atzjg.net

Sowya 教程

徐海峰*

October 28, 2025

An Awesome Publisher

* Sowya Outputs Wisdom, YA!

atzjg.net

Sowya
本教程适用于 Sowya v0.658 及以上.

出版社
首印于202?年?月 An Awesome Publisher

Sowya (数学)

书山有路勤为径，学海无涯苦作舟。

—- 韩愈

目录

Sowya简介

Sowya(原名 Calculator) 是一款运行于 Windows 系统上的计算程序. 支持大数运算(四则运算、模运算、阶乘等)、分数运算、进制转换、变量定义、不定方程求解等. 矩阵运算也在加入中.

Sowya 的发音即是吴语中“数学”的发音, 可以在 cn.bing.com/translator 中聆听.

程序文件为 sowya.exe 和 SowyaApp.exe, 前者是控制台应用软件, 后者是其 GUI 版本. 这两个文件均可在 atzjg.net 首页上下载.

若非特别指明, 以下都使用 Sowya 指代 sowya.exe 和 SowyaApp.exe.

1 目的

Sowya（数学）是一款运行于 Windows 和 Linux 上的计算软件.

开发这款软件最初是希望能做一个可以达到任意精度的计算软件, 后来为教学方便, 逐步加入矩阵、解方程等功能.

我们的目标是开发一款兼有数值计算和符号计算功能的数学软件.

1.1 历史

此项目始于2018年11月24日. 可以在命令窗口中输入 :dev_history 并回车, 列出开发历史.

Calculator 的名字一直延续到版本 v0.549. 从 v0.550 开始, 更名为 Sowya. Sowya 是吴语中数学的发音.

Sowya 的版本采用最原始的管理方式. 最初是从 v0.1开始标记, 经过几个版本后到了 v0.4, 然后增加了一位, 版本号形如 v0.4x, 直到 v0.44 我认为小的改动应该记录在 0.00x 中. 与此同时, 从 v0.441 开始正式记录开发历史. 每次一个小功能的添加或者一个 BUG 的消除都将版本号增加 0.001, 以十进制的方式递增.

下面是 Sowya 开发的简要历史.

- v0.542 是第一阶段的最后一个版本.
- 从 v0.543 开始, Calculator 支持最简单的批处理任务. 即读取文件, 执行其中的每一行语句, 并输出.
- 从 v0.550 开始, 将 Calculator 更名为 Sowya.
- 从 v0.555 开始, Sowya 引入 clox (Crafting Interpreters) , 可以进行简单的编程.
- 从 v0.580 开始, Sowya 加入插件功能.
- 自 v0.615 开始, 提供 Linux 版本. GUI程序目前停留在v0.615.
- 自 v0.618 开始, 提供 64 位程序.
- v0.646 是一个重要的版本. 此版本将程序进行了一定程度的优化(比如将 BigNumber 类中的一些设计多项式运算的函数放到了 polyn.h 和 polynMulti.h 中. 将原来sin2x.h中的内容放到了BigNumber类中. 等等). 这样去掉了BigNumber类在一些文件中的前置声明, 使得在 Linux 下编译更顺利.

详细的开发历史见附录9.2.2.

1.2 下载

下载网址: `http://atzjg.net`

首页有下面两个文件的下载链接.

- sowya.exe (控制台应用)
- SowyaApp.7z (GUI版本)

注: 请用 7zip 软件解压缩 `SowyaApp.7z`, 解压缩后有两个文件: `SowyaApp.exe` 和 `docklayout.dat`. 其中 `SowyaApp.exe` 是可执行文件, 双击即可运行; `docklayout.dat` 是 `SowyaApp.exe` 运行后界面布局设定文件, 将保存 `SowyaApp.exe` 关闭前的样式, 使得下次运行后呈现和关闭前一样的界面.

1.2.1 Windows

Sowya 最新版下载地址如下. (文件 `Sowya.7z` 中包含了 `Sowya.exe`, `Sowya32.exe` 以及GUI程序 `SowyaApp.exe`)

下载地址: http://atzjg.net/content/calculator/download/Sowya.7z

这里 `Sowya.exe` 是64位程序, `Sowya32.exe` 和 `SowyaApp.exe` 是 32 位程序, 运行于 Windows 操作系统.

1.2.2 Linux

- sowya 【64位】 http://atzjg.net/content/calculator/download/sowya
- sowya32 【32位】 http://atzjg.net/content/calculator/download/sowya32

在 Linux 中, 可以用 `readelf -h sowya32` 查看程序是 32 位还是 64 位. 这里 sowya32 是 32位 Linux 程序.

openSUSE

以 Windows 上安装的 openSUSE Tumbleweed (WSL2)为例, 系统是64位. 为运行32位程序, 必须安装一些32位的库.

执行下面的命令

```
> sudo zypper install -t pattern 32bit
> sudo zypper install glibc-devel-32bit
> sudo zypper in libstdc++6-32bit
```

将 sowya32 设置为可运行.

```
> chmod +x sowya32
> ./sowya32
```

Ubuntu

依次执行下面的命令, 详见在 64 位系统中运行 32 位程序

```
$ sudo dpkg --add-architecture i386
$ sudo apt-get update
$ sudo apt-get install libc6:i386 libstdc++6:i386

$ chmod +x sowya32
$ ./sowya32
```

1.3 适配编辑器

1.3.1 SciTE

自 v0.578 开始, Sowya.exe 开始适配 SciTE 编辑软件. 使在 SciTE 中可以编辑并运行 `*.sy` 和 `*.sc` 文件.

其中`*.sy` 是 sowya 文件, 可以被 sowya.exe 逐行执行, 形如

```
a=2
a+3*8
:mode=polyn
(-4x^3+x^2)+(3x-x^6)*2x
:mode=numerical
A=[1,2;
3,-2]
B=[1 2 3
4;3,2,0
1;0,2,0,9;1,3,5,
-1]
EulerVarphi(20221003)
hex2decimal(0xFF)
```

而 `*.sc` 是 sowya 的 `clox` 编程文件, 通常在sowya启动后需要使用命令 `:mode clox` 切换到编程模式, 然后使用 `load(filename.sc)` 加载文件的内容. 现在可以直接按 `F5` 运行并在新的输出页面中显示结果. `*.sc` 文件的内容形如:

```
setmode(1);
fun f(x)
{
    print "f(",x,") --> ";
    if(x<0)
    {
    return -1*x;
    }else
    {
    return f(x-f(x-1))/2;
    }
}
println f(2);
```

我们在 SciTE 中模仿其他语言的配置文件加入了关于sowya的配置文件 `sowya.properties`. SciTE会自动识别 `*.sy` 或 `*.sc` 文件, 按 `F5` 键后由 sowya.exe 读取并运行源文件中的内容, 并在新窗口中显示运算结果.

关于文件 `sowya.properties`, 见附录.2.

1.3.2 Vim

将下述代码添加到 `.vimrc` 中, 重启 vim. 例如: `vim file.sc` 编辑源文件 `file.sc`, 然后按 `F5` 键即可运行并显示结果.

```
function! ExecBySowya()
  let filetype_name = strpart(expand("%"), stridx(expand("%"), "."))
  execute "w"
  let n=expand('%:t')
  if filetype_name == ".sc"
    execute "silent !sowya.exe --clox %:p > ".n.".out"
  else
    if filetype_name == ".sy"
      execute "silent !sowya.exe -f %:p -o ".n.".out"
    endif
  endif
  execute "vsplit ".n.".out"
  execute "redraw!"
  set autoread
endfunction

:nmap <F5> :call ExecBySowya()<CR>
```

从代码看出, 若源文件后缀名是`.sy`, 则运行 `sowya.exe -f <文件名>.sy -o <输出文件名>`; 若后缀名是 `.sc`, 则运行 `sowya.exe --clox <文件名>.sc` 并重定向到输出文件中.

1.3.3 Visual Studio Code

依次点击 `File>Preferences>Settings` (或键盘输入 `Ctrl+,`) 打开设置页面, 在 `Search settings` 中搜索 `Code-runner: Executor Map By File Extension`. 点击 `Edit in settings.json`, 加入下面这两行

```
".sc": "cd $dir && sowya --clox $fileName",
".sy": "cd $dir && sowya -f $fileName -o $fileNameWithoutExt.out"
```

例: 使用vscode编辑文件 `fibo.sc` 并保存. 然后点击右边的运行按钮(三角形).

```
fun fib(n)
{
  if(n<2) return n;
  return fib(n-1)+fib(n-2);
}

print fib(20);
```

基本功能 2

2.1 命令

Sowya 中的命令均以冒号(:)开头, 例如 :version, 用于显示软件的当前版本号.

指令	功能
:about	Sowya 简介
:clc	SowyaApp 清除Result和Trace窗口中的内容
:dev_history	显示开发历史
:exit	退出程序, sowya.exe 中有用.
:help	显示基本的帮助信息
:h	同 :help
:more	显示更多功能
:quit	同 :exit
:q	同 :quit
:sowya	输出 Sowya 的 ASCII 字符图案
:thanks	显示“欢迎使用Sowya”
:version	显示软件当前的版本号
:v	同 :version

注: `:help funcName`或`:h funcName`将显示函数`funcName`的基本信息, 功能等同于`help(funcName)`. 这里`funcName`是指函数名.

2.2 基础运算

2.2.1 基本的四则运算

在命令窗口(Command)中输入 (5 + 2) * 3 / 7 然后回车, 将在输出窗口(Result)中得到结果.

```
in> (5+2)*3/7

out> 3
```

2.2.2 阶乘

例, 计算 20 的阶乘.

```
in> 20!
out> 2432902008176640000
```

也可以使用内置函数 `factorial(20)` 或 `Factorial()`.

2.2.3 分数运算

Sowya 提供了分数运算的功能. 在进行分数计算前, 请先设置计算模式为分数计算模式. 关于计算模式, 见“四则运算”一节.

首先, 使用命令 `:mode=fraction` 将模式切换到分数计算模式.

```
:mode=fraction

Switch into fraction calculating mode.
e.g., 1/2+1/3 will return 5/6

>>
```

这里给了一个例子, 输入 `1/2+1/3` 将返回 `5/6`. 如果在 `numerical` 模式下, 则返回 `0.83333333`.

我们再试一下其他例子, 输入 `1/2+1/3+1/4+1/5+1/6`

```
in> 1/2+1/3+1/4+1/5+1/6

out> 29|20
```

（这里使用 | 作为分数线的记号.）

如果要计算 `1+1/2+1/3+1/4+1/5+...+1/100`, 那么最好使用求和函数 `sum( , , , )`. 这个函数需要输入四个参数, 输入 `help(sum)` 会提示用法如下:

```
Usage:
sum(general_term,x,minValue,maxValue)
```

第一个参数指通项表达式, 第二个参数指自变量采用的符号, 第三和第四个参数指自变量的取值范围. 注意这里 `x, minValue, maxValue` 都是整数, 且 `minValue<=x<=maxValue`.

因此, 要计算上面的求和, 我们只需输入

```
in> sum(1/n,n,1,100)

out> 5.18737752
```

如果在分数计算模式下, 则结果是

```
14466636279520351160221518043104131447711|2788815009188499086581352357412492142272
```

2.2.4 十进制小数转为分数

这里提供了两个函数 `decimal2fraction()` 和 `decimalToFraction()`. 前者将输出整数部分与真分数部分相加的形式, 同时也输出假分数形式; 而后者只输出假分数的形式.

参见 8.4.3 和 8.4.6.

2.2.5 模运算

模运算使用 mod 或@ 运算符. 例如计算

$$(2^{2022} + 10 * 9^2 - 365) \pmod{10007}$$

输入

```
>> (2^2022+10*9^2-365)mod10007

out> 6301
```

快速模幂运算

使用 `expmod()` 函数可以快速执行模幂运算.

```
expmod(base,power,N)
```

例如计算 $3^{123456789} \mod 10^{20}$, 我们不能输入

```
>> 3^123456789mod100000000000000000000
```

因为这里将先计算 $3^{123456789}$, 而这是一个非常大的数. 而应使用 `expmod` 函数.

```
expmod(3,123456789,100000000000000000000)
```

得到结果

```
>> expmod(3,123456789,100000000000000000000)
in> expmod(3,123456789,100000000000000000000)
calculate: 3^123456789(mod 100000000000000000000)
out> 43806986775225122483

------------------------
```

2.2.6 运算符的优先级

主要的运算符有 `+, -, *, /, %, ^, !, ==, ~`. 其中 == 是比较运算符, 与之等价的是~.系统会自动替换为 ~.

~ 的优先级最低, 阶乘运算符 ! 的优先级最高. 但当碰到小括号(), 则小括号内的先算.

^ 的优先级次之, 即只比 ! 低. 例如 `2^3!` 将返回 `64`, 而不是 `40320`, 即 `(2^3)!`.

`*, /, %` 分别是乘法、除法和整除运算符, 它们的优先级相同. `a%b` 所得是不完全商, 仅当 $b|a$ 时才是完全商. 例如: `7%3` 返回 `2`.

`+, -` 的优先级仅比 ~ 高.

2.2.7 关于表达式中的空格

输入的表达式中允许有空格, Sowya 在计算之前将这些空格自动去掉.

```
in> 1.2*3+(4.5-6)/7

out> 3.38571429
```

这里看到数值计算的精度默认是八位, 最后一位四舍五入而得. 设置精度的函数是 `setprecision()`.

2.2.8 设置精度

例如, 我们将精度设定为十位, 即小数点后面保留十位有效数字.

```
in> setprecision(10)
Now the precision is: 10
```

2.2.9 计算模式

Caculator 启动后默认的计算模式是数值计算模式, 使用 `:mode` 命令可以查看当前计算模式.

```
:mode
Calculating mode: numerical

>>
```

最后的 `>>` 是控制台应用中的输入提示符. 在 GUI 版本中, 默认命令或表达式在 Command 窗口中输入, 因此可以忽略这个提示符.

更改计算模式

目前计算模式有数值计算模式、分数计算模式和多项式计算模式三种. 当输入 `:mode=xxx` （其实这里的 `xxx` 可以是任意字符, 比如输错了的情况）并回车, 系统会给出正确的提示.

```
Syntax:
        :mode=numerical
        :mode=fraction
        :mode=polyn
or
        :mode
                This will print the current calculating mode.

>>
```

也就是, 命令 `:mode=fraction` 将切换到分数计算模式, 而 `:mode=numerical` 将切换到数值计算模式.

要进行分数计算, 请先切换到分数计算模式.

2.2.10 进制转换

Sowya 提供了十进制与二进制、八进制、十六进制等之间的相互转换.

十进制转二进制

`decimal2binary()` 参见8.4.2.

二进制转十进制

`binary2decimal()` 将二进制数转换为十进制数. 见8.2.2.

任意进制转十进制

`any2decimal( , )`

功能: 任意p进制数转成十进制形式.

参数: 双参数. 第一个参数是p进制数的表达式, 第二个参数是 p.

例

```
1 in> any2decimal(12ABGH9JK8,36)
2 out> 108010759187528
```

十进制转任意进制

`decimal2any( , )` 参见8.4.1.

十进制转八进制

`decimal2octal()` 参见 8.4.5.

八进制转十进制

`octal2decimal()`

功能: 八进制数转成十进制形式

注意: 输入的八进制数以0开头.

例

```
in> octal2decimal(01234567)
out> 342391
```

十进制转十六进制

`decimal2hex()` 参见 8.4.4.

十六进制转十进制

`hex2decimal()`

功能: 十六进制数转成十进制形式

注意: 输入的十六进制数以0x开头.

例

```
in> hex2decimal(0xFF)
out> 255
```

2.3 内置常数

2.3.1 常数 π

常数 π 在计算中经常会碰到, 故 Sowya 内置了 π 的近似值, 精确到小数点后10000位.

直接键入 `pi()` 将得到 3.14159265. 这是因为默认精度是 8 位. 使用 `setprecision()` 函数将精度设置为 10000位, 然后再键入 `pi()`, 就可以获得精确到小数点后10000位的的近似值.

```
>> pi()
in> pi()
out> 3.14159265

------------------------

>> setprecision(10000)
in> setprecision(10000)
Now the precision is: 10000

```

```
----------------------
>> pi()
in> pi()
out> 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482......56375678
```

2.3.2 常数 e

常数 e 可以用函数 `exp(1)` 获得.

```
>> exp(1)
in> exp(1)
out> 2.71828182
```

e 的计算使用了公式

$$e^x = 1 + \frac{1}{1!}x^1 + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots + \frac{1}{n!}x^n + \cdots$$

因此, 也可以使用 `sum()` 函数计算 e.

```
>> setprecision(100)

>> sum(1/n!,n,0,100)
in> sum(1/n!,n,0,100)

out> 2.7182818284590452353602874713526624977572470936999595749669676277240766303535475945713821785251664274

----------------------
```

如果需要将此和式进行展开, 则在 `sum` 函数中添加参数 `expand`, 如下.

```
>> sum(1/n!,n,0,100,expand)
in> sum(1/n!,n,0,100)
1/0!+1/1!+1/2!+1/3!+1/4!+1/5!+1/6!+1/7!+1/8!+1/9!+1/10!+1/11!+1/12!+1/13!+1/14!+1/15!+1/16!+...+1/98!+1/99!+1/100!
out> 2.7182818284590452353602874713526624977572470936999595749669676277240766303535475945713821785251664274

----------------------
```

2.3.3 常数 $\ln 2$

$\ln 2$ 可以表示为很多个级数形式, ([1],P.342)

$$\ln 2 = \frac{2}{3}\sum_{k=0}^{\infty}\frac{1}{(2k+1)9^k}.$$

内置的 `__ln2__` 有 1000 位.

```
0.69314718055994530941723212145817656807550013436025525412068000949339362196969471560586332699641868754200148102057068573368552023575813055703267075163507596193072757082837143519030703862389167347112335011536449795523912047517261857493206515552473413952588295045300709532636664265410423915781495204374043038550080194417064167151864471283996817178454695702627163106454615025720740248163777338963855069526066834113727387372292895649354702576265209885969320196505855476470330679365443254763274495125040606943814710468994650622016772042452452961268794654619316517468139267250410380254625965686914419287160829380317271436778265487756648508567407764845146443994046142260319309673540257444607030809608504748663852313818167675143866747664789088143714198549423151997354880375165861275352916610007105355824987941472950929311389713
```

```
559982056543928717000721808576102523688921324497138932037843935308877482597017155910708823683627589842589185353024363
421436706118923678919237231467232172053401649256872747782344535340
```

2.4 变量

2.4.1 定义变量

变量的定义很简单, 输入 `a=1` 即定义了变量 `a`, 其值为 1. 可以键入 `disp(a)` 查看其值.

```
>> a=1
1
--------------------
>> disp(a)
out> 1

------------------------
```

命令 `:list` 则列出所有已定义的变量. 如果是 SowyaApp.exe, 则在 Result 窗口和 Variables 窗口显示变量的信息. `:list` 命令还会附带列出变量在内存中的地址.

```
>> :list
info> All variables are listed below.
int : a=1       mem addr: 00000282A98B6520
---*---*---*---

>> b=3
3
--------------------
>> :list
info> All variables are listed below.
int : a=1       mem addr: 00000282A98B6520
int : b=3       mem addr: 00000282A98B6820
---*---*---*---
```

除了直接使用等号定义变量, 还提供了函数`symbol()`定义单个或多个变量. 例如`symbol(a:1)`与`a=1`的功能是等价的, 都定义了变量`a`并赋值`1`.

```
>> symbol(a:1)
out> a has been defined.

------------------------

>> disp(a)
out> 1

------------------------
```

如果要定义多个变量, 则在变量:值之间使用逗号隔开.

```
>> symbol(b:2,c:3.14,d:-1)
out> b has been defined.
out> c has been defined.
out> d has been defined.
```

```
5
6  ------------------------
7
8  >> :list
9  info> All variables are listed below.
10 int : a=1       mem addr: 01811028
11 int : b=2       mem addr: 01810D88
12 int : c=3.14    mem addr: 018110E8
13 int : d=-1      mem addr: 01811448
14 ---*---*---*---
```

注 2.4.1 在v0.593版本之前使用`symbol(a,1)`定义变量a的值为1. 多个变量的定义在v0.593版本中加入, 且废弃了`symbol(变量名,值)`这种语法. 若在定义变量时不指定值, 例如 `symbol(a,b,c)`将定义`a,b,c`为多元多项式所要用到的变量, 其定义的变量与`:list`中显示的不同. 具体参见多元多项式一章.

2.4.2 变量代换

v0.617 版本加入了`:claim`命令, 用于声明某个字符串等于另一个字符串.

`:claim str1==str2` 声明`str1`完全等同于`str2`.

```
1  >> :claim a==b
2
3  >> a==b
4  in> b~b
5
6  out> 1
7  ------------------------
```

此声明a等于b.

```
1  >> :mode polyn
2  Switch into polynomial mode.
3
4  >> symbol(a,b)
5  out> a is a variable for polynomial.
6  out> b is a variable for polynomial.
7
8  ------------------------
9
10 >> (2a-b)^2
11 in> (2a-b)^2
12
13 out> 4a^2-4ab+b^2
14 ------------------------
15
16
17 >> (2*a-b)^2
18 in> (2*b-b)^2
19
20 out> b^2
21 ------------------------
```

使用:clear claims清除所有声明.

2.5 矩阵

2.5.1 矩阵输入

仿照简单变量的定义, 输入

```
A=[ 1 2; 3 4]
```

然后回车, 将得到一个2阶方阵. A 是矩阵的变量名. 如果仅输入

```
[ 1 2; 3 4]
```

则也会得到一个矩阵, 其具有内置变量名 __Matrix__ .

上面也可以如下输入:

```
A=[1,2;3,4]
```

或多行输入:

```
A=[1,2;
3,4]
```

2.5.2 基本规则

在矩阵输入中,

- 空格和逗号都可以作为元素之间的分隔符, 分号是行结束符号.
- 矩阵元素的输入以左中括号[开始, 以右中括号]结束.
- 逗号和空格可以同时作为分隔符;
- 多个连续的空格等同于单个空格;
- 相邻两个逗号中间有无空格都将自动插入0;

例如:

```
A=[ 1, , 3]
```

得到矩阵 A=[1,0,3].

```
A=[ 1,2 3; 4 5, 7
; 2 3]
```

注意第二行分号前面没有空格. 得到矩阵

```
[var] A
1 2 3
4 5 7
2 3 0
input> [1,2,3;4,5,7;2,3]
det(A)=13
```

例

```
A=[1;
2 3;
4 5 6;
7 8 9 10]
```

将得到矩阵

```
[var] A
1 0 0 0
2 3 0 0
4 5 6 0
7 8 9 10
input> [1; 2 3; 4 5 6; 7 8 9 10]
det(A)=180
```

测试输入

```
B=[1 2 3
4;3,2,0
1;0,2,0,9;1,3,5,
-1]
```

结果应该是

```
 type: matrix
 name: B
 size: 4x4
value:

1 2 3 4
3 2 0 1
0 2 0 9
1 3 5 -1
determinant: -143
```

2.5.3 行列式

`det()` 函数用于返回矩阵的行列式, 如果矩阵不是方阵, 则返回 `NaN`.

矩阵的元素可以是分数, 在默认的计算模式(numerical)下, 只返回行列式的计算表达式. 若要计算其行列式, 则切换到分数模式(fraction)下计算.

例

```
A=[1/2 2/3; 3/4 4/5]
```

回车后会显示下面的信息.

```
[var] A
1/2 2/3
3/4 4/5
input> [1/2 2/3; 3/4 4/5]
det(A)=1/2*4/5-3/4*2/3
--------------------
```

上面顺带计算了矩阵 A 的行列式. 也可以输入 det(A) 获得.

```
det(A)
```

为使计算所得的值是分数, 将计算模式切换至 fraction.

```
:mode=fraction
A=[1/2 2/3; 3/4 4/5]
```

回车后显示

```
[var] A
1/2 2/3
3/4 4/5
input> [1/2 2/3; 3/4 4/5]
det(A)=-1|10
--------------------
```

Sowya 可以处理元素中有表达式的行列式, 例如输入如下矩阵

```
A=[2^2-1, 6+5*3, 7-1/4;
1/3,-1+2/3,0;
1-3^2,-2,7+9/7]
```

如果在数值计算模式下, 则返回

```
>> A=[2^2-1, 6+5*3, 7-1/4;
A=[2^2-1, 6+5*3, 7-1/4;
1/3,-1+2/3,0;
1-3^2,-2,7+9/7]
input> [2^2-1,6+5*3,7-1/4;1/3,-1+2/3,0;1-3^2,-2,7+9/7]
det(A)=-88.78571428
----------------------------
 type: matrix
 name: A
value:
2^2-1   6+5*3   7-1/4
1/3     -1+2/3  0
1-3^2   -2      7+9/7

determinant: -88.78571428
--------------------
>>
```

切换到分数计算模式

```
:mode=fraction
```

然后重新输入上面的矩阵, 会得到行列式的精确值

```
>> A=[2^2-1, 6+5*3, 7-1/4;
A=[2^2-1, 6+5*3, 7-1/4;
1/3,-1+2/3,0;
1-3^2,-2,7+9/7]
input> [2^2-1,6+5*3,7-1/4;1/3,-1+2/3,0;1-3^2,-2,7+9/7]
det(A)=-1243|14
----------------------------
 type: matrix
 name: A
value:
2^2-1   6+5*3   7-1/4
1/3     -1+2/3  0
1-3^2   -2      7+9/7

determinant: -1243|14
--------------------
>>
```

如果方阵中有一些符号, 也可以计算. 输入下面的矩阵

```
A=[a+1, b-2;
c, -2]
```

在默认的数值计算模式下, 得到

```
>> A=[a+1, b-2;
A=[a+1, b-2;
c, -2]
input> [a+1,b-2;c,-2]
det(A)=-2*a-c*b+2*c-2
----------------------------
 type: matrix
 name: A
value:
a+1     b-2
c       -2

determinant: -2*a-c*b+2*c-2
--------------------
>>
```

2.5.4 行列式展开

v.570版本加入了`CofactorMatrix(A,i,j)`函数(8.3.3)和`DetExpansion(A,ri)`函数(8.4.13). 前者返回矩阵`A`在`|(i,j)|`处元素对应的余子式矩阵, 即 A 中划去第 i 行、第 j 列后由剩下元素所构成的子矩阵. 后者第二个参数`ri`指第`i`行, 例如`r2`即第2行, 也可以写`cj`, 即第`j`列, 这里`j`是某个正整数.

2.5.5 矩阵求逆

v0.544版本加入了`inv(A)`函数, 用于对矩阵 `A`求逆.

例 2.5.1 求矩阵 $A=\begin{pmatrix}1&2&3\\4&5&6\\7&8&0\end{pmatrix}$ 的逆矩阵.

```
>> A=[1 2 3;
A=[1 2 3;
4 5 6;
7 8 0]
A=[1 2 3;
input> [1,2,3;4,5,6;7,8,0]
--------------------

1       2       3
4       5       6
7       8       0

--------------------
>> inv(A)
out>
-1.77777792     0.88888896      -0.11111112
1.55555568      -0.77777784     0.22222224
-0.11111112     0.22222224      -0.11111112


------------------------
```

若需要输出分数形式, 则先切换至`fraction`模式.

```
>> :mode fraction
Switch into fraction calculating mode.
e.g., 1/2+1/3 will return 5/6

>> inv(A)
out>
-16|9   8|9     -1|9
14|9    -7|9    2|9
-1|9    2|9     -1|9


------------------------
```

检验一下. 令 `B=[-16|9 8|9 -1|9;14|9 -7|9 2|9;-1|9 2|9 -1|9]`, 然后再输入`inv(B)` 求 B^{-1}.

```
>> B=[-16|9 8|9 -1|9;14|9 -7|9 2|9;-1|9 2|9 -1|9]
input> [-16|9,8|9,-1|9;14|9,-7|9,2|9;-1|9,2|9,-1|9]
--------------------

-16|9   8|9     -1|9
14|9    -7|9    2|9
-1|9    2|9     -1|9

--------------------
>> inv(B)
out>
1       2       3
4       5       6
```

```
7       8       0

------------------------
```

2.5.6 矩阵的初等变换

自 v0.563 开始, 添加`transform()`函数用于矩阵的初等变换. 矩阵的初等变换分为初等行变换和初等列变换. 初等变换可用于求矩阵的秩, 对于可逆方阵可以求矩阵的逆, 等等.

对于上面的矩阵 A, 作初等行变换 $r_2 - r_1 * 4$, 即第二行减去第一行的4倍, 或第二行加上第一行的 (-4) 倍. 这里 r_k, c_h 分别表示矩阵 A 的第 k 行和第 h 列.

```
>> A=[1 2 3;4 5 6;7 8 0]
input> [1,2,3;4,5,6;7,8,0]
--------------------

1       2       3
4       5       6
7       8       0

--------------------
>> transform(A,r2-r1*4)
1       2       3
0       -3      -6
7       8       0
```

为使矩阵第三行第一列的元素变为零, 执行 `transfor(A,r_3-r_1*7)`.

```
>> transform(A,r3-r1*7)
1       2       3
0       -3      -6
0       -6      -21
```

我们看到`transform()`函数的第二个参数即为要执行的变换, 如果是初等行变换, 则形如`r1+r2*5`, `r2*8`; 若是初等列变换, 则形如`c2-c1*3`, `c1*(-2)`. 也可以执行两行或两列交换的操作, 语法是`r1<-->r2`. 例如, 将上面得到的矩阵进行初等变换: c_2 与 c_3 交换.

```
>> transform(A,c2<-->c3)
1       3       2
0       -6      -3
0       -21     -6

>>
```

例 2.5.2 利用初等变换计算行列式

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{vmatrix}$$

```
>> A=[ 1 2 3 4;
A=[ 1 2 3 4;
2 3 4 1;
3 4 1 2;
4 1 2 3]
A=[ 1 2 3 4;
input> [1,2,3,4;2,3,4,1;3,4,1,2;4,1,2,3]
--------------------

1       2       3       4
2       3       4       1
3       4       1       2
4       1       2       3

--------------------
```

其行列式为 160.

```
>> det(A)
out> 160

------------------------
```

下面使用初等变换求其行列式.

```
>> transform(A, r2-r1*2)
1       2       3       4
0       -1      -2      -7
3       4       1       2
4       1       2       3

>> transform(A, r3-r1*3)
1       2       3       4
0       -1      -2      -7
0       -2      -8      -10
4       1       2       3

>> transform(A, r4-r1*4)
1       2       3       4
0       -1      -2      -7
0       -2      -8      -10
0       -7      -10     -13

>> transform(A, r3-r2*2)
1       2       3       4
0       -1      -2      -7
0       0       -4      4
0       -7      -10     -13

>> transform(A, r4-r2*7)
1       2       3       4
0       -1      -2      -7
0       0       -4      4
0       0       4       36

>> transform(A, r4+r3)
1       2       3       4
0       -1      -2      -7
```

```
0       0       -4      4
0       0       0       40

>> 1*(-1)*(-4)*40
in> 1*(-1)*(-4)*40

out> 160
------------------------
```

2.5.7 初等矩阵

我们将 $E(i,j(k)), E(i(k)), E(i,j)$ 这三类矩阵称为初等矩阵. 它们的具体形式如下:

$$E(i,j(k)) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & k & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

$E(i,j(k))$ 即将单位矩阵在 (i,j) 处的元素置为 k.

$$E(i(k)) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & k & & & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

$E(i(k))$ 即将单位矩阵在 (i,i) 处的元素置为 k.

$$E(i,j) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

$E(i,j)$ 即将单位矩阵的第 i 行和第 j 行互换.

在v.571版本中,我们可以用E()函数生成这三种初等矩阵. 语法分别为E(n,i,j(k)),E(n,i(k)),E(n,i,j). 此处第一个参数n是方阵的阶数.

```
>> E(5,2,3(9))
Elementary matrix>

1       0       0       0       0
0       1       9       0       0
0       0       1       0       0
0       0       0       1       0
0       0       0       0       1


>> E(4,3(-2.1))
Elementary matrix>

1       0       0       0
0       1       0       0
0       0       -2.1    0
0       0       0       1


>> E(3,1,2)
Elementary matrix>

0       1       0
1       0       0
0       0       1


>>
```

生成后的初等矩阵可以用:list找到它们. 命名的方式我们以例子说明:

- E5_2_3x0指类型为E(n,i,j(k))的初等矩阵, 名称中x代表index, 0代表定义了第一个初等矩阵. 由于k可以是整数、小数或分数或字母等, 因此不出现在名称中.
- E4_3x1指类型为E(n,i(k))的初等矩阵, x1的涵义同上.
- E3_1_2指类型为E(n,i,j)的初等矩阵.

```
>> :list
info> All variables are listed below.
matrix : A=[1,2,3,4;2,3,4,1;3,4,1,2;4,1,2,3]    mem addr: 00000215CD7F1BF0
matrix : E3_1_2=[0,1,0;1,0,0;0,0,1]     mem addr: 00000215CD7F20F0
matrix : E5_2_3x0=[1,0,0,0,0;0,1,9,0,0;0,0,1,0,0;0,0,0,1,0;0,0,0,0,1]   mem addr: 00000215CD7F1870
matrix : E4_3x1=[1,0,0,0;0,1,0,0;0,0,-2.1,0;0,0,0,1]    mem addr: 00000215CD7F2270
---*---*---*---
```

初等矩阵左乘某个矩阵 A, 等价于将其作初等行变换; 初等矩阵右乘矩阵 A, 等价于将其作初等列变换.

E(i,j(k))*A 将A作初等行变换ri+rj*k, 相当于transform(A,ri+rj*k). 但是, 值得注意的是,

A*E(i,j(k))将A作初等列变换cj+ci*k, 相当于transform(A,cj+ci*k).

例 2.5.3 设 $A=\begin{pmatrix}1 & 6 & -1\\ 3 & 3 & 0\\ -1 & 0 & -2\end{pmatrix}$, 将第二列减去第一列的6倍, 即执行 `transform(A,c2-c1*6)`.

首先我们使用初等矩阵相乘的方法.

```
>> A=[1 6 -1;
A=[1 6 -1;
3 3 0;
-1 0 -2]
A=[1 6 -1;
input> [1,6,-1;3,3,0;-1,0,-2]
--------------------

1       6       -1
3       3       0
-1      0       -2

--------------------
>> E(3,1,2(-6))
Elementary matrix>

1       -6      0
0       1       0
0       0       1


>> :list
info> All variables are listed below.
matrix : A=[1,6,-1;3,3,0;-1,0,-2]       mem addr: 000002633A5C3DA0
matrix : E3_1_2x0=[1,-6,0;0,1,0;0,0,1]          mem addr: 000002633A5C3EA0
---*---*---*---

>> A*E3_1_2x0

out>
1       0       -1
3       -15     0
-1      6       -2

------------------------
```

下面使用初等列变换, 即执行`transform(A,c2-c1*6)`.

```
>> A
in> A
1       6       -1
3       3       0
-1      0       -2

>> transform(A,c2-c1*6)
1       0       -1
3       -15     0
-1      6       -2
```

2.5.8 矩阵转置

v0.566 版本提供了`transpose()`函数用于返回给定矩阵的转置矩阵.

例 2.5.4 设 $A=\begin{pmatrix} a & b & c & d \\ -b & a & d & -c \\ -c & -d & a & b \\ -d & c & -b & a \end{pmatrix}$, 求 A^T.

```
>> A=[a,b,c,d;
A=[a,b,c,d;
-b,a,d,-c;
-c,-d,a,b;
-d,c,-b,a]
A=[a,b,c,d;
input> [a,b,c,d;-b,a,d,-c;-c,-d,a,b;-d,c,-b,a]
--------------------

a       b       c       d
-b      a       d       -c
-c      -d      a       b
-d      c       -b      a

--------------------
>> transpose(A)
a       -b      -c      -d
b       a       -d      c
c       d       a       -b
d       -c      b       a
```

这个方阵的行列式为 $(a^2+b^2+c^2+d^2)^2$. 使用 `det(A)` 计算得

```
>> det(A)
out> a*a*a*a+a*a*b*b+a*d*d*a-a*d*b*c+a*c*d*b+a*c*a*c+b*b*a*a+b*b*b*b+b*d*c*a+b*d*b*d+b*c*c*b-b*c*a*d-c*b*d*a+c*b*b*c+c
     *a*c*a+c*a*b*d+c*c*c*c+c*c*d*d+d*b*d*b+d*b*a*c-d*a*c*b+d*a*a*d+d*d*c*c+d*d*d*d

------------------------
```

为化简, 先定义 a,b,c,d 为变量, 然后切换到`polyn` 模式, 输入上面的式子.

```
>> symbol(a,b,c,d)
out> a is a variable for polynomial.
out> b is a variable for polynomial.
out> c is a variable for polynomial.
out> d is a variable for polynomial.

------------------------

>> :mode polyn
Switch into polynomial mode.

>> a*a*a*a+a*a*b*b+a*d*d*a-a*d*b*c+a*c*d*b+a*c*a*c+b*b*a*a+b*b*b*b+b*d*c*a+b*d*b*d+b*c*c*b-b*c*a*d-c*b*d*a+c*b*b*c+c*a
     *c*a+c*a*b*d+c*c*c*c+c*c*d*d+d*b*d*b+d*b*a*c-d*a*c*b+d*a*a*d+d*d*c*c+d*d*d*d

out> x_1^4+2x_1^2x_2^2+2x_1^2x_4^2+2x_1^2x_3^2+x_2^4+2x_2^2x_4^2+2x_2^2x_3^2+x_3^4+2x_3^2x_4^2+x_4^4
```

```
15 ------------------------
```

所得结果即 $(x_1+x_2+x_3+x_4)^2$.

`transpose(A)`会将A的转置保存到名为A_t的矩阵变量中, 可用`:list`查看.

```
1  >> :list
2  info> All variables are listed below.
3  matrix : A=[a,b,c,d;-b,a,d,-c;-c,-d,a,b;-d,c,-b,a]      mem addr: 000001E1F3BC0B40
4  matrix : A_t=[a,-b,-c,-d;b,a,-d,c;c,d,a,-b;d,-c,b,a]    mem addr: 000001E1F3BC0BC0
5  ---*---*---*---
6
7  >> A_t
8  in> A_t
9  a       -b      -c      -d
10 b       a       -d      c
11 c       d       a       -b
12 d       -c      b       a
```

`transpose()`也可以直接接受以`[]`界定的矩阵数据, 此时默认矩阵名为`__Matrix__`, 并且作转置变换后, 结果更新到`__Matrix__`中, 不生成名为`__Matrix___t`的变量.

```
1  >> transpose([1 2 3; a b c; x y z; NiHao hello bonjour])
2  >> __Matrix__
3  1       2       3
4  a       b       c
5  x       y       z
6  NiHao    hello   bonjour
7
8  ---------------------
9  >> after transpose, __Matrix__ is:
10 1       a       x       NiHao
11 2       b       y       hello
12 3       c       z       bonjour
13
14 ------------------------
```

2.5.9 矩阵的运算

矩阵的基本运算有加减法、数乘和乘幂运算.

例 2.5.5 设 $A=\begin{pmatrix}1&2&1\\2&1&2\\1&2&3\end{pmatrix}$, $B=\begin{pmatrix}4&1&1\\-4&2&0\\1&2&1\end{pmatrix}$, 求 $A+B$, $A-B$, AB, BA, $AB-BA$, A^2, B^3.

```
1  >> A=[1,2,1;2,1,2;1,2,3]
2  input> [1,2,1;2,1,2;1,2,3]
3  --------------------
4
5  1       2       1
6  2       1       2
7  1       2       3
8
```

```
--------------------

>>  B=[4,1,1;-4,2,0;1,2,1]
input> [4,1,1;-4,2,0;1,2,1]
--------------------

4       1       1
-4      2       0
1       2       1

--------------------
>> A+B

out>
5       3       2
-2      3       2
2       4       4

------------------------

>> A-B

out>
-3      1       0
6       -1      2
0       0       2

------------------------

>> A*B-B*A

out>
-3      7       2
6       8       4
-1      11      4

7       11      9
0       -6      0
6       6       8

-10     -4      -7
6       14      4
-7      5       -4

------------------------

>> A^2

out>
6       6       8
6       9       10
8       10      14

------------------------

>> B^3

out>
```

```
25      39      18
-100    -32     -28
-38     15      -1

-----------------------
```

习题 2.5.1 设 A 是如下形式的 n 阶方阵, 求它的各次幂, 即 A^m, $m = 1, 2, \ldots, n$.

$$A = \begin{pmatrix} \lambda & 1 & 0 & \cdots & 0 \\ 0 & \lambda & 1 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \\ 0 & 0 & \cdots & \lambda & 1 \\ 0 & 0 & \cdots & 0 & \lambda \end{pmatrix}_n$$

2.5.10 矩阵的特征值

现有的方法已经可以计算三阶方阵的特征值. 我们以例子来说明.

例 2.5.6 求下面矩阵的特征值.

$$A = \begin{pmatrix} 2 & -1 & 2 \\ 5 & -3 & 3 \\ -1 & 0 & -2 \end{pmatrix}$$

特征值即是矩阵 A 的特征多项式 $|\lambda I - A|$ 的根, 这里 I 是 3 阶单位矩阵. 为输入方便, 这里用 x 代替 λ. 输入矩阵 $xI - A$.

```
>> A=[x-2 1 -2;
A=[x-2 1 -2;
-5 x+3 -3;
1 0 x+2]
A=[x-2 1 -2;
input> [x-2,1,-2;-5,x+3,-3;1,0,x+2]
--------------------

x-2     1       -2
-5      x+3     -3
1       0       x+2

--------------------
```

当然, 为防止输错, 我们也可以先输入矩阵 A, 再计算 $xI - A$.

```
>> A=[2 -1 2;
A=[2 -1 2;
5 -3 3;
-1 0 -2]
A=[2 -1 2;
input> [2,-1,2;5,-3,3;-1,0,-2]
--------------------

```

```
2       -1      2
5       -3      3
-1      0       -2

--------------------
>> I=[1 0 0;
I=[1 0 0;
0 1 0;
0 0 1]
I=[1 0 0;
input> [1,0,0;0,1,0;0,0,1]
--------------------

1       0       0
0       1       0
0       0       1

--------------------
>> x*I-A

out>
x       0       0
0       x       0
0       0       x

x-2     1       -2
-5      x+3     -3
1       0       x+2

------------------------
```

使用 `:list` 命令找到矩阵在内部存储空间的名称. 我们以第二种方法为例.

```
>> :list
info> All variables are listed below.
matrix : A=[2,-1,2;5,-3,3;-1,0,-2]      mem addr: 000002022C040930
matrix : I=[1,0,0;0,1,0;0,0,1]          mem addr: 000002022C041530
matrix : __tmpMatrix__tmpMatrixx_mul_I_4_minus_A_4=[x-2,1,-2;-5,x+3,-3;1,0,x+2]         mem addr: 000002022C0411B0
matrix : __tmpMatrixx_mul_I_4=[x,0,0;0,x,0;0,0,x]       mem addr: 000002022C040B30
---*---*---*---
```

其中 `__tmpMatrix__tmpMatrixx_mul_I_4_minus_A_4` 就是 $xI - A$ 所得矩阵的内部名称. 我们将等号后面的矩阵表达式拷贝, 并赋给变量`D`(可以随意命名).

```
>> D=[x-2,1,-2;-5,x+3,-3;1,0,x+2]
input> [x-2,1,-2;-5,x+3,-3;1,0,x+2]
--------------------

x-2     1       -2
-5      x+3     -3
1       0       x+2

--------------------
```

然后再求 `D` 的行列式.

```
>> det(D)
out> x*x*x-2*x*x+2*x*x-2*2*x+3*x*x-2*3*x+6*x+5*x+2*x+1
```

```
3
4 ------------------------
```

切换到polyn模式. (关于多项式的运算请见下一节.)

```
1 >> :mode polyn
2 Switch into polynomial mode.
```

将刚获得的det(D)的结果表达式拷贝后再次输入, Sowya会帮我们化简, 并将多项式降幂排列.

```
1 >> x*x*x-2*x*x+2*x*x-2*2*x+3*x*x-2*3*x+6*x+5*x+2*x+1
2
3 out> x^3+3x^2+3x+1
4 ------------------------
```

然后使用solve()函数就可以解出三个根了.

```
1 >> solve(x^3+3x^2+3x+1==0)
2
3 It is a univariate cubic equation.
4
5   x^3+3x^2+3x^1+1 == 0
6
7 Answer:
8 Let y=x+3/3
9
10 We get equation of y: y^3+py+q==0
11 where
12
13         p=b-a^2/3,    q=c-ab/3+2a^3/27,
14
15 and a,b,c,d are coefficients of the equation ax^3+bx^2+cx+d==0
16
17 By computation,
18         p = 0
19         q = 0
20 Now the equation is:
21
22         y^3+0y == 0
23
24 solution>
25         x1 = -1
26         x2 = (0*i-1)
27         x3 = (-0*i-1)
28
29
30 ------------------------
```

可见矩阵 A 的三个特征值均为 -1.

2.5.11 矩阵的秩

在 v0.572 版本中, 添加了rank()函数, 用于计算矩阵的秩.

例 2.5.7 计算下列矩阵的秩.

$$A=\begin{pmatrix}1 & 3 & 5 & -1\\ 2 & -1 & -3 & 4\\ 5 & 1 & -1 & 7\end{pmatrix}$$

```
>> A=[1 3 5 -1;
A=[1 3 5 -1;
2 -1 -3 4;
5 1 -1 7]
A=[1 3 5 -1;
input> [1,3,5,-1;2,-1,-3,4;5,1,-1,7]
--------------------

1       3       5       -1
2       -1      -3      4
5       1       -1      7

--------------------
>> rank(A)
2
>>
```

`rank(A)`采用初等行变换的方法计算矩阵`A`的秩并输出.

`rank()`函数也可以添加第二个参数, 可以是`row`,`col`或`hint`.

- `rank(A,row)`采用初等行变换计算矩阵A的秩并输出具体求解步骤, 并返回列向量组中的一个极大线性无关组;
- `rank(A,col)`采用初等列变换计算矩阵A的秩并输出具体求解步骤, 并返回行向量组中的一个极大线性无关组;
- `rank(A,hint)`当`A`的行数不超过列数时, 采用初等行变换; 当`A`的行数大于列数时, 采用初等列变换, 并显示具体求秩的过程.

```
>> rank(A,hint)
r2+r1*(-2) ==>

1       3       5       -1
0       -7      -13     6
5       1       -1      7

------------
r3+r1*(-5) ==>

1       3       5       -1
0       -7      -13     6
0       -14     -26     12

------------
r2*(-1|7) ==>

1       3       5       -1
0       1       13|7    -6|7
0       -14     -26     12

```

```
------------
r3+r2*14 ==>

1       3       5       -1
0       1       13|7    -6|7
0       0       0       0

------------

The linearly independent column vectors are:
c1,c2.
2
```

习题 2.5.2 尝试使用 `rank(A,row)` 或 `rank(A,col)` 计算 A 的秩.

2.5.12 解线性方程组

v0.575版本对函数`solve()`增加了求线性方程组的功能.

例 2.5.8 求线性方程组

$$\begin{cases} 2x_1+3x_2-x_3+x_4 &= 1, \\ 8x_1+12x_2-9x_3+8x_4 &= 3, \\ 4x_1+6x_2+3x_3-2x_4 &= 3, \\ 2x_1+3x_2+9x_3-7x_4 &= 3. \end{cases}$$

（注: 例子来源于李炯生、查建国编著《线性代数》 P.192 例2.）

先输入系数矩阵`A`和等号右边的列向量`b`, 方程组化为`Ax=b`的形式. 称`(Ab)|`为线性方程组的增广矩阵.

```
>> A=[2 3 -1 1;
A=[2 3 -1 1;
8 12 -9 8;
4 6 3 -2;
2 3 9 -7]
A=[2 3 -1 1;
input> [2,3,-1,1;8,12,-9,8;4,6,3,-2;2,3,9,-7]
--------------------

2       3       -1      1
8       12      -9      8
4       6       3       -2
2       3       9       -7

--------------------
>> b=[1;3;3;3]
input> [1;3;3;3]
--------------------

1
3
```

```
3
3

--------------------
```

然后调用`solve()`函数求解此线性方程组. 即输入`solve(A*x==b,hint)`. 第二个参数`hint`可以不加, `hint`代表打印解答过程.

```
>> solve(A*x==b,hint)

>> 2    3       -1      1       1
8       12      -9      8       3
4       6       3       -2      3
2       3       9       -7      3
r1*1|2 ==>

1       3|2     -1|2    1|2     1|2
8|1     12|1    -9|1    8|1     3|1
4|1     6|1     3|1     -2|1    3|1
2|1     3|1     9|1     -7|1    3|1

------------
r2+r1*(-8|1) ==>

1       3|2     -1|2    1|2     1|2
0       0       -5      4       -1
4|1     6|1     3|1     -2|1    3|1
2|1     3|1     9|1     -7|1    3|1

------------
r3+r1*(-4|1) ==>

1       3|2     -1|2    1|2     1|2
0       0       -5      4       -1
0       0       5       -4      1
2|1     3|1     9|1     -7|1    3|1

------------
r4+r1*(-2|1) ==>

1       3|2     -1|2    1|2     1|2
0       0       -5      4       -1
0       0       5       -4      1
0       0       10      -8      2

------------
r2*(-1|5) ==>

1       3|2     -1|2    1|2     1|2
0       0       1       -4|5    1|5
0       0       5       -4      1
0       0       10      -8      2

------------
r1+r2*1|2 ==>

1       3|2     0       1|10    3|5
0       0       1       -4|5    1|5
```

```
0       0       5       -4      1
0       0       10      -8      2

------------
r3+r2*(-5) ==>

1       3|2     0       1|10    3|5
0       0       1       -4|5    1|5
0       0       0       0       0
0       0       10      -8      2

------------
r4+r2*(-10) ==>

1       3|2     0       1|10    3|5
0       0       1       -4|5    1|5
0       0       0       0       0
0       0       0       0       0

------------

The linearly independent column vectors are:
c1,c1,
The solution is:
x= E_0 + C_1*E_1 + C_2*E_2
where E_0 is the special solution and the others form the base of the solution:
---------------
E_0     E_1     E_2
------------------------
3|5     -3|2    -1|10
0       1       0
1|5     0       4|5
0       0       1

------------------------
```

如果仅输入`solve(A*x==b)`, 则中间的初等行变换过程不输出. 我们还提供了`normal`参数, 例如`solve(A*x==b,normal)`,
这里`normal`的意思是输出的解向量前$r = \text{rank}(A)$个对应的分量 $x_{i_1}, x_{i_2}, \ldots, x_{i_r}$ 的下标 $i_1, i_2, \ldots, i_r$ 指代了矩阵 A 的极大线性无关组 $\{c_{i_1}, c_{i_2}, \ldots, c_{i_r}\}$. 若此时也希望输出具体求解过程, 则输入`solve(A*x==b,normal,hint)`.

注意这里`x`可以使用其他字母. 如果是 Pro 版本, Sowya会保存生成的解(矩阵). 键入`:list`会看到生成的`x_solution`矩阵. 这个矩阵的第一列是线性方程组`Ax=b`的一个特解, 其他列组成了该线性方程组的基础解系.

```
>> :list
info> All variables are listed below.
matrix : I=[1,0,0;0,1,0;0,0,1]          mem addr: 000002022C041330
matrix : __tmpMatrix__tmpMatrixx_mul_I_4_minus_A_4=[x-2,1,-2;-5,x+3,-3;1,0,x+2]         mem addr: 000002022C040B30
matrix : __tmpMatrixx_mul_I_4=[x,0,0;0,x,0;0,0,x]       mem addr: 000002022C0409B0
matrix : D=[x-2,1,-2;-5,x+3,-3;1,0,x+2]         mem addr: 000002022C041830
matrix : A=[2,3,-1,1;8,12,-9,8;4,6,3,-2;2,3,9,-7]       mem addr: 000002022C040D30
matrix : b=[1;3;3;3]    mem addr: 000002022C041630
matrix : Ab=[2,3,-1,1,1;8,12,-9,8,3;4,6,3,-2,3;2,3,9,-7,3]      mem addr: 000002022C040A30
matrix : x_solution=[3|5,-3|2,-1|10;0,1,0;1|5,0,4|5;0,0,1]      mem addr: 000002022C0410B0
```

```
11 ---*---*---*---
```

输入 x_solution, 得到刚才的 E_0, E_1, E_2 组成的矩阵.

```
1 >> x_solution
2 in> x_solution
3 3|5     -3|2    -1|10
4 0       1       0
5 1|5     0       4|5
6 0       0       1
```

因此原线性方程组的通解为

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \frac{3}{5} \\ 0 \\ \frac{1}{5} \\ 0 \end{pmatrix} + C_1 \begin{pmatrix} -\frac{3}{2} \\ 1 \\ 0 \\ 0 \end{pmatrix} + C_2 \begin{pmatrix} -\frac{1}{10} \\ 0 \\ \frac{4}{5} \\ 1 \end{pmatrix}$$

2.6 一元多项式

在 v0.528 版本中, 我们增加了 polyn 模式, 即可以使用命令

:mode=polyn

进入多项式计算模式. 这里只处理一元多项式, 且变元为 x. 以后根据需要进行改进, 以支持其他变元符号. (在v0.592版本中加入了:change命令, 用于改变多元多项式的主变元, 当然也适用于这里的一元多项式. 详见更改主变元一节.)

此模式下输入多项式时, 系数和 x 之间不需要加 *. 在v0.577版本中, 特增加了函数ispolyn(), 用于判断输入的式子是否是合法的多项式.

```
1 >> ispolyn(3*(9+2)x^3)
2 33x^3
3 Yes
4 >> ispolyn(3*(9+2)x^3+2x)
5 33x^3+2x
6 Yes
7 >> ispolyn(3*(9+2)x^3+2*x)
8 33x^3+2*1x
9 Yes
```

这个函数的功能暂时还比较弱, 对于每一项的系数, 支持四则运算.

因此, 在输入多项式时, 尽量采用以下规则:

- 单项式形如 2x^3, x, x^-2, -x^3, 3|4x^5 即系数是整数或分数, 不要写成复杂的表达式, 例如: (9+2)*3/2x^3.
- 单项式与单项式之间使用+或-号连接.
- 如果表达式中有两个或两个以上的多项式, 则每个多项式要括上小括号.

2.6.1 多项式的加减法和乘法

```
>> (-4x^3+x^2)+(3x-x^6)*2x

out> -2x^7-4x^3+7x^2
------------------------
```

Sowya 会在多项式进行四则运算后进行简化输出, 比如 `x^0` 视为 `1`, `1x^3` 简化为 `x^3` 等等.

```
>> (x+1)*(x^2-x+1)

out> x^3+1
------------------------
```

若输入 `-4x^3-4x^2-6x^7+6x^2`, 则会简化输出, 当然也可以使用 `simplify()` 函数.

```
>> -4x^3-4x^2-6x^7+6x^2
-4
out> -6x^7-4x^3+2x^2

------------------------

>> simplify(-4x^3-4x^2-6x^7+6x^2)
out> -6x^7-4x^3+2x^2

------------------------
```

```
>> (-4x^3-4x^2)+(-6x^7+6x^2)

out> -6x^7-4x^3+2x^2
------------------------
```

2.6.2 多项式的除法

例 2.6.1 求 x^3+x+1 被 $x+1$ 除所得的商和余式.

```
>> div(x^3+x+1,x+1)

 quotient> q(x) = x^2-x+2
remainder> r(x) = -1

(x^2-1x^1+2)+(-1)/(x^1+1)
------------------------
```

例 2.6.2 设 $f(x)=x^4+3x^3-x^2-4x-3$, $g(x)=3x^3+10x^2+2x-3$, 求 $f(x)$ 除以 $g(x)$ 所得的商 $q(x)$ 和余式 $r(x)$.

注 2.6.1 此例见李炯生、查建国编著《线性代数》P.12.

这个例子中, 系数之间的运算需要使用分数运算才能正确得到结果. 在 v0.533 版本之后, Sowya 在多项式的运算中, 凡是系数之间的运算都使用分数运算.

```
>> div(x^4+3x^3-x^2-4x-3,3x^3+10x^2+2x-3)

 quotient> q(x) = 1|3x-1|9
remainder> r(x) = -5|9x^2-25|9x-10|3

(1|3x^1-1|9)+(-5|9x^2-25|9x^1-10|3)/(3x^3+10x^2+2x^1-3)
------------------------
```

如果只想获得商, 则使用 %运算符; 若只想获得余式, 则使用 mod 运算符, 不过需要在 polyn 模式下.

```
>> (x^4+3x^3-x^2-4x-3)%(3x^3+10x^2+2x-3)

out> 1|3x^1-91|9
------------------------

>> :mode polyn
Switch into polynomial mode.

>> (x^4+3x^3-x^2-4x-3)mod(3x^3+10x^2+2x-3)

out> -5|9x^2-25|9x^1-10|3
------------------------
```

注 2.6.2 为确保运算正确, 在涉及多项式的计算时, 始终先切换到 polyn 模式.

2.6.3 多项式的幂运算

```
>> :mode polyn
Switch into polynomial mode.

>> (x+1)^8*(x^4-8x^3-12x^2-8x-2)

out> x^12-48x^10-272x^9-780x^8-1416x^7-1764x^6-1560x^5-987x^4-440x^3-132x^2-24x^1-2
------------------------
```

从 v0.532 开始, 多项式模式下可以处理非整数的系数, 但是指数仍限定为整数(使用了 C++ 中的 long long 类型, 范围为-9223372036854775808~9223372036854775807, 即 -2^63~2^63-1).

例 2.6.3 计算 $(x+2)^{100}$

注 2.6.3 需要注意的是, 当指数比较大时, 比如这里的 100 次方, 必须在 polyn 模式下计算. 否则在 numerical 模式下会非常慢.

```
>> (x+2)^100

out> x^100+200x^99+19800x^98+1293600x^97+62739600x^96+2409200640x^95+76291353600x^94+2048967782400x^93+47638500940800x
     ^92+973942685900800x^91+17725756883394560x^90+290057839910092800x^89+4302524625333043200x^88+58249564158355046400
     x^87+723958868825269862400x^86+8301395029196427755520x^85+88202322185212044902400x^84+871646478065624914329600x
     ^83+803851751993854087659520 0x^82+693850985931537212506112 00x^81+562019298604545142129950720x
     ^80+428205179889177251146629 1200x^79+307529174647681844005306368 00x^78+208585005413210294194903449600x
     ^77+133842045140143272108396380 1600x^76+81375963445207109441904999137 28x^75+469476712183887169857144225792 00x
     ^74+257342790382278893106879057100800x^73+13418588355647399426287265120256 00x
```

```
    ^72+6663023183493881094432297163161600x^71+31538309735204370513646206572298240x
    ^70+1424310762235036087713054490361856600x^69+614234016213859312826254748968550400x
    ^68+25313886728813595922536559351431168 00x^67+997664947547359368711734986203463 6800x
    ^66+37626220878928981905699719479673487360x^65+13587246428502132354836009812104314 8800x
    ^64+47004528185088457876189439349982494 7200x^63+15585711977160909716841761468678406144 00x
    ^62+49554571414562892433035344156823650304 00x^61+15114144281441682192075779967831213342720x
    ^60+44236519848121996659733990149749892710400x^59+12428355576377132394877644851596398428 1600x
    ^58+33527656903715054832693181460120516689920 0x^57+86867111068716278430159606510312247787 5200x
    ^56+21620258754880495964839724287011048338227 20x^55+51700618761670751220268905903722072113152 00x
    ^54+11880142183532853471891578377876561251532800x^53+26235313988635051417093902251144072763801600x
    ^52+55683115404449905048525833349367011580313600x^51+11359355542507780629899270003270870362383 9744x
    ^50+22273246161779962019410333339746804632 1254400x^49+41976502381816082267350243601830516422082 5600x
    ^48+76032909974610262220106101618409992009809920 0x^47+13235358402987712312388839911352850460966912 00x
    ^46+22139144964997627867995877669899313498344652 80x^45+35580768693746187644993374826623896693768192 00x
    ^44+54931713071046745837884508504261454544764928 00x^43+81450471105345174863070133299422156738789376 00x
    ^42+11596338259066092692369307113816035874675097600x^41+15848328954056993346238053055548582362055966720x
    ^40+20784693710238679798344987613834206376466841600x^39+26148485635461564907595306998049485441361510400x
    ^38+31544204893572681475829259235742236405451980800x^37+36472986908193412956427580991326960843803852800x
    ^36+40400847036768088197889012790392941242367344640x^35+42849383220814638997761074171628877075238092800x
    ^34+43488926253961126146981388711503934942032691200x^33+42209840187668151848540759631753819208443494400x
    ^32+39151156116097995917487081397568759845512806400x^31+34676738274258224955488557809275187291739914240x
    ^30+29304285865570330948300189697979031514146406400x^29+23606230280598322152797375034483108719729049600x
    ^28+18108888982376795076118534273028138195956531200x^27+13214594662815499109600011496534046791643955200x
    ^26+91621189662187460493226746375969391088731422 72x^25+60277098461965434535017596299979862558375936 00x
    ^24+37575334106160270878972008083104329906520064 00x^23+22159812421581698210675799638753835585896448 00x
    ^22+12342174006956895205946014988673022351638528 00x^21+64796413536523699831216578690533367346102272 0x
    ^20+31998228906925283867267446266930057948692480 0x^19+14828447542233668133611743391991978073784320 0x
    ^18+64316158014507476242171417121892916946534400x^17+26032730624919692764688430739813799716454400x
    ^16+98005574117344725702356445138122540109004 80x^15+34187990971166764779891783187717165154304 00x
    ^14+11003031576927234641804252060414719819776 00x^13+32508956931830465987148926542134399467520 0x
    ^12+87664602962239458841749914270924223283200x^11+21429125168547423272427756821781476802560x
    ^10+47096978392411919280061004003915333632 00x^9+92146262072110276852293268703312609280 0x
    ^8+15853120356492090641254755905946255360 0x^7+23611030318179709465698572625877401600x
    ^6+29824459349279633009303460159003033 60x^5+31067145155499617718024437665628160 0x
    ^4+25622387757113086777752113538662400x^3+15687176177824338843521702166528 00x^2+63382530011411470074835160268800x
    ^1+12676506002282294014967032053 76
4 ------------------------
```

习题 **2.6.1** 计算 $(0.2x + 1.3)^{10}$.

2.6.4 判断两个多项式是否相等

例 **2.6.4** 判断多项式 `2x^2-x+3` 与 `3-x+x^2+x^2` 是否相等.

```
1 >> (2x^2-x+3)==(3-x+x^2+x^2)
2
3 out> 1
4 ------------------------
```

返回 `1` 即表示相等, 如果返回 `0` 则表示不等. 需要注意的是, 多项式要用圆括号括起来.

2.6.5 解方程

在 v0.539 版本中, 我们扩充了 Sowya 的 `solve()` 函数的功能, 使之对多项式方程也能处理.

例 2.6.5 已知多项式 $x^{13}+x+90$ 能被 x^2-x+a 整除, 其中 a 是整数, 求 a 可能的值.

根据题设, $|a|$ 一定是 90 的因子, 因此对于变量 a, 搜索的区间可定为 $[-90,90]$. 首先进入 `polyn` 模式.

```
>> :mode=polyn
Switch into polynomial mode.
```

然后, 使用 `solve` 函数. (目前 `solve()` 函数本质上是搜索.)

```
>> solve((x^13+x+90)mod(x^2-x+a)==0,a,-90,90)
ans>> a=2

------------------------
```

2.6.6 最大公因式

设 $f(x)$ 和 $g(x)$ 是两个一元多项式, 使用辗转相除法可以得到它们的最大公因式 $d(x)$. 这里提供了函数 `Gcd(poly1, poly2)`可以方便求出两个或多个一元多项式的公因式.

例 2.6.6 求 $x^4+3x^3-x^2-4x-3$ 与 $3x^3+10x^2+2x-3$ 的最大公因式.

```
>> Gcd(x^4+3x^3-x^2-4x-3,3x^3+10x^2+2x-3)
9x+27
------------------------
```

函数 `Gcd2(poly1, poly2)` 会详细给出计算最大公因式的具体步骤, 并且给出 `u(x)`和`v(x)`, 使得

$$d(x)=u(x)f(x)+v(x)g(x).$$

```
>> Gcd2(x^4+3x^3-x^2-4x-3,3x^3+10x^2+2x-3)
out>
x^4+3x^3-1x^2-4x^1-3 == (1|3x^1-1|9)*(3x^3+10x^2+2x^1-3) + (-5|9x^2-25|9x^1-10|3)

3x^3+10x^2+2x^1-3 == (-27|5x^1+9)*(-5|9x^2-25|9x^1-10|3) + (9x^1+27)

-5|9x^2-25|9x^1-10|3 == (-5|81x^1-10|81)*(9x^1+27)


------------------------------------
f(x) = x^4+3x^3-1x^2-4x^1-3
g(x) = 3x^3+10x^2+2x^1-3

 The remainder polynomials are:

r(1) = -5|9x^2-25|9x^1-10|3
r(2) = 9x^1+27

```

```
-----u(x)*f(x)+v(x)*g(x)---------
9x^1+27 ==
        (27|5x^1-9)*f(x) + (-9|5x^2+18|5x^1)*g(x)
The greatest common divisor is:
9x+27
------------------------
```

2.6.7 多项式求导

v0.545 版本增加了 `diff()` 函数, 用于对一元多项式函数进行求导, 这里默认自变量为 `x`.

例 2.6.7 求 $x^4+3x^3-x^2-9x+7$ 的导数.

```
>> diff(x^4+3x^3-x^2-9x+7)
out> 4x^3+9x^2-2x^1-9

------------------------
```

若出现 ax^{-n} 形式的项, 也可以求导.

例 2.6.8 设 $f(x)=x^4+3x^3-x^2-9x+7-x^{-1}+10x^{-2}$, 求 $f'(x)$.

```
>> diff(x^4+3x^3-x^2-9x+7-x^-1+10x^-2)
out> 4x^3+9x^2-2x^1-9+x^-2-20x^-3

------------------------
```

2.6.8 重因式

设 $f(x)$ 是一个多项式, 具有标准分解式

$$f(x)=cp_1^{r_1}(x)p_2^{r_2}(x)\cdots p_s^{r_s}(x).$$

$f(x)$ 和 $f'(x)$ 的最大公因式必具有标准分解式

$$p_1^{r_1-1}(x)p_2^{r_2-1}(x)\cdots p_s^{r_s-1}(x).$$

于是

$$\frac{f(x)}{(f(x),f'(x))}=cp_1(x)p_2(x)\cdots p_s(x)$$

没有重因式.

只要 $f(x)$ 和 $f'(x)$ 不是互素的, 则上式给出了对 $f(x)$ 作因式分解的一个思路.

例 2.6.9 判断 $f(x)=x^5-5x^4+7x^3-2x^2+4x-8$ 是否有重因式.

首先切换至多项式计算模式.

```
>> :mode polyn
Switch into polynomial mode.
```

使用 `diff()`函数对 $f(x)$ 求导.

```
>> diff(x^5-5x^4+7x^3-2x^2+4x-8)
out> 5x^4-20x^3+21x^2-4x^1+4

------------------------
```

然后再计算 $f(x)$ 和 $f'(x)$ 的最大公因式(即 $(f(x), f'(x))$).

```
>> Gcd(x^5-5x^4+7x^3-2x^2+4x-8,5x^4-20x^3+21x^2-4x^1+4)
49|4x^2-49x+49
------------------------
```

使用 `monic_polyn()` 函数将其变为首一多项式（所谓首一多项式即最高次项的系数为 1 的多项式.）

```
>> monic_polyn(49|4x^2-49x+49)
out> x^2-4x+4
------------------------
```

当然也可以直接手算将其变为首一多项式.

```
>> (49|4x^2-49x+49)*(4|49)

out> x^2-4x^1+4
------------------------
```

而 $x^2 - 4x + 4 = (x - 2)^2$,

```
>> (x^2-4x+4)==(x-2)^2

out> 1
------------------------
```

这说明 $f(x)$ 是有重因式的. 最后计算 $\dfrac{f(x)}{(f(x), f'(x))}$.

```
>> div(x^5-5x^4+7x^3-2x^2+4x-8,x^2-4x+4)

x^3-1x^2-1x^1-2
------------------------
```

因此

$$x^5 - 5x^4 + 7x^3 - 2x^2 + 4x - 8 = (x - 2)^2 \cdot (x^3 - 1x^2 - 1x^1 - 2).$$

2.6.9 更改主变元

v0.592版本加入了`:change`命令, 以更改所用的主变元(默认是 x).

```
>> :mode polyn
Switch into polynomial mode.

>> (x-1)*(x+1)

out> x^2-1
------------------------
```

如上,x 是默认的主变元. 现在我们通过 :change 命令将其改为 t. 语法是 :change x t 或 :change x->t 或 :change x=t. 如果输错了, 则会提示正确的语法. 比如输成了 :change x-->t, 则显示如下.

```
>> :change x-->t

The command :change will change the main variable used in polynomial.
usage> :change x t
   or> :change x->t
   or> :change x=t
```

现在输入正确的命令.

```
>> :change x t

>> (t-1)*(t+1)

out> t^2-1
------------------------
```

注 2.6.4 主变元只能用单字符表示.

2.6.10 多项式插值

拉格朗日插值法(Lagrange Approximation)是以法国十八世纪数学家约瑟夫·拉格朗日命名的多项式插值方法.

v0.612版本提供了 LagrangePolyn() 函数, 用于生成拉格朗日插值多项式.

例 2.6.10 对于函数 $f(x)=\sqrt{x}$, 已知 $f(1)=1, f(4)=2, f(9)=3$. 求相应的拉格朗日插值多项式.

这里提供了三个点的坐标 (1,1),(4,2),(9,3), 我们将其作为参数传至 LagrangePolyn().

```
>> LagrangePolyn(1,1;4,2;9,3)
The points are:
(1,1) (4,2) (9,3)
---------------
ell_0 = (t-(4))/(1-(4))*(t-(9))/(1-(9))
      = (t-4)*(t-9)|24
      = 1|24t^2-13|24t^1+3|2

ell_1 = (t-(1))/(4-(1))*(t-(9))/(4-(9))
      = -((t-1)*(t-9))|15
      = -1|15t^2+2|3t^1-3|5

ell_2 = (t-(1))/(9-(1))*(t-(4))/(9-(4))
```

```
      = (t-1)*(t-4)|40
      = 1|40t^2-1|8t^1+1|10

Lagrange polyn is:
        (1)*(1|24t^2-13|24t^1+3|2)+(2)*(-1|15t^2+2|3t^1-3|5)+(3)*(1|40t^2-1|8t^1+1|10)
      = -1|60t^2+5|12t+3|5


------------------------
```

上面得到的多项式是关于 t 的多项式, 这是因为之前将主变元 x 改成了 t. 执行 `:change t x` 就可改回 x 为主变元.

2.7 多元多项式

v0.594版本初步实现了多元多项式的加减法运算. 仍然使用 `:mode polyn` 进入多项式计算模式.

2.7.1 定义多元多项式中的变量

首先使用 `symbol()` 函数定义要用到的变量, 例如定义 `x,y,z,what` 如下.

```
>> symbol(x,y,z,what)
out> x is a variable for polynomial.
out> y is a variable for polynomial.
out> z is a variable for polynomial.
out> what is a variable for polynomial.

------------------------
```

注意这些变量与 `:list` 命令输出的变量不同. 若要查看这些变量, 请使用 `:poly_vars` 命令列出.

```
>> :list
info> All variables are listed below.
---*---*---*---

>> :poly_vars
what --> x_3
y --> x_1
z --> x_2

---------------
x, x_1, x_2, x_3,
---*---*---*---
```

从上面看到, `:list` 显示为空, 即当前没有定义变量. 而 `:poly_vars` 列出了 `x,y,z,what` 所对应的内部变量 `x,x_1,x_2,x_3`. 这是为了内部处理方便以及未来功能的扩充而设定的. 这些变量被存储在内部的容器 `poly_vars` 中.

2.7.2 删除所定义的变量

使用命令 :delete 或 :del 删除已定义的多元多项式中的变量. 注意主变元 x 或用 :change 命令更改的主变元没法删除.

注 2.7.1 :list 显示的变量可用 :clear 清除.

```
>> :delete y

>> :poly_vars
what --> x_3
z --> x_2

---------------
x, x_2, x_3,
---*---*---*---

>> :del what

>> :poly_vars
z --> x_2

---------------
x, x_2,
---*---*---*---
```

2.7.3 多元多项式的加减法

切换到多项式计算模式(:mode polyn)与定义多元多项式变量(使用symbol()函数)之间并没有先后次序要求. 现在先进入多项式计算模式, 然后输入 x^2-y^3+zy+6x^2.

```
>> :mode polyn
Switch into polynomial mode.

>> x^2-y^3+zy+6x^2

out> 7x^2-y*y*y+zy
------------------------
```

此时 y和z被视为常量, 也可以认为是尚未定义的变量. 现在用 symbol() 函数定义这两个变量, 然后再输入表达式 x^2-y^3+zy+6x^2.

```
>> symbol(y,z)
out> y is a variable for polynomial.
out> z is a variable for polynomial.

------------------------

>> x^2-y^3+zy+6x^2

out> 7x^2-y^3+yz
------------------------
```

再举个例子, 输入 x^2-y^3+zyx+6x^2+7y^3-xyz, 注意此时 x,y,z 是已定义变量.

```
>> x^2-y^3+zyx+6x^2+7y^3-xyz

out> 7x^2+6y^3
------------------------
```

2.7.4 多元多项式的乘法及幂运算

先检查当前所处模式, 然后定义变量 a,b,c,d,y.

```
>> :mode
Calculating mode: polyn & numerical

>> symbol(a,b,c,d,y)
out> a is a variable for polynomial.
out> b is a variable for polynomial.
out> c is a variable for polynomial.
out> d is a variable for polynomial.
Error> Failed to define polynomial variable y with symbol() function.

------------------------
```

注意到最后定义变量 y 失败, 这是因为之前已经定义过. 尝试输入下面的表达式

- (y-x)*2
- (a+b)(c+d)
- (x+y)^5
- (a+b+c)^2
- (y+x)(ax-by)

```
>> (y-x)*2

out> 2y-2x
------------------------

>> (a+b)(c+d)

out> ac+ad+bc+bd
------------------------

>> (x+y)^5

out> x^5+5x^4y+10x^3y^2+10x^2y^3+5xy^4+y^5
------------------------

>> (a+b+c)^2

out> a^2+2ab+2ac+b^2+2bc+c^2
------------------------

>> (y+x)(ax-by)

out> xay-by^2+x^2a-xby
------------------------
```

2.7.5 BUG

v0.643 修复了多元多项式运算中的一些BUG. 假设系统中已经定义了 z,x_1,x_2 这三个变量.

```
>> symbol(z,x_1,x_2)
out> z is a variable for polynomial.
out> x_1 is a variable for polynomial.
out> x_2 is a variable for polynomial.

------------------------
```

使用 :poly_vars 查看

```
>> :poly_vars
x_1 --> x_2
x_2 --> x_3
z --> x_1

---------------
x, x_1, x_2, x_3,
---*---*---*---
```

这表明, z,x_1,x_2 分别对应到内部变量 x_1,x_2,x_3. 当用户输入 x_2+2x_3 时, 这里的 x_3 其实尚未定义. 在之前的版本会返回 3x_3 等错误结果. 这里, 系统特别地会检查是否有形如 x_j 的这种变量, 如果 poly_vars 中没有定义, 则自动定义此变量. 下面切换到多项式模式,

```
>> :mode polyn
Switch into polynomial mode.
```

输入 x_2+3x_3+x_4.

```
>> x_2+2x_3+x_4
Add var x_3 to poly_vars.
Add var x_4 to poly_vars.

out> x_2+2x_3+x_4
------------------------
```

这里看到, 系统自动添加了两个新的变量 x_3 和 x_4. 最后查看一下 poly_vars 中的内容.

```
>> :poly_vars
x_1 --> x_2
x_2 --> x_3
x_3 --> x_4
x_4 --> x_5
z --> x_1

---------------
x, x_1, x_2, x_3, x_4, x_5,
---*---*---*---
```

目前暂时不支持 x_ 后面加非数字的变量. 若输入这样的变量, 系统会退出.

```
>> x_2+2x_3+y2*x_a
Add var x_ to poly_vars.
```

2.8 复数

v0.553版本加入 `complex` 模式, 可以处理复数的加减乘除运算.

```
>> :mode complex
Switch into complex mode.

>> (1+2i)+(3-4i)

out> 4-2i
------------------------

>> (1+2i)-(3-4i)

out> -2+6i
------------------------

>> (1+2i)*(3-4i)

out> 11+2i
------------------------

>> (1+2i)/(3-4i)

out> -1|5+2|5i
------------------------

>> (1+2i)+1/2

out> 3|2+2i
------------------------
```

例 2.8.1 化简 $1+2i-3i^2+4i^3-5i^4+6i^5$.

```
>> 1+2i-3i^2+4i^3-5i^4+6i^5

out> -1+4i
------------------------
```

例 2.8.2 计算 $(1+2i)*(3-4i)-(1+2i)/(3-4i)$.

```
>> (1+2i)*(3-4i)-(1+2i)/(3-4i)

out> 56|5+8|5i
------------------------
```

例 2.8.3 计算 $\frac{1}{1+i}+\frac{1}{1-i}$ 的值.

```
>> 1/(1+i)+1/(1-i)
```

```
2
3 out> 1
4 ------------------------
```

2.9 极限

2.9.1 有理函数的极限

在 v0.600中, 我们实现了对有理函数求极限的功能. 加入了 `limit()` 函数. 例如要计算极限 $\lim\limits_{x\to x_0} f(x)$, 只需输入 `limit(f(x),x,x_0)` 或者 `lim(f(x),x,x_0)`.

`limit()`函数有三个参数, 第一个参数是有理函数的表达式 $f(x)$, 第二个参数是变量 `x`, 第三个参数是 x 趋向的点 x_0. 如果 x_0 是无穷远, 则输入 `inf` 或 `Inf` 或 `infinity` 或 `Infinity`.

例 2.9.1 计算极限 $\lim\limits_{x\to 1}\frac{x^3-1}{x-1}$.

```
1 >> limit((x^3-1)/(x-1),x,1)
2 result> 3
3
4 ------------------------
```

例 2.9.2 求 $\lim\limits_{x\to\infty}\frac{x^3+1}{x^2-2x+3}$.

```
1 >> lim((x^3+1)/(x^2-2x+3),x,Inf)
2 result> Inf
3
4 ------------------------
```

例 2.9.3 求 $\lim\limits_{t\to\infty}\frac{2t^2+t}{3t^2-2t+3}$.

```
1 >> :change x t
2
3 >> lim((2t^2+t)/(3t^2-2t+3),t,inf)
4 result> 2|3
5
6 ------------------------
```

2.10 导数

从v0.650 开始添加 `diff()` 函数, 用于对常见的函数进行求导. 下面是导数的计算公式.

2.10.1 导数的计算公式

1. $(C)' = 0$, C 为常数;

```
>> diff(C)
out> 0

------------------------
```

2. $(x^\alpha)' = \alpha x^{\alpha-1}$ $(\alpha \neq 0)$;

```
>> diff(x^alpha)
input> x^alpha

diff> x^alpha*alpha*1/x
 out> x^alpha*alpha*1/x
out>

------------------------
```

3. $(e^x)' = e^x$;

```
>> diff(e^x)
input> e^x

diff> e^x
 out> e^x
out>

------------------------

>> diff(exp(x))
input> exp[x]

diff> exp[x]
out>

------------------------
```

4. $(a^x)' = a^x \ln a$ $(a > 0,\ a \neq 1)$;

```
>> diff(a^x)
input> a^x

diff> a^x*ln[a]
 out> a^x*ln(a)
out>

------------------------
```

5. $(\ln|x|)' = \frac{1}{x}$;

这里暂不支持绝对值符号.

```
>> diff(ln(x))
input> ln[x]

diff> 1/x
```

```
 out> 1/x
out>

------------------------
```

6. $(\log_a |x|)' = \frac{1}{x \ln a}$ $a > 0,\ a \neq 1$;

```
>> diff(ln(x)/ln(a))
input> ln[x]/ln[a]

diff> 1/x*ln[a]/(ln[a]*ln[a])
 out> 1/x*ln(a)/(ln(a)*ln(a))
out>

------------------------
```

7. $(\sin x)' = \cos x$;

```
>> diff(sin(x))
input> sin[x]

diff> cos[x]
out>

------------------------
```

8. $(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$;

```
>> diff(arcsin(x))
input> arcsin[x]

diff> 1/sqrt[{1-x^2}]
 out> 1/sqrt((1-x^2)]
out>

------------------------
```

9. $(\cos x)' = -\sin x$;

```
>> diff(cos(x))
input> cos[x]

diff> -sin[x]
out>

------------------------
```

10. $(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$;

```
>> diff(arccos(x))
input> arccos[x]

diff> -1/sqrt[{1-x^2}]
 out> -1/sqrt((1-x^2)]
out>

------------------------
```

11. $(\sec x)' = \sec x \tan x$;

```
>> diff(sec(x))
input> sec[x]

diff> sec[x]*tan[x]
 out> sec(x)*tan(x)
out>

------------------------
```

12. $(\csc x)' = -\csc x \cot x$;

```
>> diff(csc(x))
input> csc[x]

diff> -csc[x]*cot[x]
 out> -csc(x)*cot(x)
out>

------------------------
```

13. $(\tan x)' = \sec^2 x$;

```
>> diff(tan(x))
input> tan[x]

diff> sec[x]^2
 out> sec(x)^2
out>

------------------------
```

14. $(\arctan x)' = \frac{1}{1+x^2}$;

```
>> diff(arctan(x))
input> arctan[x]

diff> 1/{1+x^2}
 out> 1/(1+x^2)
out>

------------------------
```

15. $(\cot x)' = -\csc^2 x$;

```
>> diff(cot(x))
input> cot[x]

diff> -csc[x]^2
 out> -csc(x)^2
out>

------------------------
```

16. $(\operatorname{arccot} x)' = -\frac{1}{1+x^2}$;

```
>> diff(arccot(x))
input> arccot[x]

diff> -1/{1+x^2}
 out> -1/(1+x^2)
out>

------------------------
```

17. $(\sinh x)' = \cosh x$;

```
>> diff(sinh(x))
input> sinh[x]

diff> cosh[x]
out>

------------------------
```

18. $(\cosh x)' = \sinh x$;

```
>> diff(cosh(x))
input> cosh[x]

diff> sinh[x]
out>

------------------------
```

19. $(\tanh x)' = 1 - (\tanh x)^2$;

```
>> diff(tanh(x))
input> tanh[x]

diff> 1-tanh[x]^2
 out> 1-tanh(x)^2
out>

------------------------
```

20. $(\coth x)' = 1 - (\coth x)^2$;

```
>> diff(coth(x))
input> coth[x]

diff> 1-coth[x]^2
 out> 1-coth(x)^2
out>

------------------------
```

2.10.2 多项式求导

Sowya 会首先判断输入的表达式(即 `diff()` 函数的参数)是否是多项式, 如果是多项式, 则使用多项式的求导算法计算.

例 2.10.1 求函数 x^3-2x^2+3x-5 的导数.

```
>> diff(x^3-2x^2+3x-5)
out> 3x^2-4x^1+3

------------------------
```

2.10.3 复合函数求导

以下例子中出现的函数 $f(x)$ 均假设可导.

例 2.10.2 求函数 $\sin(f(x))$ 的导数.

```
>> diff(sin(f(x)))
input> sin[f[x]]

diff> cos[f[x]]*f'[x]*1
 out> cos(f(x))*f'(x)*1
out>

------------------------
```

例 2.10.3 求函数 $\sin(2x)+\cos(g(x^2))$ 的导数.

```
>> diff(sin(2x)+cos(g(x^2)))
input> sin[2*x]+cos[g[x^2]]

diff> cos[2*x]*2-sin[g[x^2]]*g'[x^2]*x^2*2*1/x
 out> cos(2*x)*2-sin(g(x^2))*g'(x^2)*x^2*2*1/x
out>

------------------------
```

例 2.10.4 求函数 $f(g(x^2,x),x)$ 关于 x 的导数. 这里假设 f 和 g 的偏导数均存在.

```
>> diff(f(g(x^2,x),x))
input> f[g[x^2,x],x]

diff> f_1[g[x^2,x],x]*{g_1[x^2,x]*x^2*2*1/x+g_2[x^2,x]*1}+f_2[g[x^2,x],x]*1
out>

------------------------
```

2.11 批处理

从 v0.543 开始, 增加了从文件输入输出的功能. 如果要执行多条语句, 可以先将这些语句编辑到文本文件中. 例如 `test.sy`, 内容如下:

```
a=2
a+3*8
:mode=polyn
(-4x^3+x^2)+(3x-x^6)*2x
:mode=numerical
A=[1,2;
3,-2]
B=[1 2 3
4;3,2,0
1;0,2,0,9;1,3,5,
-1]
EulerVarphi(20221003)
hex2decimal(0xFF)
```

然后在命令符终端执行命令 `sowya.exe test.sy`, 结果将输出到文件 `test.out`.

注 2.11.1 要注意的是, `polyn` 模式下通常的计算会出错, 例如上面第五行去掉的话, `A` 和 `B` 的行列式输出为 `0`. 因此, 当不再使用 `polyn` 模式时, 要及时切换回来.

使用文本编辑器查看 `test.out`, 内容如下:

```
>> out>
out> 26
------------------------


>> Switch into polynomial mode.

>>
out> -2x^7-4x^3+7x^2
------------------------

>> Switch into numerical calculating mode.
e.g., 1/2+1/3 will return 0.83333333

>> input> [1,2;3,-2]
--------------------

1       2
3       -2

--------------------
>> input> [1,2,3,4;3,2,0,1;0,2,0,9;1,3,5,-1]
--------------------

1       2       3       4
3       2       0       1
0       2       0       9
1       3       5       -1

--------------------
>> 18354600
------------------------

>> out> 255

------------------------
```

使用下面的运行方式可以自己指定输入输出文件.

```
calculator.exe -f test.sy -o output.txt
```

这里 `-f` 指定输入文件, `-o` 指定输出文件. 输入输出文件都是文本文件, 后缀名无所谓. 这里输入文件的后缀名采用 `.sy` 是因为“数学”在吴语中的发音是 [sou ya].

例 2.11.1 计算 $(\arctan x)^2$ 的展开式.

当 $x \in [-1,1]$ 时, $\arctan x$ 可展开为

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1},$$

于是

$$\begin{aligned}
(\arctan x)^2 &= \left(x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots\right) \cdot \left(x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots\right) \\
&= x^2 - \frac{x^4}{3} + \frac{x^6}{5} - \frac{x^8}{7} + \frac{x^{10}}{9} - \frac{x^{12}}{11} + \cdots \\
&\quad - \frac{x^4}{3} + \frac{x^6}{9} - \frac{x^8}{3\cdot 5} + \frac{x^{10}}{3\cdot 7} - \frac{x^{12}}{3\cdot 9} + \cdots \\
&\quad + \frac{x^6}{5} - \frac{x^8}{5\cdot 3} + \frac{x^{10}}{5\cdot 5} - \frac{x^{12}}{5\cdot 7} + \cdots \\
&\quad - \frac{x^8}{7} + \frac{x^{10}}{7\cdot 3} - \frac{x^{12}}{7\cdot 5} + \cdots \\
&= x^2 - \frac{2}{3}x^4 + (\frac{1}{1\cdot 5} + \frac{1}{3\cdot 3} + \frac{1}{5\cdot 1})x^6 - (\frac{1}{1\cdot 7} + \frac{1}{3\cdot 5} + \frac{1}{5\cdot 3} + \frac{1}{7\cdot 1})x^8 \\
&\quad + (\frac{1}{1\cdot 9} + \frac{1}{3\cdot 7} + \frac{1}{5\cdot 5} + \frac{1}{7\cdot 3} + \frac{1}{9\cdot 1})x^{10} + \cdots
\end{aligned}$$

这里 x^{2m} 的系数为

$$\sum_{n=1}^{m} \frac{1}{(2n-1)(2m-(2n-1))}.$$

例如, 当 $m=4$ 时, x^8 的系数为

$$\sum_{n=1}^{4} \frac{1}{(2n-1)(9-2n)}.$$

利用 `sum()` 函数,

```
>> sum(1/((2*n-1)*(9-2*n)),n,1,4)
in> sum(1/((2*n-1)*(9-2*n)),n,1,4)
44|105
------------------------
```

将下面得代码保存到文件 `coeffs_of_arctan2.sy`,

```
:mode=fraction
sum(1/((2*n-1)*(7-2*n)),n,1,3)
sum(1/((2*n-1)*(9-2*n)),n,1,4)
sum(1/((2*n-1)*(11-2*n)),n,1,5)
```

```
sum(1/((2*n-1)*(13-2*n)),n,1,6)
sum(1/((2*n-1)*(15-2*n)),n,1,7)
sum(1/((2*n-1)*(17-2*n)),n,1,8)
sum(1/((2*n-1)*(19-2*n)),n,1,9)
sum(1/((2*n-1)*(21-2*n)),n,1,10)
sum(1/((2*n-1)*(23-2*n)),n,1,11)
sum(1/((2*n-1)*(25-2*n)),n,1,12)
sum(1/((2*n-1)*(27-2*n)),n,1,13)
sum(1/((2*n-1)*(29-2*n)),n,1,14)
sum(1/((2*n-1)*(31-2*n)),n,1,15)
sum(1/((2*n-1)*(33-2*n)),n,1,16)
sum(1/((2*n-1)*(35-2*n)),n,1,17)
```

然后在命令提示符窗口运行下面的命令

```
Sowya.exe -f coeffs_of_arctan2.sy -o coeffs_of_arctan2.txt
```

得到

```
>> :mode fraction
Switch into fraction calculating mode.
e.g., 1/2+1/3 will return 5/6

>> in> sum(1/((2*n-1)*(7-2*n)),n,1,3)
23|45                        (23 is a prime)
------------------------

>> in> sum(1/((2*n-1)*(9-2*n)),n,1,4)
44|105
------------------------

>> in> sum(1/((2*n-1)*(11-2*n)),n,1,5)
563|1575                    (563 is a prime)
------------------------

>> in> sum(1/((2*n-1)*(13-2*n)),n,1,6)
3254|10395
------------------------

>> in> sum(1/((2*n-1)*(15-2*n)),n,1,7)
88069|315315            (88069 is a prime)
------------------------

>> in> sum(1/((2*n-1)*(17-2*n)),n,1,8)
11384|45045
------------------------

>> in> sum(1/((2*n-1)*(19-2*n)),n,1,9)
1593269|6891885      (1593269 is a prime)
------------------------

>> in> sum(1/((2*n-1)*(21-2*n)),n,1,10)
15518938|72747675
------------------------

>> in> sum(1/((2*n-1)*(23-2*n)),n,1,11)
31730711|160044885            (31730711 is NOT a prime)
------------------------
```

```
40
41 >> in> sum(1/((2*n-1)*(25-2*n)),n,1,12)
42 186088972|1003917915
43 ------------------------
44
45 >> in> sum(1/((2*n-1)*(27-2*n)),n,1,13)
46 3788707301|21751554825
47 ------------------------
48
49 >> in> sum(1/((2*n-1)*(29-2*n)),n,1,14)
50 5776016314|35137127025
51 ------------------------
52
53 >> in> sum(1/((2*n-1)*(31-2*n)),n,1,15)
54 340028535787|2183521465125
55 ------------------------
56
57 >> in> sum(1/((2*n-1)*(33-2*n)),n,1,16)
58 667903294192|4512611027925
59 ------------------------
60
61 >> in> sum(1/((2*n-1)*(35-2*n)),n,1,17)
62 10823198495797|76714387474725
63 ------------------------
```

因此,

$$(\arctan x)^2 = x^2 - \frac{2}{3}x^4 + \frac{23}{45}x^6 - \frac{44}{105}x^8 + \frac{563}{1575}x^{10} - \frac{3254}{10395}x^{12} + \frac{88069}{315315}x^{14} - \frac{11384}{45045}x^{16} + \frac{1593269}{6891885}x^{18} + \cdots,$$

其中 $x \in [-1, 1]$.

clox语言 3

Sowya 内置了 Robert Nystrom 用 C 语言写的解释器 lox, 我们在其基础上增加了一些功能, 因此将其命名为 clox.

注 3.0.1 关于 Robert Nystrom 的解释器 lox, 可参见 https://craftinginterpreters.com/.

使用命令 `:mode clox` 进入编程模式, 此时会出现 `clox` 的提示符 `>` . 键入 `:q` 并回车即退出编程模式, 回到 `Sowya` 环境.

当进入编程模式后, Sowya 会自动加载位于 `ext` 目录下的所有以 `.sf` 为后缀的文件. 例如:

```
>> :mode clox
load ext/fibo.sf
load ext/printMultiOrders.sf
load ext/q1.sf
>
```

一般可将常用的自定义函数存放在这里. 以 `fibo.sf` 为例, 打开此文件, 其内容是

```
fun fibo(n)
{
  if(n<2) return n;
  return fibo(n-1)+fibo(n-2);
}
```

这是计算斐波那契数列的递归解法, 当然速度会非常慢. 例如打印斐波那契数列的第9项,

```
> print fibo(9);
34>
```

3.1 采用双精度类型

将上面的代码改成如下, 即数字后面加上 `d`, 则表示采用双精度类型(double).

```
fun f(n)
{
  if(n<2d) return 1;
  return f(n-1d)+f(n-2d);
}

println f(1d);
println f(2d);
println f(3d);
println f(4d);
println f(5d);
println f(6d);
println f(7d);
println f(8d);
println f(9d);
```

```
16 println f(10d);
```

假设文件保存到文件 `code\fibo.sy`. 则在 Sowya 的提示符 `>>` 下输入 `:mode clox=code\fibo.sy` 并回车就会输出结果.

```
1  >> :mode clox=code\fibo.sy
2  load ext/fibo.sf
3  load ext/printMultiOrders.sf
4  load ext/q1.sf
5  1
6  2
7  3
8  5
9  8
10 13
11 21
12 34
13 55
14 89
15
16 >>
```

注 3.1.1 ▶ 使用double类型的数进行计算要比Sowya内置的大数计算要快, 因为目前大数计算采用的是字符串类型存储.

▶ 尽量不使用递归求解, 因为递归会消耗很多内存, 效率比较低. 例如这里求解斐波那契数列, 不如Sowya内置的函数 `fibonacci(n)`来得快.

3.2 scripts

`clox` 中可以直接执行一些语句, 这些语句被称为 `scripts`. `Sowya` 中的一些命令等亦可以认为是 `scripts`.

例如, 最简单的打印 `Hello Sowya!`. 我们可以使用 `print` 指令或者 `println` 指令, 两者的功能基本相同, `println` 会在最后打印换行符.

```
1 > print "Hello Sowya!";
2 Hello Sowya!>
3
4 > println "Hello Sowya!";
5 Hello Sowya!
6 >
```

每条语句最后要加分号 `;` 作为结束. 字符串一般用双引号 `" "` 括起来. 但为了与 `Sowya` 更好对接, 在 `clox` 中直接输入的数(即不适用双引号界定的数)也被认为是字符串.

```
1 > println 2;
2 2
3 > println 2+3; // try 2-3, 2*3, 2/3
4 5
```

`print` 和 `println` 可以一次打印多项, 例如:

```
> print 2+3, 2-3, 2*3, 2/3;
5-160.66666667>
```

但是这样的输出结果挨在一起会导致分不清, 改进如下:

```
> print 2+3, "\t", 2-3, "\t", 2*3, "\t", 2/3, "\n";
5       -1      6       0.66666667
>
```

这里 `\t` 和 `\n` 分别是制表符和换行符.

3.3 变量声明

使用关键字 `var` 可以声明一个或多个变量. 例如声明变量 `a` 并赋值 `2`.

```
> var a=2;
```

可以一次声明多个变量, 中间用逗号隔开

```
> var a=3, b=4, c=a*b, e;
```

这里声明了变量 `a`, `b` 并分别赋值 `3` 和 `4`. `c` 声明为 `a*b`, 值为 `12`. 变量 `e` 并未赋值. 我们用 `println` 检验一下.

```
> println "a=",a,"\n","b=",b,"\n","c=",c,"\n","e=",e,"\n";
a=3
b=4
c=12
e=nil

>
```

`nil` 是 `clox` 中预定义的值, 等同于 C 语言中的 `NULL`.

3.4 函数

clox中的函数分为内置函数(Native function)和自定义函数.

3.4.1 内置函数

avg()

clox可以在编译前添加内置函数(Native function) , 这里我们加入了内置函数 `avg()`, 它可以接受有限多个参数.

例 3.4.1 计算 12, 3, 4, 5, 98 这五个数的平均值.

```
> println avg(12,3,4,5,98);
24.4
>
```

当加载与其同名的自定义函数 `avg()` 后, 内置函数就不起作用了.

ith_char()

例 3.4.2 使用内置函数 `ith_char()` 打印字符串 `Hello Sowya!` 的第 `i` 个字符. 这里 i 是从 `0` 开始的.

```
>> :mode clox
> var s="Hello Sowya!";
> print ith_char(s,8);
w>
```

注意, 第0个字符是 `H`.

isprime()

v0.601中在 `clox` 中添加了 `isprime()` 函数, 调用 Sowya 中同名的函数 `isprime()`. 该函数使用传统算法用于判断某个正整数是否是素数.

例 3.4.3 在clox模式下判断 89208029 是否是素数.

```
>> :mode clox

> print isprime(89208029);
true>
```

length()

`length()` 函数用于返回字符串的长度. 不过这里的字符串必须要用双引号括起来.

```
> println length("hello Sowya!");
12
```

3.4.2 自定义函数

例 3.4.4 打印 9×10^{17} 到 $9\times10^{17}+9$ 的平方.

首先进入clox编程模式,

```
>> :mode clox
>
```

然后定义函数, 不妨命名为 `echo()`, 函数以关键字 `fun` 开头, 后接函数名, 然后是一对圆括号, 括号中是参数列表, 参数与参数之间使用逗号分隔. 最后是以`{}`界定的函数体. 如果函数不复杂, 可以将函数的代码写在一行上.

```
> fun echo(){var n=9*10^17; while(n<9*10^17+10){println n*n; n=n+1;}}
```

输入 `echo();` 执行该函数.

```
> echo();
810000000000000000000000000000000000
810000000000000001800000000000000001
810000000000000003600000000000000004
810000000000000005400000000000000009
810000000000000007200000000000000016
810000000000000009000000000000000025
810000000000000010800000000000000036
810000000000000012600000000000000049
810000000000000014400000000000000064
810000000000000016200000000000000081
>
```

3.5 多行输入

在写程序时, 经常需要多行输入, 则第一个字符以 `{` 开始, 即可进入多行输入, 最后只要以 `}` 结尾即可.

例 3.5.1 打印数列 $\{2^n\}$, $n = 9, 10, 11, \ldots, 19$.

```
> {
fun echo(){
  var n=9;
  while(n<20){
    println 2^n;
    n=n+1;
  }
 }
}
> echo();
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
>
```

这和 Sowya 下执行 `printSeries(2^n,n,9,19,\n)` 得到的结果是一致的.

```
>> printSeries(2^n,n,9,19,\n)
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288


------------------------
```

例 3.5.2 对于 $m = 2^n, n = 1, 2, \ldots, 99$, 计算 m 在十进制下的长度.

```
>> :mode clox

> {
fun printLength(n){
  while(n<100){
    var m;
    m=2^n;
    print length(m),", ";
    n=n+1;
  }
}
}
> printLength(1);
1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 9, 9, 9, 10, 10, 10, 10, 11, 11, 11, 12,
      12, 12, 13, 13, 13, 13, 14, 14, 14, 15, 15, 15, 16, 16, 16, 16, 17, 17, 17, 18, 18, 18, 19, 19, 19, 19, 20, 20,
     20, 21, 21, 21, 22, 22, 22, 22, 23, 23, 23, 24, 24, 24, 25, 25, 25, 25, 26, 26, 26, 27, 27, 27, 28, 28, 28, 28,
     29, 29, 29, 30, 30, 30, >
```

3.6 导入外部文件

v0.558 版本增加了导入外部文件的功能, 比如写一个最简单的函数 `avg()`, 用于求两个数的均值. 将下面的代码编辑至 `code\avg.sy` 中.

```
fun avg(a,b){
  return (a+b)/2;
}
```

进入clox编程模式, 并使用 `load(path_to_file)` 加载含有上面函数的文件.

```
>> :mode clox
> load(code\avg.sy)
```

然后执行avg(), 该函数也可以接受变量.

```
> println avg(1,2);
1.5
```

```
> var x=31, y=27;
> println avg(x,y);
29
>
```

注意此时再执行 `println avg(12,3,4,5,98);` 会出现错误.

```
> println avg(12,3,4,5,98);
Expected 2 arguments but got 5.
[line 1] in script
```

因为这里自定义的 `avg()` 函数代替了内置的函数 `avg()`. 若要使用内置的 `avg()` 函数, 只需按 `:q` 退出 `clox` 模式, 然后再进入即可.

3.6.1 自动导入

自v0.579版本, Sowya.exe 在进入 `clox` 模式后, 会自动加载 `ext` 目录下所有 `.sf` 文件. 如果 Sowya.exe 所在目录没有 `ext` 文件夹, 则在 `:mode clox` 后会自动创建此目录.

建议将常用的函数写成 `.sf` 文件, 置于 `ext` 文件夹中.

数列4

4.1 斐波那契数列

在内置函数中, `fibonacci(n)` 或 `Fibonacci(n)` 函数用于计算第 n 个斐波那契数 F_n. 其满足递推公式 $F_n = F_{n-1} + F_{n-2}$, 这里规定 $F_0 = 0$.

```
 >> fibonacci(19)
in> fibonacci(19)

F(19) = 4181

------------------------
```

`fibonacci()` 函数可以加第二个参数 `true` 或 `t` 或 `yes` 或 `y`.

例如 `fibonacci(n,true)` 将列出前 $n+1$ 个斐波那契数.

```
>> fibonacci(19,true)
in> fibonacci(19,true)

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,

------------------------

F(19) = 4181

------------------------
```

例: 计算 F_{-5}.

```
>> fibonacci(-5)
in> fibonacci(0-5)

F(-5) = 5

------------------------
```

4.2 数列的应用

4.2.1 迭代打印字符串

前面提到的`printRecursiveSeries()`函数也可以迭代打印一些字符串.

```
>> printRecursiveSeries(8x9,x,1,10,\n)
in> printRecursiveSeries(8x9,x,1,10,\n)
1
819
88199
```

```
8881999
888819999
88888199999
8888881999999
888888819999999
88888888199999999
8888888881999999999


------------------------
```

```
>> printRecursiveSeries(o-o,o,x,6,\n)
x
x-x
x-x-x+x
x-x-x+x-x+x+x-x
x-x-x+x-x+x+x-x-x+x+x-x+x-x-x+x
x-x-x+x-x+x+x-x-x+x+x-x+x-x-x+x+x-x+x+x-x+x-x-x+x+x-x+x+x-x+x+x-x
```

现在需要打印十组 x 和 -x, 且交替出现.

```
>> printRecursiveSeries(0-o,o,x,20)
x,-x,x,-x,x,-x,x,-x,x,-x,x,-x,x,-x,x,-x,x,-x,x,-x,x,-x,x,-x,x,-x,
```

```
>> printRecursiveSeries(3*7,7,8,5,\t)
24      24      24      24      24

------------------------

>> printRecursiveSeries(3*7,7,8,5,\t,plain)
3*8     3*8     3*8     3*8     3*8
```

打印 9 行星号, 第i行的星号个数是 i.

```
>> printRecursiveSeries(y*,y, ,10,\n,plain)

*
**
***
****
*****
******
*******
********
*********
```

```
>> printRecursiveSeries(yL//>,y, ,10,\n,plain)

L//>
L//>L//>
L//>L//>L//>
L//>L//>L//>L//>
L//>L//>L//>L//>L//>
L//>L//>L//>L//>L//>L//>
L//>L//>L//>L//>L//>L//>L//>
L//>L//>L//>L//>L//>L//>L//>L//>
L//>L//>L//>L//>L//>L//>L//>L//>L//>
```

```
>> printRecursiveSeries( /<x--/*x*\--x>\ ,x,1,3,\n)
in> printRecursiveSeries(/<x--/*x*\--x>\,x,1,3,\n)
1
/<1--/*1*\--1>\
/</<1--/*1*\--1>\--/*/<1--/*1*\--1>\*\--/<1--/*1*\--1>\>\
```

4.2.2 应用递推公式计算定积分

例如: 求 $I_n = \int_0^1 \frac{x^n}{x+5}\mathrm{d}x, n = 1, 2, \ldots, 100.$

易见 I_n 满足递推公式 $I_n = \frac{1}{n} - 5I_{n-1}$. (参见问题3081.) 先求初始值 I_0,

$$I_0 = \int_0^1 \frac{1}{x+5}\mathrm{d}x = \ln(x+5)\Big|_{x=0}^{x=1} = \ln 1.2.$$

值得注意的是, I_0 的精确性影响着序列 $\{I_n\}_{n=0}^{+\infty}$ 的准确性.

另外, Sowya 设置的计算精度也应和 I_0 的一致. 例如, 若取

$$I_0 = 1 - 5\ln 1.2 \approx 0.0883922160302268689414$$

在默认精度(小数点后8位)下, 计算到第13项便开始出现负值. 初始值的精度低会导致误差在递推公式计算过程中迅速积累, 以致于误差呈指数级增长.

```
>> printRecursiveSeries(-5*I_k+1/(k+1),I_k,0.0883922160302268689414,20,\n,linenumber)
in> printRecursiveSeries(0-5*I_k+1/(k+1),I_k,0.0883922160302268689414,20,\n,linenumber)
[1]     0.0883922160302268689414
[2]     0.058038919848865655293O
[3]     0.043138730755671723535O
[4]     0.034306346221641382325O
[5]     0.028468268891793088375O
[6]     0.024325325541034558125O
[7]     0.021230512294827209375O
[8]     0.018847438525863953125O
[9]     0.016873917370680234375O
[10]    0.015630413146598828125O
[11]    0.012757024267005859375O
[12]    0.019548208664970703125O
[13]    -0.020817963324853515625O
[14]    0.175518386624267578125O
[15]    -0.810925263121337890625O
[16]    4.117126315606689453125O
[17]    -20.526808048033447265625O
[18]    102.689595800167236328125O
[19]    -513.395347420836181640625O
[20]    2567.026737104180908203125O


-----------------------
```

若要得到该序列前100项的相对精确值, 比如精度到小数点后100位, 那么先设置计算精度为100.

```
>> setprecision(100)
```

再计算 ln(1.2).

```
>> ln(1.2)
in> ln(1.2)
out> 0.18232155679395462621171802515451463319738933791448698394272645165670892748064591784934520371697116 58
```

将其赋给某个变量, 比如a

```
>> a=0.18232155679395462621171802515451463319738933791448698394272645165670892748064591784934520371697116 58
```

然后计算 $I_1 = \frac{1}{1} - 5I_0 = 1 - 5a$.

```
>> 1-5*a
in> 1-5*0.18232155679395462621171802515451463319738933791448698394272645165670892748064591784934520371697116 58

out> 0.08839221603022686894140987422742683401305331042756508028636774171645536259677041075327398141514417 10
```

最后使用 printRecursiveSeries()函数计算 I_n,

```
>> printRecursiveSeries(1/(n+1)-5*I_n,I_n
    ,0.08839221603022686894140987422742683401305331042756508028636774171645536259677041075327398141514417 10,100,\n,
    linenumber)
```

得到 I_{98}, I_{99}, I_{100} 的近似值如下.

```
[98]    0.00168631682551537898265685240458495576859384724988048932127593542122183589289812295732900081443073 16
[99]    0.00166942597343320609681674807808532216713177385160765440372133299490092154561039531436509693794735 21
[100]   0.00165287013283396951591625960957338916434113074196172798139333502549539227194802342817451531026323 95
```

习题 4.2.1 计算积分 $I_n = \int_0^1 x^n e^{x-1} \mathrm{d}x, n = 0, 1, 2, \ldots$.

4.3 $\ln(x)$ 的计算

https://zhuanlan.zhihu.com/p/545984671

编程 5

按 `:mode clox` 进入编程模式.

5.1 圆周率的计算

5.2 二分法

例 5.2.1 使用二分法求解方程 $x^3+x-4=0$.

设 $f(x)=x^3+x-4$, 则 $f(1)=-2<0, f(2)=6>0$. 于是 f 在 $(1,2)$ 上至少有一个根. $f'(x)=3x^2+1>0$. 因此函数 f 在 R 上是严格单调递增的. 故 f 在 $(1,2)$ 上有惟一的实根.

将下面的代码保存到文本文件中, 比如`D:/Sowya/code/bisect_8.1.txt`.

键入 `load("D:/Sowya/code/bisect_8.1.txt")` 加载此文件中的代码, 然后执行 `print root_of_f(1,2)` 即可.

```
/* Solve the equation x^3+x-4=0 by using binary searching method.
* f(x)=x^3+x-4, we have f(1)=-2<0, f(2)=6>0. So f has at least one root * in the interval (1,2).
* f'(x)=3x^2+1 >0. So f is strictly increasing on R. Therefore f has the only one root in (1,2).
*/
setprecision(10);

fun root_of_f(a,b)
{
  var fa, fb;//value of f(a) and f(b)
  var sign_fa, sign_fb;//sign of f(a) and f(b)
  fa=f(a);
  fb=f(b);
  sign_fa=sign(fa);
  sign_fb=sign(fb);

  if(sign_fa==0) return a;
  if(sign_fb==0) return b;

  if(sign_fa==sign_fb) return "error> function f has the same sign at the two end points.";

  var mid, fmid, sign_fmid;
  var eps=0.00000001;
  var cnt=0;
  while(1)
  {
    mid=(a+b)/2;
    cnt=cnt+1;
    fmid=f(mid);
    //println "a=",a," b=",b," mid=",mid;
    //println "fmid=",fmid, " ", eps;
    //println "-----------";
    sign_fmid=sign(fmid);
```

```
    if(sign_fmid==0)
    {
      println "cnt=",cnt;
      return mid;
    }

    fmid=abs(fmid);

    if(fmid<eps)
    {
      println "cnt=",cnt;
      return mid;
    }

    if(sign_fmid==sign_fa)
    {
      a=mid;
    }
    if(sign_fmid==sign_fb)
    {
      b=mid;
    }
  }
}

fun f(x)
{
    return x^3+x-4;
}

fun abs(x)
{
    if(x>=0) return x;
    else return (-1)*x;
}

fun sign(a)
{
    if(a>0){
        return 1;
    }else if(a<0){
        return -1;
    }else{
        return 0;
    }
}
```

程序中会判断函数 f 在两个端点处的符号是否相同, 如果相同则提示错误.

```
>> :mode clox

> load(code/BinarySearch_8.1.txt)

> print f(1);
-2> print f(2);
6> print root_of_f(1,2);
cnt=28
1.3787967006>
```

验证

```
> print f(1.3787967006);
0.000000003153533973427702436216>
```

5.3 迭代法

5.4 计算极限

例 5.4.1 求极限 $\lim_{n\to\infty}\prod_{k=2}^{n}\cos(\frac{\pi}{2^k})$.

将以下代码保存到文件 prod_cos.txt,

```
/*
* \lim_{n\rightarrow\infty}\prod_{k=2}^{n}\cos(\frac{\pi}{2^k})
*/
setprecision(100);
fun F(n){
  var A=1;
  var pi=3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 79;
  var t=1;
  for(var k=2; k<=n; k=k+1){
    t=2^k;
    A=A*cos(pi/t);
    truncate(A,120d);
  }
  return A;
}
```

这里的truncate(A,120d);是将每次计算得到的A保留小数点后120位数据.

启动 Sowya, 键入 :mode clox 进入编程模式（提示符为>）.

```
>> :mode clox

>
```

如果上面保存的文件位于 D:\Sowya\code 目录, 则输入load(code/prod_cos.txt)函数加载文件中的内容.

```
> load(D:/Sowya/code/prod_cos.txt)
>
```

然后输入print F(35);

```
> print F(35);
0.6366197723675813430764220643182049883954788220899928991272638364627387967669521231524573424544335834198496110636787696 35
```

运行print F(100); 等, 得

```
> print F(100);
0.6366197723675813430755350534900574481378385829618257949906700279082231940571953099570741477348384983469830312846171184 63>

> print F(200);
0.6366197723675813430755350534900574481378385829618257949906693762355871905369061403604552110650123442262665540364167561 18>

> print F(300);
0.6366197723675813430755350534900574481378385829618257949906693762355871905369061403604552110650123442262665540364167561 18>
```

线性规划 6

6.1 单纯形法

Sowya 在 0.628版本中加入了线性规划求解功能. 我们以例子说明使用方法.

例 6.1.1 使用单纯形法求解下面的带约束的优化问题.

$$\begin{aligned} \max z &= 5x_1 + 3x_2, \\ -2x_1 + x_2 &\leqslant 5, \\ x_1 + x_2 &\leqslant 7, \\ x_1 + 2x_2 &\leqslant 10, \\ x_1 &\geqslant 0, \\ x_2 &\geqslant 0; \end{aligned}$$

在 Sowya 中如下输入

```
>> max z=5x_1+3x_2,
-2x_1+x_2<=5,
x_1+x_2<=7,
x_1+2x_2<=10,
x_1>=0,x_2>=0;
```

注意目标函数、约束条件之间用逗号隔开, 最后使用分号结束. 按回车后得到如下结果.

```
--------------------
objModel> max
zname> z
objectiveFunction> 5x_1+3x_2
constraints>
[1] -2x_1+x_2<=5
[2] x_1+x_2<=7
[3] x_1+2x_2<=10
[4] x_1>=0
[5] x_2>=0
--------------------

======================
[Analysis]
[basic_vars]: x_1 x_2
The obj polynomial is Linear: 5x_1+3x_2
[1] -2x_1+x_2
[2] x_1+x_2
[3] x_1+2x_2

------Matrix--A----
        z       x_1     x_2     s_1     s_2     s_3     Solution
z       1       -5      -3      0       0       0       0
s_1     0       -2      +1      1       0       0       5
```

```
s_2     0       1       +1      0       1       0       7
s_3     0       1       +2      0       0       1       10

-------------------

====== Iteration [1] ======

Step 1.1
Find the minimun value of A[0]: -5
In_var: x_1

Step 1.2
pivot_element = A(2,2) = 1

------Matrix--A----
        z       x_1     x_2     s_1     s_2     s_3     Solution
z       1       -5      -3      0       0       0       0
s_1     0       -2      +1      1       0       0       5
s_2     0       1       +1      0       1       0       7
s_3     0       1       +2      0       0       1       10

-------------------
r_2/1 ==>

------Matrix--A----
        z       x_1     x_2     s_1     s_2     s_3     Solution
z       1       -5      -3      0       0       0       0
s_1     0       -2      +1      1       0       0       5
x_1     0       1       1       0       1       0       7
s_3     0       1       +2      0       0       1       10

-------------------
Perform elementary row transformations: r_i-A(i,2)* r_2

------Matrix--A----
        z       x_1     x_2     s_1     s_2     s_3     Solution
z       1       0       2       0       5       0       35
s_1     0       0       113     1       2       0       19
x_1     0       1       1       0       1       0       7
s_3     0       0       +1      0       -1      1       3

-------------------
When x_1 = 7, x_2 = 0, s_1 = 19, s_2 = 0, s_3 = 3,
z = 35

====== END ======
```

在计算过程中会给出简单的分析（见[Analysis]部分），比如基变量为 `x_1,x_2`. 目标多项式是线性的.

`In_var` 指进基变量. `pivot_element` 指枢纽元.

游戏 7

7.1 算廿四

算廿四是一个扑克牌游戏, 其规则大家耳熟能详. 一副扑克牌, 去掉大小王, 将 A(即Ace)视为 `1` 或 `11`; `2,3,...,10` 按牌面数字取值; `J, Q, K` 都算作 `10`. 随意抽取四张牌, 如果谁能用加减乘除先算得 `24`, 谁就胜.

例如, 如果抽到 `8, 3, 1, 9` 四张牌, 那么我们可以这样算得 `24`.

```
(9/3)*8*1
```

现在使用内置的 eq24() 函数, 可以轻松得到所有可能的计算表达式.

```
    >> eq24(8,3,1,9)
    in> eq24(8,3,1,9)

    8*1*9/3==24
    (8*9)/(3*1)==24
    3/(9/8-1)==24

    ------------------------

    Total: 3
```

在 v0.574 版本中, 我们扩展了 `eq24()` 函数的功能, 使其不仅能算 24, 还能算其他数值. 用法是 `eq24(a,b,c,d;h)`, 或者等价地使用别名 `suo` 或 `suo24`. 即输入 `suo(a,b,c,d;h)` 或 `suo24(a,b,c,d;h)`.

函数将 `a,b,c,d` 经过四则运算且等于 `h` 的所有等式输出, 其中 `a,b,c,d` 各使用一次. 若只输入四个参数, 比如 `suo(a,b,c,d)`, 则等同于 `suo(a,b,c,d;24)`.

7.2 端午节问题

下面的问题是在 2019年的端午节想到的, 故称为端午节问题.

问题 7.2.1 给定正整数 N, 将其作素因子分解, 各因子从小到大排序后拼接成一个正整数, 如果不是素数则再作素因子分解并将各因子按从小到大的顺序拼接成正整数. 如此进行下去, 问是否对每个正整数, 在有限步骤后终止?

两个正整数的拼接是指将这两个正整数作为字符串进行拼接, 所得仍是正整数. 例如: 若 $a = 23, b = 37$, 则 a 和 b 的“拼接”得到正整数 2337, 记为 $a.b$.

假设给定的正整数为 N, 这个游戏的过程如下:

- (1) 若 N 是素数, 则终止; 否则转到第(2)步;

- (2) 将 N 作素因子分解, 记为 $N = p_{i_1}p_{i_2}\cdots p_{i_m}$. 这里 p_{i_j} 都是素数, 且 $p_{i_1} \leqslant p_{i_2} \leqslant \cdots \leqslant p_{i_m}$;
- (3) 令 $N_1 = p_{i_1}.p_{i_2}.\ldots.p_{i_m}$. 回到第(1)步.

```
>> q_duanwu(201967)
201967==139*1453
1391453==7*7*73*389
7773389==13*701*853
13701853==11*1245623
111245623==4421*25163
442125163==442125163

out> 6

------------------------
```

若在因子分解后将各因子相加, 然后再作分解. 即上述游戏的过程更改为如下:

- (1) 若 N 是素数, 则终止; 否则转到第(2)步;
- (2) 将 N 作素因子分解, 记为 $N = p_{i_1}p_{i_2}\cdots p_{i_m}$. 这里 p_{i_j} 都是素数, 且 $p_{i_1} \leqslant p_{i_2} \leqslant \cdots \leqslant p_{i_m}$;
- (3) 令 $N_1 = p_{i_1} + p_{i_2} + \cdots + p_{i_m}$. 回到第(1)步.

这个使用 `q_DuanwuPlus(N)` 求解.

```
>> q_DuanwuPlus(201967)
201967==139*1453
139+1453==2*2*2*199
2+2+2+199==5*41
5+41==2*23
2+23==5*5
5+5==2*5
2+5==7

out> 7

------------------------
```

8 内置函数

Sowya 内置了一些有用的函数. 使用 `listfunctions()` 或 `listfunctions(1)` 列出所有内置函数. 后者(带参数1)将列出函数的具体签名. 可以使用 help(函数名) 列出此函数的具体用法.

```
>> help(listfunctions)
out> List all functions which have been registered in system.
If the bool==1, then display the prototype of the function.

Usage:
listfunctions(bool)
```

SowyaApp.exe 程序提供了帮助文档.

目前 Sowya 0.636 Pro 版本提供了 126 个函数.

8.1 A

8.1.1 any2decimal

将任意 p-进制数转为十进制数. 见2.2.10.

8.2 B

8.2.1 biggest_factor

`biggest_factor(m)` 返回小于等于正整数 m 的最大素因子.

```
>> biggest_factor(2025)
5
-----------------------

>> biggest_factor(2017)
2017
-----------------------
```

8.2.2 binary2decimal

`binary2decimal(n)` 将二进制数转成十进制形式.

单参数 `n` 为二进制数.

```
in> binary2decimal(100110100100010111001010 0)
out> 20220820
```

8.2.3 binary2graycode

`binary2graycode()` 将二进制码转换成格雷码.

8.2.4 binom

`binom(n,k)` 是组合数 $\binom{n}{k}$ 即 $C_n^k = \frac{n!}{k!(n-k)!}$.

8.3 C

8.3.1 ceil

`ceil(x)` 函数返回大于等于 x 的最小整数. 因此

$$\texttt{ceil(x)} = \begin{cases} [x], & \text{若 } x\text{是整数,} \\ [x]+1, & \text{否则.} \end{cases}$$

8.3.2 continued_fraction

`continued_fraction()` 是计算连分数的函数. 例如黄金分割数 $\phi = \frac{1+\sqrt{5}}{2}$ 可表示为

$$[1;1,1,\ldots] = 1 + \frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\cdots}}}}.$$

若记 $a_0 = 1, a_n = 1 + \frac{1}{a_{n-1}}, \forall\, n \geqslant 1$. 则

$$a_1 = 1 + \frac{1}{a_0} = 2, \quad a_2 = 1 + \frac{1}{a_1} = \frac{3}{2}, \quad a_3 = 1 + \frac{1}{a_2} = \frac{5}{3}, \ldots$$

$\phi = \lim\limits_{n\to\infty} a_n$. 易见 $a_1 = [1;1], a_2 = [1;1,1], a_3 = [1;1,1,1]$. 若要计算 a_4, 则

```
>> continued_fraction(1,1,1,1,1)
out> 8|5

Expression:
1+1/(1+1/(1+1/(1+1/(1))))

TeX Code:
1+\frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1}}}}

------------------------
```

8.3.3 CofactorMatrix

v.570版本加入了`CofactorMatrix(A,i,j)` 函数, 它返回矩阵 A 在 (i,j) 处(即元素 a_{ij})的余子式矩阵. 所谓的余子式矩阵, 即矩阵 A 划去第 i 行和第 j 列后剩余元素所组成的矩阵.

例. 设矩阵

$$A=\begin{pmatrix}1&2&3&4\\2&3&4&1\\3&4&1&2\\4&1&2&3\end{pmatrix}$$

求 A 在 $(2,3)$ 处元素的余子式矩阵.

```
>> A=[1 2 3 4;
A=[1 2 3 4;
2 3 4 1;
3 4 1 2;
4 1 2 3]
A=[1 2 3 4;
input> [1,2,3,4;2,3,4,1;3,4,1,2;4,1,2,3]
--------------------

1       2       3       4
2       3       4       1
3       4       1       2
4       1       2       3

--------------------
>> CofactorMatrix(A,2,3)
cofactor matrix at (2,3) :

1       2       4
3       4       2
4       1       3

>>
```

8.3.4 Collatz

Collatz 问题(或 $3x+1$ 猜想)是指对任意给定的正整数 x, 若是奇数, 则将其乘以 3 后加 1; 若是偶数, 则除以 2; 在经过有限步骤后, 必终止于数字 1. 这里定义了函数

$$f(x)=\begin{cases}3x+1, & 若x是奇数,\\ \frac{x}{2}, & 否则.\end{cases}$$

Collatz(n) 函数就是将这个到 1 的过程打印出来的函数. 例如,

```
>> Collatz(137)
in> Collatz(137)
137-->412-->206-->103-->310-->155-->466-->233-->700-->350-->175-->526-->263-->790-->395
-->1186-->593-->1780-->890-->445-->1336-->668-->334-->167-->502-->251-->754-->377-->1132
-->566-->283-->850-->425-->1276-->638-->319-->958-->479-->1438-->719-->2158-->1079
```

```
-->3238-->1619-->4858-->2429-->7288-->3644-->1822-->911-->2734-->1367-->4102-->2051
-->6154-->3077-->9232-->4616-->2308-->1154-->577-->1732-->866-->433-->1300-->650-->325
-->976-->488-->244-->122-->61-->184-->92-->46-->23-->70-->35-->106-->53-->160-->80-->40
-->20-->10-->5-->16-->8-->4-->2-->1

out> 90

------------------------
```

返回的 90 是指计算的步骤数.

8.3.5 continued_fraction

`continued_fraction(num,expression,detail,tex)` 将小数或分数转换为连分数, 也可以计算连分数的值.

例如: 将`677/263`写为连分数. 则可以输入:

```
>> continued_fraction(677/263)

out> (2,1,1,2,1,6,1,4)

------------------------
```

这表示

$$2+1/(1+1/(1+1/(2+1/(1+1/(6+1/(1+1/(4)))))))$$

或

$$2+\frac{1}{1+\frac{1}{1+\frac{1}{2+\frac{1}{1+\frac{1}{6+\frac{1}{1+\frac{1}{4}}}}}}}$$

上面的 LaTeX代码也可以由`continued_fraction()`生成.

```
>> continued_fraction(2,1,1,2,1,6,1,4)
out> 677|263

Expression:
2+1/(1+1/(1+1/(2+1/(1+1/(6+1/(1+1/(4)))))))

TeX Code:
2+\frac{1}{1+\frac{1}{1+\frac{1}{2+\frac{1}{1+\frac{1}{6+\frac{1}{1+\frac{1}{4}}}}}}}

------------------------
```

例. 计算连分数`(3,5,1,1,2)`.

```
>> continued_fraction(3,5,1,1,2)
out> 89|28

Expression:
3+1/(5+1/(1+1/(1+1/(2))))

TeX Code:
```

```
3+\frac{1}{5+\frac{1}{1+\frac{1}{1+\frac{1}{2}}}}

------------------------
```

更多用法请尝试:

```
>> continued_fraction(2,4,...,11)
>> continued_fraction(1,2,4,...,10)
```

8.3.6 cos

余弦函数`cos(x)`, 用法同`sin(x)`. 这里`x`默认使用弧度, 若输入的是度单位, 则加上字母`d`.

例如: `cos(1)`, `cos(1.5pi)`, `cos(3/2pi)`, `cos(30d)` .

```
>> cos(1)
out> 0.54030230

------------------------

>> cos(1.5pi)
out> 0

------------------------

>> cos(3/2pi)
out> 0

------------------------

>> cos(30d)
out> sqrt(3)/2

------------------------
```

8.3.7 cot

余切函数 `cot(x)`, 用法同`sin(x)`. 这里`x`默认使用弧度, 若输入的是度单位, 则加上字母`d`.

例如: `cot(1)`, `cot(1.5pi)`, `cot(1/6pi)` 或 `cot(30d)`.

```
>> cot(1)
0.64209261
------------------------

>> cot(1.5pi)
out> -0

------------------------

>> cot(1/6pi)
out> +sqrt(3)/2/1/2

```

```
----------------------

>> cot(30d)
+sqrt(3)/2/1/2
----------------------
```

注 8.3.1 ▶ 三角函数(sin(),cos(),tan(),cot())的参数, 如果数字后面加d, 则表示单位是度(°). 在其他表达式中d表示数是double类型.

▶ 内部计算采用了公式 $\cot(x) = \frac{\cos(x)}{\sin(x)}$. 后续将进一步化简所得到的值.

8.3.8 cubicsum

cubicsum(max,n) 函数用于查找不定方程 $x^3 + y^3 + z^3 = n$ 的整数解. 这里 $x, y, z \in [-max, max]$.

例 8.3.1 求不定方程 $x^3 + y^3 + z^3 = 10$ 的整数解, 其中 $x, y, z \in [-10, 10]$.

```
>> cubicsum(10,10)
1^3+(1)^3+(2)^3
1^3+(2)^3+(1)^3
2^3+(1)^3+(1)^3
4^3+(-3)^3+(-3)^3
out>

----------------------
```

8.4 D

8.4.1 decimal2any

decimal2any(,)

功能: 十进制数转成任意p进制形式.

参数: 双参数. 第一个参数是十进制数的表达式, 第二个参数是要转换的进制 p.

例

```
in> decimal2any(108010759187528,36)
out> 12abgh9jk8
```

8.4.2 decimal2binary

`decimal2binary()`

功能: 将十进制数转成二进制形式.

参数: 单参数, 接受十进制数.

例

```
in> decimal2binary(20220820)
out> 1001101001000101110010100
```

8.4.3 decimal2fraction

`decimal2fraction(num)` 将十进制小数转成分数, 分两种形式输出.

```
>> decimal2fraction(3.7)
out> 3+7/10==37/10
------------------------
```

8.4.4 decimal2hex

`decimal2hex(num)` 将十进制数(num)转换成十六进制形式.

注意: 返回的十六进制数以0x开头.

```
in> decimal2hex(1010)
out> 0x3F2
```

8.4.5 decimal2octal

`octal2decimal()` 将八进制数转成十进制形式.

功能: 十进制数转成八进制形式

注意: 返回的八进制数以0开头.

例

```
in> decimal2octal(1010)
out> 01762
```

8.4.6 decimalToFraction

`decimalToFraction()` 将十进制数转换为分数, 只输出假分数的形式.

```
in> decimalToFraction(3.7)
out> 37|10
```

8.4.7 deg

`deg()` 返回多项式的次数.

参数: 一元多项式, 默认变元为 `x`. 若多项式主变元是其他字母, 例如 `t`, 则需要先将 `x` 更改为 `t`.

```
>> deg(x^3-2x^2+x+3)
out> 3

------------------------
>> deg(t^3-2t^2+t+3)
out> t^3-2t^2+t+3 is NOT a polynomial.

------------------------

>> :change x=t
The main var has been changed to t.

>> deg(t^3-2t^2+t+3)
out> 3

------------------------
```

8.4.8 deg2rad

`deg2rad()` 将度转换为弧度.

例 8.4.1 将 30° 转换为弧度.

```
>> deg2rad(30)
out> 1/6pi

------------------------
```

8.4.9 det

`det(A)` 计算矩阵 A 的行列式.

关于矩阵的输入详见 2.5.

例 8.4.2 计算行列式

$$\begin{vmatrix} 1 & 6 & 5 \\ 7 & 2 & 9 \\ 4 & 8 & 3 \end{vmatrix}.$$

```
>> A=[1 6 5;
A=[1 6 5;
7 2 9;
4 8 3]
A=[1 6 5;
input> [1,6,5;7,2,9;4,8,3]
--------------------

1       6       5
7       2       9
4       8       3

--------------------
>> det(A)
out> 264

------------------------
```

8.4.10 diff

`diff()` 函数可用于对一元多项式进行求导. 最初仅实现了对多项式的求导, 从v0.650开始支持对常见的初等函数进行求导, 甚至可以对多元函数求偏导.

例 8.4.3 求 $3x^2 - 6x + 9$ 的导数.

```
>> diff(3x^2-6x+9)
out> 6x^1-6

------------------------

>> diff(3t^2-6t+9)
out>

------------------------

>> :change x t
The main var has been changed to t.

>> diff(3t^2-6t+9)
out> 6t^1-6

------------------------
```

例 8.4.4 常数 C 的导数为 0.

```
>> diff(C)
out> 0
```

```
3
4 ------------------------
```

例 8.4.5 $(x^{\alpha})' = \alpha x^{\alpha-1}, (\alpha \neq 0)$.

```
1 >> diff(x^alpha)
2 input> x^alpha
3
4 diff> x^alpha*alpha*1/x
5  out> x^alpha*alpha*1/x
6 out>
7
8 ------------------------
```

例 8.4.6 $(e^{x})' = e^{x}$.

```
1 >> diff(exp(x))
2 input> exp[x]
3
4 diff> exp[x]
5 out>
6
7 ------------------------
8 >> diff(e^x)
9 input> e^x
10
11 diff> e^x
12  out> e^x
13 out>
14
15 ------------------------
```

例 8.4.7 $(a^{x})' = a^{x} \ln a \ (a > 0, a \neq 1)$.

```
1 >> diff(a^x)
2 input> a^x
3
4 diff> a^x*ln[a]
5  out> a^x*ln(a)
6 out>
7
8 ------------------------
```

例 8.4.8 $\left(\ln |x|\right)' = \frac{1}{x}$.

这里暂不支持绝对值符号的输入, 也未加入 `abs()`函数.

```
1 >> diff(ln(x))
2 input> ln[x]
3
4 diff> 1/x
5  out> 1/x
6 out>
```

```
7
8  ------------------------
9
10 >> diff(ln(-x))
11 input> ln[{0-x}]
12
13 diff> -1/{0-x}
14  out> -1/(0-x)
15 out>
16
17 ------------------------
```

例 **8.4.9** $(\sin x)' = \cos x$.

```
1 >> diff(sin(x))
2 input> sin[x]
3
4 diff> cos[x]
5 out>
6
7 ------------------------
```

例 **8.4.10** $(\arcsin x)' = \dfrac{1}{\sqrt{1-x^2}}$.

```
1 >> diff(arcsin(x))
2 input> arcsin[x]
3
4 diff> 1/sqrt[{1-x^2}]
5  out> 1/sqrt((1-x^2))]
6 out>
7
8 ------------------------
```

例 **8.4.11** $(\cos x)' = -\sin x$.

```
1 >> diff(cos(x))
2 input> cos[x]
3
4 diff> -sin[x]
5 out>
6
7 ------------------------
```

例 **8.4.12** $(\arccos x)' = -\dfrac{1}{\sqrt{1-x^2}}$.

```
1 >> diff(arccos(x))
2 input> arccos[x]
3
4 diff> -1/sqrt[{1-x^2}]
5  out> -1/sqrt((1-x^2))]
6 out>
```

```
7
8 ------------------------
```

例 8.4.13 $(\sec x)' = \sec x \tan x$.

```
1 >> diff(sec(x))
2 input> sec[x]
3
4 diff> sec[x]*tan[x]
5  out> sec(x)*tan(x)
6 out>
7
8 ------------------------
```

例 8.4.14 $(\csc x)' = -\csc x \cot x$.

```
1 >> diff(csc(x))
2 input> csc[x]
3
4 diff> -csc[x]*cot[x]
5  out> -csc(x)*cot(x)
6 out>
7
8 ------------------------
```

例 8.4.15 $(\tan x)' = \sec^2 x$.

```
1 >> diff(tan(x))
2 input> tan[x]
3
4 diff> sec[x]^2
5  out> sec(x)^2
6 out>
7
8 ------------------------
```

例 8.4.16 $(\arctan x)' = \dfrac{1}{1+x^2}$.

```
1 >> diff(arctan(x))
2 input> arctan[x]
3
4 diff> 1/{1+x^2}
5  out> 1/(1+x^2)
6 out>
7
8 ------------------------
```

例 8.4.17 $(\cot x)' = -\csc^2 x$.

```
1 >> diff(cot(x))
2 input> cot[x]
3
```

```
diff> -csc[x]^2
 out> -csc(x)^2
out>

------------------------
```

例 8.4.18 $(\operatorname{arccot} x)' = -\frac{1}{1+x^2}$.

```
>> diff(arccot(x))
input> arccot[x]

diff> -1/{1+x^2}
 out> -1/(1+x^2)
out>

------------------------
```

例 8.4.19 $(\sinh x)' = \cosh x$.

```
>> diff(sinh(x))
input> sinh[x]

diff> cosh[x]
out>

------------------------
```

例 8.4.20 $(\cosh x)' = \sinh x$.

```
>> diff(cosh(x))
input> cosh[x]

diff> sinh[x]
out>

------------------------
```

例 8.4.21 $(\tanh x)' = 1 - (\tanh x)^2$.

```
>> diff(tanh(x))
input> tanh[x]

diff> 1-tanh[x]^2
 out> 1-tanh(x)^2
out>

------------------------
```

例 8.4.22 $(\coth x)' = 1 - (\coth x)^2$.

```
>> diff(coth(x))
input> coth[x]
```

```
3
4 diff> 1-coth[x]^2
5  out> 1-coth(x)^2
6 out>
7
8 ------------------------
```

`diff()` 也可以对复合函数求导. 以下出现的函数(比如 $f(x)$, $g(x)$ 等)我们总假设它们是可导的, 如果是多元函数则假设它们的偏导数均存在.

例 8.4.23 求函数 $\sin(f(x))$ 的导数.

```
1 >> diff(sin(f(x)))
2 input> sin[f[x]]
3
4 diff> cos[f[x]]*f'[x]*1
5  out> cos(f(x))*f'(x)*1
6 out>
7
8 ------------------------
```

例 8.4.24 求函数 $\sin(2x)+\cos(g(x^2))$ 的导数.

```
1 >> diff(sin(2x)+cos(g(x^2)))
2 input> sin[2*x]+cos[g[x^2]]
3
4 diff> cos[2*x]*2-sin[g[x^2]]*g'[x^2]*x^2*2*1/x
5  out> cos(2*x)*2-sin(g(x^2))*g'(x^2)*x^2*2*1/x
6 out>
7
8 ------------------------
```

例 8.4.25 求函数 $f(g(x^2,x),x)$ 关于 x 的导数.

```
1 >> diff(f(g(x^2,x),x))
2 input> f[g[x^2,x],x]
3
4 diff> f_1[g[x^2,x],x]*{g_1[x^2,x]*x^2*2*1/x+g_2[x^2,x]*1}+f_2[g[x^2,x],x]*1
5  out> f_1(g(x^2,x),x)*(g_1(x^2,x)*x^2*2*1/x+g_2(x^2,x)*1)+f_2(g(x^2,x),x)*1
6 out>
7
8 ------------------------
```

这里 `f_1`, `f_2` 指 `f` 对第一分量和第二分量的偏导数.

Sowya 中默认主变元为 `x`. 因此如果函数中有其他变元, 在使用 `diff()` 函数求导时, 认为是对主变元 `x` 求偏导.

例 8.4.26 求 $f(t)\cdot x$ 关于 x 的导数.

```
1 >> diff(f(t)*x)
2 input> f[t]*x
3
```

```
diff> f[t]+f'[t]*0*x
out>

------------------------
```

如果将主变元改为 t, 再执行 diff() 函数, 则是对 t 求导.

```
>> :change x=t
The main var has been changed to t.

>> diff(f(t)*x)
input> f[t]*x

diff> f'[t]*1*x
 out> f'(t)*1*x
out>

------------------------
```

例 8.4.27 求函数 $\sin(2x)+\cos(f(t^2))$ 关于 t 的导数.

```
>> diff(sin(2x)+cos(f(t^2)))
input> sin[2*x]+cos[f[t^2]]

diff> -sin[f[t^2]]*f'[t^2]*t^2*2*1/t
 out> -sin(f(t^2))*f'(t^2)*t^2*2*1/t
out>

------------------------
```

下面我们还是将主变元该回 x, 并对上面的函数求导.

```
>> :change t x
The main var has been changed to x.

>> diff(sin(2x)+cos(f(t^2)))
input> sin[2*x]+cos[f[t^2]]

diff> cos[2*x]*2-sin[f[t^2]]*f'[t^2]*0
 out> cos(2*x)*2-sin(f(t^2))*f'(t^2)*0
out>

------------------------
```

例 8.4.28 求 $e^x \cdot x$ 的导数.

```
>> diff(exp(x)*x)
input> exp[x]*x

diff> exp[x]+exp[x]*x
out>

------------------------
```

例 8.4.29 设 $f(x)=u(x)^{v(x)}$, 其中 $u(x)>0$ 且 $u(x)\neq 1$. 这样的函数称为幂指函数. 假设 $u(x),v(x)$ 均可导, $f(x)$ 的导数为

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\left[u(x)^{v(x)}\right] &= \left[e^{v(x)\ln u(x)}\right]' \\ &= e^{v(x)\ln u(x)}\cdot\left[v'(x)\ln u(x)+v(x)\cdot\frac{u'(x)}{u(x)}\right] \\ &= u(x)^{v(x)}\left[v'(x)\ln u(x)+v(x)\cdot\frac{u'(x)}{u(x)}\right].\end{aligned}$$

```
>> diff(u(x)^v(x))
input> u[x]^v[x]

diff> u[x]^v[x]*{v'[x]*1*ln[u[x]]+v[x]*u'[x]*1/u[x]}
 out> u(x)^v(x)*(v'(x)*1*ln(u(x))+v(x)*u'(x)*1/u(x))
out>

------------------------
```

例 8.4.30 求 $(\sin x)^{\cos x}+(\cos x)^{\sin x}$ 的导数.

```
>> diff(sin(x)^cos(x)+cos(x)^sin(x))
input> sin[x]^cos[x]+cos[x]^sin[x]

diff> sin[x]^cos[x]*{-sin[x]*ln[sin[x]]+cos[x]*cos[x]/sin[x]}+cos[x]^sin[x]*{cos[x]*ln[cos[x]]-sin[x]*sin[x]/cos[x]}
 out> sin(x)^cos(x)*(-sin(x)*ln(sin(x))+cos(x)*cos(x)/sin(x))+cos(x)^sin(x)*(cos(x)*ln(cos(x))-sin(x)*sin(x)/cos(x))
out>

------------------------
```

例 8.4.31 求 x^x 的导数.

```
>> diff(x^x)
input> x^x

diff> x^x*{ln[x]+x*1/x}
 out> x^x*(ln(x)+x*1/x)
out>

------------------------
```

例 8.4.32 求 x^{x^x} 的导数.

```
>> diff(x^x^x)
input> x^(x^x)

diff> x^(x^x)*{x^x*{ln[x]+x*1/x}*ln[x]+x^x*1/x}
 out> x^(x^x)*(x^x*(ln(x)+x*1/x)*ln(x)+x^x*1/x)
out>

------------------------
```

8.4.11 disp

`disp(var)` 用于显示变量 `var` 的值. 如果是数则直接返回.

例 8.4.33 定义变量 a 为 2, 用 `disp()` 函数显示变量的值. 另假设矩阵 A 如8.4.9中已定义, 用 `disp()` 函数显示其值.

```
>> a=2
2
--------------------
>> disp(a)
out> 2

------------------------

>> A
in> A
1       6       5
7       2       9
4       8       3

>> disp(A)
out> [1,6,5;7,2,9;4,8,3]

------------------------
```

8.4.12 div

`div(polyn1,polyn2)` 返回多项式 `polyn1` 除以 `polyn2` 得到的商和余式.

```
>> div(x^3+1,x^2+x+1)

 quotient> q(x) = x-1
remainder> r(x) = 2

(x^1-1)+(2)/(x^2+x^1+1)
------------------------
```

8.4.13 DetExpansion

v0.570加入了函数`DetExpansion(A,ri)`, 这里第二个参数`ri` 表示按第 i 行展开, 也可以是`cj`, 表示按第 j 列展开.

例. 对于方阵

$$A=\begin{pmatrix}1&2&3&4\\2&3&4&1\\3&4&1&2\\4&1&2&3\end{pmatrix}$$

可以使用det(A)求其行列式, 也可以按某一行(或某一列)展开, 将四阶行列式写成若干个三阶行列式的线性和. 下面将 $|A|$ 按第一行展开,

```
>> A=[1 2 3 4;
A=[1 2 3 4;
2 3 4 1;
3 4 1 2;
4 1 2 3]
A=[1 2 3 4;
input> [1,2,3,4;2,3,4,1;3,4,1,2;4,1,2,3]
--------------------

1       2       3       4
2       3       4       1
3       4       1       2
4       1       2       3

--------------------
>> DetExpansion(A,r1)
expansion>

1*(-1)^(1+1)*|A1_1| + 2*(-1)^(1+2)*|A1_2|  + 3*(-1)^(1+3)*|A1_3|  + 4*(-1)^(1+4)*|A1_4|
where

A1_1:
------------
3       4       1
4       1       2
1       2       3

A1_2:
------------
2       4       1
3       1       2
4       2       3

A1_3:
------------
2       3       1
3       4       2
4       1       3

A1_4:
------------
2       3       4
3       4       1
4       1       2


>>
```

8.5 E

8.5.1 E

`E(n,i,j(k))` 用以生成 n 阶初等(变换)矩阵, 即线性代数教材中常用的 $E(i,j(k))$. `E(n,i(k))` 将生成 n 阶初等矩阵 $E(i(k))$.

```
>> E(4,2,3(5))
Elementary matrix>

1       0       0       0
0       1       5       0
0       0       1       0
0       0       0       1


>> E(5,3(2))
Elementary matrix>

1       0       0       0       0
0       1       0       0       0
0       0       2       0       0
0       0       0       1       0
0       0       0       0       1


>> :list
info> All variables are listed below.
matrix : E4_2_3x0=[1,0,0,0;0,1,5,0;0,0,1,0;0,0,0,1]     mem addr: 0000022E379FF260
matrix : E5_3x1=[1,0,0,0,0;0,1,0,0,0;0,0,2,0,0;0,0,0,1,0;0,0,0,0,1]     mem addr: 0000022E379FEC60
---*---*---*---
```

8.5.2 Eisenstein

`Eisenstein(polyn)` 利用艾森斯坦因(Eisenstein)判别法判定多项式 `polyn` 是否在有理数域上不可约.

定理 8.5.1 ([2], P.33) 设 $f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_0$ 是一个整系数多项式. 如果有一个素数 p, 使得

- $p\nmid a_n$;
- $p|a_k$, $k=0,1,2,\ldots,n-1$;
- $p^2\nmid a_0$.

那么 $f(x)$ 在有理数域上是不可约的.

例 8.5.1 对于任意的 $n\in\mathbb{Z}^+$, 多项式 x^n+2 在有理数域上是不可约的.

(见 [2], [3].)

```
>> Eisenstein(x^2+2)
out> True, it is an irreducible polynomial on Q.
(The prime number that satisfies the Eisenstein conditions is 2.)

-----------------------
```

注 8.5.1 如果返回 `false`, 只能说明该多项式不满足艾森斯坦因判别条件, 不能由此推断其在有理数域上可约.

8.5.3 EulerBrackets

由下面递推关系定义的函数, 称为欧拉括号.

$$[k_1, k_2, \ldots, k_{m+1}] = [k_1, k_2, \ldots, k_m]k_{m+1} + [k_1, k_2, \ldots, k_{m-1}]$$

这里 k_i (i=1,2,3,...) 是正整数.

`EulerBrackets()` 可以接受任意有限个参数. 当这些参数形成一个等差数列时, 可以简化输入. 例如: `EulerBrackets(3,4,5,6,7,8,9,10)` 等价于 `EulerBrackets(3,4,...,10)` .

```
>> EulerBrackets(3,4,...,10)
in> EulerBrackets(3,4,...,10)
2263381

-----------------------

>> EulerBrackets(3,4,5,6,7,8,9,10)
in> EulerBrackets(3,4,5,6,7,8,9,10)
2263381

-----------------------
```

8.5.4 EulerVarphi

欧拉函数(Euler's totient function) $\varphi(m)$ 计算的是比 m 小且与 m 互素之正整数的个数. 也可以定义为与 m 互素的数就模 m 分成类的个数, 或者定义为模 m 的任一完全剩余组中与 m 互素的个数. (详见 [4].)

这里我们定义了函数 `EulerVarphi()`. 若计算 $\varphi(20221003)$, 只需键入 `EulerVarphi(20221003)`.

```
>> EulerVarphi(20221003)
in> EulerVarphi(20221003)
out> 18354600
```

8.5.5 eq24

这是一个名为“算廿四”的游戏（详见7.1）. 函数 `eq24(a,b,c,d)` 对输入的四个正整数 a,b,c,d 遍历可能的算式（算式要求仅进行加减乘除运算, 且每个数只能参与一次）, 如果结果等于 24 则输出该算式.

```
>> eq24(2,4,6,8)

2*6+4+8==24
4*(6-2)+8==24
2*6*8/4==24
6*8/(4-2)==24
4*8-2-6==24
(2+6)*4-8==24
(6+8)*2-4==24
(8/4+2)*6==24
8-4*(2-6)==24

------------------------

Total: 9
```

8.5.6 exp

`exp(x)` 用于计算指数函数 e^x 的值. 指数函数 e^x 可表示为

$$e^x = \sum_{n=0}^{+\infty} \frac{x^n}{n!}, \quad \forall\, x \in \mathbb{R}.$$

使用该公式计算 $\exp(x)$ 并不是最好的方法. 特别的, 当 $x = 1$ 时, 至少有一个更快的算法

$$e = \sum_{n=0}^{+\infty} \frac{2n+2}{(2n+1)!}.$$

这里的 `exp(x)` 采用了另一个更快的计算公式

$$e^x = 1 + \sum_{k=1}^{+\infty} \frac{[3k(3k-1) + 3kx + x^2]}{(3k)!} x^{3k-2}.$$

注意对于 $x^2 + 3kx + 3k(3k-1)$,

$$\Delta = (3k)^2 - 4 \cdot 3k(3k-1) = 12k - 27k^2 = 3k(4-9k) < 0$$

对所有 $k \geqslant 1$ 成立, 故 $x^2 + 3kx + 3k(3k-1) > 0$ 对所有 x 成立.

如果 $x = n$ 是正整数, 可采用快速求幂法计算 e^n, 但速度不如上面的级数快. 上面的级数在 x 不是正整数时, 计算就会受很大影响.

若采用浮点数计算 $e^x = \exp(x)$, 则将其转换为下面的形式

$$e^x = 2^n \cdot e^{x_1},$$

这里 $x = n \ln 2 + x_1$. 取适当的 n, 使得

$$-\frac{1}{2}\ln 2 \leqslant x_1 \leqslant \frac{1}{2}\ln 2.$$

$$\ln 2 \approx 0.693147180559945309417232121458 18$$

$$\frac{1}{\ln 2} \approx 1.442695040888963407359924681 0019$$

令 $M = \frac{x}{\ln 2} + \frac{1}{2}$, $m = \frac{x}{\ln 2} - \frac{1}{2}$, 则 n 满足

$$m \leqslant n \leqslant M.$$

8.5.7 expmod

`expmod(base,power,N)` 使用快速模幂算法计算 `base^power mod N` 的值. 这比先计算 `base^power` 再计算模 `N` 要快得多.

例 8.5.2 计算 $3^{123456789} \mod 10^{20}$.

这里, 我们不能输入

```
3^123456789mod100000000000000000000
```

因为这里将先计算 $3^{123456789}$, 而这是一个非常大的数. 我们使用 `expmod()` 函数进行计算, 即输入

```
expmod(3,123456789,100000000000000000000)
```

得到结果

```
>> expmod(3,123456789,100000000000000000000)
calculate: 3^123456789(mod 100000000000000000000)
out> 43806986775225122483

------------------------
```

8.6 F

8.6.1 factorial

`factorial(n)` 是阶乘函数, 返回 $n!$ 的值. `factorial(20)` 等同于输入 `20!`.

```
>> factorial(20)
2432902008176640000
------------------------

>> 20!

out> 2432902008176640000

------------------------
```

8.6.2 Factorial

`Factorial(n)` 也是阶乘函数, 计算 n 的阶乘, 即 $n!$. 与 `factorial(n)` 不同的是, 它还将 n 表为标准分解式. 关于标准分解式, 参见[4].

例 8.6.1 求 20! 的标准分解式.

```
in> Factorial(20)
out> expr = 2^18*3^8*5^4*7^2*11^1*13^1*17^1*19^1

2432902008176640000
```

8.6.3 factorise

`factorise(n)` 将正整数 n 作因子分解.

```
>> factorise(123456789)
out> 3*3*3607*3803

------------------------
```

8.6.4 Factorise

函数 `Factorise(n)` 将对正整数 n 进行因子分解, 并输出为指数形式.

```
>> Factorise(719712)
in> Factorise(719712)
719712 == 2*2*2*2*2*3*3*3*7*7*17
2^5*3^3*7^2*17^1
out> 719712 == 2^5*3^3*7^2*17

------------------------
```

而 `factorise()` 函数不会将结果简化.

```
>> factorise(719712)
in> factorise(719712)
2*2*2*2*2*3*3*3*7*7*17
out> 2*2*2*2*2*3*3*3*7*7*17

```

6 ----------

8.6.5 Factorise_analysis

设 n 是正整数, 利用函数 `Factorise_analysis(n)` 可以将 n 分解为

$$n = 2^{\gamma} \cdot p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_s^{\alpha_s} \cdot q_1^{\beta_1} q_2^{\beta_2} \cdots q_t^{\beta_t}$$

的形式, 这里 p_i, q_j 是互异的奇素数, 模 4 分别余 1 和 3, 即 $p_i \equiv 1 \pmod 4$, $q_j \equiv 3 \pmod 4$. $i = 1, 2, \ldots, s$; $j = 1, 2, \ldots, t$. α_i, β_j 以及 γ 都是非负整数.

`Factorise_analysis(n)` 将计算一些特定的函数, 例如下面定义的三个函数 $\delta(n)$, $\varepsilon(n)$, $\xi(n)$.

$$\delta(n) = \left[\prod_{i=1}^{s}(\alpha_i + 1)\right]\left[\prod_{j=1}^{t}\frac{(-1)^{\beta_j}+1}{2}\right],$$

$$\varepsilon(n) = \left[\frac{(-1)^{\gamma}+1}{2}\right]\left[\prod_{i=1}^{s}\frac{(-1)^{\alpha_i}+1}{2}\right]\left[\prod_{j=1}^{t}\frac{(-1)^{\beta_j}+1}{2}\right],$$

$$\xi(n) = (-1)^{\gamma} \cdot \left[\frac{(-1)^{\prod_{i=1}^{s}(\alpha_i+1)}-1}{2}\right]\left[\prod_{j=1}^{t}\frac{(-1)^{\beta_j}+1}{2}\right].$$

特别的, 当 $n = 1$ 时, 可得 $\gamma = \alpha_i = \beta_j = 0$, $i = 1, 2, \ldots, s$, $j = 1, 2, \ldots, t$. 因此,

$$\delta(1) = 1, \quad \varepsilon(1) = 1, \quad \xi(1) = -1.$$

由这三个函数可表示双平方和问题的解数公式.

定理 8.6.1 若用 $r_2(n)$ 表示二次不定方程 $x^2 + y^2 = n$ 整数解的数目, 则

$$\begin{aligned} r_2(n) &= \mathrm{Card}\{(x, y) \in \mathbb{Z}^2 \ : \ x^2 + y^2 = n\} \\ &= 4\delta(n). \end{aligned}$$

定理 8.6.2 若用 $r_2^+(n)$ 表示二次不定方程 $x^2 + y^2 = n$ 正整数解的数目, 则

$$\begin{aligned} r_2^+(n) &= \mathrm{Card}\{(x, y) \in \mathbb{Z}_+^2 \ : \ x^2 + y^2 = n\} \\ &= \delta(n) - \varepsilon(n). \end{aligned}$$

定理 8.6.3 若用$r_{2*}^{+}(n)$表示二次不定方程 $x^2+y^2=n$ 无序正整数解的数目, 则

$$\begin{aligned} r_{2*}^{+}(n) &= \mathrm{Card}\{(x,y)\in\mathbb{Z}_{+}^{2} \,:\, x^2+y^2=n \quad \text{且 } x\leqslant y\} \\ &= \frac{\delta(n)+\xi(n)}{2}. \end{aligned}$$

例子:

```
>> Factorise_analysis(20230320)
in> Factorise_analysis(20230320)
out> 20230320 == 2^4 * 5*97 * 3*11*79

        delta(20230320) = 0
        epsilon(20230320) = 0
        xi(20230320) = 0

------------------------
```

参见问题3106

8.6.6 Farey

函数 `farey(n)` 或 `Farey(n)` 将生成 Farey 序列. Farey 序列是指集合

$$\mathscr{F}_n = \{\frac{a}{b} \mid 0\leq a\leq b\leq n,\ (a,b)=1\}$$

按从小到大的顺序排列而成一个分数序列. 例如 $\mathscr{F}_5$ 为

$$\frac{0}{1},\quad \frac{1}{5},\quad \frac{1}{4},\quad \frac{1}{3},\quad \frac{2}{5},\quad \frac{1}{2},\quad \frac{3}{5},\quad \frac{2}{3},\quad \frac{3}{4},\quad \frac{4}{5},\quad \frac{1}{1}.$$

```
>> Farey(5)
in> Farey(5)
out> (0/1,1/5,1/4,1/3,2/5,1/2,3/5,2/3,3/4,4/5,1/1)

Total = 11

------------------------
```

8.6.7 Fibonacci

斐波那契数列是由递推公式 $F_n=F_{n-1}+F_{n-2}$ 以及初始条件 $F_0=F_1=1$ 得到的. `fibonacci(n)` 或 `Fibonacci(n)` 将返回斐波那契数列的第 n 项.

```
>> fibonacci(102)
in> fibonacci(102)

F(102) = 927372692193078999176

------------------------
```

如果要打印斐波那契数列的前 20 项, 则加一个参数 `t` 或 `true` 或 `y` 或 `yes`.

```
>> fibonacci(20,t)
in> fibonacci(20,t)

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,

------------------------

F(20) = 6765

------------------------
```

如果第二个参数输入的是 `formula`, 则显示斐波那契数列的通项公式.

```
>> fibonacci(20,formula)

Fibnacci series, Fibonacci(n) or denote by F(n), is defined by F(n)=F(n-1)+F(n-2). F(0)=0, F(1)=1. Here n is a
     nonnegative integer.
We define F(-n)=(-1)^{n-1}F(n).
The first few number of Fibonacci series are:
 0,1,1,2,3,5,8,13,21,34,55,...
They can be generated by function
 Fibonacci(10,yes)

The formula of F_n=F(n) is:
F_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n-\frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n

And we infer that
F_n = \frac{1}{2^{n-1}}\cdot(C_n^1+C_n^3\cdot 5+C_n^5\cdot 5^2+\cdots+C_n^m\cdot 5^{\frac{m-1}{2}})

------------------------

F(20) = 6765

------------------------
```

8.6.8 firstfactor

`firstfactor(n)` 的功能是返回正整数 n 最小的因子.

```
>> firstfactor(20230217)
in> firstfactor(20230217)
7

------------------------
```

8.6.9 fix_variables

函数 `fix_variables(s)` 用于固定变量. 其参数为 1 或 0. 当参数为 1 时锁定所有变量, 即无法通过赋值将其改变; 当参数为 0 时, 则解锁所有变量.

比如定义变量 `a` 为 3, 默认可以更改 `a` 的值.

```

>> a=3
--------------------
>> a=2
--------------------
```

如果我们不希望 a 的值发生改变, 则使用`fix_variables(1)`.

```
>> fix_variables(1)
in> fix_variables(1)
Now variables are fixed

------------------------

>> a
in> 2

out> 2

------------------------


>> a=4
 The variable a has been defined. Try again.
```

使用 `fix_variables(0)` 将所有变量解锁, 即恢复到默认的可变更状态. 未来将增加此函数的功能, 比如固定或解锁某个特定变量.

8.6.10 floor

`floor(x)` 返回 x 的整数部分, 即不超过 x 的最大整数. 这个值数学上通常记作 $[x]$ 或 $\lfloor x \rfloor$.

```
>> floor(3.14)
3
------------------------

>> floor(-3.14)
-4
------------------------
```

8.6.11 free_plugin

函数 `free_plugin(pn)` 的作用是卸载名为 `pn` 的插件. 当然这个名为 `pn` 的插件之前已经被自动加载或者使用 `load_plugin()` 函数进行加载过.

关于插件的使用以及开发, 详见9.

8.7 G

8.7.1 gcd

`gcd(n1,n2,...,nk)` 用于计算两个或两个以上的正整数的公因子, 这里 $k \geqslant 2$.

```
>> gcd(9,129)
in> gcd(9,129)
3

------------------------

>> gcd(9,12,18)
in> gcd(9,12,18)
3

------------------------
```

8.7.2 gcd2

`gcd2(a,b)` 使用辗转相除法计算两个正整数 a, b 的最大公因子, 同时将辗转相除法过程的每一步进行输出.

例 8.7.1 求 27, 6, 9 这三个数的最大公因数, 并打印辗转相除法的具体过程.

```
>> gcd2(27,6,9)
27==4*6+3
6==2*3+0
(27,6,0)
gcd = 3

3==0*9+3
9==3*3+0
(3,9,0)
gcd = 3

3
------------------------
```

由此可见, 这里`gcd2(27,6,9)` 是先计算 27 和 6 的最大公因数, 将所得结果 3 再与第三个数 9 作辗转相除法求最大公因数.

8.7.3 Gcd

`Gcd(p1,p2,...,pn)` 计算两个或两个以上多项式的最大公因式, 这里 $n \geqslant 2$. 注意 `p1, p2, ..., pn` 均是一元多项式. 详见2.6.6.

例 8.7.2 计算 x^2-1, x^3-1, x^4-1 的最大公因式.

```
>> Gcd(x^2-1,x^3-1,x^4-1)
x-1
-----------------------
```

8.7.4 Gcd2

`Gcd2(polyn1,polyn2)` 使用辗转相除法计算两个一元多项式的最大公因式, 并且将每一步计算得到的商和余式输出. 最后将最大公因式 $d(x)$ 表示为

$$d(x) = u(x) \cdot f(x) + v(x) \cdot g(x).$$

详见2.6.6.

8.7.5 getValueFromMemAddr

该函数将根据变量地址返回变量的值. 使用`:list`命令可列出当前所有定义的变量, 并附有变量的地址.

```
>> a=2
--------------------
>> :list
info> All variables are listed below.
int : a = 2     mem addr: 01579AD0
---*---*---*---

>> getValueFromMemAddr(01579AD0)
in> getValueFromMemAddr(01579AD0)
2
```

8.7.6 getprecision

`getprecision()` 函数用于获取当前的计算精度. 例如:

```
>> getprecision()
in> getprecision()
Now the precision is: 8
```

8.7.7 goldbach

`goldbach(n)` 对于正整数 n 验证满足哥德巴赫(Goldbach)猜想. 即对于大于等于 6 的偶数都可以表示为两个素数的和. `goldbach(n)` 试图列出所有可能的表示法.

```
>> goldbach(90)
90 == 7 + 83
90 == 11 + 79
90 == 17 + 73
90 == 19 + 71
90 == 23 + 67
90 == 29 + 61
90 == 31 + 59
90 == 37 + 53
90 == 43 + 47
in> goldbach(90)

Total: 9
------------------------
```

对于弱哥德巴赫猜想,即任一大于5的奇数都可以表示为三个素数之和,已被完全证明.2013年5月13日,法国国家科学研究院和巴黎高等师范学院的数论领域的研究员哈洛德•贺欧夫各特,在线发表两篇论文宣布彻底证明了弱哥德巴赫猜想.

```
>> goldbach(23)
23 == 3 + 3 + 17
23 == 3 + 7 + 13
23 == 5 + 5 + 13
23 == 5 + 7 + 11
in> goldbach(23)

Total: 4
------------------------
```

8.7.8 graycode2binary

格雷码(Gray Code)是由贝尔实验室的弗兰克·格雷(Frank Gray, 1887–1969)在20世纪40年代提出,并在1953年取得美国专利“Pulse Code Communication”.最初目的是在使用PCM(Pusle Code Modulation)方法传输数字信号的过程中降低错误可能.*

`graycode2binary()` 将格雷码(也是二进制形式)转换成相应的二进制数.

```
>> graycode2binary(100111001)
in> graycode2binary(100111001)
out> 111010001
```

8.8 H

8.8.1 help

`help(funcName)` 显示函数`funcName`的简介及用法等.

* 见 https://zhuanlan.zhihu.com/p/29254973

```
>> help(help)
in> help(help)
out> Display the introduction of the function.

Usage:
help(funcName)
```

8.8.2 hex2decimal

`hex2decimal(n)` 将十六进制数转换为十进制数. 详见进制转换2.2.10.

8.9 I

8.9.1 IndefiniteEquation

求解不定方程使用 `IndefiniteEquation()` 函数. 对于二元一次不定方程 $ax + by = 1$, 则使用

```
IndefiniteEquation(a,b)
```

这里 a, b 是系数.

例 8.9.1 求解不定方程 $7x + 9y = 1$.

```
>> IndefiniteEquation(7,9)
out> Solve the equation: 7*x+9*y = 1
9==1*7+2
7==3*2+1
1 3
test: -1==9*3-7*4
7*4-9*3 == 1
x = 4+9t
y = -3-7t


-----------end---------
1

------------------------
```

如果右边的系数不为1, 即形如 $ax + by = c$, 则使用

```
IndefiniteEquation(a,b;c)
```

此方程有解当且仅当 $\gcd(a, b)|c$.

例 8.9.2 求解不定方程 $7x + 9y = 5$.

这里 $\gcd(7, 9) = 1$, 可整除 5. 故方程有解. 先计算相应的不定方程 $7x + 9y = 1$, 上面得到解 $x = 4 + 9t$, $y = -3 - 7t$. 最后只需将 4 和 -3 分别乘以 5 即可.

```
>> IndefiniteEquation(7,9;5)
Solve the equation: 7*x+9*y = 5
9==1*7+2
7==3*2+1
1 3
test: -1==9*3-7*4
7*20-9*15 == 1
x = 20+9t
y = -15-7t


-----------end---------
```

如果方程形如 a1∗x1+a2∗x2+...+aK∗xK=b, 则使用

```
IndefiniteEquation(a1,a2,...,aK;b)
```

例 8.9.3 求解不定方程 $25x - 13y + 7z = 4$.

```
>> IndefiniteEquation(25,-13,7;4)
Solve the indefinite equation :
25*X1-13*X2+7*X3 = 4

u2=4-2*u1
---------------
X2 = +2*u0+1*u2-u1
X1 = +1*X2-1*u2+u0
X3 = -3*X1+1*X2+u2

------------------------
```

例 8.9.4 求解不定方程 $x + 2y + 3z = 4$.

```
>> IndefiniteEquation(1,2,3;4)
Solve the indefinite equation :
1*X1+2*X2+3*X3 = 4

X3=4-X1-2*u1
---------------
X2 = -1*X3+u1
X1 = +X1

------------------------
```

例 8.9.5 求解不定方程 $7x_1 + 4x_2 - 2x_3 + 3x_4 = 2$.

```
>> IndefiniteEquation(7,4,-2,3;2)
Solve the indefinite equation :
7*X1+4*X2-2*X3+3*X4 = 2

X4=2-X1-2*u2
---------------
X1 = +X1
X3 = +3*X1+2*X2+1*X4-u2

```

```
----------------------
```

对于 $b = 1$ 的情形, 可以直接输入

```
IndefiniteEquation(a1,a2,...,aK)
```

8.9.2 index_of_prime

给定素数 p, 函数`index_of_prime(p)`返回其在素数集合 Primes 中的位置, 即是第几个素数. 例如希望了解素数 19 是第几个素数, 我们输入`index_of_prime(19)`, 此时会提醒先运行函数`Primes(100)`.

```
>> index_of_prime(19)
in> index_of_prime(19)
hint> You should call function Primes(100) first.
```

`Primes(100)` 将列出不超过 100 的所有素数.

```
>> Primes(100)
2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,

Time used:     000
in> Primes(100)
out> 25
```

然后再运行`index_of_prime(19)`,

```
>> index_of_prime(19)
in> index_of_prime(19)
out> 8
```

返回 8, 表明 19 是第 8 个素数.

8.9.3 InfixToPostfix

`InfixToPostfix(InfixExpression)` 将中缀表达式`InfixExpression` 转换为后缀表达式.

例 8.9.6 将 $7 * (2 + 1/3 - 4) + 5$ 转换为后缀表达式.

```
>> InfixToPostfix(7*(2+1/3-4)+5)
out> 7,2,1,3,/,+,4,-,*,5,+,

----------------------
```

8.9.4 integrate

`integrate(polyn)` 对于一元多项式`polyn`求不定积分.

例 8.9.7 求不定积分 $\int (3x^2 - 6x + 9)\mathrm{d}x$.

```
>> integrate(3x^2-6x+9)
out> x^3-3x^2+9x^1

------------------------
```

8.9.5 inv

`inv(A)` 返回方阵 A 的逆矩阵. 参见2.5.5.

8.9.6 iscomposite

`iscomposite(n)` 检测正整数 n 是否是合数. 例如: 2023 是合数, 具有因子 7.

```
>> iscomposite(2023)
in> iscomposite(2023)
2023 is a composite number.
It has a divisor: 7
```

8.9.7 ispolyn

`ispolyn(polyn)` 判定所给的表达式 `polyn` 是否是一元多项式. 参见 2.6.

8.9.8 isprime

`isprime(n1,n2,...,nk,"hint")` 函数用于检测所给的正整数是否是素数. 它可以接受单个参数, 也可以接受多个参数, 当最后一个参数是字符串 `"hint"` 时, 还可以给出提示.

- `isprime(n)` 检测正整数 n 是否是素数, 返回 `true` 或 `false`;
- `isprime(n1,...,nk)` 检测多个正整数是否是素数, 返回其中素数的个数;
- `isprime(n1,...,nk,"hint")` 检测一个或多个正整数是否是素数, 并输出提示, 返回其中素数的个数.

例 8.9.8 20230411 是素数.

```
>>  isprime(20230411)
true
------------------------
```

如果所给参数是表达式, 则先计算其值, 然后再判断.

例 8.9.9 判断 $2^{23}-1$ 是否是素数.

```
>> isprime(2^23-1)
false
------------------------
```

注 8.9.1 对于长度大于 20 的数, `isprime(n)` 会调用内部的函数isCompositeByLittleFermat()检测是否是合数, 以及其他算法(例如Miller-Rabin算法)从概率上判定是素数的可能性有多大. 当输入的参数 n 形如 2^k-1 时, 调用 Lucas-Lehmer 测试, 判断是否是梅森素数.

例 8.9.10 判断 $2^{79}-1$ 是否是素数, 并给出提示.

```
>> isprime(2^79-1,"hint")
sqrtnum=777472127993.868721

Hint> Since len(604462909807314587353087)=24 > 20, we use function isCompositeByLittleFermat() to check whether it is
      a composite.

rs> 604462909807314587353087 is not a prime.

the following computation will take a few while.

--------------------
we choose sqrtnum : 777472127999
 ... since we use the traditional algorithm, ...
 ... please wait a minute ...
 ... Press Ctrl+C to stop it. ...
 ... ::: ... ::: ...
out> 604462909807314587353087 is not a prime, which has a divisor: 26870
------------------------
```

需要注意的是, `isprime()`最后使用的是传统的检测方法, 效率很低, 对于较大的素数无法在较短时间内判断是否是素数. 如果不想等待, 请按 Ctrl+C 终止程序运行.

例 8.9.11 判断 137, 159, 237, 289, 367 中哪些是素数, 共有多少个素数.

```
>> isprime(137,159,237,289,367,"hint")
137 is a PRIME number.
159 is NOT a prime number.
237 is NOT a prime number.
289 is NOT a prime number.
367 is a PRIME number.
2
------------------------
```

8.9.9 ispseudoprime

`ispseudoprime(n1,n2,...,nk,"hint")` 函数用于检测所给的正整数是否是伪素数. 它可以接受单个参数, 也可以接受多个参数, 当最后一个参数是字符串 `"hint"` 时, 则给出提示.

- `ispseudoprime(n)` 检测正整数 n 是否是伪素数, 返回 `true` 或 `false`;
- `ispseudoprime(n1,...,nk)` 检测多个正整数是否是伪素数, 返回其中伪素数的个数;
- `ispseudoprime(n1,...,nk,"hint")` 检测一个或多个正整数是否是伪素数, 并输出提示, 返回其中伪素数的个数.

`ispseudoprime(n)` 检测正整数 n 是否是伪素数. 伪素数是指满足费马小定理的合数. 对于伪素数, 我没有 Fermat 小定理.

定理 8.9.1 设 p 是一个素数, 且 $0 < a < p$, 则有 $a^{p-1} \equiv 1 \pmod{p}$.

例 8.9.12 判断 2047, 341, 561, 645 是否是伪素数.

```
>> ispseudoprime(2047)
true

------------------------

>> ispseudoprime(341,561,645)
3
------------------------

>> ispseudoprime(341,561,645,"hint")
341 is a pseudo prime.
561 is a pseudo prime.
645 is a pseudo prime.
3
------------------------
```

这说明表示 2047, 341, 561, 645 都是伪素数.

例 8.9.13 1000以内的伪素数只有三个: 341, 561, 645.

输入 `:mode clox` 进入编程模式, 输入下面的代码

```
{
fun PseudoPrimes(N){
  var cnt=0;
  for(var k=0; k<=N; k=k+1){
    if(ispseudoprime(k)){
      println k;
      cnt=cnt+1;
    }
  }
  return cnt;
}
}
```

然后调用此函数

```
> print PseudoPrimes(1000);
341
561
645
3>
```

最后显示有 3 个伪素数.

习题 8.9.1 求 10000 以内的伪素数.

关于伪素数, 参见问题2296.

8.10 J

8.10.1 Jacobi

雅可比符号 $\left(\frac{a}{p}\right)$ 是勒让德符号在保持勒让德符号的同余性质、Gauss互反律等性质下的推广. 这里不要求 p 必须是素数, p 是正的奇数, a 与其互素. 勒让德符号 $\left(\frac{a}{p}\right)$ 在计算过程中必须将分子 a 进行因子分解, 在 a 非常大时会比较麻烦, 这也导致函数 `Legendre(a,p)` 的实现比较复杂. 而这里的 `Jacobi(a,p)` 函数的实现就比较直接, 每当分子是偶数, 需要将因子 2 都提取出来, 即 $a = 2^k b$, 这里 b 是奇数. 从而再应用Gauss互反律、同余性质进行计算, 将分母不断变小.

但某些情况下, 雅可比符号仍需要调用因子分解的程序. 例如当 $a = 2$ 时, 必须将 p 进行因子分解.

例: 计算 $\left(\frac{438}{593}\right)$.

```
>> Jacobi(438,593,show)
in> Jacobi(438,593,show)

(438|593) = (2|593)(219|593)
(2|593) = 1

(64|155) = (1|155)

(155|219) = -(219|155) = -(64|155) = -1
(219|593) = (593|219) = (155|219) = -1
-------
result> -1

------------------------
```

例: 计算 $\left(\frac{853}{1409}\right)$.

```
>>  Jacobi(853,1409,show)
in> Jacobi(853,1409,show)

(556|853) = (139|853)

(6|19) = (2|19)(3|19)
(2|19) = -1

(3|19) = -(19|3) = -(1|3) = -1
(19|139) = -(139|19) = -(6|19) = -1
(139|853) = (853|139) = (19|139) = -1
(853|1409) = (1409|853) = (556|853) = -1
-------
result> -1
```

```
15
16 ------------------------
```

例: 计算 $\left(\frac{2}{262992639}\right)$.

```
1  >> Jacobi(2,262992639,show)
2  in> Jacobi(2,262992639,show)
3
4  (2|262992639) = (2|13)(2|3)(2|7)(2|963343)
5
6  (2|13) = -1
7  (2|3) = -1
8  (2|7) = 1
9  (2|963343) = 1
10 -------
11 result> 1
12
13 ------------------------
```

8.11 L

8.11.1 LagrangePolyn

`LagrangePolyn(x0,y0;x1,y1;...,xn,yn)` 根据给定的 $n+1$ 个点, 计算拉格朗日插值多项式. 参见 2.6.10.

8.11.2 lcm

`lcm(a,b)` 返回正整数 a, b 的最小公倍数(Least Common Multiple). 也可以计算多个正整数的最小公倍数.

```
1  >> lcm(8,9)
2  in> lcm(8,9)
3  72
4
5  ------------------------
6
7  >> lcm(8,9,12,27)
8  in> lcm(8,9,12,27)
9  216
10
11 ------------------------
```

8.11.3 Legendre

这里实现的函数 `Legendre(a,p)` 计算的是数论中的勒让德符号 $\left(\frac{a}{p}\right)$. 其具体定义可参见问题2855. 这里 p 是奇素数, a 是不能被 p 整除的非零整数. 勒让德符号 $\left(\frac{a}{p}\right)$ 的值要么是 1 要么是 -1. 但函数

`Legendre(a,p)` 当检测到 a 或 p 不满足定义要求时会返回 0. 当 p 是奇数但非素数时, 使用下面的雅可比符号计算, 即 `Jacobi(a,p)`.

```
>> Legendre(438,593)
in> Legendre(438,593)


result> -1

------------------------
```

如果要显示计算的具体步骤, 可以加入第三个参数 `show`.

```
>> Legendre(438,593,show)
in> Legendre(438,593,show)

(438|593) = (2|593)(3|593)(73|593)

(2|593) = 1

(3|593) = (593|3) = (2|3) = -1

(73|593) = (593|73) = (9|73) = 1

-------
result> -1

------------------------
```

我们再看一个例子, 计算 $\left(\frac{2023}{1231}\right)$.

```
>> Legendre(2023,1231,show)
in> Legendre(2023,1231,show)
(2023|1231) =
   (792|1231) =
(792|1231) = (11|1231)(2|1231)

(11|1231) = -(1231|11) = -(10|11) = 1

(2|1231) = 1

-------
result> 1

------------------------
```

注: 这两个例子参考自 A. K. 苏什凯维奇著叶乃膺译《数论初等教程》P.104–106.

8.11.4 len

`len(str)` 计算字符串`str`的长度.

```
>> len(18446744073709551616)
in> len(18446744073709551616)
out> 20
```

```
>> len(2^64-1)
in> len(2^64-1)
hint> 2^64-1 contains non digit char, it is regarded as a string.
out> 6
```

可以试一下 `len(我喜欢Sowya)`.

8.11.5 lim

`lim(expr,x,x_0)` 用于求极限 $\lim_{x\to x_0}$ expr. 这里 `expr` 是一元有理函数 $f(x)$. 参见 2.9.1.

8.11.6 limit

`limit(expr,x,x_0)` 同 `lim(expr,x,x_0)`.

8.11.7 list_pi_x

`list_pi_x(A,B)` 将打印 $\pi(x)$, 这里 $x \in [A, B]$. 例如:

```
>> list_pi_x(10,20)
x       pi(x)
----------------
10      4
11      5
12      5
13      6
14      6
15      6
16      6
17      7
18      7
19      8
20      8
----------------
```

8.11.8 list_plugins

`list_plugins()` 列出所有插件的名称.

```
>> list_plugins()
[plugins] Auto-loaded plugins:
Pi
---Total: 1 ---


----------------
[myplugins] user plugins:
---Total: 0 ---

```

```
out> 0

------------------------
```

8.11.9 listfunctions

`listfunctions()` 列出当前版本所有可用的内置函数†

```
>> listfunctions()
Collatz
EulerBrackets
EulerVarphi
Factorial
Factorise
Factorise_analysis
Farey
...
```

如果加入参数 1, 即`listfunctions(1)`, 则列出各函数的原型.

```
>> listfunctions(1)
Collatz(num)
EulerBrackets(k1,k2)
EulerVarphi(positiveinteger)
Factorial(positiveinteger)
Factorise(positiveInteger)
Factorise_analysis(positiveInteger)
Farey(positiveinteger)
...
```

8.11.10 listpseudoprimes

`listpseudoprimes(N)` 列出小于 N 的所有伪素数.

例如: 小于 1000 的伪素数仅有三个.

```
in> listpseudoprimes(1000)
341
561
645

------------------------
Total: 3 pseudo primes.
```

8.11.11 ln

`ln(n)` 计算 n 的自然对数.

†注意, 某些函数并不在 Basic 版本中.

```
>> ln(2.71828)
in> ln(2.71828)
out> 0.99999933
```

8.11.12 load_plugin

`load_plugin(plugin_name)` 加载位于 `myplugins` 目录下名为 `plugin_name.dll` 的插件. 见 9.2.

8.11.13 log

`log(n)` 同 `ln(n)`.

8.12 M

8.12.1 mobius

`mobius(x)` 就是 Möbius 函数 $\mu(x)$. 若 x 含有平方因子, 则 $\mu(x) = 0$; 若 x 不含有平方因子, 且有 ω 个不同的因子, 则定义 $\mu(x) = (-1)^{\omega}$. 具体参见问题2765.

例: 计算 $\mu(2023)$.

```
>> mobius(20230406)
in> mobius(20230406)
out> 1
```

8.12.2 monic_polyn

`monic_polyn(polyn)` 将多项式变为首一多项式(即首项系数为 1 的多项式). 这里参数`polyn`是一元多项式, 自变量默认是 x. 例如:

```
>> monic_polyn(3x^2+6x-9x+2)
in> monic_polyn(3x^2+6x-9x+2)
out> x^2-x+2|3
```

8.12.3 multiplicativeOrder

定义 8.12.1 给定正整数 b 和 n, 满足下式

$$b^e \equiv 1 \pmod{n}$$

的最小正整数 e 被称为 b 模 n 的 *multiplicative order*, 或称为 *haupt-exponent or modulo order.*

v0.588 版本中添加了函数 `multiplicativeOrder(b,n)`, 返回 b 模 n 的乘法阶(multiplicativeOrder).

```
>> multiplicativeOrder(7,2024)
110
-----------------------
```

8.12.4 multipolyn_items

`multipolyn_items(polyn)` 将多元多项式 `polyn` 的各个项输出, 对于未事先声明的符号, 则认为是常数.

```
>> multipolyn_items(2xy^2+xy-y^2+2)
split> (2*y^2+y)x+(-y^2+2)
into items>

(2*y^2+y)x
(-y^2+2)
-----------------------
Total: 2 items.
```

8.13 N

8.13.1 nextMRprime

`nextMRprime(n)` 使用 Miller-Rabin 算法生成大于 n 的最小素数.

```
>> nextMRprime(10000000000000000000000000000000)
in> nextMRprime(10000000000000000000000000000000)
out> 10000000000000000000000000000033
```

8.13.2 nextprime

`nextprime(n)` 使用传统算法返回大于 n 的最小素数.

```
>> nextprime(20230408)
in> nextprime(20230408)
out> 20230411
```

8.13.3 norm_p

对于 p-adic 数(可参见问题1782), 有 p-范数的概念. 这里提供了函数`norm_p(x,p)`, 用于计算有理数 x 关于以素数 p 为基底的 p-adic 范数.

```
>> norm_p(5/294,7)
in> norm_p(5/294,7)
out> 49
```

8.14 O

8.14.1 octal2decimal

`octal2decimal(n)` 将八进制数转换为十进制. 详见进制转换一章.

8.14.2 ord_p

若 x 是正整数, 则 $\mathrm{ord}_p(x)$ 定义为 $\max\{r : p^r \mid x\}$, 称为 n 关于素数 p 的阶.

若 $x = \frac{a}{b}$ 是有理数, p 是素数, 定义

$$\mathrm{ord}_p(x) = \mathrm{ord}_p(\frac{a}{b}) = \mathrm{ord}_p(a) - \mathrm{ord}_p(b)$$

`ord_p(x,p)`

```
>> ord_p(252,7)
in> ord_p(252,7)
out> 1
```

8.15 P

8.15.1 p

`p(n)` 返回第 n 个素数, 数论教材中通常记作 p_n. 如果没有先生成素数集合(数组)`Primes`, 则会提示先执行 `Primes(N)`, 这里`N` 通常是一个比 n 大一些的正整数. 例如: 显示第23个素数.

```
>> p(23)
in> p(23)
hint> You should call function Primes(100) first.

------------------------

>> Primes(100)
2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,

Time used:     000
in> Primes(100)
out> 25

------------------------

>> p(23)
in> p(23)
out> 83
```

8.15.2 pi

`pi()` 将返回 π 在当前计算精度下的近似值; 如果是含有参数 x 的函数 `pi(x)` 则返回小于等于 x 的素数个数, 这里 x 是正整数, 即数论中的函数 $\pi(x)$.

```
>> pi()
in> pi()
out> 3.14159265

------------------------

>> pi(100)
in> pi(100)
out> 25

------------------------
```

8.15.3 polyn_items

`polyn_items(polyn)` 返回多项式 `polyn` 的各个项. 如果多项式未进行化简, 则先进行化简.

```
>> polyn_items(2x^2-3+4x-5x^3+6x^2+7x-8)
split> -5x^3+8x^2+11x^1-11
into items>

-5x^3
8x^2
11x^1
-11
------------------------
Total: 4 items.
```

8.15.4 prevMRprime

`prevMRprime(n)` 使用 Miller-Rabin 算法生成小于 n 的最大素数.

```
>> prevMRprime(10000000000000000000000000000000)
in> prevMRprime(10000000000000000000000000000000)
out> 99999999999999999999999999999973
```

8.15.5 prevprime

`prevprime(n)` 使用传统算法返回小于 n 的最大素数.

```
>> prevprime(1234567890)
in> prevprime(1234567890)
out> 1234567811
```

8.15.6 Primes

`Primes(N)` 列出不超过 N 的所有素数, 以逗号作为间隔符. 同时将这些素数记录在内存中, 方便其他函数（如 `p(n)`等）访问.

例 8.15.1 列出 1000 以内的所有素数.

```
>> Primes(1000)
2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,
157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,
317,331,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,
499,503,509,521,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,
683,691,701,709,719,727,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,
883,887,907,911,919,929,937,941,947,953,967,971,977,983,991,997,

Time used:     000
out> 168

------------------------
```

显示 1000 以内共有 168 个素数.

8.15.7 PrimitivePolyn

定义 8.15.1 *([2],P.30)* 如果一个非零的整系数一元多项式 $g(x)=b_nx^n+b_{n-1}x^{n-1}+\cdots+b_1x+b_0$ 的系数 $b_n,b_{n-1},\ldots,b_1,b_0$ 没有异于 ±1 的公因子, 即它们是互素的, 则称 $g(x)$ 是一个本原多项式*(Primitive polynomial).*

v0.629版本加入了函数 `PrimitivePolyn(polyn)`, 用于判定多项式 `polyn` 是否是本原多项式(primitive polynomial). 当然参数 `polyn` 必须是一元多项式.

例 8.15.2 判断多项式 $2x^5-3x^3+4x^2$ 是否是本原多项式.

```
>> PrimitivePolyn(2x^5-3x^3+4x^2)
out> true

------------------------
```

8.15.8 print3x1Maps

`print3x1Maps(from, to)` 利用`Collatz()`函数打印`Collatz(n)`,`n=from,...,to`.

```
>> print3x1Maps(12,15)
12-->6-->3-->10-->5-->16-->8-->4-->2-->1

13-->40-->20-->10-->5-->16-->8-->4-->2-->1

14-->7-->22-->11-->34-->17-->52-->26-->13-->40-->20-->10-->5-->16-->8-->4-->2-->1

```

```
15-->46-->23-->70-->35-->106-->53-->160-->80-->40-->20-->10-->5-->16-->8-->4-->2-->1
```

8.15.9 printRecursiveSeries

`printRecursiveSeries()` 打印由递推公式所定义的数列的前N项

用法:

`printRecursiveSeries(递推公式,通项,初始值,打印项数,分隔符)`

- 第一个参数是递推关系表达式, v0.537版本中这个函数只能处理简单的递推关系式;
- 第二个参数是通项;
- 第三个参数是初值;
- 第四个参数是打印前多少项;
- 第五个参数是分隔符, 是可选参数, 默认是逗号. 也可以是换行符 `\n`, 制表符 `\t`, 或者两个换行符 `\n\n` 或其他任意字符串.

分隔符默认是逗号, 且默认不打印行号. 例如: `printRecursiveSeries(n^2,n,1,10,;)` 并不是打印 $1^2, 2^2, \ldots, 10^2$; 而是打印十项 n^2, 每次 n 都等于 1, 且用分号作为分隔符.

```
>> printRecursiveSeries(n^2,n,1,10,;)
1;1;1;1;1;1;1;1;1;1;
```

例如, Hanoi 塔问题的递推关系为 $a_n = 2a_{n-1} + 1, n = 2, 3, \ldots$. 初值 $a_1 = 1$. 这里首先将递推关系改写为 $a_{n+1} = 2a_n + 1$.

```
>> printRecursiveSeries(2*a_n+1,a_n,1,10,\n)
in> printRecursiveSeries(2*a_n+1,a_n,1,10,\n)
1
3
7
15
31
63
127
255
511
1023
```

如果要打印行号, 则在最后添加参数 `linenumber`.

```
>> printRecursiveSeries(2*a_n+1,a_n,1,10,\n,linenumber)
[1]     1
[2]     3
[3]     7
[4]     15
[5]     31
[6]     63
[7]     127
[8]     255
[9]     511
[10]    1023
```

对于由递推关系 $h_{n+1}=(4n-2)h_n$, 初值 $h_1=1$ 确定的序列 $\{h_n\}$, 打印前 10 项.

```
>> printRecursiveSeries((4*n-2)*h_n,h_n,1,10,\n)
in> printRecursiveSeries((4*n-2)*h_n,h_n,1,10,\n)
1
2
12
120
1680
30240
665280
17297280
518918400
17643225600


------------------------
```

v0.541版本对`printRecursiveSeries()`作了改进, 可以处理形如

$$a_{2n+1}=\frac{a_n+n+1}{1+a_n*(n+1)}$$

的迭代式. 假设初始项是 a_3, 其值为 2, 则如下输入:

```
>> printRecursiveSeries(a_{2*n+1}==(a_n+n+1)/(1+a_n*(n+1)),a_3=2,10,\n,linenumber)
in> printRecursiveSeries(a_{2*n+1}~(a_n+n+1)/(1+a_n*(n+1)),a_3=2,10,\n,linenumber)
[3]     2
[7]     0.66666667
[15]    1.36842105
[31]    0.75862069
[63]    1.29604366
[127]   0.77782653
[255]   1.28058400
[511]   0.78241332
[1023]  1.27686257
[2047]  0.78354694


------------------------
```

- 这里第一个参数仍是迭代式, `a_{2*n+1}==(a_n+n+1)/(1+a_n*(n+1))` 必须使用两个等号, 并且 a_{2n+1} 的下标应写成`2*n+1`, 方便运算.
- 第二个参数是初始项的赋值表达式, 这里是`a_3=2`.
- 第三个参数是打印的项数.
- 第四个参数是分隔符, 跟之前一样, 可以使用`\t`,`\n`,`\n\n` 或其他字符.
- 第五个参数可选, 若写`linenumber`或`lineNumber`或`line_number`, 则在每一项前打印行号.

如果要输出分数形式, 则先执行`:mode=fraction`.

```
>> :mode fraction
Switch into fraction calculating mode.
e.g., 1/2+1/3 will return 5/6

>> printRecursiveSeries(a_{2*n+1}==(a_n+n+1)/(1+a_n*(n+1)),a_3=2,10,\n,linenumber)
in> printRecursiveSeries(a_{2*n+1}~(a_n+n+1)/(1+a_n*(n+1)),a_3=2,10,\n,linenumber)
```

```
[3]     2|1
[7]     2|3
[15]    26|19
[31]    22|29
[63]    950|733
[127]   5318|6837
[255]   880454|687541
[511]   693690|886603
[1023]  454634426|356055883
[2047]  11062298746|14118233579


------------------------
```

`printRecursiveSeries()`还可以用于牛顿迭代法求解一元高次方程.

例 8.15.3 求方程 $x^5-4x^3-5=0$ 在 $x=1$ 附近的根.

首先切换到多项式计算模式

```
>> :mode polyn
```

然后对 `f(x)=x^5-4x^3-5` 求导,

```
>> diff(x^5-4x^3-5)
out> 5x^4-12x^2
```

计算`(x^5-4x^3-5)/(5x^4-12x^2)`的商和余式

```
>> (x^5-4x^3-5)/(5x^4-12x^2)
in> (x^5-4x^3-5)/(5x^4-12x^2)

out>
 quotient> q(x) = 1|5x
remainder> r(x) = -8|5x^3-5

1|5x^1
------------------------
```

注意, 从 `v0.625` 开始, 要得到上面的结果必须使用`div()` 函数.

```
>> div(x^5-4x^3-5,5x^4-12x^2)

 quotient> q(x) = 1|5x
remainder> r(x) = -8|5x^3-5

(1|5x^1)+(-8|5x^3-5)/(5x^4-12x^2)
------------------------
```

如果仅需得到商`q(x)`, 则可以使用 `%` 运算符.

```
>> (x^5-4x^3-5)%(5x^4-12x^2)

out> 1|5x^1
------------------------
```

为提高计算的精度, 设定数值计算精度为100.

```
>> setprecision(100)
Now the precision is: 100

------------------------
```

然后使用printRecursiveSeries()函数计算迭代公式

$$x_n - \frac{f(x_n)}{f'(x_n)} = x_n - (\frac{1}{5}x_n + \frac{-\frac{8}{5}x^3 - 5}{5x^4 - 12x^2}) = \frac{4}{5}x_n + \frac{\frac{8}{5}x^3 + 5}{5x^4 - 12x^2}$$

这里不妨迭代100次.

```
>> printRecursiveSeries(4/5*x_n+(8/5*x_n^3+5)/(5*x_n^4-12*x_n^2),x_n,1,100,\n,linenumber)
[1]     1
[2]     -0.142857142857142857142857142857142857142857142857142857142857142857142857142857142857142857142857142857
[3]     -20.68684146042636608674344523401127174712080372457730948296986032835089438863023768684146042636608673092
[4]     -16.565023688617589109177537826433863432338260623780573240602345316225300420691945262445633989624803938736
[5]     -13.2714938151392127327086618324645366416606984746301979545605997238256662746354987884645019492889088259888
[6]     -10.64160729106856435253004280224457254617986844856116673479583215765055870684575566382988446238459617659104
[7]     -8.543927904421098082103266940566651336467393310841794835891510921669912488167846412120893765286900283972832
[8]     -6.8736750146567822836834747098409137278847103761224278023001309485647375068881413772811833775034090294782656
[9]     -5.54751385154459757117959853911308426694533117603318730701466364274880256207172022811686144943490408568261248
[10]
     -4.499428383849649530577659730552688681850614839831792743978496600619449792627229845215124665110187212646089984
...  ...
[95]    -1.75238826131581457566420214399657304232924718679223682144779151802603527445993530886755041057951064050000000
39951342135619708337669405749916685322147717924146834921482445434074138489452088000512
[96]    -1.75238826131581457566420214399657304232924718679223682144779151802603527445993530886755041057951064050000000
319610737084957666701355245999333482577181743393174679371859563472593107915616704004096
[97]    -1.75238826131581457566420214399657304232924718679223682144779151802603527445993530886755041057951064050000000
2556885896679661333610841967994667860617453947145397434974876507780744863324933632032768
[98]    -1.75238826131581457566420214399657304232924718679223682144779151802603527445993530886755041057951064050000000
20455087173437290668886735743957342884939631577163179479799012062245958906599469056262144
[99]    -1.75238826131581457566420214399657304232924718679223682144779151802603527445993530886755041057951064050000000
163640697387498325351093885951658743079517052617305435838392096497967671252795752450097152
[100]   -1.75238826131581457566420214399657304232924718679223682144779151802603527445993530886755041057951064050000000
1309125579099986602808751087613269944636136420938443486707136771983741370022366019600777216
```

最后检验一下是否满足原方程.

```
>> x=-1.75238826131581457566420214399657304232924718679223682144779151802603527445993530886755041057951064050000000
1309125579099986602808751087613269944636136420938443486707136771983741370022366019600777216
----------------------------
 type: double
 name: x
value: -0
--------------------
>> (x)^5-4*(x)^3-5
in> (-1.75238826131581457566420214399657304232924718679223682144779151802603527445993530886755041057951064050000000
1309125579099986602808751087613269944636136420938443486707136771983741370022366019600777216)^5-4*(-1.7523882613
158145756642021439965730423292471867922368214477915180260352744599353088675504105795106405000000013091255790999
86602808751087613269944636136420938443486707136771983741370022366019600777216)^3-5

out> 0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001002461454
590868985858028958701885768370959989508849182986411587235592821887757059985818749869892582054243034647638488407
901913896321641765488022531054488166815940051659363431567227735167118930346767284506514535706498178164690029478
```

```
140095962457775116208647992617386963768387997477842777302515091340650933937993006942658375621314215193463439745
037455603051203056146511073088557155558112290821596229489082921661083813733704994891080747399745082810964794306
543580924476256376253826109387804215305344784878332518373086856953137405176106411680592827231552151954544698390
746095027370895087163214113614348073715487095091316799164127967421654418127385877665845623664963936231766775622
796511525075087479189744606535365707603087814408564247201833277255254306352830707925464072037286554907982664097
492173581179456846031855032016969567238231489207648412237826212174368084969570519846818479360385996554 24
------------------------
```

8.15.10 printSeries

`printSeries()` 函数用于打印数列 $\{a_n\}_{n=1}^{\infty}$ 的前 N 项. 语法是

```
printSeries(general_term,x,minValue,maxValue,delimiter)
```

其中 `general_term` 指数列的通项 a_n, 第二个参数 `x` 指其中的变元, `minValue` 和 `maxValue` 是 `x` 的变化范围, 最后一个参数 `delimiter` 是可选的, 作为打印的分隔符, 默认采用逗号.

例如打印 $n^2, n = 1, 2, \ldots, 10$, 则可以输入

```
>> printSeries(n^2,n,1,10,;)
1;4;9;16;25;36;49;64;81;100;
```

这里第一个参数是通项 n^2, 第二个参数是变元 n, 第三个参数是变元 n 的起始值, 第四个参数是变元的最终值, 最后一个参数是分隔符.

例. 打印数列 $\left\{\frac{(2n-2)!}{(n-1)!}\right\}_{n=1}^{\infty}$ 的前 12 项, 使用换行分隔符 `\n`.

```
>> printSeries((2*n-2)!/((n-1)!),n,1,12,\n)
in> printSeries((2*n-2)!/((n-1)!),n,1,12,\n)
1
2
12
120
1680
30240
665280
17297280
518918400
17643225600
670442572800
28158588057600
```

8.16 Q

8.16.1 q_DuanwuPlus

这里的 `q_DuanwuPlus(N)` 和 `q_duanwu(N)` 是用来求解端午节问题的. 详见 7.2.

8.16.2 q_duanwu

详见 7.2.

8.16.3 QR

QR(A) 将方阵 A 作 QR 分解. 关于矩阵的QR分解, 有如下定理. （参见 [3] 或问题3396.）

定理 8.16.1 任意一个 n 阶可逆实方阵 A 均可表为一个实正交方阵 O 和一个对角元全为正数的上三角方阵 R 的乘积, 即 $A = OR$. 而且这种表法惟一.

注 8.16.1 QR分解中的Q实际上应该是指O, QR分解也可称为**正交上三角分解**.

例 8.16.1 求矩阵

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

的 QR 分解.

首先输入矩阵 A,

```
>> A=[1 2 2;2 1 2;1 2 1]
input> [1,2,2;2,1,2;1,2,1]
--------------------

1       2       2
2       1       2
1       2       1

--------------------
```

然后输入 QR(A), 得到正交矩阵 O 和上三角矩阵 R, 如下.

```
>> QR(A)

---O--------------
0.40824829      0.57735027      0.70710678
0.81649658      -0.57735027     0.00000001
0.40824829      0.57735027      -0.70710678

-----------------

---R--------------
2.44948974      2.44948974      2.85773803
0       1.73205081      0.57735027
0       0       0.70710678

-----------------
```

使用 :list 列出内存中的矩阵变量

```
>> :list
info> All variables are listed below.
matrix : A=[1,2,2;2,1,2;1,2,1]          mem addr: 00000138D0931480
matrix : AO=[0.40824829,0.57735027,0.70710678;0.81649658,-0.57735027,0.00000001;0.40824829,0.57735027,-0.70710678]
      mem addr: 00000138D0931600
matrix : AR
     =[0.40824829,0.57735027,0.70710678;0.81649658,-0.57735027,0.00000001;0.40824829,0.57735027,-0.70710678][2.44948974,2.44948974,2.8577
         mem addr: 00000138D0931300
---*---*---*---
```

这里的 `AO` 和 `AR` 即上面的 `O` 矩阵和 `R` 矩阵. 命名时前面加上 `A` 表示它们从属于矩阵 `A`.

验证 $AO \cdot AR$ 是否等于 A.

```
>> AO*AR

out>
0.9999999977275446      2.0000000005347633      1.9999999966065100
1.9999999954550892      0.9999999926478705      2.0000000008329323
0.9999999977275446      2.0000000005347633      0.9999999999625732

------------------------
```

这里显然是存在一些误差的.

8.17 S

8.17.1 sin

Sowya v0.581 加入了正弦函数 `sin()`.

例如计算 `sin(12.25)`.

```
>> sin(12.25)
-0.31111935
------------------------
```

这是在默认的 8 位精度下的结果. 微软计算器的计算结果为

```
-0.31111935498112732258349594512964
```

我们设置计算精度为100, 然后再计算`sin(12.25)`.

```
>> setprecision(100)
Now the precision is: 100

------------------------

>> sin(12.25)
out> -0.31111935498112732258349594512964293507211103309188263439617013378696332785304909496343144312884861900

------------------------
```

上面的参数默认是弧度, 如果要计算 $\sin 30°$, 则输入 `sin(30d)`. 最后一个字母d是degree的意思.

```
>> sin(30d)
out> 1/2

------------------------
```

$30° = \frac{\pi}{6}$, 可以输入`sin(1/6pi)`或`sin(1|6pi)`.

```
>> sin(1/6pi)
out> 1/2

------------------------

>> sin(1|6pi)
out> 1/2

------------------------
```

目前支持 $\sin(\frac{k}{n}\pi)$ 的求值, 其中 $n = 1, 2, 3, 4, 5, 6$. 而 $k \in \mathbb{Z}$.

当然对于 $\sin(a\pi)$, 如果 a 是能转换为上述形式分数的小数, 则也可以求值. 比如:

```
>> sin(1.5pi)
out> -1

------------------------
```

其他例子

```
>> sin(1/5pi)
out> 1/2*sqrt((5-sqrt(5))/2)

------------------------

>> sin(2/5pi)
out> 1/2*sqrt((5+sqrt(5))/2)

------------------------
```

8.17.2 solve

这里提供的函数 `solve()`, 主要用于求解一部分整数方程. 语法为

```
solve(equation,x,minValue,maxValue)
```

第一个参数 `equation` 是方程的表达式, 其中等号要用连续的两个'=' 表示; 第二个参数 `x` 是指方程表达式中未知量的名称; `maxValue`, `minValue` 指 `x` 的上下界 ($x \in [\text{minValue}, \text{maxValue}]$), 当然这里的 `x`, `minValue`, `maxValue` 都是整数.

例, 求解方程 $x^{340} \equiv 1 \pmod{341}$. 我们可以如下解决

```
solve(x^340mod341==1,x,1,340)
```

例. 求解方程 $2^x = x^2$. 这个方程有三个实数解. 如果限定 x 为整数, 则有两个解 $x = 2$, $x = 4$.

```
>> solve(2^x==x^2,x,1,10)
ans>> x=2
ans>> x=4

------------------------
```

v0.550 版本对`solve()`函数进行了更新, 使其可以求解一元一次方程、一元二次方程.

例如, 求解方程 $8x+9=0$.

```
>> solve(8x+9==0)

  8x^1+9 == 0

solution>
        x = -1.125
```

若切换至分数计算模式, 再次运行, 则可以得到解 $x=-\frac{9}{8}$.

```
>> :mode fraction
Switch into fraction calculating mode.
e.g., 1/2+1/3 will return 5/6

>> solve(8x+9==0)

  8x^1+9 == 0

solution>
        x = -9|8
```

对于一元二次方程 $ax^2+bx+c=0, a\neq 0$, 我们知道它的两个解是 $x_{1,2}=\dfrac{-b\pm\sqrt{b^2-4ac}}{2a}$. `solve()` 函数就是利用此求解公式计算这两个解的. 当判别式 $\Delta=b^2-4ac<0$ 时, 得到两个共轭复根.

例. 求解方程 $x^7+7x+8=0$.

```
>> solve(x^2+7x+8==0)

  x^2+7x^1+8 == 0

solution>
        x1 = -1.43844719
        x2 = -5.56155282
```

若在分数计算模式下, 我们会得到下面形式的解.

```
>> solve(x^2+7x+8==0)

  x^2+7x^1+8 == 0

  Delta=b^2-4ac=17
  sqrt(Delta)=1*sqrt(17)

solution>
        x1 = (-7+1*sqrt(17))/2
        x2 = (-7-1*sqrt(17))/2
```

对于一元三次方程 $x^3+ax^2+bx+c=0$, 作变换 $y=x+\frac{a}{3}$, 可变换为 $y^3+py+q=0$ 的形式. 我们这里采用荷兰数学家许德(Johannes Hudde, 1628–1704)的方法求解.

例 8.17.1 求方程 $x^3-x^2-x-2=0$ 的根.

```
in> solve(x^3-x^2-x-2==0)

It is a univariate cubic equation.

  x^3-1x^2-1x^1-2 == 0

Answer:
    Let y=x-1/3

We get equation of y: y^3+py+q==0
where

    p=b-a^2/3,    q=c-ab/3+2a^3/27,

and a,b,c,d are coefficients of the equation ax^3+bx^2+cx+d==0

By computation,
    p = -4|3
    q = -65|27
Now the equation is:

    y^3-4|3y-65|27 == 0

    Delta = (q/2)^2+(p/3)^3 = 49|36

    sqrt(Delta) = sqrt((q/2)^2+(p/3)^3) = 7|6

    q/2 = -65|54
    -q/2 + sqrt(Delta) = 64|27
    -q/2 - sqrt(Delta) = 1|27
    u = 4|3
    v = 1|3

solution>
    x1 = 2
    x2 = 4|3w+1|3w^2+1|3 = -1|2+1|2*sqrt(3)*i
    x3 = 4|3w^2+1|3w+1|3 = -1|2-1|2*sqrt(3)*i

Where w is
    -1/2+i*sqrt(3)/2


------------------------

>>
```

例 8.17.2 求方程 $x^3+1=0$ 的根.

```
in> solve(x^3+1==0)

It is a univariate cubic equation.
```

```
4
5   x^3+1 == 0
6
7  Answer:
8      p = 0
9      q = 1
10
11     Delta = (q/2)^2+(p/3)^3 = 1|4
12
13     sqrt(Delta) = sqrt((q/2)^2+(p/3)^3) = 1|2
14
15     q/2 = 1|2
16     -q/2 + sqrt(Delta) = 0
17     -q/2 - sqrt(Delta) = -1
18     u = 0
19     v = -1
20
21 solution>
22     x1 = -1
23     x2 = 0w-1w^2-0 = 1|2+1|2*sqrt(3)*i
24     x3 = 0w^2-1w-0 = 1|2-1|2*sqrt(3)*i
25
26 Where w is
27     -1/2+i*sqrt(3)/2
28
29
30 ------------------------
31
32 >>
```

例 8.17.3 求方程 $x^3-6x+2=0$ 的根.

```
1  in> solve(x^3-6x+2==0)
2
3  It is a univariate cubic equation.
4
5    x^3-6x^1+2 == 0
6
7  Answer:
8      p = -6
9      q = 2
10
11     Delta = (q/2)^2+(p/3)^3 = -7
12
13     sqrt(Delta) = sqrt((q/2)^2+(p/3)^3) = sqrt(7)*i
14
15     q/2 = 1
16     -q/2 + sqrt(Delta) = (sqrt(7)*i-1)
17     -q/2 - sqrt(Delta) = -(sqrt(7)*i+1)
18     u = sqrtn(-1+sqrt(7)*i, 3)
19     v = sqrtn(-1-sqrt(7)*i, 3)
20
21 solution>
22     x1 = sqrtn(-1+sqrt(7)*i, 3)+sqrtn(-1-sqrt(7)*i, 3)-0
23     x2 = sqrtn(-1+sqrt(7)*i, 3)w+sqrtn(-1-sqrt(7)*i, 3)w^2-0
24     x3 = sqrtn(-1+sqrt(7)*i, 3)w^2+sqrtn(-1-sqrt(7)*i, 3)w-0
25
```

```
Where w is
    -1/2+i*sqrt(3)/2


------------------------

>>
```

8.17.3 solve_n3plus_until_m3_eq_p3

`solve_n3plus_until_m3_eq_p3(min,max)` 用于求解方程

$$n^3 + (n+1)^3 + (n+2)^3 + \cdots + m^3 = p^3$$

这里 $n \in [min, max]$.

例 8.17.4 若令 $min = 1, max = 1000$, 则有如下15个解.

```
>> solve_n3plus_until_M3_eq_p3(1,1000)
in> solve_n3plus_until_M3_eq_p3(1,1000)
out> 3^3 + ... + 5^3 = 6^3
3^3 + ... + 22^3 = 40^3
6^3 + ... + 30^3 = 60^3
6^3 + ... + 69^3 = 180^3
11^3 + ... + 14^3 = 20^3
11^3 + ... + 109^3 = 330^3
15^3 + ... + 34^3 = 70^3
34^3 + ... + 158^3 = 540^3
213^3 + ... + 365^3 = 1581^3
213^3 + ... + 555^3 = 2856^3
273^3 + ... + 560^3 = 2856^3
291^3 + ... + 339^3 = 1155^3
406^3 + ... + 917^3 = 5544^3
556^3 + ... + 654^3 = 2805^3
646^3 + ... + 798^3 = 3876^3

---------
Total: 15 solutions.
```

8.17.4 solve_x3plusy3_zn

`solve_x3plusy3_zn(min,max)`用于求解方程 $x^3 + y^3 = z^n$, 这里参数`min`和`max`是指 x 和 y 的范围, 即 $\min \leqslant x, y \leqslant \max$.

```
>> solve_x3plusy3_zn(2,8)

2^3 + 2^3 == 2^4
2^3+2^3 == 4^2
3^3+6^3 == 3^5
4^3 + 4^3 == 2^7
4^3+8^3 == 24^2
```

```
8^3 + 8^3 == 2^10
8^3+8^3 == 32^2
out>
---------
Total: 4 solutions.
```

8.17.5 sort

v0.589 版本增加了 `sort()` 函数, 用于对一列数进行排序. 这列数可以是十进制小数、整数或分数.

例 8.17.5 将数列 $1,28,28,14,14,14,7,28,14,28,28,4,14,28,28,7,4,28,28,7,28,14,7,7,7,28,28,2$ 进行排序.

```
>> sort(1,28,28,14,14,14,7,28,14,28,28,4,14,28,28,7,4,28,28,7,28,14,7,7,7,28,28,2)
1,2,4,4,7,7,7,7,7,7,14,14,14,14,14,28,28,28,28,28,28,28,28,28,28,28
------------------------
```

`sort()` 默认是从小到大排序. 如果是将上面的这列数降序排列, 则在最后加上参数 `desc`, 用逗号隔开.

```
>> sort(1,28,28,14,14,14,7,28,14,28,28,4,14,28,28,7,4,28,28,7,28,14,7,7,7,28,28,2;desc)
28,28,28,28,28,28,28,28,28,28,28,28,14,14,14,14,14,14,7,7,7,7,7,7,4,4,2,1
------------------------
```

现在要求将这列数视为字符串, 且降序排列. 则使用参数 `string|desc`.

```
>> sort(1,28,28,14,14,14,7,28,14,28,28,4,14,28,28,7,4,28,28,7,28,14,7,7,7,28,28,2;string|desc)
7,7,7,7,7,7,4,4,28,28,28,28,28,28,28,28,28,28,28,28,2,14,14,14,14,14,14,1
------------------------
```

例 8.17.6 将数列 $2,\frac{1}{2},\frac{1}{3},\frac{2}{3},20.3,\frac{7}{6}$ 进行排序.

```
>> sort(2,1/2,1/3,2/3,20.3,7|6)
0.33333333,0.5,0.66666667,1.16666667,2,20.3
------------------------
```

切换到分数模式下排序.

```
>> :mode fraction
Switch into fraction calculating mode.
e.g., 1/2+1/3 will return 5/6

>> sort(2,1/2,1/3,2/3,20.3,7|6)
(1|2),(1|3),(2|3),(7|6),2|1,203|10
------------------------
```

`sort()` 函数还可以对矩阵中的元素进行排序.

例 8.17.7 设 $A=\begin{pmatrix}1 & \frac{1}{2}\\ -3.2 & 4\end{pmatrix}$, 求矩阵 A 的秩和行列式, 并将矩阵 A 中的元素排序.

```
>> A=[1,1/2;-3.2,4]
input> [1,1/2;-3.2,4]
--------------------

1       1/2
-3.2    4

--------------------
>> A.rank()
in> A.rank()
2

>> A.det()
in> A.det()
28|5

>> A.sort()
in> A.sort()
-16|5   1|2
1|1     4|1
```

8.17.6 sqrt

`sqrt(n)` 求 n 的平方根.

```
>> sqrt(2)
out> 1.41421356

------------------------
```

如果要求精确到小数点后100位, 则先执行 `setprecision(100)`.

```
>> setprecision(100)
Now the precision is: 100

------------------------

>> sqrt(2)
out> 1.41421356237309504880168872420969807856967187537694807317667973799073247846210703885038753432764157 27

------------------------
```

8.17.7 sqrtn

`sqrtn(N,s)` 返回 $N^{\frac{1}{s}}$. 例如

```
>> sqrtn(97,3)
out> 4.59470089220703980609429646442230898912097549271904969222139061272491909988047118926272584262302985 99

------------------------
```

8.17.8 squaresum

`squaresum(max,n)` 将 n 表示为平方和, 即返回 $x^2+y^2=n$ 的整数解. 这里 $1 \leqslant x \leqslant \max$.

```
>> squaresum(10,25)
3^2+(-4)^2
3^2+(4)^2
4^2+(-3)^2
4^2+(3)^2
5^2+(0)^2
out>

------------------------
```

8.17.9 subset

`subset(N,m)` 返回形如 $(x_1, x_2, \ldots, x_m)$ 的 m-维向量, 这里 $x_i \in \{0, 1, 2, \ldots, N-1\}$, 且 $x_1 < x_2 < \cdots < x_m$.

```
>> subset(8,5)
0 1 2 3 4
0 1 2 3 5
0 1 2 3 6
0 1 2 3 7
0 1 2 4 5
0 1 2 4 6
0 1 2 4 7
0 1 2 5 6
0 1 2 5 7
0 1 2 6 7
0 1 3 4 5
0 1 3 4 6
0 1 3 4 7
0 1 3 5 6
0 1 3 5 7
0 1 3 6 7
0 1 4 5 6
0 1 4 5 7
0 1 4 6 7
0 1 5 6 7
0 2 3 4 5
0 2 3 4 6
0 2 3 4 7
0 2 3 5 6
0 2 3 5 7
0 2 3 6 7
0 2 4 5 6
0 2 4 5 7
0 2 4 6 7
0 2 5 6 7
0 3 4 5 6
0 3 4 5 7
0 3 4 6 7
0 3 5 6 7
0 4 5 6 7
```

```
1 2 3 4 5
1 2 3 4 6
1 2 3 4 7
1 2 3 5 6
1 2 3 5 7
1 2 3 6 7
1 2 4 5 6
1 2 4 5 7
1 2 4 6 7
1 2 5 6 7
1 3 4 5 6
1 3 4 5 7
1 3 4 6 7
1 3 5 6 7
1 4 5 6 7
2 3 4 5 6
2 3 4 5 7
2 3 4 6 7
2 3 5 6 7
2 4 5 6 7
3 4 5 6 7
```

8.17.10 sum

`sum(general_term,x,minValue,maxValue,expand)` 用于求和. 这在前面已经介绍过了.

- 第一个参数`general_term`指通项表达式;
- 第二个参数`x`指自变量采用的符号;
- 第三和第四个参数指自变量的取值范围. 注意这里 `x, minValue, maxValue` 都是整数, 且 `minValue<=x<=maxValue`;
- 最后一个参数是可选的, `expand`表示将求和式展开.

这里仅举一个简单的例子, 求 $\sum_{i=1}^{100} i^2$.

```
>>  sum(i^2,i,1,100)
338350
------------------------

>>  sum(n^2,n,1,10,expand)
1^2+2^2+3^2+4^2+5^2+6^2+7^2+8^2+9^2+10^2
out> 385
385
------------------------

>> n=10
--------------------
>> n*(n+1)*(2*n+1)/6
in> 10*(10+1)*(2*10+1)/6

out> 385
------------------------
```

8.17.11 suo

`suo(a,b,c,d;h)` 等同于函数 `eq24(a,b,c,d;h)`. 详见算廿四问题7.1.

8.17.12 suo24

`suo24` 和 `suo` 都是 `eq24` 的别名. 因此, `suo24(a,b,c,d;h)` 等同于函数 `eq24(a,b,c,d;h)`. 详见算廿四问题7.1.

8.17.13 symbol

函数 `symbol(x:val)` 用于定义变量并赋值. 这里x变量名, 冒号后面的 `val` 是赋予该变量的值. 比如 `symbol(x:36)` 的作用是定义变量 `x` 并赋予值 36. 若仅写 `symbol(x)`, 则仅定义变量 `x`, 并不赋值.

多个变量的赋值用逗号隔开, 若要赋值则在变量名后面加上冒号和对应的值. 详见 2.4.1.

```
>> symbol(x:3,y:4,z:5)
out> x has been defined.
out> y has been defined.
out> z has been defined.

------------------------

>> :list
info> All variables are listed below.
int : x=3       mem addr: 0000029A22542090
int : y=4       mem addr: 0000029A22542290
int : z=5       mem addr: 0000029A22541E10
---*---*---*---
```

8.18 T

8.18.1 tan

`tan(x)` 是正切函数. 求 x 的正切值. 内部的计算方式是

$$\tan x = \frac{\sin x}{\cos x}.$$

例 8.18.1 依次计算 $\tan 1$, $\tan(1.5\pi)$, $\tan \pi$, $\tan(\frac{3}{2}\pi)$ 和 $\tan 30°$.

```
>> tan(1)
1.55740773
------------------------

>> tan(1.5pi)
out> -Inf

------------------------

>> tan(1pi)
out> -0

------------------------

>> tan(3/2pi)
out> -Inf

------------------------

>> tan(30d)
+1/2/sqrt(3)/2
------------------------
```

8.18.2 tau

`tau(m)` 计算正整数 m 的因子总个数. 这是数论中常用的一个函数 $\tau(m)$.

例 8.18.2 24 的因子有 $1, 2, 3, 4, 6, 8, 12, 24$. 因此 $\tau(24) = 8$.

```
>> tau(24)
8
------------------------
```

8.18.3 transform

`transform(A, expr)` 函数执行矩阵的初等变换. 这里 `expr` 形如 `r1+r2*3` 等等. 详见 2.5.6.

8.18.4 transpose

`transpose(A)` 返回矩阵 A 的转置矩阵 A^T. 详见 2.5.8.

8.18.5 trimzeros

`trimzeros(num)` 去除数值`num`前后不必要的`0`. 注意数值`010`即`10`, 因此最后一个`0`是有效的.

```
>> trimzeros(001002300)
out> 1002300

------------------------
>> trimzeros(001002300.030200)
out> 1002300.0302

------------------------
```

8.19 X

8.19.1 xor

`xor(num1,num2)` 计算两个数的异或. 这里`num1`和`num2`都要求是二进制数.

```
>> xor(10011101101,1011101110001)
out> 1001110011100

------------------------
```

9 插件开发

自v0.580开始, Sowya 加入插件功能. 这里的插件暂时都以 `.dll` 的形式发布, 即只能在 Windows 系统下运行. 文件放置于 Sowya.exe 同级目录下的 `plugins` 目录下. 另一个 `myplugins` 是用户插件目录, 存放用户自定义的插件. 两者的区别是, `plugins` 目录下的插件会在 sowya.exe 启动时自动加载. 如果成功加载, 则显示

```
Pi loaded!

All modules have been loaded. Fine!
```

这里第一行 `Pi loaded!` 表明加载了一个名为`Pi`的插件.

9.1 plugin_pi

`plugin_pi.dll` 是用于计算 π 的插件, 这是发布的第一个插件. 具体代码见https://gitee.com/nooin_-195231242/plugin-pi.

9.2 插件的使用

在 `>>` 提示符下, 输入 `:dev_history` 可以查看最新加入的功能.

- `list_plugins()` – 列出所有插件的名称.
- `load_plugin(plugin_pi)` – 加载 `plugin_pi.dll`, 括号内即不带后缀的文件名
- `free_plugin(Pi)` – 卸载名为 Pi 的插件, 注意这里的插件名可能与插件的文件名不同. 插件名可以由 `list_plugins()` 列出.

我们检查一下自动加载的插件.

```
>> list_plugins()
[plugins] Auto-loaded plugins:
Pi
---Total: 1 ---


----------------
[myplugins] user plugins:
---Total: 0 ---

out> 0

------------------------
```

这里列举了自动加载的插件与用户加载的插件.

当sowya.exe启动时, 系统会自动加载 plugins 目录下所有 .dll 文件. 而 myplugins 目录中的插件(即.dll文件)则需要使用 load_plugin() 手动加载. 例如, 将 plugin_pi.dll 拷贝到 myplugins 目录, 在 >> 提示符下输入 load_plugin(plugin_pi) 则加载此插件.

```
>> load_plugin(plugin_pi)
Pi loaded!
out> 0

-----------------------
```

加载成功会显示 Pi loaded!.

9.2.1 插件的重命名

上面的 plugin_pi.dll 也可以改名为 Pi.dll, 然后键入 load_plugin(Pi) 就可以加载插件.

9.2.2 插件中函数的调用

注意这里输出了模块（插件）的名称 Pi. 这在调用其函数时要用到, 卸载（即使用 free_plugin() 释放）插件时也要用到. 例如: 输入 Pi->help() 将显示使用方法:

```
>> Pi->help()
[plugin] Pi
Usage:
        Pi->print()      -- print the Pi 314159265358979...1508984279927
        Pi->help()        -- print the usage.
        Pi->about()  -- About the author.
```

Pi->about() 一般显示作者信息以及插件介绍.

```
>> Pi->about()
[plugin] Pi
This programe is written by Dik T. Winter and Achim Flammenkamp. It can compute Pi to 15,000 decimal digits.
```

这里介绍了此插件的功能是计算 π. 算法来源于 Dik T. Winter 和 Achim Flammenkamp. 使用 Pi->print() 可打印, 共 15000 位.

为同时支持控制台应用程序Sowya.exe和GUI应用程序SowyaApp.exe, 插件中对于 print() 函数提供了对应的 Print(), 以及 Help() 和 About(). 小写的函数(例如: Pi->print(), Pi->help(), Pi->about())将直接打印结果到控制台终端, 适用于Sowya.exe; 而 Pi->Print(), Pi->Help(), Pi->About() 将结果以字符串返回, 适用于 SowyaApp.exe.

附录

附录

.1 开发历史

命令 `:dev_history` 可以列出开发历史.

```
>> :dev_history
Version: 0.441
Date: 24/11/2018
        exp(x)

Version: 0.442--0.443
Date: 04/01/2019
        log(x), ln(x), and fixed some bugs, such as operator==,
Now we can compare .9==0.9 and so on.

Version: 0.444
Date: 04/01/2019
        Variables can be defined.
        fix_variables(1)
        a="1+2/3"
        a=2+3*9
        :clear

Version: 0.445
Date: 08/03/2019
        Add two functions:
        binary2decimal()
        decimal2binary()

Version: 0.446
Date: 17/03/2019
        Add extra file in project, functions.h:
        which contains self defined functions.
        Fixed function sqrtn()

Version: 0.447
Date: 21/03/2019
        :change the function declaration IntDivision(string &, string &) to IntDivision(const string &, const string
      &)
        And change power operation to fast algorithm.

Version: 0.448
Date: 01/04/2019
        :Improved the function plus(), and add char ( in operators[], so can deal with expression a+b*c+(d*e+f)*g if a
      ,b,...,g are defined.

Version: 0.449
Date: 01/04/2019
        :Add function: solve(equation,x,minValue,maxValue).

Version: 0.450
Date: 06/06/2019
        exp(x)
```

```
47
48 fixed bug in computation IntDivision, that is: num1 % num2
49 add function decimal2octal(n)
50
51 add function decimal2hex(n)
52
53 add function firstFactor(n)
54
55 add function factorise(n)
56
57 Version: 0.451
58 Date: 27/07/2019
59 fixed bug of internal function ChangeInfix()
60 add function decimal2fraction(n)
61
62 add internal function isInteger(n)
63
64 add internal isPositiveInteger(n)
65
66 add internal isNegativeInteger(n)
67
68 add function !n=1!+2!+...+n!
69
70 Version: 0.452
71 Date: 27/08/2019
72 add function eq24(a,b,c,d)
73
74 Version: 0.453
75 Date: 06/09/2019
76 10%+10%
77 5%%3
78
79 Version: 0.454
80 Date: 19/09/2019
81 add function Collatz() and print3x1maps(a,b)
82
83 Version: 0.455
84 Date: 21/09/2019
85 add Lucas-Lehmer test for Mersenne primes in function isprime()
86 For example: isprime(2^89-1)
87
88 Version: 0.456
89 Date: 11/10/2019
90 Fixed some BUGs founded on 09-10-2019
91 For the previous versions, if you input the following :
92 >> p=3511
93 >> 2^(p-1)-1mod(p^2)
94
95 You will get 2^(3511-1)-1@(3511^2
96 That is, the last char ) is lost.
97
98 Version: 0.457
99 Date: 23/11/2019
100 Add function sum(general_term,n,min,max) on 09-10-2019
101
102 Version: 0.458
103 Date: 01/12/2019
```

```
104 Using hash function to deal with functions, add several functions, such as listfunctions(), help(funcName) and etc. on
          01-12-2019
105
106 Version: 0.459
107 Date: 15/12/2019
108 Fix bug: -(-2) and -3^2 and etc. on 15-12-2019
109
110 Version: 0.460
111 Date: 24/01/2020
112 Fix bug: a=3
113  a==3
114  and etc. on 24-01-2020
115
116 Set values: a=3
117  b=a
118  and etc. on 24-01-2020
119
120 Version: 0.461
121 Date: 21/02/2019
122 Fix bug: binom(5,5)=Inf on 21-02-2020
123
124 Version: 0.462
125 Date: 05/03/2020
126 Add exmpo() in function list. on 05-03-2020
127
128 Version: 0.463
129 Date: 11/03/2020
130 Fix BUG: sum(i!,i,1,20) on 11-03-2020
131
132 Version: 0.464
133 Date: 11/03/2020
134 Fix BUG: sqrt(x) does not work when x is not an integer on 24-03-2020
135
136 Version: 0.465
137 Date: 02/04/2020
138 Set default precision to 8, and make some functions do quick calculation when precision is high. Date April 2, 2020
139
140 Version: 0.466
141 Date: 14/04/2020
142 Add funtion getprecision() and fix bug of function setprecision() when no parameter is given. Date April 14, 2020
143
144 Version: 0.467
145 Date: 16/04/2020
146 Modify the command class so that print the functions list ordered by name. Date April 16, 2020
147
148 Version: 0.468
149 Date: 29/04/2020
150 Add command :mode, and enable do fraction calculation. Date April 29, 2020
151
152 Version: 0.469
153 Date: 08/07/2020
154 Make it possible to do the fraction calculation. Date July 08, 2020
155
156 Version: 0.470
157 Date: 10/07/2020
158 Make function sum() work under the fraction calculation mode. Date July 10, 2020
159
160 Version: 0.471
```

```
Date: 22/07/2020
Compare fractions. (Comparer les fractions.) And we make the computing x! possible. For example, 3.4!=3.4*2.4*1.4.
      Date July 22, 2020

Version: 0.472
Date: 25/07/2020
Fix a bug, the result of  x mod 1 should be zero.
And add function continued_fraction(num,expression,detail,tex). Date July 25, 2020

Version: 0.473
Date: 28/07/2020
Still use function continued_fraction() to compute continued fraction.
e.g., continued_fraction(3,5,1,1,2). Date July 25, 2020

Version: 0.474
Date: 01/08/2020
Add function Farey(n) to generate Farey series. Date August 01, 2020


==================

>> Version: 0.475
Date: 09/08/2020
Version 0.475 is the first version for register user. Date August 09, 2020

Version: 0.476
Date: 29/08/2020
Allow function sqrt() accept negative number. Date August 29, 2020

Version: 0.477
Date: 30/08/2020
Let function pi() return pi and pi(x) return the number of primes which is not greater than x. And add function
      list_pi_x(A,B) to print pi(x), where x is in [A,B]. Date August 30, 2020

Version: 0.478
Date: 01/09/2020
Fix bug in function q_DuanwuPlus(). Add function reverse() and reverse_plus(). If input '_' , then it will return the
      previous result. Date September 01, 2020

Version: 0.479
Date: 02/09/2020
Set system variables. Date September 02, 2020

Version: 0.480
Date: 06/09/2020
Permet au syst\'{e}me d'effecteur des op\'{e}rations symboliques. Date September 06, 2020

Version: 0.481
Date: 09/09/2020
Fix bugs in the symbolic operations such as +,-,*,/. Date September 06, 2020

Version: 0.482
Date: 22/10/2020
Change the infinity symbol Inf to string 'Inf' to avoid the error during arithmetic computation. Improved the
      operations including pre_power and etc. Date October 22, 2020

Version: 0.483
Date: 11/11/2020
```

```
Improved function eq24() such that it works under fraction mode. And improved the function fr_multiplication() and
      inputcmd2expression(). Date November 11, 2020

Version: 0.484
Date: 24/11/2020
Add functions: Primes(n), p(n) and index_of_prime(n). Date November 24, 2020

Version: 0.485
Date: 10/12/2020
Improved Factorise() and the calculation of n!. Try 10000! or  Factorise(10000!). Date December 10, 2020

Version: 0.486
Date: 18/12/2020
Fix bug in function Factorise(). Date December 18, 2020

Version: 0.487
Date: 24/12/2020
Fix greate bugs in symbol calculations. Date December 24, 2020

Version: 0.488
Date: 26/12/2020
Fix bugs in symbol calculations. Date December 26, 2020

Version: 0.489
Date: 27/12/2020
Fix bugs in symbol calculations. Date December 27, 2020

Version: 0.490
Date: 28/12/2020
Such expressions (x-1)(x+2) are allowed. Date December 28, 2020

Version: 0.491
Date: 14/03/2021
Add function Fibonacci(). Date March 14, 2021

Version: 0.492
Date: 21/03/2021
Improve the functions gcd() and lcm() and let them support multiparameters.
Fix bugs in functions help() and Factorise().
Date March 21, 2021

Version: 0.493
Date: 22/03/2021
Fix bugs in functions log() or ln().
Date March 22, 2021

Version: 0.494--0.499
Date: 12/04/2021
Improve functionality of Calculator.
Date April 12, 2021

Version: 0.494
Date: 11/05/2021
Improved the algorithm of computing !n
Date May 11, 2021

Version: 0.495
Date: 23/06/2021
```

```
Add Mobius function: mobius(x).
Date June 23, 2021

Version: 0.496
Date: 06/09/2021
Add function: printSeries(1/n,n,1,10).
Date September 6, 2021

Version: 0.497
Date: 12/10/2021
Add function: hex2decimal(0xABCDEF123456).
              octal2decimal(01234567).
              any2decimal(num,p)
Date October 12, 2021

Version: 0.498
Date: 28/11/2021
Add function: goldbach(n). Verify the Goldbach conjecture.
Date November 28, 2021

Version: 0.499
Date: 03/12/2021
Improved function: goldbach(n).
Date December 03, 2021

Version: 0.500
Date: 14/03/2022
Add function subset(N,m) on Pi Day.
Date March 14, 2022

Version: 0.501
Date: 06/07/2022
For convenience, we change the name of functions expmo() to expmod().
Date June 13, 2022
Fix function exp(x), here x may be negative.
Date July 4, 2022
Add function decimal2any(N, base). We also add functions for p-adic numbers, such that ord_p() and norm_p().
Date July 6, 2022

Version: 0.502
Date: 21/07/2022
Change if else statements to a switch statement for string.
Date July 21, 2022

Version: 0.503
Date: 24/07/2022
Matrix definition, A=[1,2,3;4,5,6;7,8,9].
Date July 24, 2022

Version: 0.504
Date: 25/07/2022
Fix some bugs.
Date July 25, 2022

Version: 0.505
Date: 28/07/2022
Add function EulerBrackets(), qid=2965.
Date July 28, 2022
```

```
330
331 Version: 0.506
332 Date: 29/07/2022
333 Fix some bugs in matrix inputting.
334 Date July 29, 2022
335
336 Version: 0.507
337 Date: 07/08/2022
338 Add function Factorial(n), return the standard decomposition of n!.
339 Date August 7, 2022
340
341 Version: 0.508
342 Date: 10/08/2022
343 Add function IndefiniteEquation(a,b), solve the indefinite equation ax+by=1.
344 Date August 10, 2022
345
346 Version: 0.509
347 Date: 11/08/2022
348 Fix bugs of IndefiniteEquation(a,b).
349 Date August 11, 2022
350
351 Version: 0.510
352 Date: 20/08/2022
353 We change the interface of the CalculatorApp by Using UPP’s Docking technology. And we start to write the manual of
        Calculator.
354 Date August 20, 2022
355
356 Version: 0.511
357 Date: 22/08/2022
358 Fix bug of fraction operation described in qid=2973.
359 Date August 22, 2022
360
361 Version: 0.512
362 Date: 24/08/2022
363 Update the manual of Calculator and add download link to update Calculator.
364 Date August 24, 2022
365
366 Version: 0.513
367 Date: 28/08/2022
368 Update the manual of Calculator and fix some bugs.
369 Date August 24, 2022
370
371 Version: 0.514
372 Date: 29/08/2022
373 Add dialog for register and add the backup serialization data as used in the DockingExample2 in U++ reference.
374 Date August 24, 2022
375
376 Version: 0.515
377 Date: 01/09/2022
378 Add two functions disp() and det().
379 Date September 1, 2022
380
381 Version: 0.516
382 Date: 02/09/2022
383 Fix bugs.
384 Date September 2, 2022
385
386 Version: 0.517
```

Date: 04/09/2022
Add function solve_n3plus_until_M3_eq_p3(n,M) to find the solutions of the equation:
n^3+(n+1)^3+...+M^3=q^3.
Date September 4, 2022

Version: 0.518
Date: 05/09/2022
Improve matrix input rules, see qid=2978.
Date September 5, 2022

Version: 0.519
Date: 09/09/2022
Fix mod function, we allow negative numbers. For example, -680mod47.
Date September 9, 2022

Version: 0.519
Date: 10/09/2022
Happy Mid-Autumn Festival!
September 10, 2022

Version: 0.520
Date: 11/09/2022
Fix the bug described in qid=2928, about the precedence of operators
Try (2*1+1)*1!*2^(2*1) and 9!3!.
Here 9!3! means 9!(3!).
Date September 11, 2022

Version: 0.521
Date: 13/09/2022
Solve the indefinite equation like 25x-13y+7z=4.
Try input ‘IndefiniteEquation(25,-13,7;4)‘.
Date September 13, 2022

Version: 0.522
Date: 18/09/2022
Solve the indefinite equations like x+2y+3z=4 and 7x1+4x2-2x3+3x4=2.
Try input ‘IndefiniteEquation(1,2,3;4)‘ or ‘IndefiniteEquation(7,4,-2,3;2)‘.
Date September 18, 2022

Version: 0.523
Date: 30/09/2022
Input matrix with elements in the form of expression.
Try input ‘[2^2-1,6+5*3,7-1/4;1/3,-1+2/3,0;1-3^2,-2,7+9/7]‘.
Date September 30, 2022

Version: 0.524
Date: 03/10/2022
Add function EulerVarphi(n), which return the amount of numbers that less than n and coprime to n.
Date October 03, 2022

Version: 0.525
Date: 16/11/2022
Make the function binom() support simple symbolic operations.
For example, binom(n,k) will return (n)!/((k)!*(n-k)!).
Date November 16, 2022

Version: 0.526
Date: 07/12/2022

```
445 Fix bug of IndefiniteEquation()
446 For the case IndefiniteEquation(a,b;c), it will use the same algorithm of IndefiniteEquation(a,b).
447 Date November 16, 2022
448
449 Version: 0.527
450 Date: 09/12/2022
451 Add a parameter in function sum(). Try sum(n^2,n,1,100) and sum(n^2,n,1,100,expand). The last parameter ‘expand‘ will
        print the expression.
452 Date December 09, 2022
453
454 Version: 0.528
455 Date: 16/12/2022
456 Add the operations of polynomials, such as addition, subtraction and multiplication.
457 First we switch to polynomial mode by typing ‘:mode=polyn‘, then type ‘(-4x^3+x^2)+(3x-x^6)*2x‘ and push Enter key.
458 Here we only deal with polynomial of variable x.
459 Date December 16, 2022
460
461 Version: 0.529
462 Date: 17/12/2022
463 Simplify the output of the result polynomial. Delete the zero terms(for example, 0x^2) in the polynomial.
464  ‘(x+1)*(x^2-x+1)‘ will return 1x^3+1x^0.
465 Date December 17, 2022
466
467 Version: 0.530
468 Date: 01/01/2023
469 Add division of polynomials.
470  For example ‘(x^3+x+1)/(x+1)‘ will return quotient: x^2-1x^1+2
471 remainder: -1.
472
473 Date January 01, 2023
474
475 Version: 0.531
476 Date: 03/01/2023
477 Add the power of the polynomial.
478  Try ‘(x^2+2)^3‘.
479
480 Date January 03, 2023
481
482 Version: 0.532
483 Date: 12/01/2023
484 Make handling large number coefficients of polynomial possible.
485  Try ‘(x^2+2)^100‘.
486
487 Date January 12, 2023
488
489 Version: 0.533
490 Date: 27/01/2023
491 Fix bug[qid=3056] of polynomial calculation. The coefficients are calculated by using fractions.
492  Try ‘(x^4+3x^3-x^2-4x-3)/(3x^3+10x^2+2x-3)‘.
493
494 Date January 27, 2023
495
496 Version: 0.534
497 Date: 30/01/2023
498 The function isprime(n) allows negative numbers as arguments.
499 Try ‘isprime(-1291)‘
500
501 Improve the algorithm of function solve_n3plus_until_m3_eq_p3().
```

Try ‘solve_n3plus_until_m3_eq_p3(10,100)‘

Date January 30, 2023

Version: 0.535
Date: 31/01/2023
Add function Legendre(a,p) to calculate the value of Legendre symbol (a|p), where a is coprime with p and p is a prime number.

Try ‘Legendre(438,593)‘

Date January 31, 2023

Version: 0.536
Date: 02/02/2023
Add function Jacobi(a,p) to calculate the value of Jacobi symbol (a|p), where a is coprime with p and p is an odd number.

Try ‘Jacobi(438,593)‘ or ‘Jacobi(438,593,show)‘

Date February 2nd, 2023

Version: 0.537
Date: 04/02/2023
Add function printRecursiveSeries(Iteration_expression, x, initialValue, N, delimiter).

Try ‘printRecursiveSeries(x/2,x,2,10,\n)‘
‘printRecursiveSeries((4*n-2)*h_n,h_n,1,10,\n)‘

Date February 4th, 2023

Version: 0.538
Date: 10/02/2023
Fix bug of function IndefiniteEquation().
Try ‘IndefiniteEquation(51,24;906)‘

Fix bugs in polynomial operations. Add function simplify(polynomial).
Try ‘simplify(x+2-3x+x^2-3x^2)‘.

Date February 10, 2023

Version: 0.539
Date: 11/02/2023
Fix bugs in polynomial operations. Add operations (poly1)%(poly2), (poly1)mod(poly2) and (poly1)==(poly2).
Make sure add to enclose parentheses around polynomials.

Try ‘(x^2000)%(x^4+x^3+2x^2+x+1)‘,
‘(x^2000)mod(x^4+x^3+2x^2+x+1)‘,
‘(x+x^2-3)==(-3+x^2+x)‘.

Add function ‘deg()‘ to get the degree of polynomial.
Try ‘deg(x^2-x^3+7-x^9)‘.

Date February 11, 2023

Version: 0.540

Date: 13/02/2023
Make the function decimal2binary() and binary2decimal() able to handle decimals.
Try ‘setprecision(100)‘
‘decimal2binary(2023.0213)‘.

Date February 13, 2023

Version: 0.541
Date: 16/02/2023
Fix bug of function printRecursiveSeries() and improve it.
Try ‘printRecursiveSeries((a_n+n+1)/(1+a_n*(n+1)),a_n,2,10,\n,linenumber)‘
‘printRecursiveSeries(a_{n+1}==(a_n+n+1)/(1+a_n*(n+1)),a_3=2,10,\n,linenumber)‘.

Date February 16, 2023

Version: 0.542
Date: 24/02/2023
Fix bug of function printRecursiveSeries().
Try ‘printRecursiveSeries(-5*I_k+1/(k+1),I_k,0.08,10,\n,linenumber)‘.

Date February 24, 2023

Version: 0.543
Date: 16/03/2023
Implement batching, i.e. read a file and execute the statements in it sequentially.
First write some expressions or functions in a text file, named for example test.sy, with content below.

```
a=2
a+3*8
:mode=polyn
(-4x^3+x^2)+(3x-x^6)*2x
:mode=numerical
A=[1,2;
3,-2]
B=[1 2 3
4;3,2,0
1;0,2,0,9;1,3,5,
-1]
EulerVarphi(20230316)
hex2decimal(0x134b0ac)
```

Save it and run the command:

```
        calculator.exe test.sy
```

We also can specify the output file by using the following way.

```
        calculator -f test.sy -o output.txt
```

Date March 16, 2023

Version: 0.544
Date: 24/03/2023
Add function Factorise_analysis(n) to decompose a positive integer into the product of two types of prime numbers. One type is equal to 1(mod 4), another is equal to 3(mod 4).
And compute some functions such as delta(n), epsilon(n) and xi(n).

Try Factorise_analysis((3 * 24 * 2023) ^ 2)

Change the internal logic of the mode.
Try ‘:mode help‘ .

Date March 24, 2023

Version: 0.545
Date: 25/03/2023
Add function Gcd(poly1, poly2) to compute the greatest common divsor(factor) of polynomials.
Try ‘Gcd(x^4+3x^3-x^2-4x-3,3x^3+10x^2+2x-3)‘

‘Gcd2(x^4+3x^3-x^2-4x-3,3x^3+10x^2+2x-3)‘

Add function monic_polyn() to transfer a polynomial to monic polynomial.

‘Try monic_polyn(3x^2+6x-9x+2)‘.

Date March 25, 2023

Version: 0.546
Date: 01/04/2023
Fix errors of function Gcd() and Gcd2(). Add the function diff() to differentiate polynomial function.

Try ‘diff(x^4+3x^3-x^2-9x+7)‘.

Date April 01, 2023

Version: 0.547
Date: 02/04/2023
We modified the sqrt() function so that it returns results of the form m*sqrt(n) in fractional calculation mode.

Try ‘:mode fraction
sqrt(972)‘.

Date April 01, 2023

Version: 0.548
Date: 06/04/2023
Improved the algorithm used by function listpseudoprimes(n). Now it runs fast.

Date April 06, 2023

Version: 0.549
Date: 09/04/2023
Fix bug in function printRecursiveSeries(). And we can type ‘:help funcName‘ or ‘:h funcName‘ to display the introduction of the function, just like input ‘help(funcName)‘.

Date April 09, 2023

Version: 0.550
Date: 11/04/2023
Fix bug in function diff(polyn). Update the function solve() and make it to solve equation like ax^2+bx+c==0.

Try ‘solve(x^2+7x+8==0)‘ or ‘solve(x^2==-7x-8)‘

We add the two functions solve() and eq24() to the basic version. And we rename Calculator to Sowya. But we will still use calculator for the name of the class inside this project.

Date April 11, 2023

```
671
672 Version: 0.551
673 Date: 19/04/2023
674 Make the function sqrtn() deal with the fraction. For example, Try ‘sqrtn(64/27,3)‘.
675 And fix bug in function sqrt().
676 We also make the function solve() can solve the equation like ax^3+bx^2+cx+d==0.
677
678 Try ‘solve(x^3-3x^2+2x^1==0)‘
679
680 Date April 19, 2023
681
682 Version: 0.552
683 Date: 01/05/2023
684 Fix bug of sum() function under fraction mode, Try
685
686         ‘:mode fraction‘
687         ‘sum(1/8∗((i-1)/8)^2,i,1,8)‘.
688
689 Date May 01, 2023
690
691 Version: 0.553
692 Date: 02/05/2023
693 Add the function of complex number operations. Try
694
695         ‘:mode complex‘
696         ‘(1+2i)+(3-4i)‘
697         ‘(1+2i)-(3-4i)‘
698         ‘(1+2i)∗(3-4i)‘
699         ‘(1+2i)/(3-4i)‘
700
701 Date May 02, 2023
702
703 Version: 0.554
704 Date: 07/05/2023
705 Add function inv(A) to compute the inverse matrix of A. Try
706
707         ‘A=[1,2,3;4,5,6;7,8,0]‘
708         ‘inv(A)‘
709         ‘:mode fraction‘
710         ‘inv(A)‘
711
712 Date May 07, 2023
713
714 Version: 0.555
715 Date: 04/06/2023
716 By using clox (http://www.craftinginterpreters.com/), we make programing in Sowya possible. Try
717
718         ‘:mode clox‘
719         ‘fun echo(){var n=900000000000000000; while(n<900000000000000009){print n∗n;n=n+1;}}‘
720         ‘echo();‘
721         ‘:q‘
722
723 Date June 04, 2023
724
725 Version: 0.556
726 Date: 08/06/2023
727 In mode clox, we can input multiple lines. Try
728
```

```
>>:mode clox
>{
fun echo(){
  var n=9;
  while(n<20){
    print 2^n;
    n=n+1;
  }
 }
}
>echo();
>:q

>>
```

Date June 08, 2023

Version: 0.557
Date: 11/06/2023
In mode clox, we add native function length() to return the length of string and function ith_char(s,i) to return the i-th char of string s.
Try

```
        >>:mode clox
        var x="Hello Sowya!";
        print length(x);
        print ith_char(x,6);
```

Date June 11, 2023

Version: 0.558
Date: 14/06/2023
In mode clox, we add native function avg() to calculate the average value of several numbers.
Try

```
        >>:mode clox
        print avg(12,3,4,5,98);
        var x=12;
        var y=29;print avg(x,y);
```

And we also add load file function. Suppose we write a function named avg(a,b) in the file avg.sy in the folder code.

Then try

```
        >>:mode clox
        load(code\avg.sy)
        print avg(2,3);
```

Date June 14, 2023

Version: 0.559
Date: 17/06/2023
In mode clox, we add native functions setprecision() and setmode().
Try

```
        >>:mode clox
        print 1/2+1/3;
        setprecision(20);
        print 1/2+1/3;
```

```
        setmode(1);
        print 1/2+1/3;

Date June 17, 2023

Version: 0.560
Date: 21/06/2023
In mode clox, we improved the print function. It will print newline when ‘\n‘ accurred.
Try

        >>:mode clox
        print 1/2+1/3;
        print "\n";
        var a=2;
        var b=3;
        var c=a*b;
        print a,"\n", b,"\n",c,"\n";

Also, we improved the variable declaration. Now we can define multiple variables in one line.
Try

        var a=2, b=3, c=a*b;



Date June 21, 2023

Version: 0.561
Date: 04/07/2023
In mode clox, we fix a bug that we can not run the native function avg. The reason is that we forget claim the global
      variable pA in class Calculator.

Date July 04, 2023

Version: 0.562
Date: 08/07/2023
A bug is fixed. Try

        :mode fraction
        printSeries((-1)^n*1/(2*n+1),n,0,10)
        (1/5+1/3)^(-2)
        (-1/3)^(3/2)


Date July 08, 2023

Version: 0.563
Date: 24/07/2023
Several functions are opened in the basic version. See which functions are available,

        listfunctions()

Add operations for the matrix elementary transformations. Try

        A=[1 2 3;4 5 6;7 8 0]
        transform(A, r2-r1*4)
        transform(A, r3-r1*7)
        transform(A, c1<-->c3)
```

```
        :mode fraction
        transform(A, c2*(-1/3))


Date July 24, 2023

Version: 0.564
Date: 25/07/2023
Make elementary transformation available for matrix with x. Try

        A=[5x,1,2,3;x,x,1,2;1,2,x,3;x,1,2,2x]
        transform(A,r1<-->r3)
        transform(A, r2-r1*x)
        transform(A, r3-r1*5x)
        transform(A, r4-r1*x)transform(A, r2*(-1))
        transform(A, r3*(-1))
        transform(A, r4*(-1))
        transform(A, r3-r2*10)
        transform(A, r4-r2*2)
        transform(A, r2+r3*x)
        transform(A, r3-r4)


Date July 25, 2023

Version: 0.565
Date: 30/07/2023
Fix bug in multiplication for fraction. Try

        A=[0,a,2,1;-a,0,d,3;-2,-d,0,1;-1,-3,-1,0]
        det(A)


Date July 30, 2023

Version: 0.566
Date: 31/07/2023
Add function transpose() to return the transpose matrix of the given matrix. For the above matrix A, Try

        transpose(A)


Date July 31, 2023

Version: 0.567
Date: 02/08/2023
Addition, subtraction and multiplication of matrices are implemented. Try

        A=[1 2;3 4]
        B=[-2 0;0 3]
        A+B
        A-B
        A*B
        A^2
        A+B*A


Date August 02, 2023
```

```
901
902 Version: 0.568
903 Date: 04/08/2023
904 Use unordered_map<> to store the variables.
905
906
907 Date August 04, 2023
908
909 Version: 0.569
910 Date: 05/08/2023
911 Fix a bug. In polyn mode, ‘x*x*x-3x*x+2x-5‘ will be transformed into ‘x^3-3x^2+2x-5‘. Try it.
912
913
914 Date August 05, 2023
915
916 Version: 0.570
917 Date: 07/08/2023
918 Add two functions CofactorMatrix(A,i,j) and DetExpansion(A,r2). The first returns the (i,j)-th element of matrix A.
      The second will expand matrix A along ri(row i) or cj(column j).
919
920 Try
921
922         A=[1,0,-2;1,1,3;-2,3,1]
923         CofactorMatrix(A,2,3)
924         DetExpansion(A,r1)
925         DetExpansion(A,c2)
926
927 Date August 07, 2023
928
929 Version: 0.571
930 Date: 10/08/2023
931 Add E() function to generate elementary matrix. E(n,i,j(k)) will generate the elmentary matrix E(i,j(k)) of order n.
      And E(n,i(k)) will generate the elementary matrix E(i(k)) of order n.
932 Try
933         E(4,2,3(5))
934         E(5,3(2))
935         :list
936
937 Date August 10, 2023
938
939 Version: 0.572
940 Date: 19/08/2023
941 Add rank() function to calculate the rank of matrix.
942 Try
943         A=[1 3 6;1 2 3;1 4 9]
944         rank(A)
945         rank(A,hint)
946
947 Date August 19, 2023
948
949 Version: 0.573
950 Date: 25/08/2023
951 Fix a bug in rank() function. To return the rank of matrix A, simply input rank(A). If we want to calculator the row
      rank of A, then input rank(A,row), which will use elementary row transformation of A to calculate the rank. Of
      course, you can input rank(A,col) to return the column rank of A. While rank(A,hint) also return the rank of A,
      it will use elementary row transformation when the rows of A is less or equal than the columns of A, and use the
      elementary column transformation when the rows of A is greater than the columns of A.
952
```

```
Date August 25, 2023

Version: 0.574
Date: 29/08/2023
We extend functionality to function eq24(). Also we add aliases ‘s24()‘ and ‘suo()‘.

Try
        eq24(8,6,4,4)
        s24(11,10,5,1;24)
        suo(8,6,6,4;28)

Date August 29, 2023

Version: 0.575
Date: 01/09/2023
We extend functionality to function solve() to solve linear equations.

Try
        A=[1,1,-3,-1;3,-1,-3,4;1,5,-9,-8]
        b=[1;4;0]
        solve(A*x==b)

Date September 01, 2023

Version: 0.576
Date: 02/09/2023
Try
        A=[1,1,-3,-1;3,-1,-3,4;1,5,-9,-8]
        b=[1;4;0]
        solve(A*x==b,normal)
        solve(A*x==b,hint)
        solve(A*x==b,normal,hint)

Date September 02, 2023

Version: 0.577
Date: 09/10/2023
We add the ispolyn() function to check if the input string is a polynomial of x.
Try
        ispolyn(3*(9+2)x^3)

Date October 09, 2023

Version: 0.578
Date: 21/10/2023
Adapt to SciTE so that *.sy , *.sc files can be run in SciTE. where *.sy is a sowya file, which can be executed line
      by line by sowya.exe; And *.sc is a sowya clox programming file and can be executed by sowya.exe.
Date October 21, 2023

Version: 0.579
Date: 26/10/2023
When switching to clox mode, Sowya will automatically load the files in the ‘ext‘ directory. If this directory does
      not exist, it will create it. Now we can put the code of the custom function into the ‘ext‘ directory to enhance
      the functionality of Sowya.
Date October 26, 2023

Version: 0.580
Date: 14/12/2023
```

We add plugin system.
        list_plugins()   -- (list the names of all plugins)
        load_plugin(Pi) -- (load the plugin named Pi)
        free_plugin(Pi) -- (unload the plugin named Pi)

For example, when we load Pi plugin, we can type ‘Pi->print()‘ to print 15,000 decimal digits of Pi.
The plugin file is .dll file which is located in plugins or myplugins directory.
Date December 14, 2023

Version: 0.581
Date: 21/12/2023
Add sine function. Try:
        sin(1)
        sin(1.2)

Date December 21, 2023

Version: 0.582
Date: 25/12/2023
Add functionality to the sine function. Try:
        sin(1.5pi)
        sin(2/3pi)

Date December 25, 2023

Version: 0.583
Date: 29/12/2023
Add function deg2rad(). Try:
        deg2rad(15)

Date December 29, 2023

Version: 0.585
Date: 5/1/2024
Add cosine function. Try:
        cos(1)
        cos(1.5pi)
        cos(30d)

Date January 5, 2024

Version: 0.586
Date: 7/1/2024
Add functions: ceil() and floor(). Try:
        ceil(1.23)
        floor(-1.5)

Date January 7, 2024

Version: 0.587
Date: 9/1/2024
Add functions tau(m) and biggest_factor(m). tau(m) will return the number of all factors of positive integer m. And biggest_factor(m) returns the biggest factor less than m. If m is a prime then biggest_factor(m) returns m itself. For example, 2024 == 2^3∗11∗23, then tau(2024)=(3+1)∗(1+1)∗(1+1)=16. Try:
        tau(2024)
Date January 9, 2024

Version: 0.588

Date: 13/1/2024
Add function multiplicativeOrder(a,m), which will return the multiplicative order of a of modular m, if (a,m)=1. Try:
        multiplicativeOrder(7,2024)
Date January 13, 2024

Version: 0.589
Date: 16/1/2024
We add the function sort(), which will sort several numbers (including decimals, integers or fractions). Try:
        sort(1,28,28,14,14,14,7,28,14,28,28,4,14,28,28,7,4,28,28,7,28,14,7,7,7,28,28,2)
        sort(1,28,28,14,14,14,7,28,14,28,28,4,14,28,28,7,4,28,28,7,28,14,7,7,7,28,28,2;desc)
        sort(1,28,28,14,14,14,7,28,14,28,28,4,14,28,28,7,4,28,28,7,28,14,7,7,7,28,28,2;string|desc)
        sort(2,1/2,1/3,2/3,20.3,7|6)
        :mode fraction
        sort(2,1/2,1/3,2/3,20.3,7|6)
Date January 16, 2024

Version: 0.590
Date: 6/2/2024
Try:
        A=[1,1/2;-3.2,4]
        A.rank()
        A.det()
        A.sort()

Date February 6, 2024

Version: 0.591
Date: 23/3/2024
Add function InfixToPostfix() to convert infix expression to postfix expression. Try:
        InfixToPostfix(a+b*c+(d*e+f)*g)
Date March 23, 2024

Version: 0.592
Date: 31/3/2024
Fix some bugs in computation of polynomial. See qid=3067.
Add the command :change to change the main variable in polynomial. When we input :mode polyn, sowya is in polyn mode, ‘x‘ is the default variable. If we want to deal with polynomial f(t), we can input :change x t to change the main variable ‘x‘ to ‘t‘.
        Try:
                :change x->t
                :change x=t

Add the commands :delete and :del to remove variable in poly_vars, where poly_vars is a map used to store the variables used in the multivariate polynomial.
We use symbol(u2) to add variable ‘u2‘ to the poly_vars. If we wang to delete all the variables in poly_vars, use :clear poly_vars

        Try
                symbol(y)
                symbol(what)
                symbol(z)
                :poly_vars
                :del what
Date March 31, 2024

Version: 0.593
Date: 8/4/2024
Improvements to the symbol() function.

```
1118
1119 Try:
1120         symbol(x,y,z,what)
1121         :poly_vars
1122         symbol(a:2)
1123         symbol(b:3,c:4)
1124         :list
1125 Date April 8, 2024
1126
1127 Version: 0.594
1128 Date: 15/4/2024
1129 Support addition and subtraction of multivariate polynomials.
1130
1131 Try:
1132         :mode polyn
1133         symbol(x,y,z)
1134         x^2-y^3+zy+6x^2
1135         x^2-y^3+zy+6x^2+7y^3-zy
1136 Date April 15, 2024
1137
1138 Version: 0.595
1139 Date: 22/4/2024
1140 Fix some bugs in computation of polynomials.
1141
1142 Try:
1143         :mode polyn
1144         x^1.2
1145         x^(1.2+2.3)+x^2
1146 Date April 22, 2024
1147
1148 Version: 0.596
1149 Date: 23/4/2024
1150 Fix some bugs in computation of polynomials.
1151
1152 Try:
1153         :mode polyn
1154         (x+2+x^2)/0.2
1155 Date April 23, 2024
1156
1157 Version: 0.597
1158 Date: 4/5/2024
1159 Support multiplication of multivariate polynomials.
1160
1161 Try:
1162         :mode polyn
1163         symbol(y)
1164         (2x-1)(y+2)
1165         (x+y)(x+y)(x+y)(x+y)(x+y)
1166 Date May 4, 2024
1167
1168 Version: 0.598
1169 Date: 5/5/2024
1170 Fix some bugs in doing multiplication of the multivariate polynomials. Power operations on multivariate polynomials
          are supported.
1171
1172 Try:
1173         :mode polyn
1174         symbol(a,b,c,d,y)
```

```
        (y-x)*2
        (a+b)(c+d)
        (x+y)^5
        (a+b+c)^2
Date May 5, 2024

Version: 0.599
Date: 9/5/2024
Modify internal function dealWithInputCommand() in Calculator.cpp.
Date May 9, 2024

Version: 0.600
Date: 13/5/2024
Fix bug of Fibonacci() function. Try
        fibonacci(1).
Add function limit() or lim() to calculate the limit of a rational function at a certain point.

Try:
        limit(((x^3-1)(x+1))/(x-1),x,1)

Date May 13, 2024

Version: 0.601
Date: 20/5/2024
Add function isprime() in clox mode. That is we can check prime number in programs.

Try:
        :mode clox
        load(code/n2plus4.sy)
        println n2plus4(100);


Date May 20, 2024

Version: 0.602
Date: 21/5/2024
Fix dealWithInputCommand() in myFunctions.h


Date May 21, 2024

Version: 0.603
Date: 23/5/2024
Fix the determinant calculation of the matrix with letters. See qid=3317.

        Try:
        A=[x-3,-5;5,x-3]
        det(A)
        B=[0,1,1,1;1,0,x,x;1,x,0,x;1,x,x,0]
        det(B)

Date May 23, 2024

Version: 0.604
Date: 31/5/2024
Fix the internal functions isEqualTo(), gcd() and solve_eqn_of_deg2().

        Try:
```

```
        solve(x^2+x+1==0)
        solve(2x^2+10==0)

Date May 31, 2024

Version: 0.605
Date: 16/6/2024
Fix the solve_eqn_of_deg2() and add native function binom() in clox mode.

        For example:
        https://www.zhihu.com/question/658915012/answer/3531071669

Date June 16, 2024

Version: 0.606
Date: 22/6/2024
Fix a bug in printRecursiveSeries(). See qid=3331. Support simple derivative operations.

        For example:
        diff(exp(sin(x^2+x+1)))

Date June 22, 2024

Version: 0.607
Date: 15/7/2024
Add C style comment /* */ in clox mode, see clox/scanner.c. For example:
        println /* sowya is ...*| /* * **/ "hello sowya";

Date July 15, 2024

Version: 0.608
Date: 3/8/2024
Fix bug in production of polynomials.

        :mode polyn

        symbol(a,b,c)

        (a+b+c)(a+b-c)(a-b+c)(-a+b+c)

Date August 3, 2024

Version: 0.609
Date: 9/8/2024
Fix some bugs.

        :mode polyn
        (3x^3)/(2x^2)
        (3x)/(2x^2)
        1/x

Date August 9, 2024

Version: 0.610
Date: 20/8/2024
Fix some bugs.

        :mode polyn
```

```
        symbol(x,y)
        (x+y-1)(x+y+1)

Date August 20, 2024

Version: 0.611
Date: 7/9/2024
Add factorial() in clox mode.

        :mode clox
        println factorial(9);
        println factorial(9,6)

Date September 7, 2024

Version: 0.612
Date: 19/9/2024
Add function LagrangePolyn().

        LagrangePolyn(1,1;4,2;9,3)

Date September 19, 2024

Version: 0.613
Date: 8/10/2024
Update the function sin().

        setprecision(50)
        sin(50)

Date October 08, 2024

Version: 0.614
Date: 12/10/2024
Improve the function transform().

        D1=[4,2,-1,1;6,3,-1,2;12,5,-3,4;16,3,-2,2]
        transform(D1,-2*c1)
        transform(D1,c1-8*c3)

Date October 12, 2024

Version: 0.615
Date: 13/10/2024
Fix bug since v0.568. Improved function inputcmd2expr() in source file Calculator.cpp.

        a=2
        b=3
        c=6
        (a+b)*c

Date October 13, 2024

Version: 0.616
Date: 02/11/2024
Add sine and cosine functions in clox mode.

        :mode clox
```

```
1349         setprecision(50);
1350         print sin(50);
1351         print cos(50);
1352
1353 Date November 02, 2024
1354
1355 Version: 0.617
1356 Date: 11/11/2024
1357 Add :claim to make a claim.
1358
1359         :claim a==b
1360         a==b
1361         (a-b)^2
1362         :mode polyn
1363         symbol(a,b)
1364         (a-2-b)^2
1365
1366 Date November 11, 2024
1367
1368 Version: 0.618
1369 Date: 17/11/2024
1370 When :claim sin(2x)=2sin(x)*cos(x), it expands the expression of sin(2x).
1371
1372         :claim sin(2x)=2sin(x)*cos(x)
1373         sin(2t)
1374         sin(x-t)
1375         sin(9-x2)
1376         sin(2(x-a))
1377         Try ‘sin(2sin(x))‘.
1378
1379 Date November 17, 2024
1380
1381 Version: 0.619
1382 Date: 27/11/2024
1383 Add EulerVarphi() in clox mode.
1384
1385         :mode clox
1386         print EulerVarphi(9);
1387
1388 Date November 27, 2024
1389
1390 Version: 0.620
1391 Date: 05/12/2024
1392 For functions printSeries(), when the last parameter is \null, then the delimiter is the empty string ’’.
1393
1394 Try,
1395         printSeries(n+1,n,0,10,\null)
1396
1397 Date December 05, 2024
1398
1399 Version: 0.621
1400 Date: 18/12/2024
1401 In clox mode, we add two functions: Shorten(string, Nd) and truncate(number, Nd), where N is a positive integer.
1402
1403 Try,
1404         :mode clox
1405         print Shorten("Hello Sowya",8d);
1406         print truncate("Hello Sowya",8d);
```

```
        print truncate(1.532088886237956070404785301110833347871664914160791110362546,10d);

Date December 18, 2024

Version: 0.622
Date: 22/12/2024
Add parameter for Primes() function.

Try,
        Primes(100)
        Primes(100,true)
        Primes(100,false)
        Primes(1000,yes)

Date December 22, 2024

Version: 0.623
Date: 26/12/2024
We can assign a variable to another.

Try,
        A=[1 2;3 4]
        B=A
        B

Date December 26, 2024

Version: 0.624
Date: 17/01/2025
Fix bug [qid=3374].
Date January 17, 2025

Version: 0.625
Date: 20/01/2025
Enhance the operation of unary polynomials to enable the processing of four-rule operations of rational functions.
For example:
        :mode polyn
        x/(x+1)+1/x
        1/x/x
        (1/x)*x
Date January 20, 2025

Version: 0.626
Date: 06/03/2025
Using function QR(A), we can perform a QR decomposition of an invertible matrix A.
For example:
        A=[1 2 2;2 1 2;1 2 1]
        QR(A)
Date March 06, 2025

Version: 0.627
Date: 27/03/2025
Add ln() function in clox mode.
And we fixed a lot of compilation warnings.

Date March 27, 2025

Version: 0.628
```

Date: 02/04/2025
Implement a linear programming solver. Try the following examples. Every example is ended with a semicolon.

max z=5x_1+3x_2,
-2x_1+x_2<=5,
x_1+x_2<=7,
x_1+2x_2<=10,
x_1>=0,x_2>=0;

max z=2x_1+3x_2, 2x_1+2x_2<=12, x_1+2x_2<=8, 4x_1<=16, 4x_2<=12, x_1>=0,x_2>=0;

Date April 02, 2025

Version: 0.629
Date: 22/04/2025
Add exp() function in clox mode.
We add a function named PrimitivePolyn() to determine whether a given polynomial with integer coefficients is a primitive polynomial.

  Try PrimitivePolyn(2x^5-3x^3+4x^2)

Date April 22, 2025

Version: 0.630
Date: 24/04/2025
Function Eisenstein() Use Eisenstein’s discriminant method to determine whether a polynomial is irreducible.

  Try Eisenstein(x^2+2)

Date April 24, 2025

Version: 0.631
Date: 30/04/2025
Fix bug in multiplication.

Try
        2*3^n
        J=[3^n n*3^(n-1);0 3^n]
        A=[1 2;0 2]
        A*J

Date April 30, 2025

Version: 0.632
Date: 05/05/2025
Fix bug in sqrtn(a,n).

Try
        14!
        sqrtn(87178291200,14)

Date May 05, 2025

Version: 0.633
Date: 15/05/2025
Add tan() and cot() functions. Fix some bugs in clox mode.

```
Try
        :mode clox
        print sqrt(4)+sqrt(9);

Date May 15, 2025

Version: 0.634
Date: 09/06/2025
Fix some bugs in clox mode.

Try
        :mode clox
        print sin(cos(1));

Date June 09, 2025

Version: 0.635
Date: 13/06/2025
Add function pi() in clox mode.

Try
        :mode clox
        print sqrt(2)/2+tan(1)-pi()/2-ln(2);
        print pi(20);

Date June 13, 2025

Version: 0.636
Date: 16/06/2025
Use the virtual machine (VM) of clox to perform calculations.

Try
        sqrt(2)/2+tan(1)-pi()/2-ln(2)
        pi(20)
        1+sqrt(4)+5^2

Date June 16, 2025

Version: 0.637
Date: 20/06/2025
Recall we have fixed function ConvertInfixToPostfix() on 2024-11-17 in v0.617.
And now we no longer need InsertCommaInPostfix().

Try
        A=[1 2;3 4]
        B=[3 4;2 1]
        A*B-B*A

Date June 20, 2025

Version: 0.638
Date: 22/06/2025
Fix :change command. And fix bug relate to poly_varnames since v0.625. (49|4x^2-49x+49)*(4|49) will return (196x^2-784
     x^1+784)/(196).
Now it will return x^2-4x^1+4

Try
        (49|4x^2-49x+49)*(4|49)
```

```
        (x-(4))/(1-(4))*(x-(9))/(1-(9))

        :mode polyn
        symbol(x,y,z)
        x^2-y^3+zyx+6x^2+7y^3-xyz

Date June 22, 2025

Version: 0.639
Date: 07/07/2025
Fix functions sin(), cos(), tan() and cot().
If the parameter is of the form 30d, then it means 30 degree.

        Here d does NOT stand for double.

Date July 7, 2025

Version: 0.640
Date: 08/07/2025
Allow function gcd2() to accept multiple parameters.
Try

        gcd2(27,6,9)

Date July 8, 2025

Version: 0.641
Date: 09/07/2025
Add a welcome page at startup. Type ':home' will return to the start page. And fix bug in function Gcd(). Change the
     name of function s24() to suo24().
Try

        :sowya
        Gcd(x^2-1,x^3-1,x^4-1)

Date July 9, 2025

Version: 0.642
Date: 12/07/2025
Improve the functions isprime() and ispseudoprime().
Try
        :h isprime
        isprime(20250712)
        isprime(127,29,31)
        isprime(202507127,127,72172017,"hint")

        :h ispseudoprime
        ispseudoprime(341)
        ispseudoprime(341,561,654)
        ispseudoprime(456,561,645,"hint")

Date July 12, 2025

Version: 0.643
Date: 13/07/2025
Fix bugs in multi-variable polynomial operations. (Add checkUndefinedVariables() in polyMulti.cpp)
Try
        symbol(z,x_1,x_2)
```

```
        :poly_vars
        :mode polyn
        x_2+2x_3+x_4


        It will check the undefined variables like x_3 and x_4.

Date July 13, 2025

Version: 0.644
Date: 14/07/2025
Delete A from BigNumber.cpp. Eliminate some warnings during compilation. For example, for building under windows, we use fopen_s, strcpy_s and strcat_s instead of fopen, strcpy and strcat respectively.

Date July 14, 2025

Version: 0.645
Date: 15/07/2025
Simplify the BigNumber class, moving the polynomial-related functions to polyn.cpp and polynMulti.cpp. Transfer some basic functions from the BigNumber class and myFunctions.h to tools.h.

Date July 15, 2025

Version: 0.646
Date: 15/07/2025
Move the contents of sin2x.h to BigNumber.h. And This is the new importent version of Sowya.

Date July 15, 2025

Version: 0.647
Date: 21/07/2025
Fix some bugs related the index of array out of range. Try
        pi=1
        pi+1

Here, pi is recognized as an internal function. So we add a mode named ‘cmd‘. In this mode, the expression will be interpreted and executed by Sowya.
Try
        :mode cmd
        pi=1
        :list
        pi+2
Date July 21, 2025

Version: 0.648
Date: 29/07/2025
Add functions polyn_items() and multipolyn_items(). It will split a polynomial into items and print them.
Try
        polyn_items(2x^2-3+4x-5x^3+6x^2+7x-8)
        multipolyn_items(2xy^2+xy-y^2+2)

Date July 29, 2025

Version: 0.649
Date: 01/08/2025
Fix a bug related to calculation in clox mode. For example, 1/99^3 will work in old version of Sowya. But failed in version v0.648.
Try
```

```
        setprecision(10000)
        1/99^3

Date August 1, 2025

Version: 0.650
Date: 01/09/2025

A diff() function has been added, used for taking derivatives.
Try
        diff(f(g(x^2,x),x))
        diff(sin(2x)+cos(f(x^2)))

Date September 1, 2025

Version: 0.651
Date: 03/09/2025

Fix bug in the derivative function diff().
Try
        diff(f(t)*x)
        diff(sin(2x)+cos(f(t^2)))

Date September 3, 2025

Version: 0.652
Date: 06/09/2025

Fix bug in the derivative function diff().
Try
        diff(cos(f(x)))
        diff(sin(2x)+cos(f(t^2)))
        :change x=t
        diff(sin(2x)+cos(f(t^2)))
Date September 6, 2025

Version: 0.653
Date: 14/09/2025

Fix bug in the derivative function diff().
Try
        diff(sin(x)*cos(x))
        diff(cos(x^2)*cos(x^2))
Date September 14, 2025

Version: 0.654
Date: 16/09/2025

Fix bug in the derivative function diff().
Try
        diff(u(x)/v(x))
Date September 16, 2025

Version: 0.655
Date: 21/09/2025

Fix bug in the Calculator::EvalMatrixPostfixExpression().
And we add the derivatives of the functions tan(), cot(), sec(), csc() and sqrt().
```

```
Try
        A=[3 1;0 2]
        A^2
        diff(tan(x))
        diff(cot(x)*x)
        diff(sec(x))
        diff(csc(x))
        diff(sqrt(f(x)))

Date September 21, 2025

Version: 0.656
Date: 28/09/2025
Add the derivatives of the functions arcsin(), arccos(), arctan(), arccot(), sinh(), cosh(), tanh() and coth().
Try
        diff(arcsin(x))
        diff(arccos(x))
        diff(arctan(x))
        diff(arccot(x))
        diff(sinh(x)+cosh(x))

Date September 28, 2025

Version: 0.657
Date: 30/09/2025
Fix BUG[20250930], and add factorial operation in clox mode.
Try
        diff(f(x,x))
        diff(f(x,g(x)))
        diff(f(g(x,x^2),x))
        9!3

Date September 28, 2025

Version: 0.658
Date: 14/10/2025
Fix BUG[20251014-1]. To check, please calculate the determinant of the following matrix and compare it with the result
      of this polynomial -(a-b)(a-c)(a-d)(b-c)(b-d)(c-d)(a+b+c+d).

Try     A=[1,a,a^3,bcd;1,b,b^3,acd;1,c,c^3,abd;1,d,d^3,abc]
        symbol(a,b,c,d)
        :mode polyn
        det(A)
        ‘Copy the result into the input and press Enter, and we will get the simplified result of det(A).‘
        -(a-b)(a-c)(a-d)(b-c)(b-d)(c-d)(a+b+c+d)
        ‘Make the subtraction of the two expressions, the result should be zero.‘

Fix BUG[20251014-2]. To check, please calculate the determinant of the following matrix.

Try     B=[1,a,b,c,d;-a,1,e,f,g;-b,-e,1,h,i;-c,-f,-h,1,j;-d,-g,-i,-j,1]
        symbol(a,b,c,d,e,f,g,h,i,j)
        :mode polyn
        det(B)
        ‘Copy the result into the input and press Enter, and we will get the simplified result of det(B).‘

Date October 14, 2025
```

```
1805 ------------------
```

.2 SciTE

下面是文件 `sowya.properties` 中的代码.

```
1  # Define SciTE settings for Sowya files.
2
3  file.patterns.sowya=*.sy
4  file.patterns.clox=*.sc
5
6  *source.patterns.sowya=$(file.patterns.sowya);
7  *source.patterns.clox=$(file.patterns.clox);
8
9  shbang.sowya=sy
10 shbang.clox=sc
11
12 filter.sowya=sowya (sy)|$(file.patterns.sowya)|
13 filter.clox=sowya (sc)|$(file.patterns.clox)|
14
15 *filter.sowya=$(filter.sowya)
16 *filter.clox=$(filter.clox)
17
18 *language.sowya=Sow&ya|sy|F9|
19 *language.clox=clo&x|sc|F10|
20
21 lexer.$(file.patterns.sowya)=sowya
22 lexer.$(file.patterns.clox)=clox
23
24 keywords.$(file.patterns.sowya)=\
25 :help :h :more :dev_history :clc :thanks :about :exit :quit :q :version :mode \
26 clox numerical fraction polyn mod \
27 CofactorMatrix Collatz DetExpansion E EulerBrackets EulerVarphi \
28 Factorial Factorise Factorise_analysis Farey Fibonacci Gcd Gcd2 \
29 IndefiniteEquation Jacobi Legendre Primes any2decimal \
30 binary2decimal binary2graycode binom continued_fraction \
31 cubicsum cubicsum2 decimal2any decimal2binary decimal2fraction \
32 decimal2hex decimal2octal decimalToFraction deg det diff disp \
33 eq24 exp expmod factorial factorise farey fibonacci firstfactor fix_variables \
34 gcd gcd2 getValueFromMemAddr getprecision goldbach graycode2binary \
35 help hex2decimal index_of_prime inv iscomposite ispolyn isprime \
36 ispseudoprime lcm len list_pi_x listfunctions listpseudoprimes ln log \
37 mobius monic_polyn nextMRprime nextprime norm_p octal2decimal \
38 ord_p p pi prevMRprime prevprime print3x1Maps printRecursiveSeries \
39 printSeries q_DuanwuPlus q_duanwu rank register reverse reverse_plus \
40 round s24 setprecision simplify solve solve_n3plus_until_m3_eq_p3 \
41 solve_x3plusy3_zn sqrt sqrtn squaresum subset sum suo symbol \
42 transform transpose trimzeros xor
43
44
45 keywords.$(file.patterns.clox)=\
46 print println var nil avg ith_char fun echo \
47 while return if length and
48
49
```

```
comment.block.clox=#~
block.start.$(file.patterns.clox)=10 {
block.end.$(file.patterns.clox)=10 }

word.characters.$(file.patterns.sowya)=$(chars.alpha)$(chars.numeric)_$@%&

comment.block.sowya=#~
block.start.$(file.patterns.sowya)=10 {
block.end.$(file.patterns.sowya)=10 }

#colour.sowya.heredoc=$(colour.embedded.comment)

#fold.sowya.package=1
#fold.sowya.pod=1


#fold.sowya.at.else=1
#fold.sowya.comment.explicit=0

# sowya styles
# Default
style.sowya.32=$(font.base)
# White space
style.sowya.0=fore:#808080
# Comment: /* */.
style.sowya.1=$(colour.code.comment.box),$(font.code.comment.box)
# Line Comment: //.
style.sowya.2=$(colour.code.comment.line),$(font.code.comment.line)
# Doc comment: block comments beginning with /** or /*!
style.sowya.3=$(colour.code.comment.doc),$(font.code.comment.doc)
# Number
style.sowya.4=$(colour.number)
# Keyword
style.sowya.5=$(colour.keyword),bold
# Double quoted string
style.sowya.6=$(colour.string)
# Single quoted string
style.sowya.7=$(colour.char)
# UUIDs (only in IDL)
style.sowya.8=fore:#804080
# Preprocessor
style.sowya.9=$(colour.preproc)
# Operators
style.sowya.10=$(colour.operator),bold
# Identifiers
style.sowya.11=
# End of line where string is not closed
style.sowya.12=fore:#000000,$(font.monospace),back:#E0C0E0,eolfilled
# Verbatim strings for C#
style.sowya.13=fore:#007F00,$(font.monospace),back:#E0FFE0,eolfilled
# Regular expressions for JavaScript
style.sowya.14=fore:#3F7F3F,$(font.monospace),back:#E0F0FF,eolfilled
# Doc Comment Line: line comments beginning with /// or //!.
style.sowya.15=$(colour.code.comment.doc),$(font.code.comment.doc)
# Keywords2
style.sowya.16=fore:#B00040
# Comment keyword
style.sowya.17=fore:#3060A0,$(font.code.comment.doc)
```

```
# Comment keyword error
style.sowya.18=fore:#804020,$(font.code.comment.doc)
# Global class
style.sowya.19=fore:#DD9900
# Raw strings for C++0x
style.sowya.20=$(colour.string),back:#FFF3FF,eolfilled
# Triple-quoted strings for Vala
style.sowya.21=$(font.monospace),fore:#007F00,back:#E0FFE0,eolfilled
# Hash-quoted strings for Pike
style.sowya.22=$(font.monospace),fore:#007F00,back:#E7FFD7,eolfilled
# Preprocessor stream comment
style.sowya.23=fore:#659900
# Preprocessor stream doc comment
style.sowya.24=$(colour.code.comment.doc)
# User defined literals
style.sowya.25=fore:#C06000
# Task Marker
style.sowya.26=fore:#BE07FF,$(font.code.comment.line)
# Escape sequence
style.sowya.27=$(colour.string)

# Inactive states are 64 greater than their active counterparts

# White space
style.sowya.64=fore:#C0C0C0,$(traits.inactive)
# Comment: /* */.
style.sowya.65=$(style.sowya.1),fore:#90B090,$(traits.inactive)
# Line Comment: //.
style.sowya.66=$(style.sowya.2),fore:#90B090,$(traits.inactive)
# Doc comment: block comments beginning with /** or /*!
style.sowya.67=$(style.sowya.3),fore:#D0D0D0,$(traits.inactive)
# Number
style.sowya.68=$(style.sowya.4),fore:#90B0B0,$(traits.inactive)
# Keyword
style.sowya.69=$(style.sowya.5),fore:#9090B0,$(traits.inactive)
# Double quoted string
style.sowya.70=$(style.sowya.6),fore:#B090B0,$(traits.inactive)
# Single quoted string
style.sowya.71=$(style.sowya.7),fore:#B090B0,$(traits.inactive)
# UUIDs (only in IDL)
style.sowya.72=$(style.sowya.8),fore:#C0C0C0,$(traits.inactive)
# Preprocessor
style.sowya.73=$(style.sowya.9),fore:#B0B090,$(traits.inactive)
# Operators
style.sowya.74=$(style.sowya.10),fore:#B0B0B0,$(traits.inactive)
# Identifiers
style.sowya.75=$(style.sowya.11),fore:#B0B0B0,$(traits.inactive)
# End of line where string is not closed
style.sowya.76=$(style.sowya.12),fore:#000000,$(traits.inactive)
# Verbatim strings for C#
style.sowya.77=$(style.sowya.13),fore:#90B090,$(traits.inactive)
# Regular expressions for JavaScript
style.sowya.78=$(style.sowya.14),fore:#7FAF7F,$(traits.inactive)
# Doc Comment Line: line comments beginning with /// or //!.
style.sowya.79=$(style.sowya.15),fore:#C0C0C0,$(traits.inactive)
# Keywords2
style.sowya.80=$(style.sowya.16),fore:#C0C0C0,$(traits.inactive)
# Comment keyword
```

```
style.sowya.81=$(style.sowya.17),fore:#C0C0C0,$(traits.inactive)
# Comment keyword error
style.sowya.82=$(style.sowya.18),fore:#C0C0C0,$(traits.inactive)
# Raw strings for C++0x
style.sowya.84=$(style.sowya.20),fore:#B090B0,$(traits.inactive)
# Triple-quoted strings for Vala
style.sowya.85=$(style.sowya.21),fore:#90B090,$(traits.inactive)
# Hash-quoted strings for Pike
style.sowya.86=$(style.sowya.22),fore:#90B090,$(traits.inactive)
# Preprocessor stream comment
style.sowya.87=$(style.sowya.23),fore:#A0C090,$(traits.inactive)
# Preprocessor stream doc comment
style.sowya.88=$(style.sowya.23),fore:#C0C0C0,$(traits.inactive)
# User defined literals
style.sowya.89=fore:#D7A090,$(traits.inactive)
# Task Marker
style.sowya.90=fore:#C3A1CF,$(font.code.comment.line),$(traits.inactive)

# Braces are only matched in operator style
braces.sowya.style=10


command.go.$(file.patterns.sowya)=sowya $(FileNameExt)

command.go.$(file.patterns.clox)=sowya --clox $(FileNameExt)
#-o $(FileName).out

#command.compile.$(file.patterns.sowya)=sowya -c -w $(FileNameExt)
#command.name.0.$(file.patterns.sowya)=Lint
#command.0.$(file.patterns.sowya)=sowya -MO=Lint,all $(FileNameExt)

#command.name.1.$(file.patterns.sowya)=Check Syntax
#command.1.$(file.patterns.sowya)=sowya -cw $(FileNameExt)

#command.name.2.$(file.patterns.sowya)=Code Profiler
#command.2.$(file.patterns.sowya)=sowya -d:DProf $(FileNameExt)

#command.name.3.$(file.patterns.sowya)=Profiler Parser
#command.3.$(file.patterns.sowya)=C:\sowya\bin\dprofpp.bat $(FileDir)\tmon.out
```

参考文献

以引用顺序排列.

[1] 梅加强 数学分析. 高等教育出版社, 2011 (cited on page 13).

[2] 北京大学数学系几何与代数教研室代数小组 编. 高等代数 (cited on pages 98, 125).

[3] 李炯生、查建国. 线性代数 (cited on pages 98, 131).

[4] 叶乃膺译 A. K. 苏什凯维奇著. 数论初等教程. 高等教育出版社, 2011 (cited on pages 99, 102).