カルキングプロフェッショナルの概要

カルキング8に対して追加した事柄に焦点を当てて説明いたします。ただしここではカルキング8に関しては触れません。これらに関してはカルキング8の方の説明をご覧いただきたいと思います。

●製品提供形態の変更

パッケージからオンライン配布への変更です。このことにより多くの利点が認められます。環境負荷が少ない。いち早く変更･修正を反映できます。紙のマニュアルからPDF形式のマニュアルへの移行に伴い多くの利点が発生します。より迅速に製品出荷が可能になります。

●計算機能の大幅な拡張

学術参考書、設計書等で記載されている数式を、そのままの計算式で計算できるかどうかが、いわゆる数学ソフトの最重要目標の一つです。この目標達成のために、カルキングプロフェッショナルが実現した項目は以下のようなものです。詳しくは、このご案内の後半部で、説明しています。

(1)数学関数、数学機能の大幅追加

(2)数学関数の計算精度の大幅強化

(3)数値計算のみならず、記号処理機能の大幅強化

(4)実数モード計算機能と同程度の計算精度を持つ複素演算機能の実現

(5)カルキングスクリプト機能（プログラミング）の進化

一例をあげてみます。ここでは、いくつか数学での専門用語が出てきますが、これらの意味を御存じでなくても気になさらないでください。

以下に示されている数式は、リーマンのゼータ関数の計算で、引数が、4 で割って余りが 1 になる場合の、効率の良い公式です。ここではベルヌーイ数が使われています。カルキングではベルヌーイ数はBernoulli_constという名前で定義されています。これを公式のように B で参照するためには

```
{B,k}=search_name("Bernoulli_const")
```

を実行することにより可能になります。(別名機能)

ここではζ(5)の値を計算します。

n=(5-1)/2　　代入定義

以下では30桁の精度で計算しています。二番目のΣの上限値は公式は∞ですが一応 100 とします。

$$\frac{(2p)^{2n+1}}{2n}\sum_{k=0}^{n/2}(-1)^k(2n+2-4k)\frac{B_{2k}}{(2k)!}\frac{B_{2n+2-2k}}{(2n+2-2k)!}-2\sum_{k=1}^{100}\frac{e^{2pk}\left(1+\frac{4pk}{2n}\right)-1}{k^{2n+1}\left(e^{2pk}-1\right)^2}$$

=1.036927755143369926331365486 46

Bはベルヌーイの定数です。

注　カルキングでは組み込み関数としてこの公式をもとに高精度、高速版を組み込んでいます。

ζ(5)=1.036927755143369926331365486 46

このようにユーザの方が、実際の数式をもとにしたいろいろな検算等を、効率よく実現できます。
汎用言語（C、C++、JAVA、FORTRAN等）で、このような公式をもとに計算するプログラムを作成しようとすると、
ベテランの方でも10倍以上の時間がかかるでしょう。
組み込みシステムプログラムとして、C言語でプログラム化するような場面でも、カルキングは活用できます。
Cのプログラムと公式では、表現上のギャップが大きくなっています。
カルキングでこのギャップを埋める作業を、画面上で容易に行えます。

これまでカルキングは複素数については、複素数型を持っている汎用言語と同じ計算精度（64ビット）でした。
これは有効桁数 15, 6 桁に相当します。
今回のカルキングプロフェッショナルでは、実数型と同じ、1000桁まで可能となりました。
これでカルキングプロフェッショナルには、数学ソフトとして新しい世界が見えてきました。
多くの技術計算では、このような精度は全く必要ありません。
しかし実際に高精度複素数に関するプログラムを多数開発していく過程で、この意義に気付きました。
これによりストレス軽減、文法規則の単純さ、数式やプログラムの表現上の美しさ、単純さが実現できました。
例として、先ほどのリーマンのゼータ関数の計算では、Σの上限値をほとんど気にせずに計算できます。
a+b-c=a-c+b　この式のように数学の世界では正しくても、コンピュータの世界では、a+b が許される上限値を
超え計算できないが、a-c+b の順序では正しく計算されるようなことが、頻繁に起こります。
カルキングでは稀です。

次に強調したいのは、記号計算機能に関してです。
汎用言語（C、C++、JAVA,FORTRAN等）と比べて数学ソフトの大きな特徴は、数値計算だけでなく
記号計算機能も備わっていることです。
カルキングプロフェッショナルでは、この記号計算処理を「高水準」でプログラミングできる機能を、初めて
搭載しました。 この機能を利用して、Laplace変換、逆Laplace変換、関数の級数展開、部分分数分解
多変数因数分解、常備微分方程式の級数展開等の処理プログラムを開発いたしました。
これらのいくつかは　カルキングプロフェッショナルに実装されています。

進化の始まり

カルキングスクリプト関数は、これまで、カルキングファイルの数式からのみ呼び出すことができ、カルキング
システム記述言語のC++ソースからは呼び出すことができませんでした。
遂にこの大きな限界を超えるための技術を確立いたしました。
これにより、スクリプト機能は第二のシステム記述言語になりました。
ユーザの方もカルキングスクリプトを活用することができます。
複雑で、かつ多大な労力がかかるために諦めていた課題へ是非挑戦していただきたいと思います。

以上計算機能を中心に説明いたしましたが、数式ワープロ面、作図機能、表計算機能等に関しても新機能を
追加いたしました。

プロテクトファイルの実現

カルキングファイル内の情報の非表示、ファイルの改変の防止等を目的として、プロテクトファイルを実現
しました。今回、数学機能を実装したライブラリ関数群もこのプロテクトファイル形式で提供されています。
ごく一部に関しては、弊社の企業ノウハウとして非表示にしております。
ユーザの方々も、このような非表示の情報を含むカルキングファイルの販売も可能になりました。

以下従来のカルキングバージョン8に対して新たに追加した部分のみの新機能を中心に概要を説明いたします。

新機能（カルキングプロフェッショナル版の独自機能）

● ワープロ面

大規模ファイルでのスクロールの適正化

グリッド表示の高速化

テキスト貼り付けの改善　（分割フレームとして貼り付ける）

「他のアプリケーション」からの貼り付けの「テキスト」貼り付けでは、
一つのフレームにまとめて貼り付けていました。
今回は段落や空白行で、フレームを分割して貼り付けるようにしました。
これにより、その後の編集がより便利になりました。

カーソル移動の柔軟性とより適正化

フレームID番号を指定してジャンプ

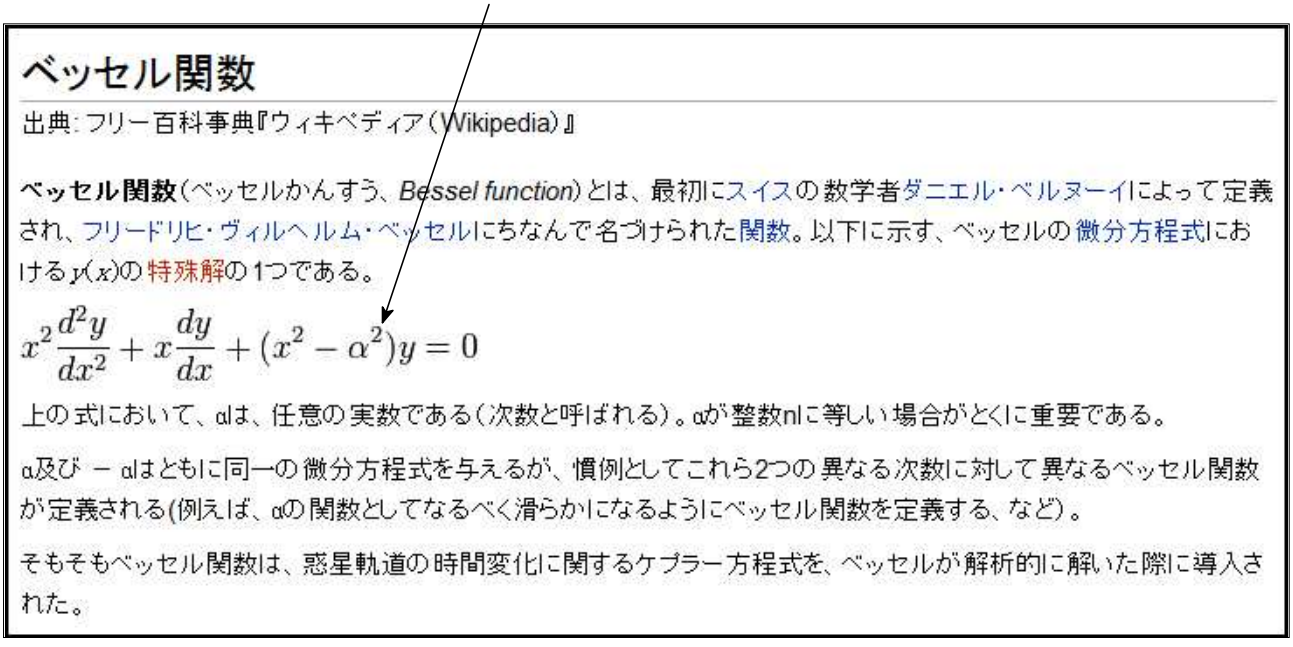
カルキングでのスクリプトの実行中に、発生したエラーメッセージに、
どのフレームで発生したかを示すフレーム番号を出力することにしました。
この番号をもとに、そのフレームにジャンプできるようにしました。

作図オブジェクトをOLE貼り付けの上に上書き可能

カルキングでは文章数式、表、関数グラフ、作図部品等の重ね書きができます。
このときに作図部品群のレイヤーの表示順序を指定できますが、他のアプリケーションの
オブジェクトに関してはレイヤーの表示順序指定ができませんでした。
今回これを可能にしました。

次のように、他のアプリケーションの図面にカルキングの作図機能で矢印の上書き等が可能になりました。

使用例　　$J_{\alpha}(x)$　　添字のαとは　　Wikipediaとベッセル関数で検索した画面



ベッセル関数

出典: フリー百科事典『ウィキペディア（Wikipedia）』

ベッセル関数（ベッセルかんすう、*Bessel function*）とは、最初にスイスの数学者ダニエル・ベルヌーイによって定義され、フリードリヒ・ヴィルヘルム・ベッセルにちなんで名づけられた関数。以下に示す、ベッセルの微分方程式における$y(x)$の特殊解の1つである。

$$x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+(x^2-\alpha^2)y=0$$

上の式において、αは、任意の実数である（次数と呼ばれる）。αが整数nに等しい場合がとくに重要である。

α及び $-\alpha$はともに同一の微分方程式を与えるが、慣例としてこれら2つの異なる次数に対して異なるベッセル関数が定義される(例えば、αの関数としてなるべく滑らかになるようにベッセル関数を定義する、など)。

そもそもベッセル関数は、惑星軌道の時間変化に関するケプラー方程式を、ベッセルが解析的に解いた際に導入された。

● 近似代数計算

従来、近似モードの代数計算は実行できませんでした。あとで述べますLaplace変換を利用して、
電気回路の過渡現象のような、現実に即した問題を解くときには、どうしても近似モードの代数計算が
不可欠になります。部分分数分解と逆Laplace変換で実現しました。
一般には置き換え計算を経由すれば実現できます。

$$\text{partial\_fract\_decompose}\left(\frac{4s^4+2s^2-67}{s^6+5s^5-12s^4+28s^3-12s^2+2s+123}\right)\downarrow$$

$$=\frac{0.74108671s-1.1667342}{s^2-3.1192958s+3.8995381}+\frac{-0.16970508s-0.19071969}{s^2-0.37536538s+3.436221}+\frac{-0.40035634}{s+7.223988}+\frac{-0.17102529}{s+1.2706731}$$

近似モードでの部分分数分解　　表示精度8桁

$$L^{-1}\left\{\frac{0.74108671s-1.1667342}{s^2-3.1192958s+3.8995381}+\frac{-0.16970508s-0.19071969}{s^2-0.37536538s+3.436221}+\frac{-0.40035634}{s+7.223988}+\frac{-0.17102529}{s+1.2706731}\right\}\downarrow$$

$$=-0.400356e^{-7.22399t}-0.171025e^{-1.27067t}-0.120688e^{0.187683t}\sin(1.84418t)-\downarrow$$

$$0.169705e^{0.187683t}\cos(1.84418t)+0.954282e^{1.55965t}\sin(1.21121t)+0.741087e^{1.55965t}\cos(1.21121t)$$

近似モードでの逆Laplace変換　　表示精度8桁

● 基本演算での複素数計算の高精度化(最大1000桁複素数)

高度数学関数を多数サポートしました。
同時にIEEE754浮動小数点をはるかに超える計算精度を複素数でもサポートしました。
スクリプトでも使えます。以下は100桁計算の例です

$$e^{\sqrt{3}i}=-0.160556538574690627402747929079680481541644337729381562435090840093843709084146049310857014719128989 7 + 0.987026644990353783993324392439670388957092614144764957307888640040540682154936103974525800342238616 9i$$

● 既存関数群の機能強化

ln関数 (1000桁、1000桁複素数化)

$$\ln\left(10\times10^{5000}\right)=11515.228050063222465773975264876505402213108544632493 65$$

ガンマ関数 (高精度化、高精度複素数化)

$$\Gamma(800)=9.638162641692325051807992472187854507444455022700127989542637425424635647838663367088634622444237914\times10^{1973}$$

arg関数 (1000桁、1000桁複素数化)

$$\arg(2.1+4.3i)=1.116494240101580404559999633576820266517862674440016518040884009154937431479787748820492817064272144$$

三角関数群の（高精度化、高精度複素数化）

$\sin(2+3i)$=9.154499146911429573467299544609832559158860568765182977899 83 - 4.168906959966564350754813058853754843573565604758055889965 48i

双曲線関数群の（高精度化、高精度複素数化）

$\cosh(0.67+0.42i)$=1.125813963629033520843406589261291427760014390761465775063 17 + 0.294103086867861172130642890540612107521029892193905363853864i

累乗数（高精度化、高精度複素数化）

$\left(\sqrt{7}\right)^{4.123i}$=-0.644902121935772854970261462754260788190492876863423566641847415097607388286865767021758847441200540 7 - 0.764265171993816266846186278154460938605858659860978192246690944242380790377472420391363458796137605 9i

● 数学関数群の追加

nは整数　　νは実数　　zは複素数

第一種ベッセル関数　　例　$J_{\nu}(z)$

計算例　$J_{2.5}(3.09)$=0.424040957083466148981630775183219905827357785360 32

$J_{5}\left(\sqrt{3}+\sqrt{7i}\right)$=-0.044776521185492601345775728206910671728074622999615 + 0.191297283636313914328185584054955532217890808019 51i

微分例　$\frac{d}{dx}J_{n}(x)=\frac{-nJ_{n}(x)+xJ_{n-1}(x)}{x}$

第二種ベッセル関数　　例　$Y_{\nu}(z)$

計算例　$Y_{5.2}(12.5)$=-0.235581135081821174481120006384113236437647040980 96

$Y_{4}\left(\frac{1}{3}\left(\sqrt{3}+\sqrt{7i}\right)\right)$=2.106872654055747060238429428283024821108 + 9.688697193026150506992235632635764300182i

微分例　$\frac{d}{dx}Y_{4}(x)=\frac{xY_{3}(x)-4Y_{4}(x)}{x}$

ハンケル関数1　　例　$H_{n}^{(1)}(z)$

ハンケル関数2　　例　$H_{n}^{(2)}(z)$

変形ベッセル関数1　　例　$I_{n}(z)$

計算例　$I_{3}(0.378)$=0.00113529588528147

$I_{5}\left(\frac{1}{3}\left(\sqrt{3}+\sqrt{7i}\right)\right)$=-0.000956400876853642 + 0.000781172472933835i

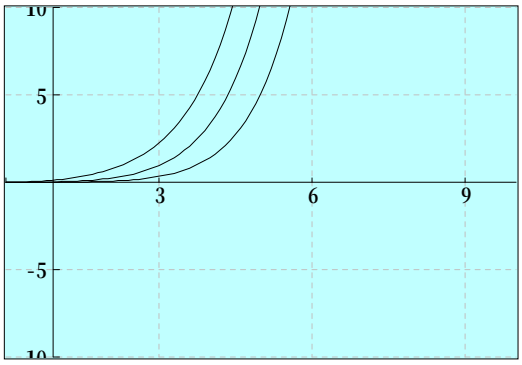
微分例　$\frac{d}{dx}I_{3}(x)=\frac{xI_{2}(x)-3I_{3}(x)}{x}$

関数グラフ

$y=I_2(x)$

$y=I_3(x)$

$y=I_4(x)$



変形ベッセル関数2　　例　$K_n(z)$

エルミート多項式　　例　$H_n(z)$

ラゲール多項式　　例　$L_n(z)$

拡張ラゲール多項式　　例　$L_n^{\alpha}(z)$

第1種チェビシェフ多項式　　例　$T_n(z)$

第2種チェビシェフ多項式　　例　$U_n(z)$

Sin積分関数　　例　$Si(z)$

計算例　$Si(12.6)=1.49220613972864$　　$Si(x)=\int_0^x \frac{\sin t}{t}dt$　(数学での定義)

$Si(12.6)=1.49220613972864$

Cos積分関数　　例　$Ci(z)$

指数積分関数　　例　$E_n(x)$

Ψ関数　　例　$\Psi(z)$

ゼータ関数　　例　$\zeta(x)$

計算例　$\zeta(5)=1.036927755143369926331365486457034168057080919501 9$

$\zeta(7.24)=1.007022550411735191843252270016010855383988935567 7$

$\zeta(s)=\sum_{n=1}^{\infty}\frac{1}{n^s}$　(数学での定義)

超幾何級数

$${}_0F_1\left(;\frac{1}{2};2+3i\right)=-3.12472212875648+13.8634771324033i$$

$${}_2F_1\left(\frac{1}{2},1;\frac{3}{2};0.5\right)=1.24645048028047$$

● 一般システム関数

ispolynomial関数　多項式かどうかを判定します。

Pochhammer symbol

$(8)_6=1235520$　　$8\times9\times10\times11\times12\times13=1235520$

Pochhammer symbolの括弧は関数のカッコです。

● スクリプト(プログラミング)機能の強化

実行機能

スクリプト内での記号式計算、関数定義、方程式機能の実行

無条件ループ

従来繰り返しはfor文のみでしたが無限ループ型も追加しました。

$$
\left[\begin{array}{l}
arr_k=arr_k+n \\
k=k+fun(t) \\
\mathbf{break}\quad k\geqq 100
\end{array}\right.
$$

last文

カルキングにはgoto文はありません。そこでブロックからの脱出のための制御文を新設しました。
これによりプログラミングがよりやさしく表現可能になりました。

$$
\left[\begin{array}{l}
k=k+n \\
a=\zeta(k) \\
\mathbf{last}\quad a>1.000023 \\
n=2n+1
\end{array}\right.
$$

return文無しの関数

従来は必ず値を返す必要がありましたが、返す必要のない手続きもサポートされました。

return文での多重代入機構

関数の入力パラメータは複数個可能でしたが、戻り値は従来一個でした。
これを複数個可能にしました。

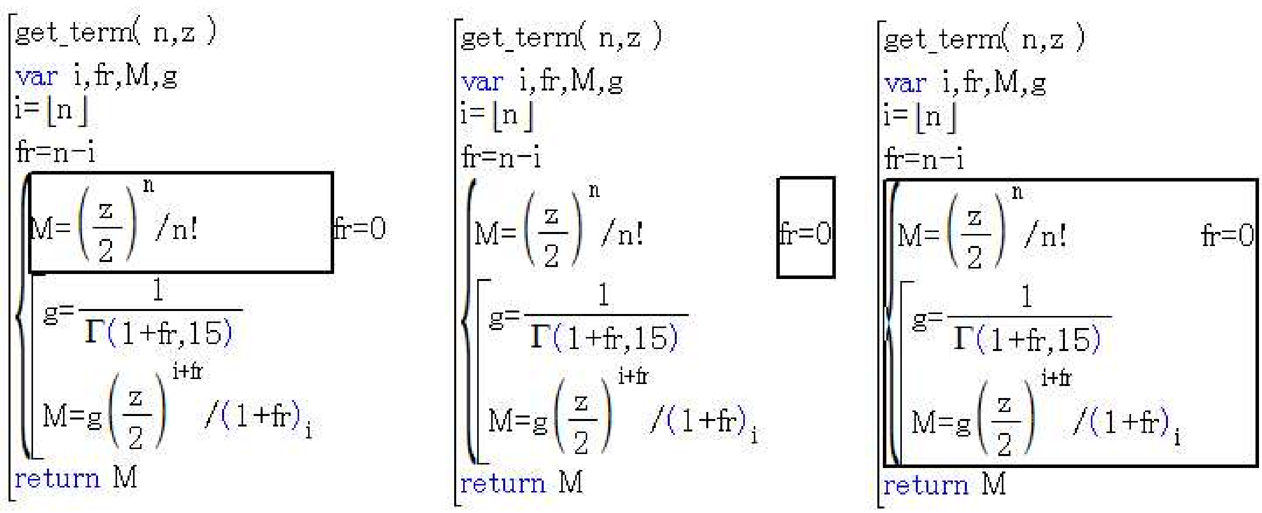
プロパティ機能の明確化

スクリプト関数は、基本的にプロパティーの指定が不要です。
関数が呼ばれた時のプロパティーが引き継がれるためです。
これにより、汎用プログラミングが容易になっています。
いくつかの例外は、配列宣言、特定の指定（分数に制限したいとか、
計算精度を必ず100桁にしたいとか）の時のみです。

編集機能

編集箇所の枠表示

次のようにカーソル位置を含むブロックが表示されます。
これにより編集が一段と容易になります。



カーソル移動の正確化

条件式の付加･削除が簡単に行えます

スクリプトファイルサイズの縮小

スクリプト関数群（スクリプトでのみ使用可能）

以下の記号処理関数群を使って、数学的記号処理の核心部分が記述可能になりました。
今回の新規サポート関数のかなりの部分の実装で使われました。
代表例として無限級数展開、ラプラス変換、ラプラス逆変換、部分分数分解等があります。

symbolic_calculus関数（代数計算関数）

symbolic_assign関数（代数代入定義関数）

replace関数（置き換え計算関数）

function関数（関数定義関数）

prime_factor関数（素因数分解関数）

factor関数（因数分解関数）

デバッグ機能

エラー発生時点での関数履歴表示

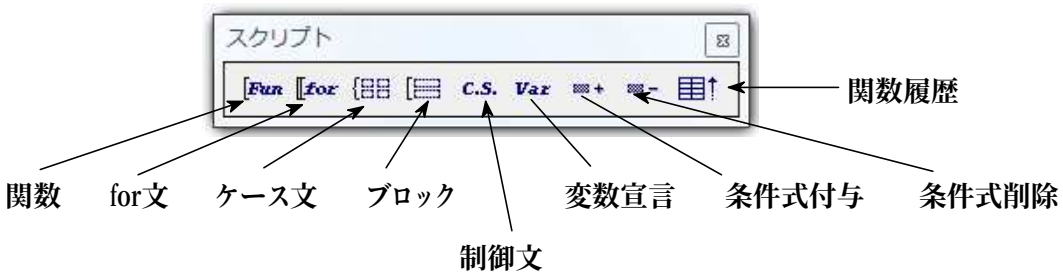
stop文やエラー発生時点までの式の実行回数表示

次の表は関数履歴の例です。
appeare_cnt関数内のstop文で止まった状況です。
appeare_cnt関数はget_bind_eq関数から呼ばれています。
get_bind_eq関数はSQPE関数から呼ばれています。...
グリーン表示されたSQPE、symbol_factor関数はライブラリ関数です。

function_name	フレームID
appeare_cnt	378
get_bind_eq	408
SQPE	736
setup	144
factor	1609
fact	2287
symbol_factor	2316
symbol_factor_main	1

この表の情報をもとに必要な関数に素早くジャンプすることができます。

スクリプト機能のツールバー



● 表計算の再実行

行計算、列計算、行集計、列集計等が対象です。
表を含めて必要な計算式等を選択して「実行」－「再実行」で可能になりました。

表を選択してここをチェックします。

a	b	$\sqrt{a^2+b^2}$	$\sin^2 a+\cos^2 b$
2.1	0.3	2.1213	1.65780
4.7	4.7	6.6468	1.00000
5.8	9.1	10.7912	1.11403
1.2	12.3	12.3584	1.79941

a,bの値を変更後、
表を選択して再実行

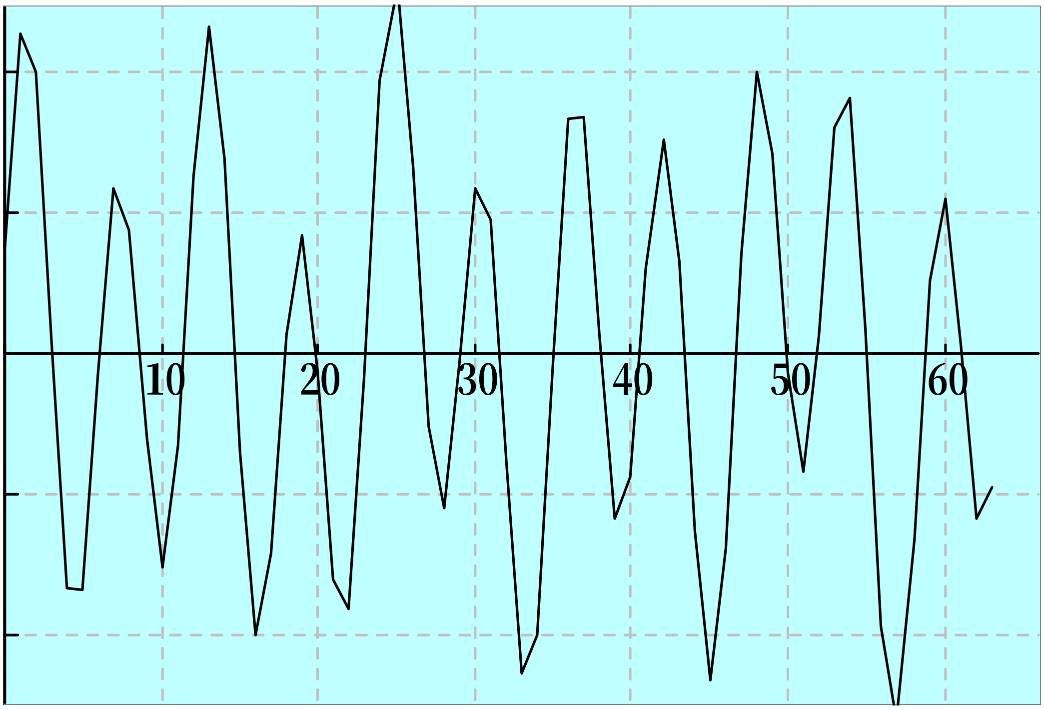
a	b	$\sqrt{a^2+b^2}$	$\sin^2 a+\cos^2 b$
8.1	2.3	8.4202	1.38461
7.7	2.7	8.1597	1.79382
6.8	7.1	9.8311	0.71275
3.2	16.3	16.6111	0.69199

● 1次元高速フーリエ変換(FFT)

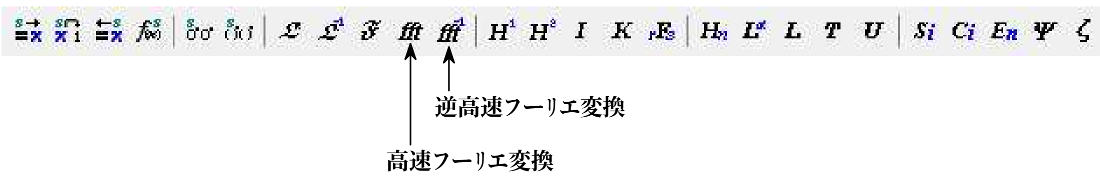
2の巾乗個の波形実測データをもとに1次元周波数分析を行います

f={3, 9.10949732073801, 8.04256821819622, 0.380941892044495, -6.69132749216299, -6.7349198
0695517, -0.595938306693562, 4.70968260727719, 3.53553390593273, -2.43145003571863, -6.085
1356747581, -2.63559027698414, 5.0677391917244, 9.31745398255832, 5.5519308557423, -2.8425
2728303621, -8, -5.67090770399218, 0.563113181927043, 3.34634562252514, -0.475538003343329
, -6.44194998196602, -7.25567760685487, -0.689741972191845, 7.77817459305203, 10.451324621
6704, 5.28877337572582, -2.09685708677876, -4.39522689797245, -0.207160949932867, 4.709146
82007863, 3.81796973464787, -3.00000000000002, -9.10949732073802, -8.04256821819621, -0.380
941892044467, 6.691327492163, 6.73491980695516, 0.595938306693545, -4.70968260727719, -3.5
3553390593273, 2.43145003571864, 6.0851356747581, 2.63559027698413, -5.06773919172441, -9.
31745398255832, -5.55193085574229, 2.84252728303621, 8, 5.67090770399218, -0.5631131819270
4, -3.34634562252514, 0.475538003343325, 6.44194998196602, 7.25567760685486, 0.68974197219
1842, -7.77817459305203, -10.4513246216704, -5.28877337572582, 2.09685708677876, 4.3952268
9797243, 0.207160949932821, -4.70914682007864, -3.81796973464786}

このデータをカルキングのデータグラフで表示すると次のようになります。



この64個のデータを高速フーリエ変換します。
計算誤差で見づらくなるので小数点下6桁までの表示としました。
下記の fft がシステム関数でツールバーから入力します。



x=fft(f)　　　　代入定義

次にxの値を表示してみましょう。

x={0, 0, 0, 0.500000*i*, 0, 1.500000, 0, 0, 0, 0, 0, 3.500000*i*, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, - 3.500000*i*, 0, 0, 0, 0, 0, 1.500000, 0, - 0.500000*i*, 0, 0}

このxに対して逆高速フーリエ変換を行います。
ここでも見やすいように下6桁表示しています。これは確かに元のデータです。

$fft^{-1}(x)$={3.000000, 9.109496, 8.042568, 0.380942, -6.691327, -6.734920, -0.595938, 4.709682, 3.535534, -2.431450, -6.085136, -2.635590, 5.067739, 9.317454, 5.551930, -2.842527, -8.000000, -5.670907, 0.563113, 3.346345, -0.475538, -6.441950, -7.255677, -0.689742, 7.778174, 10.451324, 5.288773, -2.096857, -4.395227, -0.207161, 4.709146, 3.817969, -3.000000, -9.109496, -8.042568, -0.380942, 6.691327, 6.734920, 0.595938, -4.709682, -3.535534, 2.431450, 6.085136, 2.635590, -5.067739, -9.317454, -5.551930, 2.842527, 8.000000, 5.670907, -0.563113, -3.346345, 0.475538, 6.441950, 7.255677, 0.689742, -7.778174, -10.451324, -5.288773, 2.096857, 4.395227, 0.207161, -4.709146, -3.817969}

● ラプラス変換、ラプラス逆変換

連立線形常微分方程式の非数値解を求めるための各種道具立てを実現しました。

(1)ラプラス変換（新規）

(2)線形連立方程式の記号解

(3)部分分数分解機能（新規）

(4)ラプラス逆変換（新規）

(5)仮想関数定義機能（新規）

以上はいずれも数学的表現に基づいた入力、表示、解法を実現しました。
以下に簡単な例で解法を示します。

$y'(t)+2y(t)=e^{-t}$　　$y(0)=3$

yを仮想関数として定義します。

$y(t)=\varnothing$　　未知数の関数を仮想関数定義します。

微分方程式をラプラス変換します。

$$\mathcal{L}\{y'(t)\}+\mathcal{L}\{2y(t)\}=\mathcal{L}\{e^{-t}\} \quad (1)$$

$\mathcal{L}\{y'(t)\}=\frac{d}{dt}\mathcal{L}\{y(t)\}$　　ラプラス変換と微分の可換性を使います。

$\mathcal{L}\{2y(t)\}=2\mathcal{L}\{y(t)\}$　　定数はくくりだせます。

$\mathcal{L}\{e^{-t}\}=\frac{1}{s+1}$　　ラプラス変換を実行します。（代数計算を実行します）

以上により方程式(1)は以下のようになります。

$$\frac{d}{dt}\mathcal{L}\{y(t)\}+2\mathcal{L}\{y(t)\}=\frac{1}{s+1} \quad (2)$$

ラプラス変換の微分機能により

$\frac{d}{dt}\mathcal{L}\{y(t)\}=s\mathcal{L}\{y(t)\}-y(0)$　　ラプラス変換の微分（代数計算を実行）

また y(0)=3　より、方程式(2)は次のようになります。

$$s\mathcal{L}\{y(t)\}-3+2\mathcal{L}\{y(t)\}=\frac{1}{s+1}$$

ここでカルキングの代数方程式の記号解の機能を利用します。
未知数は $\mathcal{L}\{y(t)\}$ なので、
カルキングの置換機能を使ってXに置き換えます。
方程式は

$$sX-3+2X=\frac{1}{s+1}$$

置換表

$\mathcal{L}\{y(t)\}$	X

この方程式をXに関して記号解を指定して解きます。（「実行」－「方程式」－「一元多項式」）

$$X = \frac{3s+4}{s^2+3s+2}$$ 方程式の記号解

この式の右辺に対して「部分分数分解」を行います。

$$\text{partial\_fract\_decompose}\left(\frac{3s+4}{s^2+3s+2}\right)=\frac{2}{s+2}+\frac{1}{s+1}$$

従って $$\mathcal{L}\{y(t)\}=\frac{2}{s+2}+\frac{1}{s+1}$$

この式に対して逆ラプラス変換を行います。

$$\mathcal{L}^{-1}\{\mathcal{L}(y(t))\}=\mathcal{L}^{-1}\left\{\frac{2}{s+2}+\frac{1}{s+1}\right\}$$

従って $$y(t)=\mathcal{L}^{-1}\left\{\frac{2}{s+2}+\frac{1}{s+1}\right\}$$

$$\mathcal{L}^{-1}\left\{\frac{2}{s+2}+\frac{1}{s+1}\right\}=e^{-t}+2e^{-2t}$$ 逆ラプラス変換の実行（代数計算）

得られた最終解 $y(t)=e^{-t}+2e^{-2t}$

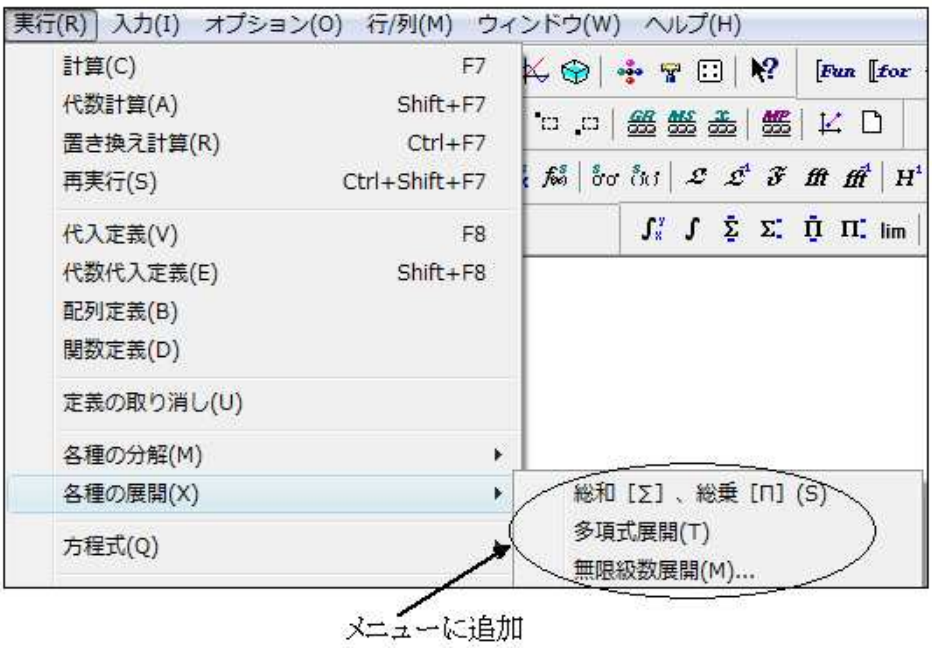
● 部分分数分解

ラプラス変換で微分方程式を解くとき等に利用します。

$$\text{partial\_fract\_decompose}\left(\frac{1}{s^2+3s-20}\right)=\frac{-0.105999788000636}{s+6.2169905660283}+\frac{0.105999788000636}{s-3.2169905660283}$$

● 関数の展開

実行メニューに追加されました。



総和∑、総乗Πの級数展開

$$\sum_{k=1}^{10}\frac{1}{k}x^k=\frac{1}{1}x+\frac{1}{2}x^2+\frac{1}{3}x^3+\frac{1}{4}x^4+\frac{1}{5}x^5+\frac{1}{6}x^6+\frac{1}{7}x^7+\frac{1}{8}x^8+\frac{1}{9}x^9+\frac{1}{10}x^{10}$$

$$\prod_{k=1}^{10}\frac{2k+1}{k^2}=\frac{1+2}{1^2},\frac{1+4}{2^2},\frac{1+6}{3^2},\frac{1+8}{4^2},\frac{1+10}{5^2},\frac{1+12}{6^2},\frac{1+14}{7^2},\frac{1+16}{8^2},\frac{1+18}{9^2},\frac{1+20}{10^2}$$

多項式展開（多項式関数群）

$\text{polynomial\_expand}(H_4(x))=16x^4-48x^2+12$ **エルミート関数の展開**

$\text{polynomial\_expand}(L_5(x))=-\frac{1}{120}x^5+\frac{5}{24}x^4-\frac{5}{3}x^3+5x^2-5x+1$ **ラゲール関数の展開**

無限級数展開（マクローリン展開等）

展開次数はダイアログで指定します。

$$\text{taylor\_expand}(\sin(3x),13)=3x-\frac{9}{2}x^3+\frac{81}{40}x^5-\frac{243}{560}x^7+\frac{243}{4480}x^9-\frac{2187}{492800}x^{11}+\frac{6561}{25625600}x^{13}$$

$$\text{taylor\_expand}(3\sin x-4\sin^3 x,13)=3x-\frac{9}{2}x^3+\frac{81}{40}x^5-\frac{243}{560}x^7+\frac{243}{4480}x^9-\frac{2187}{492800}x^{11}+\frac{6561}{25625600}x^{13}$$

コメント：この級数展開により二つの式が等価らしいことが分かります。

● 配列の先頭のインデックス値

これまで配列の先頭要素の添字番号は 1 に固定されていました。

例 $a_{1..100}=0$ **配列定義**

主な理由は、行列、ベクトル、表等の先頭要素の添字番号は通例 1 であり、これに合わせていました。
しかし、数学の級数等では以下の例のように、 0 番目からの記法が圧倒的に多いのも事実です。
下記の級数を k=1 にした公式に変更すると見づらくなります。
実際に、多くの公式を従来の方法で作成しましたが、かなり似通っているために混乱を起こしやすく、
数学の表記通りの式で計算ができるというカルキングの基本方針に反します。
そこで、

$b_{0..100}=0$ **のように0から始まる配列も認めることとしました。**

Bはベルヌーイの定数です。

{B,k}=search_name("Bernoulli_const")

n=2

$$\frac{(2\pi)^{2n+1}}{2n}\sum_{k=0}^{n/2}(-1)^k(2n+2-4k)\frac{B_{2k}}{(2k)!}\frac{B_{2n+2-2k}}{(2n+2-2k)!}-2\sum_{k=1}^{100}\frac{e^{2\pi k}\left(1+\frac{4\pi k}{2n}\right)-1}{k^{2n+1}\left(e^{2\pi k}-1\right)^2}=1.03692775514337$$

このように簡潔に数学の公式をそのまま使って計算ができるようになりました。

● 多重代入文

効果

(1)簡潔な表現

{x,y,z}={0,0,0}　　変数x,y,zにそれぞれ0を代入します。
3つの代入文を1行にまとめられます。

{x,y}={y,x}　　変数 x,y の値を交換します。同時代入機能

$\{a_i, a_j\}=\{a_j, a_i\}$　　配列要素a_i , a_jの値を交換します。同時代入機能

同時代入機能のために、わざわざ作業変数を導入する必要はありません。

(2)関数の戻り値に複数の異なるタイプの値を設定できます。
これが多重代入機能のもっとも適用意義のあるケースです。

{x,y,a,s}=fun(t,w,v)　　ここでfunは三つの変数を引数とする関数で、戻り値として、二つの実変数値、一つの配列、一つの文字列を返します。
簡潔な表現に加えて、文法的な美しさを実現しています。
つまり、入力パラメータが複数であれば、出力も複数にできるという対称性を実現しています。

● 仮想関数定義機能（内容は定義されていない）

f(x)=∅　　関数定義

g(x)=∅　　関数定義

$$\frac{d}{dx}(f(x)g(x))=f(x)\left(\frac{d}{dx}(g(x))\right)+\frac{d}{dx}(f(x))(g(x))$$

微分した時に0にならないので、Laplace変換等で、微分方程式の関数を求める時に、
未知の関数の処理のために使います。

● 空または未定義等のオブジェクトφ

a=∅　　代入定義

b={∅,∅,∅,∅,∅}　　代入定義

スクリプトプログラムを簡潔で理解しやすい形に書けます。
数学においてゼロの発見は大きな進歩をもたらしましたが、このφはカルキングスクリプトに大きな進歩をもたらしました。 このφの利用によって、プログラムが簡潔かつ理解しやすくなりました。

● 配列、文字列に対して＆を結合子として使用可能

"ab"　&　"cde"="abcde"

{1,2,3}　&　{4,5,6,7}={1, 2, 3, 4, 5, 6, 7}

多くのプログラミング言語には配列の連結機能はありません。
この機能はカルキングにおいて特に重要です。
この機能が無ければプログラム開発は大きな障害に直面していたと思われます。

● 配列同士の比較演算子

配列要素の比較はもちろん、複数の異なるタイプのデータを含んだ二つの配列を、
一つの＝記号で比較できます。

$a=\left\{2,3,\left\{"\sqrt{x^2+b^2}",110,200\right\},\{10,20\}\right\}$　　代入定義

$b=\left\{2,3,\left\{"\sqrt{x^2+b^2}",110,200\right\},\{10,20\}\right\}$　　代入定義

```
( for k = 1 to 100 step 1 )
break   a=b        ← この条件比較でTRUEとなる
…
```

このような簡潔なプログラム例はいままでのプログラミング言語ではほとんど実現不可能でしょう。

● 数学記号ツールバーへの記号の追加

円周率

オイラー定数

円周率（π）の扱いの柔軟性を合わせて実現しました。二種類のπを使い分けられます。
プロパティの精度に無関係にある特定の値を使いたいことがあります。
教育現場、建築等の業務計算では 3.14 等のように固定したいときがあります。
このときは、basic.lbrでライブラリ登録されたπを利用します。（ギリシア文字パレットから入力します。）
プロパティの指定精度に基づいて任意精度のπを使いたいときは数学記号文字盤の
システム定数のπを使います。このとき最大1000桁です。オイラー定数も最大１０００桁です。

● 組み込み関数object_type関数の機能拡張

従来、スカラー変数、行列、表を判定するための関数でしたが拡張しました。

object_type(123)="number"	数値
object_type("abc")="string"	文字列
object_type(sin)="system_function"	システム組み込み関数
object_type(calking_gamma)="user_function"	ライブラリ定義関数
object_type(s)="others"	s=∅　の時

● ライブラリ関数および定数の大幅強化

（システム関数とは別の、標準ライブラリ登録されたもの）

文字列基本関数

システム機能

数値計算基本関数

代数計算基本関数

エラー処理基本関数

π、e(自然対数), γ（オイラー定数)の1000桁　「数学記号」文字版から提供

ベルヌーイ定数400個（500桁）

リーマンゼータ関数の400個(400桁)

● decompose_string関数の強化

数式を分析、加工する上での不可欠のシステム関数で今回サポートされた新関数群を網羅しています。

例　decompose_string("$J_2(5)$")={53, "2", "5"}

decompose_string("$Y_3(5)$")={54, "3", "5"}

decompose_string("$H_2^{(1)}(5)$")={55, "2", "5"}

decompose_string("$H_3^{(2)}(6)$")={56, "3", "6"}

decompose_string("$I_2(6)$")={57, "2", "6"}

decompose_string("$K_1(3)$")={58, "1", "3"}

decompose_string("$H_4(7)$")={76, "4", "7"}

● エラー対策

エラーメッセージ機能の強化（式番号、スクリプト関数の表示）

エラー発生時の関数履歴表示機能やフレーム番号へのジャンプ機能と合わせて
強力なデバッグ機能となります。プロフェッショナル版ではスクリプトのデバッグ支援機能を重視しました。

● スクリプト機能補足説明

カルキングスクリプト機能の改善、強化に努めてまいりました。
文法仕様の簡素化を保持したままで、仕様の一貫性、美しさを重視してきました。
特に重要な点は記号処理手続きをスクリプト言語で記述可能にしたことです。
一般的にプログラミング言語は数値計算処理の手続きを記述するものです。
記号処理手続きは数値計算処理手続きに比べてかなり複雑です。
今回、記号処理手続き用の関数の実現により、「ユーザ」の方がスクリプト言語で
いろいろな記号処理機能を実現できることが可能になります。

● スクリプトの例

ここでは記号処理計算例を紹介したいのですが、プログラム量が多いため、
代わりに数値計算のスクリプトを紹介します。

以下では実際にSi(z)の実装で使われているSIN積分を取り上げます。

$$Si(z)=\int_0^z \frac{\sin t}{t}dt$$　　定義

$$Si(z)=\sum_{n=0}^{\infty}\frac{(-1)^n z^{2n+1}}{(2n+1)(2n+1)!}$$　　級数での公式

これをカルキングのスクリプトで記述した時の例を以下に示します。
この関数の許されているパラメータは、実数、複素数、及びこれらを要素とする配列です。
しかも計算精度は1000桁まで可能です。
これほど汎用性に富んだ仕様ですが、驚くほど簡単にプログラム記述ができます。

関数名をcalking_Siとします。

```
calking_Si( x )
var v,d,q,u
d=x
q=x^2
u=d
( for k = 2 to 10000000 step 2 )
 v=u
 d=-dq/k(k+1)
 u=v+d/k+1
 break  v=u
return v
```

級数での公式からでは、このスクリプトは同じものを計算しているとは思えませんが、
実際に計算するときは、各ステップの段階で計算されたものを活用して、計算の重複を回避し、
高速計算を可能としています。

計算例

calking_Si(3.6021)=1.82168950512375

50桁精度

calking_Si(3.6021)=1.82168950512374697831482461028163171188447585283154914124977831085722478991108093867357214968942804546448909738614554544874476659422713533243029619336549334909739444697619537128792846230205848077323 59

200桁精度

calking_Si(3.6021)=1.82168950512374697831482461028163171188447585283154914124977831085722478991108093867357214968942804546448909738614554544874476659422713533243029619336549334909739444697619537128792846230205848077323 59

複素数パラメータ（精度100桁）

calking_Si(3.6021+0.792*i*)=1.889715177242137395127357758628981086971171583625158154753126407381310019036957310538281301338048671 - 0.118245793767528785427029520527735175771281085106624515416239596803881727128604478252056876945551937 6*i*

配列パラメータ

calking_Si({1.2, 4.5, 7.2, 8.1, 0.34, 10.4})={1.1080471990137185899, 1.6541404143792439837, 1.475089055447246115, 1.5863666224636431063, 0.33782400214611531729, 1.6311171372314991051}

計算応用編

$$\frac{\sqrt{\text{calking\_Si}(\{1.2,\ 4.5,\ 7.2,\ 8.1,\ 0.34,\ 10.4\})}+3.6}{2}$$
={2.32631910449216, 2.44306695109826, 2.40726622157156, 2.42975523468719, 2.09061314584259, 2.43857598162435}