタ 18 のアドレスで指定したセルをマスク
UMi 12           ; メモリレジスタ 12 のアドレスで指定したセルをマスク解除
```

その他のコマンド

- index – index コマンドは、ベクトルアドレス指定を使うマクロで使用します。カレント行のインデックス番号を計算で参照できるようにします。インデックス番号は、データ選択の先頭を基準にしています。インデックスコマンドは選択した最初の行に 0 の値を返します。
- size – size コマンドはカレントなデータ選択の行列数を決定します。選択が存在しなければ、データウィンドウ全体のサイズが決定されます (空白の行列を含む)。行数は、X レジスタに置かれ、列数は Y レジスタに置かれます。

1.4.7 文字列の使用

マクロ計算機では 1 つのプログラムで 100 までの文字列を許容されます。文字列はデータウィンドウの列名、入力を促す確認、または出力表示のラベルに使用します。

文字列は 31 文字まで使用できます。文字列の最初と最後に、二重引用符 (") を使用する必要があります。

例

```
" クリアする最初の列 "    ; 文字列のサンプル
```

アルファレジスタは、文字列の使用における重要な要素です。アルファレジスタは、文字列の一時的な記憶場所です。プログラムの実行中に文字列が検出されると、文字列はアルファレジスタに置かれます。

注意 : アルファレジスタとそのメモリロケーションは、通常データのメモリロケーションと計算機スタックから独立しています。データウィンドウから文字列を読み書きすることが可能です。データウィンドウからリコールされたテキストは、アルファレジスタにストアされ、データウィンドウに置かれるテキストは、アルファレジスタから書き込まれます。

- STOa n – STOa コマンドは、文字列がアルファレジスタに置かれるとこれをストアします。文字列は、0 から 9 までのアルファメモリロケーションのいずれかにストアされます。
- RCLa n – RCLa コマンドは特定のアルファメモリロケーションの内容をリコールし、その文字列をアルファレジスタに置きます。
- Ncol nnn – このコマンドは、アクティブなデータウィンドウのデータ列に名前を付けるために使用します。列番号 (0–999) の指定及び選択した列は、アルファレジスタの内容でリネームされます。

例

```
" 月 "          ; この文字はアルファレジスタに置かれます
STOa 1          ; 文字をアルファメモリロケーション 1 にストアします
RCLa 1          ; アルファメモリ 1 の内容をアルファレジスタにリコールします
Ncol 120        ; 列 120 の名前をアルファレジスタの内容にリネームします
```

- sload – このコマンドは、アルファ文字列または RCLa コマンドで指定されたプロットスクリプトをロードします。スクリプトは、起動ディレクトリ (プログラム、スタイルまたはマクロ)、またはカレントセッション中にロードしたスクリプトの最後のディレクトリからロードされます。
- srun – このコマンドは、カレントのプロットスクリプトを実行します。

例

```
"Script5"       ; この文字列はプロットスクリプトの名前です
sload           ; プロットスクリプトのロード
srun            ; カレントのプロットスクリプトを実行
```

1.4.8 入力と出力

- prmt n – prmt コマンドは実行中にマクロに 1 つの値を入力するために使用します。プログラムが実行される度に変わる情報や定数がある場合に、このコマンドは役に立ちます。

  このプロンプトコマンドが実行されると、タイトル付きのダイアログ、プロンプトの文字列および編集

フィールドが表示されます。フィールドにはデフォルト値が表示され、この値は変更することができます。デフォルト値は、prmt コマンドを実行されたときにアクセスする同じメモリレジスタに値をストアすることによって定義できます。

Figure 1-13 は階乗マクロの以下の部分が実行されたときに表示されるダイアログです。

```
" 数値入力 "                                    ; アルファレジスタに文字列を入力
STOa 2                                  ; アルファレジスタをアルファメモリ 2 にストア
5.0                                ; これはダイアログに表示されるデフォルト値です
STO 02                                  ; この値をメモリレジスタ 02 にストア
" 階乗 "                          ; この文字列はダイアログのタイトルとして使用されます
prmt 2                    ; ユーザに値の入力を促します。その値をレジスタ 02 にストア
( 以下残りのマクロ )
```

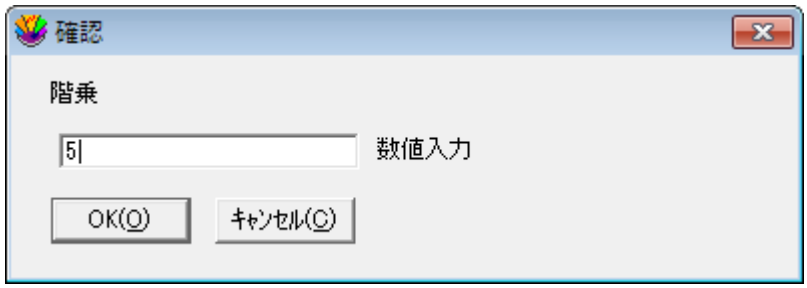

Figure 1-13 プロンプトダイアログ

このコマンドは、アルファレジスタ、アルファメモリロケーション、数値メモリロケーションの内容を使用します。アルファレジスタの内容は、ダイアログのタイトルとして使用し、アルファメモリ **n** は、プロンプト文字列として使用されます。**OK** をクリックすると、ダイアログに入力された値がデータメモリロケーション **n** に置かれます。そのため、最初の 10 のデータメモリレジスタ (0- 9) は、10 のアルファメモリ (0 – 9) と一致します。

- **inpt n – inpt** 関数もプログラムに入力を提供するために使用します。プロンプトコマンドに比べてコマンドは、一度に 6 つの異なる値を入力できる点が優れています。

  inpt コマンドを実行すると、ダイアログにタイトルと、6 つまでの異なる文字列、数値フィールドが表示されます。各フィールドにはデフォルト値が表示され、この値は変更することができます。右側のフィールドに対応するメモリレジスタのデータに定数をストアすることで、デフォルト値を定義することができます。

  Figure 1-14 は以下のクリアマクロが実行されたとき表示されるダイアログです。
  マクロ全体についてはセクション 1.3.5 を参照してください。

```
" クリアする最初の列 "                ; アルファレジスタに文字列を入力
STOa 0                                ; アルファレジスタ 0 にストア
0.0                                   ; デフォルト値を入力
STO 00                                ; レジスタ 00 にデフォルト値をストア
" クリアする最後の列 "                ; アルファレジスタに文字列を入力
STOa 1                                ; アルファレジスタ 1 にストア
1.0                                   ; デフォルト値を入力
STO 01                                ; レジスタ 01 にデフォルト値をストア
" クリアマクロ – 複数列 "             ; アルファレジスタにタイトル文字列を入力
inpt 2                                ; 2 つの値を入力。レジスタ 00 と 01 にストアします
( 以下残りのマクロ )
```

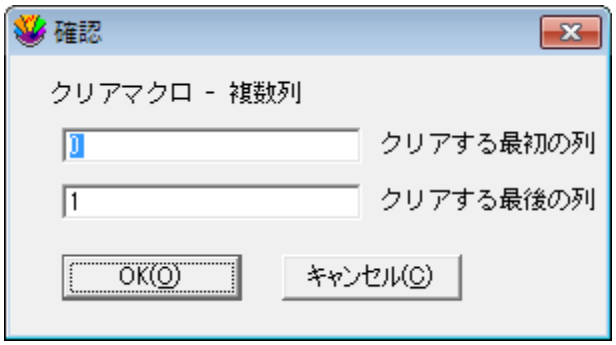

Figure 1-14 入力ダイアログ

このコマンドはアルファレジスタの内容を使用して、6 つのアルファとデータメモリロケーション (0–5) でを使用します。ダイアログのタイトルは、アルファレジスタの内容から採用されます。アルファメモリロケーションの内容は、デフォルト値を含むフィールドの右に表示されます。OK をクリックすると、ダイアログに入力された値は、適切なデータメモリレジスタに配置されます。

注意 : inpt コマンドについては、n の値は 1 から 6 の間でなければなりません。

- **view n** – **view** コマンドは計算結果を表示するのに役立ちます。9 つまでの変数が一度に表示できます。

  このコマンドを実行すると、タイトルと最大 9 つの異なる出力値を含むダイアログが表示されます。それぞれの出力変数は、その意味を記述するための文字列を持つこともできます。出力値は、メモリレジスタ 00 から 08 までにストアされ、文字列は、これに対応するアルファメモリロケーションにストアされます。ダイアログのタイトルは、アルファレジスタから採用されます。

  Figure 1-15 は入力ダイアログで、Figure 1-16 は以下の階乗マクロが実行されたときの出力ダイアログです。

```
" 数値入力 "                          ; 文字列をアルファレジスタに入力
STOa 1                          ; アルファレジスタをアルファメモリ 1 にストア
" 階乗 "                  ; この文字列はダイアログのタイトルとして使用されます
prmt 1              ; ユーザーに値の入力を促します。これをレジスタ 01 にストア
1
STO 00                                  ; メモリレジスタ 00 に 1 をストア
LBL 00
RCL 01                          ; レジスタ 01 の内容を X レジスタにリコール
MUL 00                          ; 00 の内容と X レジスタの内容を掛け算
0.0
DSE 01          ; レジスタ 01 を減少させ、X レジスタに対しこれをテストします
GTO 00                  ; レジスタ 01> X レジスタならば、LBL00 に移動
" 結果 ="                               ; アルファレジスタに文字列をロード
view 1                          ; メモリレジスタ 00 から )1 つの出力変数を表示
STOP
```

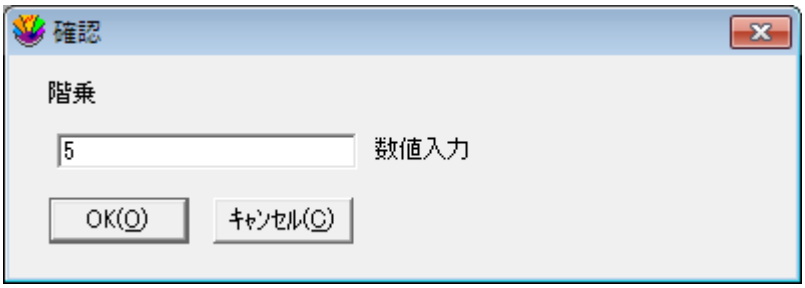

Figure 1-15 プロンプトダイアログ

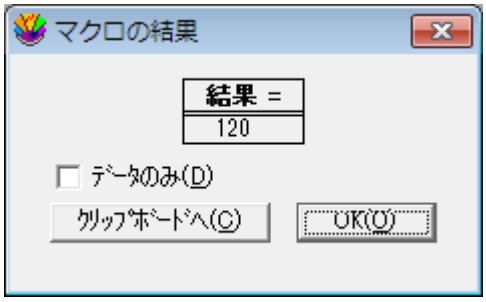

Figure 1-16 ビューダイアログ

1.4.9 レジスタ計算

単数値関数

- abs – この関数はXレジスタにある値の絶対値を決定します。
- erf – Xレジスタにある値の誤差関数を計算します。
- erfc – Xレジスタにある値の相補誤差関数を計算します。
- exp – eの値(2.7182...)をXレジスタで指定した累乗を計算します。
- factorial – Xレジスタの値の階乗を計算します。
- ln – Xレジスタの値の自然対数(底e)を計算します。
- log – Xレジスタの値の常用対数(底10)を計算します。
- ran# – 0と1の間で乱数を発生させます。
- sqrt – Xレジスタの値の平方根を計算します。
- +/– – Xレジスタの値の符号を変更します。
- pi – Xレジスタに π(3.14159...)の近似値を置きます。
- 1/x – Xレジスタの値の逆数を計算します
- x^2 – Xレジスタの値の二乗を計算します。
- 10^x – Xレジスタの値の10の累乗に計算します。
- int – この関数はXレジスタの整数部分を決定します。Xレジスタに5.08がある場合、intコマンドにより5.00がXレジスタにストアされます。
- frac – この関数はXレジスタの小数部分を決定します。Xレジスタに5.08がある場合、fracコマンドにより0.08がXレジスタにストアされます。

複数値関数

- x<>y – XとYレジスタの値を交換します。
- y^x – Yレジスタの数値をXレジスタで指定した累乗にします。Yの負値は、Xが整数の時だけ許されます。
- corr – XとYレジスタで指定した列の線形相関を計算します。この関数を使用するには、比較されるふたつの列は、データポイント数が同じでなければなりません。
- p–>r – XとYレジスタの数(Rと φ)を直交座標(XとY)に変換します。φ は計算機の設定によって、度またはラジアンで表現できます。
- r–>p – XとYレジスタの数(XとY)を極座標(Rと φ)に変換します。φ は計算機の設定によって、度またはラジアンで表現できます。

加算

- + – Yレジスタの値をXレジスタの値に追加します。
- Add nn – Xレジスタの値を指定されたメモリレジスタ **nn** に追加します。
- Addi nn – Xレジスタの値をメモリレジスタ **nn** で指定したメモリロケーションに追加します。
- rsum nnn – 列 **nnn** の累積和（running sum）を計算し、結果をXレジスタに置きます。

減算

- – – Y レジスタの値から X レジスタの値を減算します。
- Sub nn – メモリレジスタ nn から X レジスタの値を減算します。
- Subi nn – メモリレジスタ nn で指定したメモリロケーションから X レジスタの値を減算します。
- diff nnn – 各値と列 nnn に続く値の違いを計算します。

乗算

- * – Y レジスタの値と X レジスタの値を掛け算します。
- Mul nn – メモリレジスタ nn と X レジスタの値を掛け算します。
- Muli nn – メモリレジスタ **nn** で指定したメモリロケーションと X レジスタの値を掛け算します。

除算

- / – Y レジスタの値を X レジスタの値で割り算します。
- Div nn – メモリレジスタ nn を X レジスタの値で割り算します。
- Divi nn – メモリレジスタ nn で指定したメモリロケーションを X レジスタの値で割り算します。
- mod – mod 関数は XmodY の整数値を計算し、X レジスタに結果を置きます。

分布

- norm – この関数は X レジスタの値の正規分布を計算します。
- inorm – この関数は X レジスタの値の逆正規分布関数を計算します。

三角関数コマンド

- cos – X レジスタの値のコサインを求めます。マクロ計算機の設定により、値はラジアンまたは度になります。
- inv–cos – X レジスタの値の逆コサインを求めます。マクロ計算機の設定により、値はラジアンまたは度になります。
- inv–sin – X レジスタの値の逆サインを求めます。マクロ計算機の設定により、値はラジアンまたは度になります。
- inv–tan – X レジスタの値の逆タンジェントを求めます。マクロ計算機の設定により、値はラジアンまたは度になります。
- sin – X レジスタの値のサインを求めます。マクロ計算機の設定により、値はラジアンまたは度になります。
- tan – X レジスタの値のタンジェントを求めます。マクロ計算機の設定により、値はラジアンまたは度になります。
- –>deg – X レジスタにラジアンの値が含まれていると、この関数は度で対応する値を計算します。
- –>rad – X レジスタに度の値が含まれていると、この関数はラジアンで対応する値を計算します。

その他のコマンド

- CM – メモリレジスタの内容を 0 にリセットします。
- FIX n – このコマンドを使用すると、計算機は指定した有効桁数にまるめた数字を表示します。
- SCI n – このコマンドを使用すると、計算機は一桁の数字を左に、そして指定した桁数を小数点の右に表示します。
- tbl – このコマンドは x 列と y 列のデータによって定義された関数に基づいて、x の線形近似を行います。x と y の列番号は、0 – 999 の範囲で設定できます。結果は、x の値が与えられるため、この関数は y の線形

推定値を求めます。このコマンドはスタックの最後の 3 つのレジスタ (X、Y、Z) を使用します。

Z = x 値
Y = y 列番号
X = x 列番号

例

c0 が x 列、c1 が y 列、そして 2.5 が x 値 (Z=2.5、Y=1、X=0) と仮定します。さらに、c0 は 0 で始まり 1 で (0、1、2、3...) 増加する級数で、c1 は c0 の二乗です。次のマクロは、2.5 の二乗値を推測するために、c1 の二乗値の間を線形補間します。

```
2.5                                  ; これは x 値 (Z) です
1.0                                  ; これは y 列番号 (Y) です
0.0                                  ; これは x 列番号 (X) です
tbl                                  ; 線形近似を計算します
STO 00                               ; 結果をレジスタ 00 にストア
alpha " 結果 ="                      ; この文字列はダイアログの値の前にきます
STOa 0                               ; 文字列をアルファメモリ 0 にストア
alpha " 式 :"                        ; この文字列をダイアログのタイトルとして使用
view 1                               ; 線形近似の結果を表示
STOP                                 ; プログラムの実行を中断
```

マクロの結果は Figure 1-17 に示します。

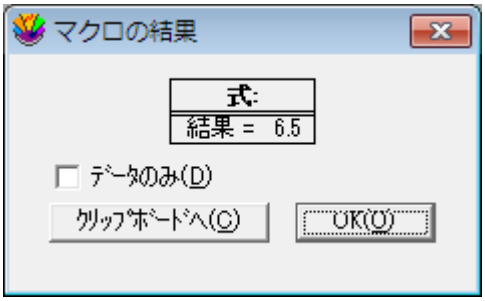

Figure 1-17 2.5 の二乗の線形近似

1.4.10 統計

指定したデータグループの統計

以下のコマンドは特定のデータセットに統計処理を行います。使用するデータは、コマンドが実行される前に入力された行列アドレスで決まります。これらのコマンドはデータウィンドウ全体で操作できます。これらのコマンドのアドレスは、スタック下の 4 つのレジスタ (S0 – S3) にストアされます。これらのコマンド全ては以下に示すように同じパターンに従います。

S3 には開始行の番号が含まれます。
S2 には終了行の番号が含まれます。
S1 には開始列の番号が含まれます。
S0(X レジスタ) には終了列の番号が含まれます。

コマンドが実行された後は、スタックから 4 つのアドレスが削除され、関数の結果は X レジスタに置かれます。

- **cmin** – 指定したデータ範囲内の最小値を求めます。
- **cmax** – 指定したデータ範囲内の最大値を求めます。
- **csum** – データ範囲の全ての値の合計を計算します。

- **npts** – 指定したデータ範囲内のポイント数を求めます。
- **mean** – データ範囲の平均を計算します。
- **median** – データ範囲の中央値を求めます。
- **rms** – データ範囲の二乗平均平方根 (root mean square) を計算します。
- **std** – データ範囲の標準偏差を計算します。
- **var** – データ範囲の分散を計算します
- **stderr** – データ範囲の標準誤差を計算します。
- **skew** – データ範囲の歪度を計算します。
- **kurtosis** – データ範囲の尖度を計算します。

例

```
0                   ; 最初の値、開始行アドレス (S3) は、スタックに置かれます。
127                 ; 2 つめの値、終了行アドレス (S2) は、スタックに置かれます。
100                 ; 3 つめの値、開始列番号 (S1) は、スタックに置かれます。
101                 ; 最後の値、終了列番号 (S0) は、スタックに置かれます。
cmin                ; このデータ範囲内の最小値を求め、その結果を X レジスタに置きます。
```

行統計

次のコマンドで行ごとの統計処理を行うことができます。使用するデータは、コマンドが実行される前に入力された列アドレスで決まります。これらのコマンドは、データウィンドウ全体で操作できます。これらのコマンドのアドレスは、スタック下の 2 つのレジスタ (S0 と S1) にストアされます。これらのコマンド全ては以下に示すように同じパターンに従います。

S1 開始列の数値を含みます。
S0 (X レジスタ) 終了列の数値を含みます。

コマンドが実行された後は、スタックから 2 つのアドレスが削除され、関数の結果は、X レジスタに置かれます。

- **rstat 00** – 指定された列範囲にわたる最小値を求めます。
- **rstat 01** – 指定された列範囲にわたる最大値を求めます。
- **rstat 02** – 指定された列範囲にわたるすべての値の合計を計算します。
- **rstat 03** – 指定された列範囲にわたるポイント数を求めます。
- **rstat 04** – 指定された列範囲にわたる平均値を求めます。
- **rstat 05** – 指定された列範囲にわたる中央値を求めます。
- **rstat 06** – 指定された列範囲にわたる二乗平均平方根 (RMS) を求めます。
- **rstat 07** – 指定された列範囲にわたる標準偏差を求めます。
- **rstat 08** – 指定された列範囲にわたる分散を求めます。
- **rstat 09** – 指定された列範囲にわたる標準誤差を求めます。
- **rstat 10** – 指定された列範囲にわたる歪度を求めます。
- **rstat 11** – 指定された列範囲にわたる尖度を求めます。

1.4.11 回帰関数

データウィンドウのいずれかの列に回帰曲線が適用されている場合には、次のコマンドが役に立ちます。各コマンドは特定の x の値 について f(x) を求めます。X レジスタの値は、x の値として使用されます。適切な回帰曲線を持つデータ列のアドレスは、コマンドの nnn の値として使用されます。

注意 : x が元のデータ範囲にある場合のみ、解は正確です。元の範囲外に x が位置する場合は、解は線形補間されます。

非線形回帰曲線

- gen nnn – このコマンドは X レジスタで指定された値で、列 nnn に適用する一般回帰曲線の値を計算します。
- genf nnn – このコマンドは gen nnn コマンドに似ています。相違点は genf nnn コマンドは、x の値を求めるために使用される一般回帰曲線を決定するためにアルファレジスタを使用することです。これは、複数の一般回帰曲線がデータ列に適用される場合に便利です。

例

```
"fit1"          ; 一般フィットの名前は、アルファレジスタに置かれます。
25.48           ;x の値を X レジスタに置きます
genf 150        ;150 列に適用された回帰曲線 fit1 に基づいて y の値を決定します。
```

最小自乗回帰曲線

- lin nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用される線形回帰曲線の値を計算します。
- poly nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用される多項回帰曲線の値を計算します。
- expr nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用される指数回帰曲線の値を計算します。
- logr nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用される対数回帰曲線の値を計算します。
- pow nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用される累乗回帰曲線の値を計算します。

スムージング回帰曲線

- smh nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用されるスムーズ回帰曲線の値を計算します。
- wgt nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用される加重回帰曲線の値を計算します。
- spln nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用される 3 次スプライン回帰曲線の値を計算します。
- intp nnn – このコマンドは X レジスタで指定した値で、列 nnn に適用される補間回帰曲線の値を計算します。

例

```
56.74           ;x の値を X レジスタに置きます
spln 102        ; 列 102. に適用される 3 次スプライン回帰曲線に基づいて y の値を決定します
```

1.4.12 その他のコマンド

- peek nn – このコマンドは X レジスタの内容を変更することなく、指定した任意のメモリレジスタ nn の値

を表示します。このコマンドの結果は、マクロ計算機のディスプレイ領域に置かれます。間接的なリコールまたはストア操作のアドレスは、プログラムで計算されるマクロをデバッグするのに重要な助けとなります。

- abort – abort コマンドはマクロの実行を中断するために使用します。これは､プログラムの実行中にエラーが検出された場合に特に役立ちます。
- version – このコマンドは現在の KaleidaGraph のバージョン番号を返します。バージョン番号は、100 で乗じて X レジスタに置かれます。
- up – このコマンドはスタックの値の全てを 1 ポジション上げます。
- down – このコマンドはスタックの値の全てを 1 ポジション下げます。

例

スタックレジスタ	元のスタック	Down コマンド後のスタック	Up コマンド後のスタック
S3	5	5	8
S2	8	5	15
S1	15	8	29
S0	29	15	0