Introduction to the use of SMath Studio
Prepared by Gilberto E. Urroz, May 2010

Where to find *SMath Studio*?

SMath Studio 0.88 is freeware that produces notebook type of mathematical calculations. You can download *SMath Studio* at this web site:

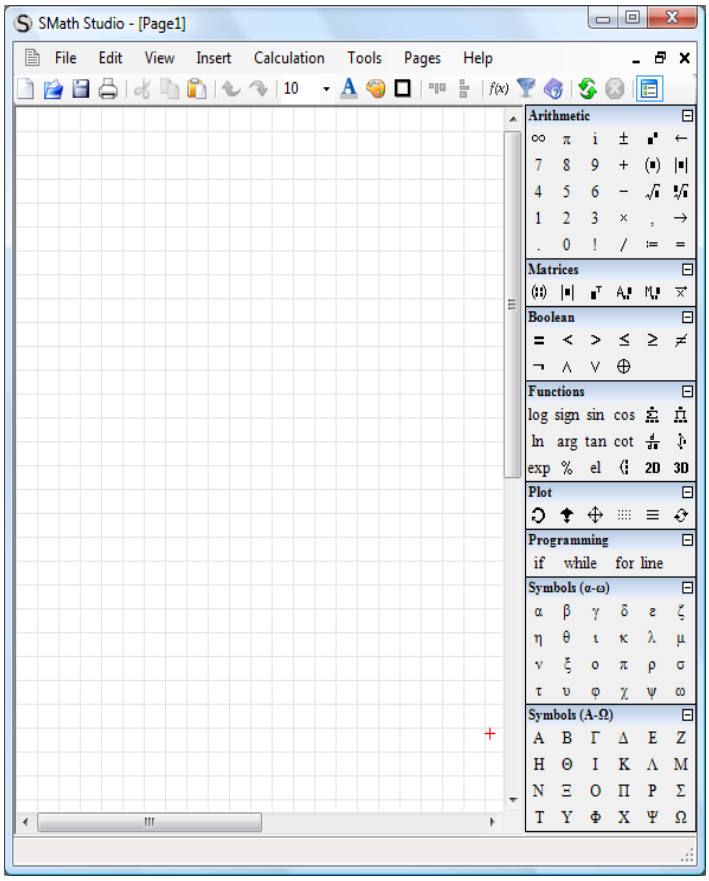
http://en.smath.info/forum/default.aspx?g=posts&t=425

Getting started

You will find the icon for *SMath Studio* in the folder *SMath\SMath Studio* under your *Program Files* folder in your *Windows* machine, or under your *Program Files (x86)* folder if you have a *64-bit* version of *Windows*. The program is called *SMathStudio_Desktop.exe*. You may want to create a shortcut to the program to place in your desktop or in your *Quick Launch*. Double click on the program icon or on the shortcut to open the *SMath Studio* interface.

The *SMath Studio* interface

To get started, find the *SMath Studio Desktop* icon in your *Start>Programs* button in Windows. The *SMath Studio* interface is shown here.



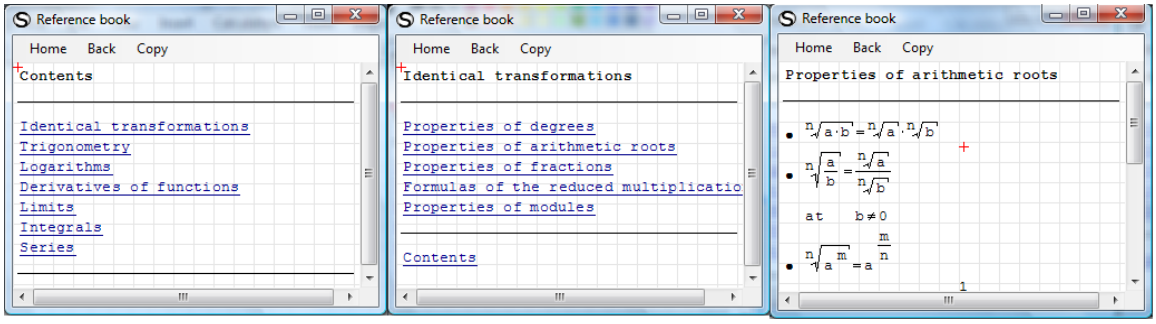
The interface shows a main window with menus, a toolbar, and a number of palettes on the right-hand side of the interface. The palettes contain mathematical, graphical, and programming functions that can be placed in the main window with the purpose of calculating mathematical expression, producing graphs, or building small programs.

The *SMath Studio* menus

The *SMath Studio* interface contains menus entitled *File, Edit, View, Insert, Calculation, Tools, Pages,* and *Help*. Besides these 8 menus, there is also a menu indicated by a page icon located to the left of the menu bar. We'll refer to this menu as the *Control* menu. Explore the different menus to become acquainted with the various options available in them. The operation of the *Control*, *File,* and *Edit* menus is very similar to those of other *Windows* applications, hence, the item therein contained would be familiar to *Windows* users. Other menus are presented below.

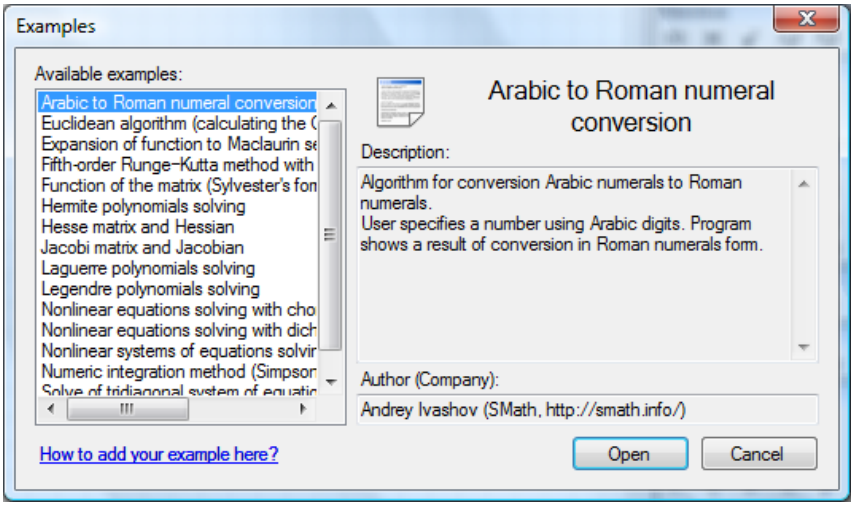
The *Help* menu

The *Help* menu contains the option *Reference book*... which represents a modest attempt by the author to address basic mathematical operations and definitions. The contents of the *Reference book* are shown in the figure below. To find the contents of a particular section of the *Reference book*, click on that section. For example, to see the contents of the section entitled *Identical Transformations*, click on the corresponding link to produce the screen in the middle. If you click on the link entitled *Properties of arithmetic roots*, you get the screen shown to the right in the figure below. This provides some basic properties of limits that can be useful to review those concepts from algebra.

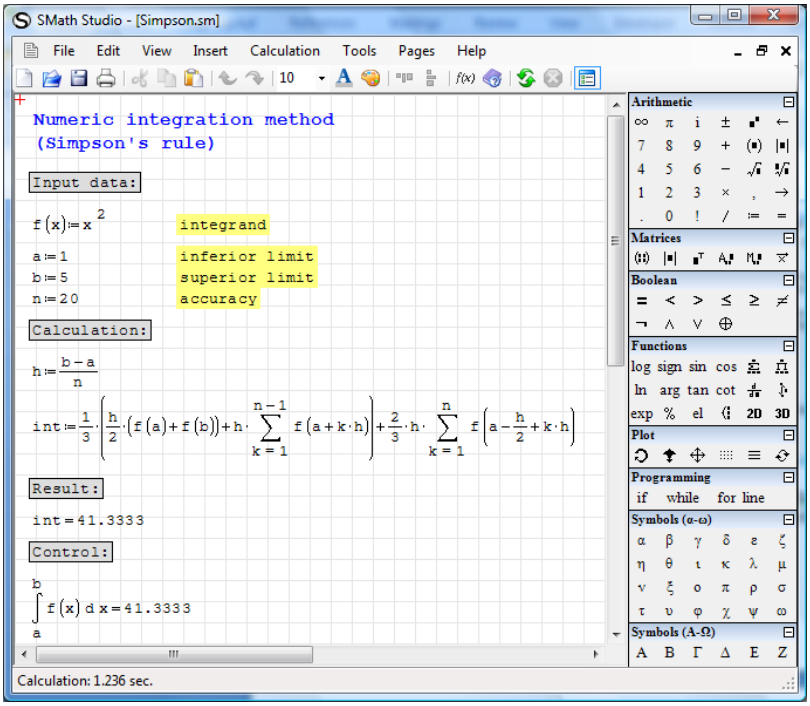


You can go back one page in the *Reference book* by pressing the option *Back*, or return to the Contents page by pressing the option *Home*. The menu option *Copy* lets you copy equations from the *Reference book* that you can then paste in the main interface. To copy a specific equation, first highlight the equation you want to copy, then press *Copy*, and paste the equation (using, for example, *Cntl-v*) onto the main screen.

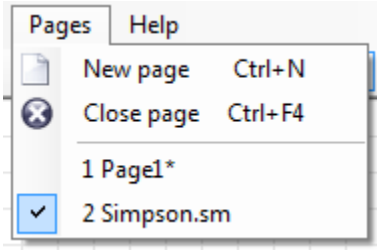
The *Help* menu also contains the option *Examples*, which opens the following menu of examples:



Click on a given example to select it, and press the [Open] button. The worksheet corresponding to the example *Numeric integration method (Simpson's rule)* is shown below.



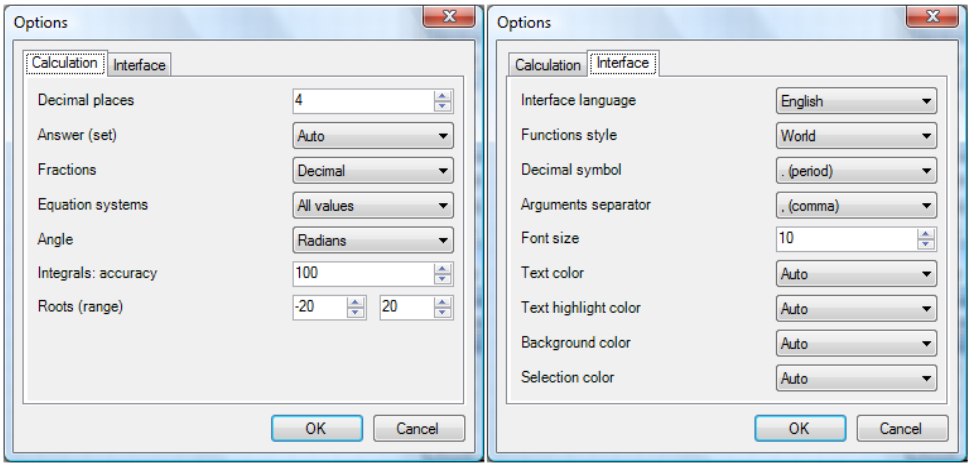
The *Pages* menu



The *Pages* menu shows the pages (worksheets) currently open. After having opened the worksheet for the example shown above, the *Pages* menu will show two pages open, namely, *Page1*,the default page open when we started *SMath Studio*, and *Simpson.sm*, the worksheet we just opened from the *Examples* menu.
The *Pages* menu also shows the options *New page* and *Close page*, whose operation is obvious.

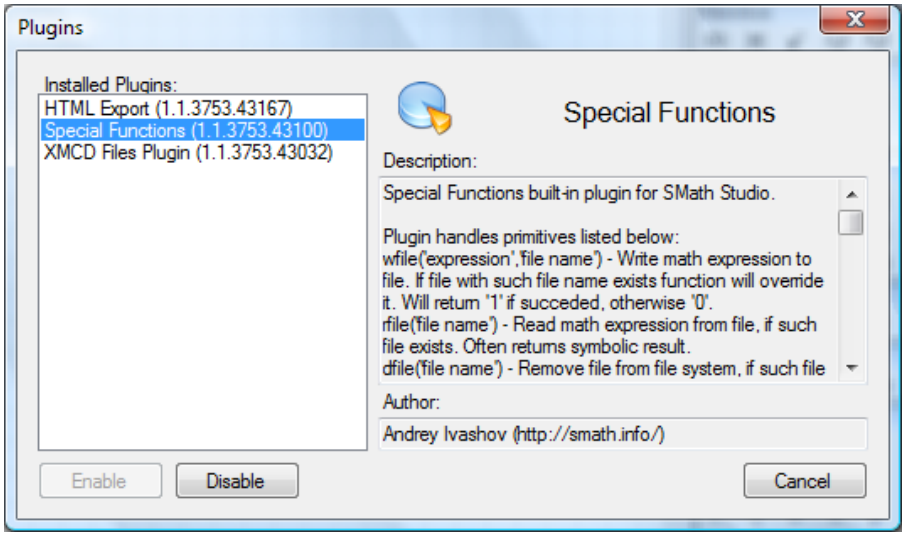
The *Tools* menu

The *Tools* menu has two items (*Options*... and *Plugins*...). The *Options*... item opens the following dialogue form with two tabs as shown below.



The *Calculation* tab in the *Options* interface lets you modify basic settings for mathematical calculations, whereas the *Interface* option deals with properties of the interface window. Click on the different drop-down menus to select the setting you would like to change.

The *Plugins* item in the *Tools* menu produces the following interface:

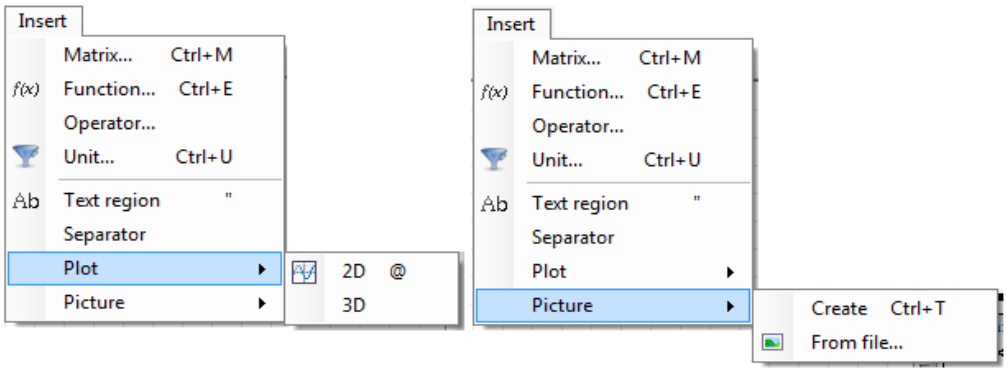


SMath Studio 0.88 includes the three plugins shown above, namely:

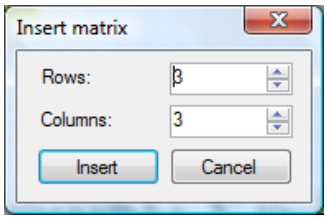
- *HTML Export*: allows to save files to HTML format
- *Special Functions*: handles a number of functions. Scroll down the cursor to the right to see all functions available. A brief description of the functions is included.
- *XMCD Files*: allows to save and open XMCD files. This format is used by the commercial software *Mathcad*.

The *Insert* menu

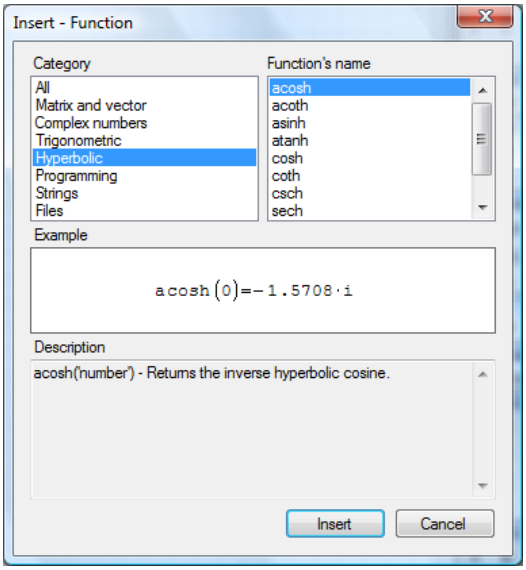
The *Insert* menu allows inserting a variety of items into the worksheet (or page), as indicated in the figure below:



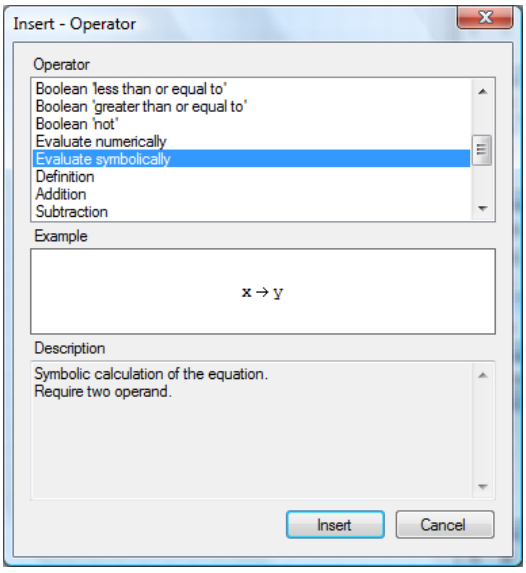
When you select the option *Matrix..* , for example, it produces an entry form that allows you build a matrix of a pre-determined size:



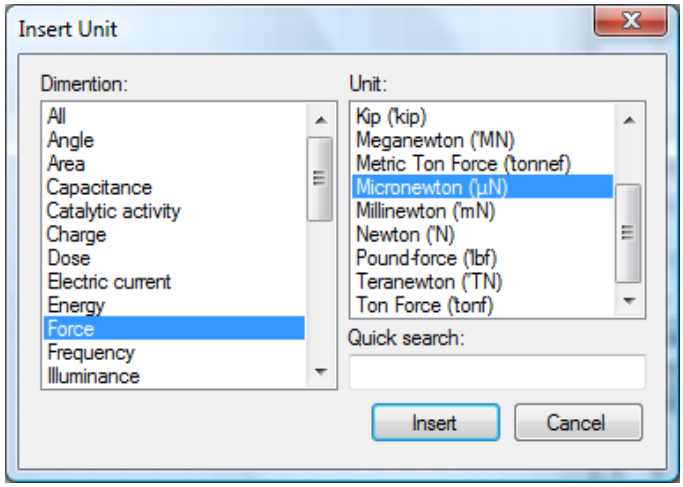
The option *Function…* opens up a menu of mathematical functions, i.e.,



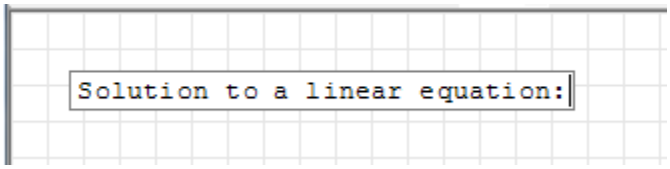
Selecting the option *Operator…* in the *Insert* menu produces a list of Boolean, arithmetic, and other operators, e.g.,



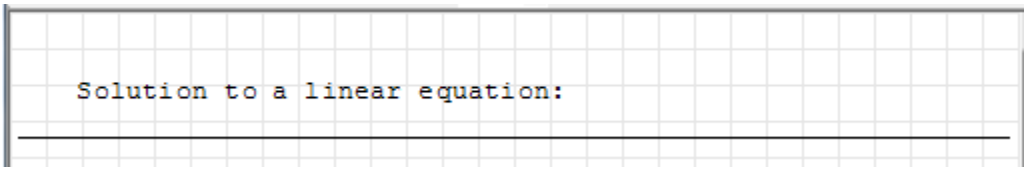
The option *Unit...* allows inserting units in calculations. The input form for *Units...* is shown below, highlighting the listing of units of *Force*. The listing shows the name of the unit (e.g., Micronewton) and the symbol used for its representation (e.g., μN).



The option *Text region* in the *Insert* menu allows the user to insert text fields in the worksheet with the purpose of documentation. This is equivalent to pressing the double quote (") after clicking in any position in the worksheet. For example:



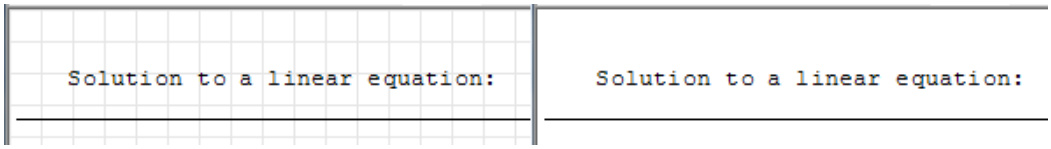
Inserting a *Separator* simply means inserting a horizontal line to separate regions in the worksheet, e.g.,



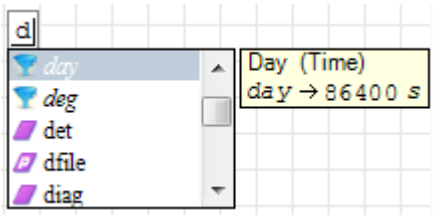
The insertion of plots and pictures will be illustrated in other sections of this document.

The *View* menu

The *View* menu includes an option for activating of deactivating a grid in the main screen (*Grid*). Here is the same worksheet with and without a grid:



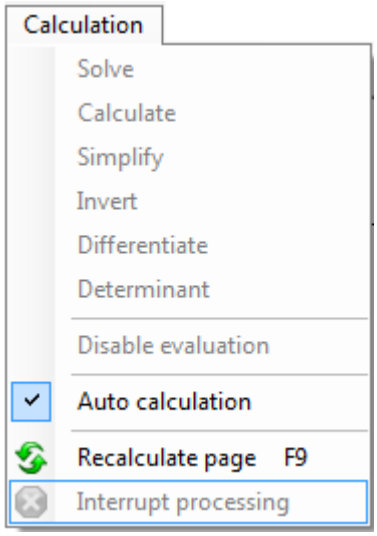
The *View* menu also includes the option *Dynamic Assistance* that activates or deactivates that particular option. When the *Dynamic Assistance* is active, every time that you type a letter in the main screen, a drop down menu of functions starting with that particular letter becomes available. You can then scroll down and select a particular function. For example, if you type the letter *d* the following dynamic assistance menu becomes available:



The drop-down menu shows functions *day* (a unit, described in the box to the right in terms of the unit *s*, or seconds), *deg* (a unit, *deg*rees), *det* (determinant of a matrix), *dfile* (*d*elete *file*), *diag* (create a diagonal matrix out of a vector), etc. Scrolling the cursor to the right, up or down, will provide additional function or unit definitions.

The *Calculation* menu

The *Calculation* menu offer options useful when calculating symbolic or numeric expressions. Unless a particular calculation has been selected, these options show as inactive (shadow options) in the menu, e.g.,

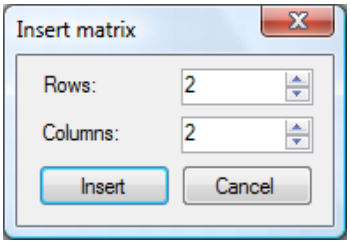


The only active options shown above are *Auto calculation* and *Recalculate page*. The meaning of these, and the other options shown above, is obvious.

To illustrate the use of some of the other items in the *Calculation* menu, click somewhere in the main screen and enter the expression 5+7/3. Then, click on the expression, and drag the cursor over the expression to highlight it. The highlighted expression should look as shown in the figure to the right.

Click on the *Calculation* menu and select, for example the *Calculate* option. This produces the result *22/3*. Try using the other available options such as *Simplify,* and *Invert.*

To check the use of the option *Determinant* first you need to enter a matrix in the main screen. Click somewhere in the main screen, and use the option *Insert* > *Matrix*, and change the number of rows and columns to 2, i.e.,



This results in the expression:

Click on each of the place holders, one at a time, and enter the values 2, 5, 3, and 1, so that the resulting matrix is:

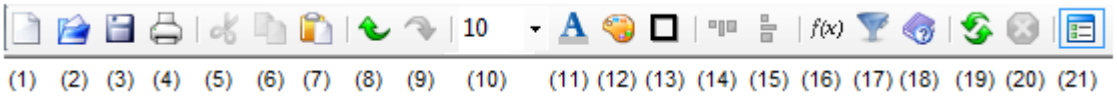
Click somewhere inside the matrix, and drag the cursor over the matrix to highlight it:

Next, click on the *Calculation* menu and select the *Determinant* option. The result is the value *-13*, which is the determinant of the matrix shown above, namely, $2 \times 1 - 5 \times 3 = -13$.

Note: I'm not familiar with the use of the options *Solve* and *Differentiate* in the *Calculation* menu. Therefore, I am not including any examples of those options. Examples of the function *solve* and of derivatives in *SMath Studio* are presented later in this document.

The *SMath Studio* toolbar

The *SMath Studio* toolbar contains 19 icons briefly described below:



1. New page
2. Open (existing worksheet)
3. Save (current worksheet)
4. Print (current worksheet)
5. Cut
6. Copy
7. Paste
8. Undo (recent action)
9. Redo (recent action)
10. Font size
11. Text color
12. Background color
13. Control border (frames selection)
14. Align horizontally
15. Align vertically
16. Function (insert a function)
17. Unit (insert unit)
18. Reference book (see *Help*)
19. Recalculate page
20. Interrupt process
21. Show/hide side panel

Items (1) through (4) manipulate new or existing worksheets. Items (5) through (7) are well-known editing functions. Items (8) and (9) will un-do and re-do the most recent action. Items (10) through (12) adjust font or background properties. Item (13) allows you to put a frame over an entry, for example, to show a solution to a problem. Items (14) and (15) re-align selected cells. Item (16) and (17) open the *Function* and *Unit* menus, as shown earlier under the *Insert* menu. Item (18) is also available under the *Help* menu. Items (19) and (20) were shown earlier in the *Calculation* menu. Item (19) shows or hides the palettes on the right-hand side of the page.

Using *SMath Studio* as a calculator

SMath Studio (or, simply, *SMath*) can be used as a calculator. For example, open a new page, and click on the main window in an area near the top left corner. A small red cross will indicate the location where you want to enter a calculation. Type the following:

$$1+2/3+4/5 =$$

The result is the following expression:

$$1+\frac{2}{3+\frac{4}{5}}=1.5263$$

Notice the way that *SMath* interprets the two fractions. Try the following example also: click somewhere else in your worksheet and type:

$$5 + 2/3\ [\rightarrow]+ (4/5) =$$

The result is now:

$$5+\frac{2}{3}+\frac{4}{5}=6.4667$$

Suppose that you want to calculate the expression $\frac{2+3}{4}$. Use the following keystrokes:

2 + 3 [space] 4 =

Here [space] means to press the *space* bar. The result is:

$$\frac{2+3}{4}=1.25$$

The figure above shows the result immediately after entering the equal sign (=). If you click somewhere else in the worksheet, then the result of this operation is shown as:

$$\frac{2+3}{4}=1.25$$

More complex expressions can be built using the palette functions shown on the right-hand side of the screen, for example:

$$\frac{5+\sqrt{3-\frac{1}{5+3}}}{\ln\left(3+\frac{5}{4}\right)+\exp\left(-\frac{1}{10}\right)}=2.8471$$

Expressions such as the one shown above can be edited by clicking on the location where the editing is to be done, and then entering factors or functions. For example, to modify the expression above, we could click on the right side of the 3 within the square root in the numerator, i.e.,

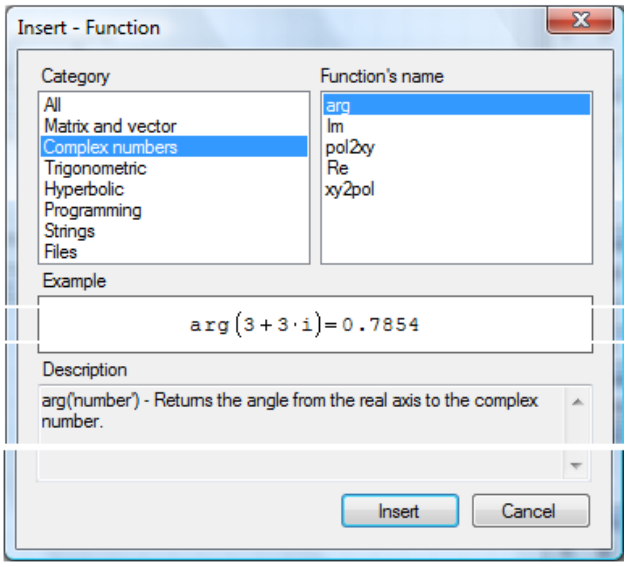
$$\frac{5+\sqrt{3}-\frac{1}{5+3}}{\ln\left(3+\frac{5}{4}\right)+\exp\left(-\frac{1}{10}\right)}=2.8471$$

and type * 3 [space] √ 4. Then, select the 1 within the *exp* function in the denominator and type over the expression *exp(1.2)*. The modified expression will be the following:

$$\frac{5+\sqrt{\frac{3\cdot 5}{\sqrt{4}}-\frac{1}{5+3}}}{\ln\left(3+\frac{5}{4}\right)+\exp\left(-\frac{\exp(1.2)}{10}\right)}=3.5648$$

Functions in *SMath*

Calculations in *SMath* may involve mathematical functions such as *sin*, *cos*, *exp*, etc. You can insert any function by using the menu option *Insert* > *Function*... or by pressing item (15) [*f*(*x*)] in the toolbar. As indicated earlier, collections of functions under the headings *All, Matrix and vector, Complex numbers, Trigonometric, Hyperbolic*, and *Programming* are available. The figure below shows the options for complex number functions:

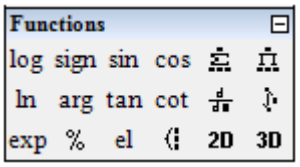


In this example, function *arg* is selected. The argument (*arg*) of a complex number is the angle that a vector representing the complex number in the complex plane forms with the real (*x*) axis. As indicated in the figure above, as you select a particular function the *Insert – Function* form provides an *Example* as well as a brief *Description* of the function.

Typically, you will start an expression in the worksheet and, at the proper location, insert the function that you need. For example, calculate the expression that uses the hyperbolic function *asinh*:

$$4+2\cdot\operatorname{asinh}(3+2\cdot i)=7.9668+1.1413\cdot i$$

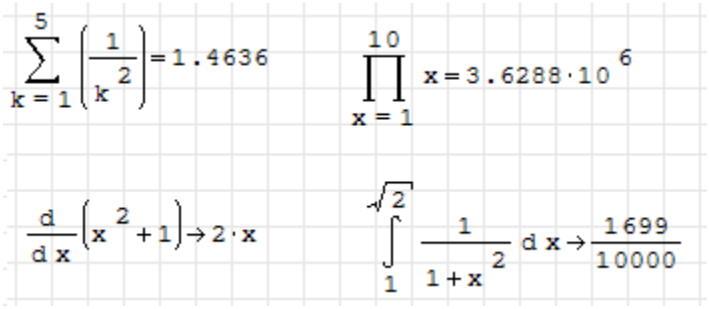
Some functions are available for insertion in the *Functions* palette shown to the right: → → → → → → → → → → → →



To enter any of those functions simply place a cursor in the desired position and click on the name of the function, e.g., type:

$$\ln\left(\frac{1}{\sin(3.2)}\right)=2.8409+3.1416\cdot i$$

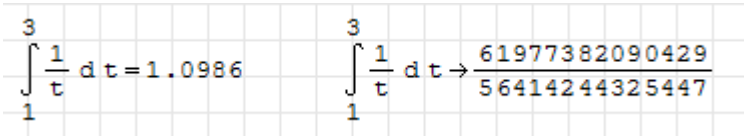
Some of the functions in the *Functions* palette include operations typical from Calculus, such as summation, products, derivatives, and integrals, e.g.,



In these expressions we used symbols that are available in the *Functions* palette as well as other symbols available in the *Arithmetic* palette: Infinity (∞), Symbolic evaluation (→) , and π (the ratio of the length of the circumference to the diameter of a circle).

Numeric versus symbolic evaluation
An expression that results in a number is evaluated using the numerical evaluation symbol (=), whereas, an expression that results in a symbolic output needs the symbolic evaluation symbol (→). Both symbols are available in the *Arithmetic* palette. The numerical evaluation symbol (=) can be typed directly from the keyboard (it's just the *equal* sign). The symbolic evaluation symbol (→) can also be entered by typing *Ctrl+period* (*Ctrl+.*).

The examples above show the summation and product producing numeric results, while the derivative and the definite integral produce symbolic results. Notice the difference between numeric and symbolic results in the following integral:



In the current version of *SMath Studio, t*he main difference between numeric and symbolic results is whether a result is shown in its decimal form (numeric) or as a fraction (symbolic) and other symbolic results such as square roots, etc. A proper symbolic result in the integral above would have been the expression *ln(3)*, however, *SMath Studio* does not handle such type of symbolic calculations when the result involves a function definition.

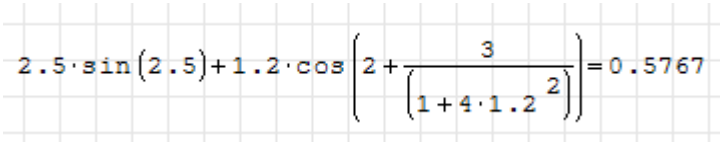
For an open-source, free program that would handle such type of symbolic calculations please refer to this web site: http://www.neng.usu.edu/cee/faculty/gurro/Maxima.html.

Note: Use function *eval* to convert from symbolic to numeric results, e.g., *eval*(*sqrt*(3)).

An alternative for entering functions in expressions is simply to type the name of the function. For example, click in another area of the main *SMath Studio* interface, and type the following expression:

2.5*sin(2.5)+1.2*cos(2+3/(1+4*1.2^2))

The results of this operation will be shown as:



The *Insert - Function* menu in *SMath Studio*

As indicated earlier, the *Insert - Function* menu in *SMath Studio* can be obtained by pressing the [*f*(*x*)] button in the tool bar. This menu includes function groups labeled *All, Matrix and vector, Complex numbers, Trigonometric, Hyperbolic*, *Programming, Strings*, and *Files*.

- The *Complex numbers, Trigonometric, Hyperbolic*, and *Programming* function groups include a relatively small number of easily recognizable functions.
- The *Matrix and vector* group includes a total of 30 functions useful for the manipulation of and calculations with vectors and matrices.
- The *Strings* group includes functions used in manipulating and operations with strings. These can be useful in programming.
- The *Files* group includes functions for reading, writing, and deleting files, as well as for importing data into a worksheet.

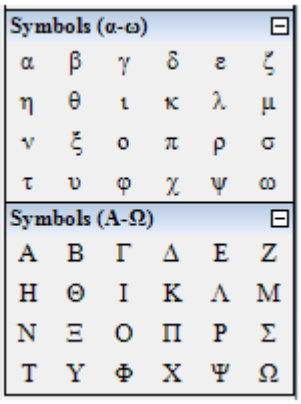
As indicated before, examples and brief explanations of the functions are available in the *Insert-Function* menu.

The *All* menu includes all existing functions in *SMath Studio*. Herein we group some of those functions according to their applications:

- Functions specific to *SMath Studio*: *eval*, *range*, *sys, error,*
- Functions for real numbers: *abs, exp*, *Gamma, ln, log, log10, mod, nthroot, numden, perc, round, sign, sqrt*
- Functions for algebraic manipulation: *expand*
- Functions for solving equations: *polyroots, solve*
- Functions for Calculus applications: *diff*, *int, product, sum*
- Functions for interpolation: *ainterp, cinterp*, *linterp*
- Statistics functions: *random*

Entering Greek letter in *SMath Studio*

Greek characters can be entered by using the *Symbols* palettes in the interface. These palettes, showing lower-case and upper-case Greek letters, are shown in the figure to the right.

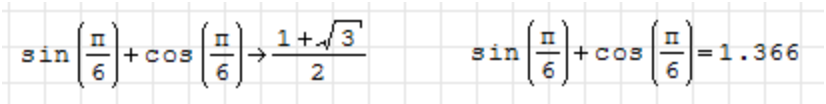


An alternative way to enter Greek letters is to type a letter in the English alphabet followed by *Ctrl-g*. This will generate a corresponding Greek letter. For example, typing *g Ctrl-g* produces the Greek letter γ (gamma). The corresponding upper case character would be entered as *G Ctrl-g*, resulting in the letter Γ (upper-case gamma).

The table below shows the letters of the Greek alphabet and its closest equivalent English letters.

Lower case	Upper case	Letter name	English equivalent	Lower case	Upper case	Letter name	English equivalent
α	A	Alpha	*a*	ν	N	Nu	*n*
β	B	Beta	*b*	ξ	Ξ	Xi	*x*
γ	Γ	Gamma	*g*	o	O	Omicron	*o*
δ	Δ	Delta	*d*	π	Π	Pi	*p*
ε	E	Epsilon	*e*	ρ	P	Rho	*r*
ζ	Z	Zeta	*z*	σ	Σ	Sigma	*s*
η	H	Eta	*h*	τ	T	Tau	*t*
θ	Θ	Theta	*th*	υ	Y	Upsilon	*u*
ι	I	Iota	*i*	ϕ	Φ	Phi	*ph*
κ	K	Kappa	*k*	χ	X	Chi	*ch*
λ	Λ	Lambda	*l*	ψ	Ψ	Psi	*ps*
μ	M	Mu	*m*	ω	Ω	Omega	*o*

Clicking on any of the letters in the *Symbols* palette will copy that letter to any entry point or text in the worksheet. To illustrate this fact we perform symbolic and numeric calculations with trigonometric functions that features the value π.

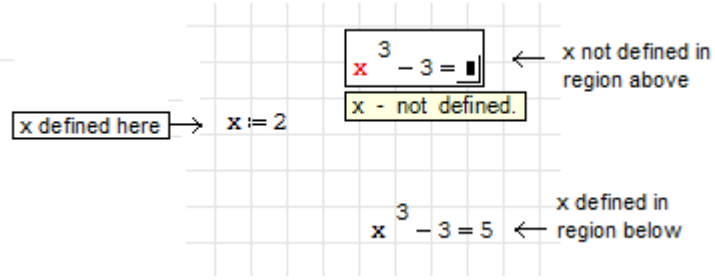


Notice that SMath Studio assigns the proper value to the symbol π in the calculation.

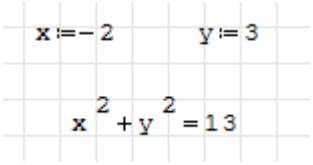
Defining variables in *SMath Studio*

To define a variable in *SMath Studio* use the *assignment operator* (:=). To enter this operator simply press the *colon* key in your keyboard (:). Click anywhere in the worksheet and make the following variable assignment: $x := 2$, by typing: x : 2. The value of *2* is now stored in the name *x*. The value $x = 2$ will be replaced into any

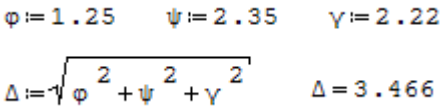
expression containing the name *x* that is located below or to the right of the assignment statement. To check this fact, type an equal sign in a location above the assignment statement *x:=2* and fill the placeholder to the left with the expression x^2-*3*. The result is inconclusive, as shown below. Then, repeat this operation in a location below or to the right of the assignment statement *x:=2*. The result now is 5, as illustrated below.



Once you have made variable assignments, you can use the assigned variables to calculate expressions. As an example, assign the values *x: =-2* and *y:=3*, then calculate the following expression: x^2+y^2. The result may look somewhat like this:



The following example shows variable assignments that use Greek letters:



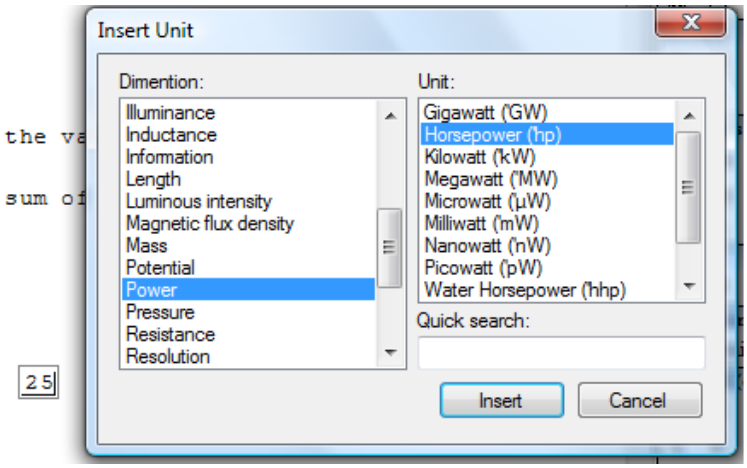
The keystrokes used to produce these entries are the following:

```
<click>  f Cntl-g : 1.25   <click>   y  Cntl-g : 2.35
<click>  g Cntl-g : 2.22
<click>  D Cntl-g : sqrt f Cntl-g ^ 2 + y Cntl-g ^ 2 + g Cntl-g ^ 2
<click>  D Cntl-g  =
```

The symbol <click> indicates that the user must click somewhere in the worksheet.

Using units

Units are important in calculations in the physical sciences. Units in *SMath Studio* can be incorporated by using the *Unit* button in the toolbar (see item (17) in page 9). For example, to enter the quantity *25 hp* (25 horsepower), click somewhere in the worksheets, type 25 and the click on the *Unit* button in the toolbar. Scroll down to find the unit category *Power*, and select the unit *horsepower ('hp)*, as shown below.



Press [Insert] to obtain the following result: 25 *hp*

If you next press the equal sign, the quantity entered will be expressed using the basic units of the SI (*Systeme International*, or International System of units), e.g.,

$$25\ hp = 18642.4968\ W$$

Notice that the units are shown in blue italics font. This way the symbol ***m*** is different than the symbol **m** in your worksheet, so you can use ***m*** for ***m****eters* and **m** for mass, for example.

Explore the units menu to see all the available unit categories (*Angle, Area, Capacitance, ...,*
Volume), and the available units. For example, for *Angle*, you have available the units *Degree(º), Degree ('deg), Radian ('rad),* and *Revolution ('rev)*.

Entering units without the Unit menu
An alternative way to enter units is to type the single quote, or apostrophe, symbol (') followed by the unit symbol. This approach requires that you know the proper symbols to enter. If the option *View>Dynamic Assistance* is active, you will get a menu of possible functions and units to insert. If you are not sure of the symbol for the units to insert use the approach shown earlier utilizing the full units menu. Here is an example entering the quantity *100 µF* (100 micro Farads, a unit of electric capacitance). Click in your worksheet and type: 100 ' m *Cntl-g* F, then type the equal sign (=). The result is shown below:

$$100\ \mu F = 1 \cdot 10^{-4}\ F$$

Notice that when you type the single quote (apostrophe) symbol, *SMath Studio* shows the following symbol to indicate an impending unit insertion (call this symbol the *unit placeholder*):

100 ■

Try the following example where we perform a calculation using units:

```
2.5 'mg * 10.5 'ft [space] / ( 1.25 'hr [▶][▶] ^ 2 =
```

The result is given, by default, in units of the International System (SI):

$$2.5\ mg\cdot\frac{10.5\ ft}{(6.8\ hr)^2}=1.3351\cdot10^{-14}\ N$$

Converting units in a result

The result thus obtained can be expressed in other units by clicking to the right of the ***N***, which produces a small, black, rectangular placeholder as shown below:

$$2.5\ mg\cdot\frac{10.5\ ft}{(6.8\ hr)^2}=1.3351\cdot10^{-14}\ N\blacksquare$$

Click on the placeholder so that the placeholder gets selected, as illustrated here:

$$2.5\ mg\cdot\frac{10.5\ ft}{(6.8\ hr)^2}=1.3351\cdot10^{-14}\ N\blacksquare$$

Then, type the replacement unit(s). For example, if we want to show the previous result using *micronewtons* (μN), type, in the placeholder, `'m Cntl-g N`, i.e.,

$$2.5\ mg\cdot\frac{10.5\ ft}{(6.8\ hr)^2}=1.3351\cdot10^{-8}\ \mu N$$

By a similar procedure we can replace the units with *millinewtons* (*mN*) as follows: click on the right-hand side of the ***μN*** units, press the [← Backspace] key twice to remove the units, so that the unit placeholder shows up, i.e.,

$$2.5\ mg\cdot\frac{10.5\ ft}{(6.8\ hr)^2}=\blacksquare\blacksquare$$

Now, type `mN`, to get[1]:

$$2.5\ mg\cdot\frac{10.5\ ft}{(6.8\ hr)^2}=1.3351\cdot10^{-11}\ mN$$

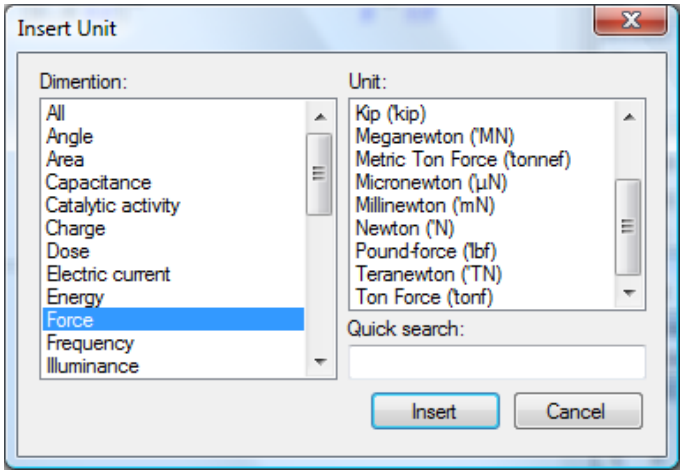
[1]When the unit place holder is active, there is no need to type the single-quote (apostrophe) symbol to enter units.

Defining your own units

Suppose that you wanted to see the result in *nanonewtons* (*nN*). Attempting to replace the *micronewtons* (*mN*) with *nanonewtons* (*nN*), we obtain the following result:

$$2.5\ mg\cdot\frac{10.5\ ft}{(6.8\ hr)^2}=1.3351\cdot10^{-14}\ \frac{m\ kg}{s^2}\ nN$$

The unit substitution was not accomplished because the *nanonewton* unit (*nN*) is not defined. You can check this by using the option *Insert* > *Unit* ... and select the *Force* category in the resulting menu:



The units are listed in alphabetical order and, obviously, there is no *nanonewton* (*nN*) units listed. (Press the [Cancel] button to remove the menu.) Hence the impossibility of replacing μN with *nN*.

You can, however, define your own units if they're missing. For example, to define the symbol for *nanonewtons* (*nN*) you can type:

```
'nN : 10 ^ -9 'N
```

This definition must be entered at a location in the worksheet above the point where the calculation takes place. In this case, the conversion to *nanonewton* (*nN*) is successfully performed as shown below:

Definition of nanonewton: $nN := 10^{-9}\ N$

$$2.5\ mg\cdot\frac{10.5\ ft}{(6.8\ hr)^2}=1.3351\cdot10^{-5}\ nN$$

Physical constants available

The collection of units in *SMath Studio* include two physical constants referred to by their typical symbol. These are the speed of light in vacuum (referred to as *c)*, and the standard acceleration of gravity (referred to as *g)*. In *SMath Studio* you can show the values of those constants using the notation for units. For example, to see the value of *c* type `'c =` , or type `'g =` to see the value of g.

$$c = 2.9979 \cdot 10^{8} \frac{m}{s} \qquad g = 9.8066 \frac{m}{s^2}$$

Units of the English System

In the United States of America, and in a few other countries, the traditional system of units, known as either the English system (ES) or imperial system, is still very much in use. Therefore, *SMath Studio* includes in their collection of units such ES units as *horsepower* (see example above), *lb* (pounds), *ft* (foot, or feet), *in* (inches), etc. For example, the values of the speed of light in vacuum and the standard acceleration of gravity in units of the English system are:

$$c = 9.8357 \cdot 10^{8} \frac{ft}{s} \qquad g = 32.174 \frac{ft}{s^2}$$

Are pounds units of mass or force?

Modern usage in the physical sciences requires that the pound, *lb*, be used as a unit of force, with the corresponding unit of mass being the *slug,* defined as *1 slug = 14.5939 kg*. However, in buying produce (e.g., meat, grain, etc.), the pound can be used as a unit of mass with the conversion *1 kg = 2.2 lb*. In *SMath Studio* the pound, *lb,* as well as the ounce, *oz*, are used as a units of mass (Note: *1 lb = 16 oz*). The corresponding unit of force is the *pound-force*, *lbf*, defined as *1 lbf = 4.4482 N*. Thus, in physical science applications use the *pound-force*, *lbf*, when you are required to use *pounds* as a unit of force. Another commonly used unit of force is the *kip* (*kilopound*) equal to *1000 lbf*.

What about the kilogram?

The kilogram is the basic unit of mass in the International System (SI). Nevertheless, there is a unit of force called the *kilogram-force*, *kgf*, corresponding to the weight of *1 kg*. The definition of *kgf* is, therefore, *9.8066 N*.

The cgs *system*

Before the adoption of the *Systeme International* (SI, or International System), back in 1960, metric units used in the physical sciences were grouped into two systems, called the *MKS* (*meter-kilogram-second*) and the *cgs* (*centimeter-gram-second*). The *MKS* system became the basis of the *SI*. The *cgs* system is used for applications requiring small masses and/or distances. The units of force and work (or energy) in the *MKS* (now the *SI*) system are the *newton* (*N*) and the *joule* (*J*), respectively. The units of force and work/energy in the *cgs* system are the *dyne* and the *erg*. The units in the *SI* and the *cgs* systems are related by: $1\ N = 10^{5}\ dyne$, $1\ J = 10^{7}\ erg$. *SMath Studio* includes the *dyne*, but not the *erg*, in its collection of units.

Working with variables and units

Units can be attached to numerical values assigned to variables. The results of operations with these variables will be given in the basic SI units. Here are some examples:

Example 1: Universal gravitation law. Calculating the force between masses *m1* and *m2*, separated by a distance *d*. The formula, shown below, requires us to use the universal gravitational constant *G*, found elsewhere:

$G := 6.673\,10^{-11}\ \frac{m^3}{kg\ s^2}$ $\quad d := 5.67\,10^{8}\ km$

$m1 := 1.2\,10^{18}\ kg$ $\quad m2 := 2.6\,10^{15}\ kg$

$F := \frac{G \cdot m1 \cdot m2}{d^2}$ $\quad F = 0.6476\ N$

$F = 64760.4117\ dyne$ $\quad F = 0.1456\ lbf$

Example 2: Relativistic mass – The equation below, belonging to Einstein's Special Theory of Relativity, is used to calculate the mass of a body moving at a velocity *v* if the rest mass is *m0*.

$m0 := 4.5\,10^{-6}\ kg$ $\quad v := 150\,10^{5}\ \frac{km}{hr}$

$m := \frac{m0}{\sqrt{1-\left(\frac{v}{c}\right)^2}}$ $\quad m = 4.5004 \cdot 10^{-6}\ kg$

Notice that in the equation above we used the symbol for *c* from the collection of Units in *SMath Studio*.

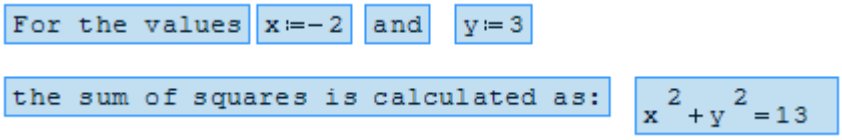
Documenting the worksheet

In order to document your worksheet, you can add text to it by clicking on any place in the worksheet and typing the double-quote symbol ” followed by the text. (Alternatively, you can use the option *Insert>Text Region* in the worksheet menu). For example, the following figure shows assignment statements interposed with text regions to produce a readable document.

For the values $x := -2$ and $y := 3$

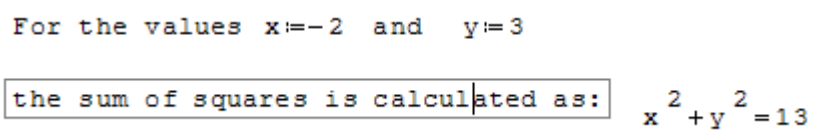
the sum of squares is calculated as: $x^2 + y^2 = 13$

By dragging the cursor over the region where these fields were entered you can see all the fields involved, both text regions and mathematical fields:

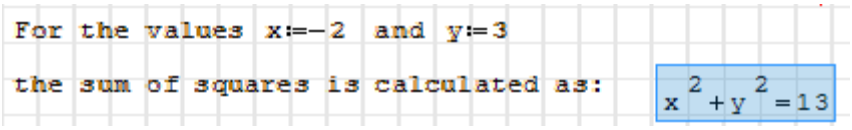


Selecting worksheet fields for editing and repositioning

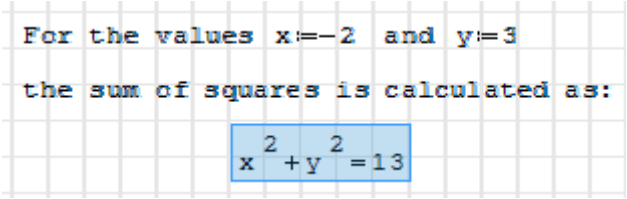
If you click on any of the text lines or the operations shown above, the corresponding field will be shown enclosed in a rectangle. This indicates that the field can be edited for text or math calculations. An example is shown below:



A text or calculation field can also be selected by clicking in a region near the field and dragging the mouse towards the field. In this case, the field is enclosed by a frame with blue boundaries and a blue background, e.g.,



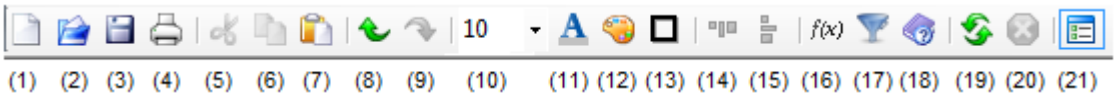
When selected in this way, a field can be dragged and positioned somewhere else in the worksheet. For example, the result shown above could be re-organized as follows:



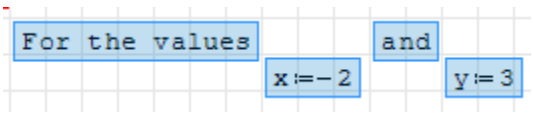
A field selected by dragging can be erased from the worksheet by pressing the [*delete*] or the [*backspace*] key. The selected field can also be copied and pasted by using *Ctrl-C* and *Ctrl-V*, respectively. By dragging on top of two or more fields, multiple fields can be selected simultaneously for editing.

Horizontal and vertical alignment of cells

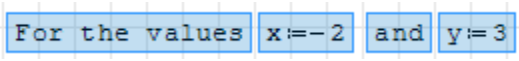
Items (13) and (14) in the *SMath Studio* toolbar (see figure below) provide for horizontal and vertical alignment of cells in the worksheet.



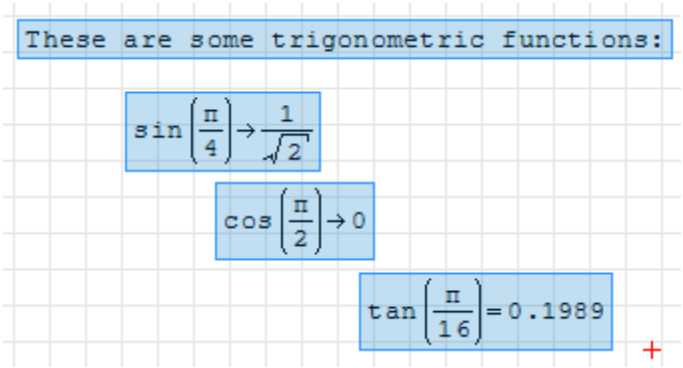
Consider the following text and calculation cells that are misaligned horizontally:



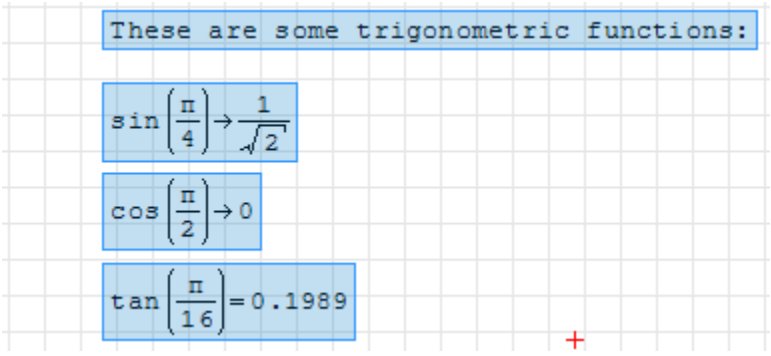
The four cells were selected by dragging the mouse, while holding the left button, over the cells. Next, press button (14) in the toolbar to get the following result:



Now, for an exercise on vertical alignment, consider the following vertically misaligned cells:



After pressing button (15) in the toolbar we get the following result:



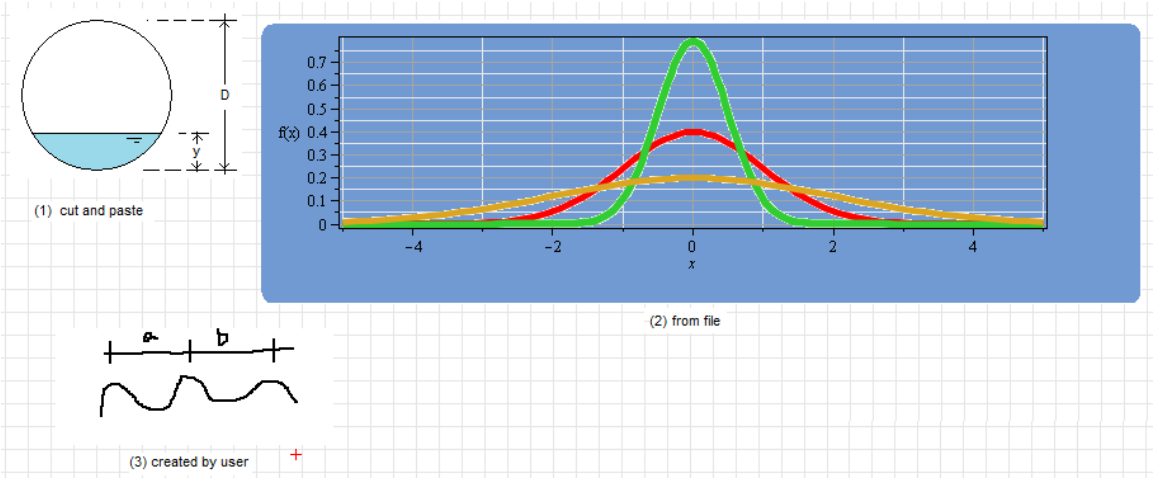
Adding images to the worksheet
Images scanned into the computer or produced by other graphic software can be incorporated into your worksheet in different ways.

1 – Copying a figure from other software, clicking and using *Paste* (or *Ctrl-v*).

2 – Using *Insert* > *Picture* > *From file* ..., to insert a picture contained in a file.

3 – Using *Insert* > *Picture* > *Create* ..., to insert a picture frame where you can draw your own picture. Use the mouse as a pencil. *SMath Studio* does not provide for other drawing tools

Examples of the three cases are illustrated below. The only images allowed to insert from a file are *bitmap* images. Through the cut-and-paste procedure, you can paste images from any format.

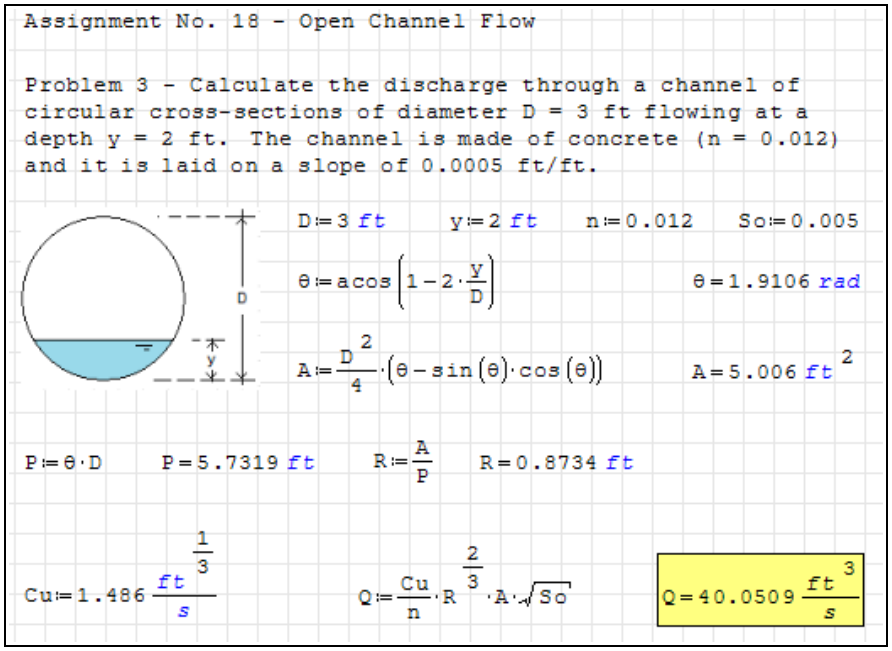


By selecting the field containing a figure, the figure can be repositioned in the worksheet. Be careful, however, when dragging your mouse across a figure whose field has not been highlighted. In such case, the mouse dragged across the figure will add lines to the figure that you may not intended to add. In such case use the *Undo* button in the menu.

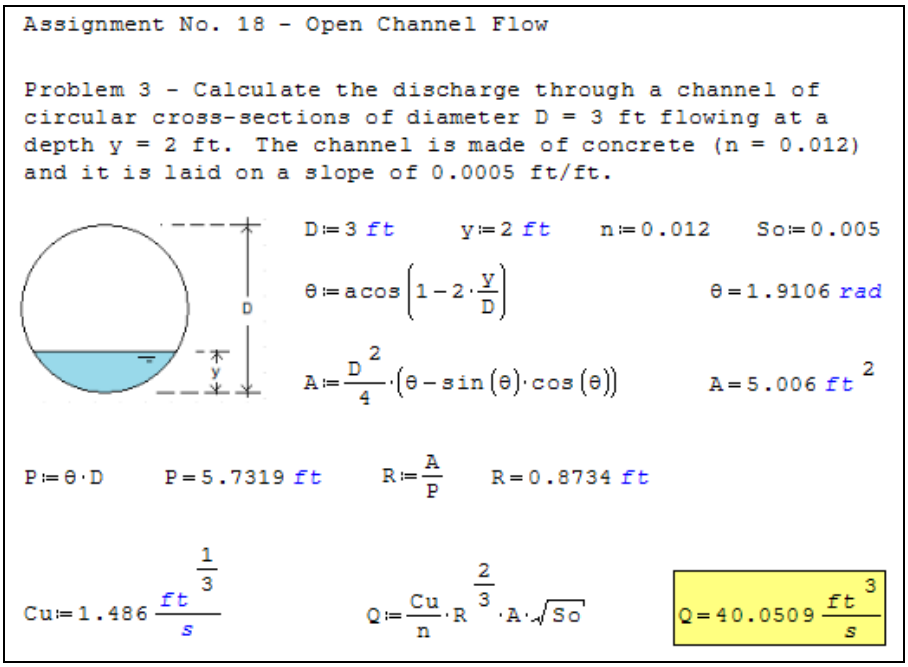
With text and calculation cells, plus image cells imported into or created within *SMath Studio*, you can produce technical reports as would be required for class assignments or for an engineering project.

Application – Writing an assignment in *SMath* Studio

The figure below shows a class assignment in fluid mechanics/hydraulics prepared in *SMath Studio*.



If you prefer, you can click off the grid using the *View* menu to produce the following:



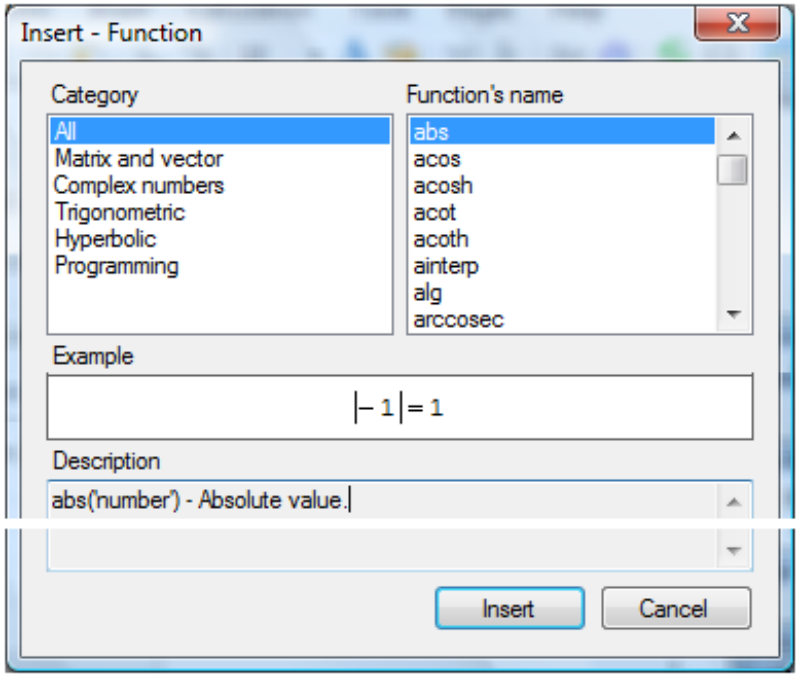
Note: To enter multiple lines in a text box, use *Shift-Enter* at the end of each line.

Predefined functions for real and complex numbers in *SMath Studio*
This section shows examples of predefined, or intrinsic, functions for both real and complex numbers available in *SMath Studio*.

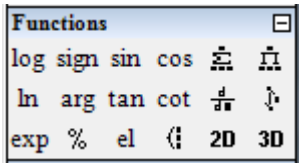
Functions for real numbers
The following functions are available for application to real numbers:

- abs absolute value
- exp exponential function
- Gamma Gamma (Γ) function
- ln natural logarithm, i.e., logarithm of base *e*
- log logarithm of any base
- log10 logarithm of base 10
- mod modulus
- nthroot the *n*-th root of a number
- numden decompose a fraction into numerator and denominator
- perc percentage
- round rounds to an integer
- sign extracts the sign
- sqrt square root
- random generates a random number

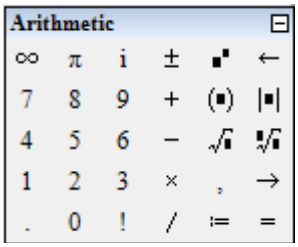
These functions are available, unclassified, by using the *Insert* > *Function* menu and then selecting the *All* category of functions:



Some of these functions are also available in the *Functions* palette: The *Function* palette includes also trigonometric functions (*sin, cos, tan,cot*), calculus expressions (summation, product, derivative, integral), functions that apply to matrices (*el*), and functions that apply to graphs (the last three symbols in the last line).



Some of these functions are also available in the *Arithmetic* palette. These include the absolute value (*abs*), the square root (*sqrt*), and the *n*-th root (*nthroot*) functions. Also shown in the *Arithmetic* palette are the following items:



- Mathematical Constants: Positive infinity (∞), Pi (π), Imaginary unit (i)
- Numerical Digits: 0-9
- Arithmetic operators: ±, +, -, ×, /, power
- Evaluation operators: Definition (:=), Numerical Evaluation (=), Symbolic Evaluation (→)
- Postfix Operators: Factorial (!)
- Editing Characters: Decimal point (.), Comma (,), Backspace (←)

Since trigonometric and hyperbolic functions apply also to real numbers, we provide a list of those functions available under the *Function – Insert* form (see above) under the headings *Trigonometric* and *Hyperbolic*:

Trigonometric:

• sin	sine	• asin	inverse sine
• cos	cosine	• acos	inverse cosine
• tan	tangent	• atan	inverse tangent
• cot	cotangent	• acot	inverse cotanget
• sec	secant	• arcsec	inverse secant
• csc	cosecant	• arccosec	inverse cosecant

Hyperbolic:

• sinh	hyperbolic sine	• csch	hyperbolic cosecant
• cosh	hyperbolic cosine	• asinh	inverse hyperbolic sine
• tanh	hyperbolic tangent	• acosh	inverse hyperbolic cosine
• coth	hyperbolic cotangent	• atanh	inverse hyperbolic tangent
• sech	hyperbolic secant	• acoth	inverse hyperbolic cotanget

Examples of functions applied to real numbers

These functions can be inserted from the *Functions – Insert* form (*Insert > Function* menu), the *Functions* palette, or simply by typing the name of the function into a region of the *SMath Studio* worksheet. The following are examples of real-number functions in *SMath Studio:*


```
//EXAMPLES OF FUNCTIONS FOR REAL NUMBERS:                    //Using "Ctrl+." instead of "="
//abs: type "abs(-3.25)=" to produce:          |-3.25|=3.25          |-3.25|→13/4
//exp: type "exp(-0.5)=" to produce::          exp(-0.5)=0.6065     exp(-0.5)→exp(-1/2)
//"exp(-0.5)"is same as"e^(-0.5)":             e^-0.5=0.6065        e^-0.5→1/√e
```

```
//Gamma: type "Gamma(1.5)=" to produce:        Gamma(1.5)=0.8862    Gamma(1.5)→886228183571491/1000000000000000
//ln: type "ln(3.2)=" to produce:              ln(3.2)=1.1632       ln(3.2)→ln(16/5)
//"exp" and "ln" are inverse functions:        exp(ln(5))=5         ln(exp(-3.2))=-3.2
//log: type "log(10,2)=" to produce:           log2(10)=3.3219      log2(10)→ln(10)/ln(2)
//log10: type "log10(8.2)=" to produce:        log10(8.2)=0.9138    log10(8.2)→log10(41/5)
//mod: type "mod(18,5)=" to produce:           mod(18, 5)=3         mod(18, 5)→3
```

The *mod* function applies to integers only, and it's described in the following example:

```
//Function "mod" calculates the integer residual (r) of the ratio of two integers m,n,
//where m>n and q is the integer quotient, i.e.: m/n = q + r/m. Thus, if m is a multiple
//of n, r = 0, and mod(m,n) = 0. Otherwise, mod(m,n) = r < n. See the following examples:
mod(5, 1)=0   mod(5, 2)=1   mod(5, 3)=2   mod(5, 4)=1   mod(5, 5)=0

// Thus, function "mod" can be used to determine if an integer m is a multiple of
// another integer n, for if that is the case then "mod(m,n) = 0".
```

```
//nthroot: type "nthroot(81,3)=":          ∛81=4.3267          ∛81→∛81
```

Function *numdem*, shown below, separates a fraction into a numerator and a denominator:

```
//numden: type "numdem(27.4)=":          numden(27.4)=(27.4 1)ᵀ          numden(27.4)→(137 5)ᵀ
```

$$\text{numden}\left(\frac{1+\sqrt{2}}{\sqrt{3}+\sin\left(\frac{\pi}{6}\right)}\right)=\begin{pmatrix}2.7182\cdot 10^{15}\\ 1.9604\cdot 10^{15}\end{pmatrix}\qquad \text{numden}\left(\frac{1+\sqrt{2}}{\sqrt{3}+\sin\left(\frac{\pi}{6}\right)}\right)\rightarrow\begin{pmatrix}1+\sqrt{2}\\ \sqrt{3}+\sin\left(\frac{\pi}{6}\right)\end{pmatrix}$$

Notice that, in its numerical evaluation, the last example shows both numerator and denominator multiplied by 1015. These two factors obviously cancel when the fraction is put together again, but it serves to emphasize that *SMath Studio* calculates values with 15 decimals.

More functions for real numbers are shown next:

```
//numden: type "numdem(27.4)=":        numden(27.4)=[27.4; 1]      numden(27.4)→[137; 5]

//perc: type "perc(10,25)=" :          perc(10, 25)=2.5            perc(10, 20)→perc(10, 20)
```

```
//round: "round(x,n)" rounds up a floating-point value x to n decimal figures:

          round(10.23446, 4)=10.2345          round(-3.12567, 4)=-3.1257
          round(10.23446, 3)=10.234           round(-3.12567, 3)=-3.126
          round(10.23446, 2)=10.23            round(-3.12567, 2)=-3.13
          round(10.23446, 1)=10.2             round(-3.12567, 1)=-3.1
          round(10.23446, 0)=10               round(-3.12567, 0)=-3

// Function "sign(x)" returns the values -1, 0, or 1, depending on whether
// x is negative, zero, or positive, e.g.,

    sign(-3.5)=-1              sign(0.0)=0              sign(3.5)=1

//sqrt: type "sqrt(23.54)=" to produce:   √23.54 = 4.8518   √23.54 → √1177 / √50
```

Function *rand* is used to produce random numbers, as indicated below:

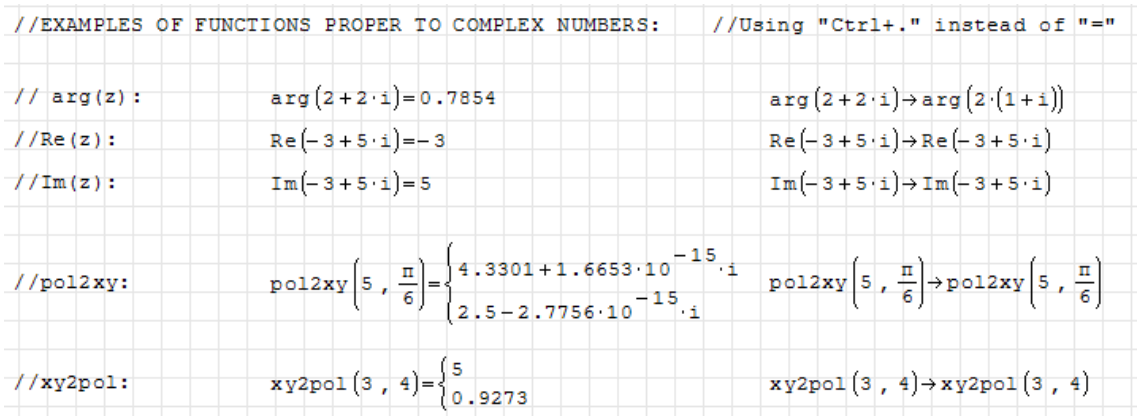
```
// Function "rand(x)" produces a random number uniformly distributed between 0 and x.
//  The argument "x" must be a positive number.  Other examples:

   random(10)=2        random(100)=9        random(200)=54       random(1000)=951

// Function "random" returns integer numbers.  If we were to need a random number
// between 0 and 1, with n decimal figures, use: random(10^n)/10^n, e.g.,

        random(10^2)/10^2 = 0.8        random(10^3)/10^3 = 0.321        random(10^4)/10^4 = 0.9049

// To generate a uniformly-distributed random number in the interval [a,b],
// with a<b, use:  a + (b-a)*random(10^n)/10^n, where n = 2, 3, 4, ..., e.g.:

                        20+(80-20)·random(10^3)/10^3 = 53.6

// You can write your own function "myrandom" to calculate random numbers:

                    myrandom(a, b, n) := a + ⌊(b-a)·random(10^n)/10^n⌋

//Examples:  myrandom(50, 100, 5)=76.378     myrandom(50, 100, 5)=78.82
             myrandom(50, 100, 5)=84.0755    myrandom(50, 100, 5)=69.4585

 The following example shows how to produce a row vector of random values in the range
 [50,100] using n=5:

  for k∈1..10                          1) Press "for" in the "Programming" palette
    x[1,k] := myrandom(50, 100, 5)     2) Type k in the first place holder in "for"
                                       3) Type "range(1,10)" in the second place holder in "for"
                                       4) Type "x[1,k <space bar> :" below the "for" line
                                       5) Click outside of the region, and type "x="

  x=(65.848 68.176 90.855 77.108 94.498 81.336 81.732 88.2085 90.936 66.7025)
```


Functions exclusive for complex numbers

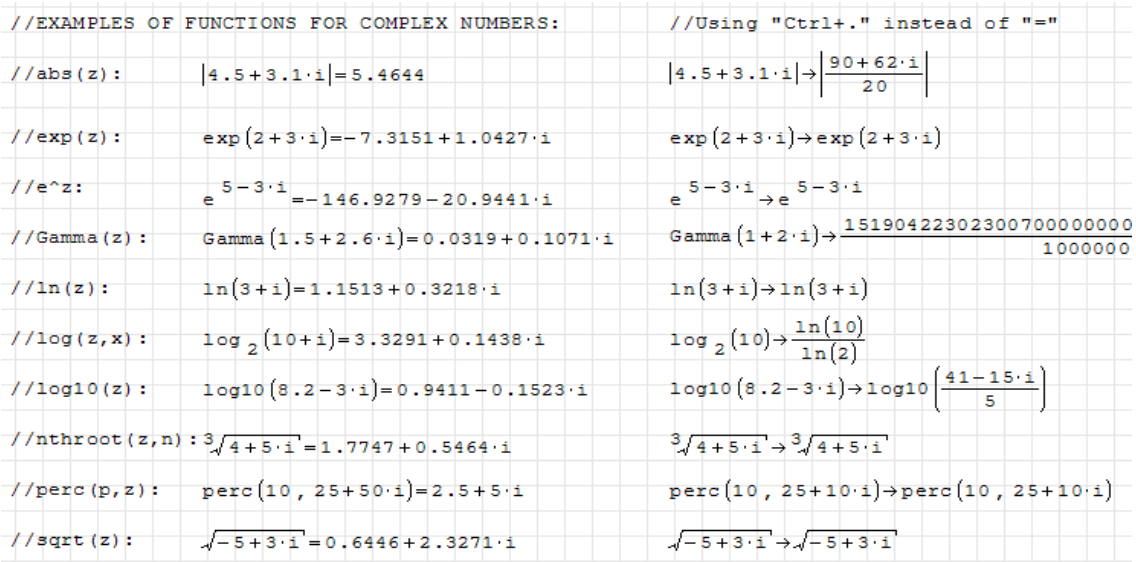
The *Function – Insert* form provides the following functions that apply exclusively to complex numbers (let $z = x+iy$ represent a complex number):

- arg — angle in complex plane, $arg(z) = atan(y/x)$
- Im — imaginary part, $Im(z) = y$
- pol2xy — convert polar coordinates to rectangular coordinates
- Re — real part, $Re(z) = x$
- xy2pol — convert rectangular coordinates to polar coordinates

The following examples show applications of these functions to complex numbers:

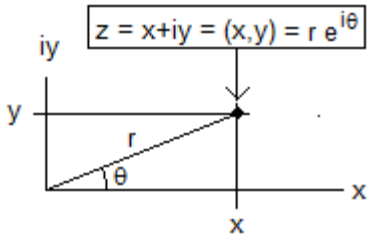


The function *abs,* when applied to a complex number, produces the modulo (length) of the complex number. Function *abs* is not included in the listing of *Complex Numbers* functions in the *Insert – Function* form. However, *abs*, and many other functions that we applied to real numbers above, can be applied to complex numbers as illustrated next:



Rectangular and polar representation of complex numbers

A complex number written in the form $z = x + iy$ is in its rectangular (or Cartesian) representation. Thus, it can be written also as the ordered pair (x,y), and be represented in an Argand diagram in which the abscissa is x and the ordinate is iy. An alternative way to represent point (x,y) is through its polar representation whose coordinates are (r,θ). The proper way to write the polar representation of a complex number is through the use of Euler's formula: $e^{i\theta}=\cos(\theta)+i\sin(\theta)$. With this result,



$$z=x+iy=r\cos(\theta)+i\,r\sin(\theta)=r(\cos(\theta)+i\sin(\theta))=r\,e^{i\theta}$$.

SMath Studio provides functions *xy2pol* to convert from rectangular (x,y) into polar (r,θ) coordinates, and *pol2xy* to convert from polar (r,θ) to rectangular (x,y) coordinates. Thus, with these functions one can go easily go from rectangular to polar representations of a complex number, and vice versa.

In the following example we convert from rectangular to polar representations of a complex number:

```
// EXAMPLE: Rectangular to polar

z1:=3+4·i                       // z1 in rectangular form

r1:=|z1|          r1=5          // components of polar form

θ1:=arg(z1)       θ1=0.9273

xy2pol(3 , 4)={5                // direct way to calculate polar form
              {0.9273           // with "xy2pol"
```

The following example shows a conversion from polar to rectangular representations of a complex number:

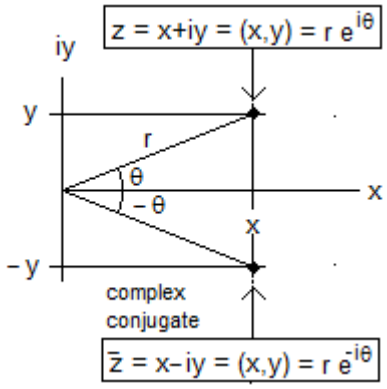
```
// EXAMPLE: Polar to rectangular

          i·π/6
z2:=10·e                        // z2 in polar form

z2=8.6603+5·i                   // shown directly in rectangular form

x2:=Re(z2)    x2=8.6603         // or you can separate components
                                //  of the rectangular form with Re, Im
y2:=Im(z2)    y2=5

pol2xy(10 , π/6)={8.6603+3.3307·10^-15·i   // Alternatively, use
                 {5-5.5511·10^-15·i        // pol2xy to calculate
                                           // rectangular components

Note: the imaginary parts of the results from "pol2xy" contain
numbers so small (e.g., 3.3307x10^(-15))that they're basically zero.
```


Operations with complex numbers

The following examples show operations with complex numbers in *SMath Studio*:

```
//Operations with complex numbers:
z1:=3.2-1.5·i          z2:=-5.2+2.2·i
// addition:           z1+z2=-2+0.7·i
// subtraction:        z1-z2=8.4-3.7·i
// multiplication:     z1·z2=-13.34+14.84·i
// division:           z1/z2=-0.6255+0.0238·i
// conjugate:          z1c:=Re(z1)-i·Im(z1)
                       z1c=3.2+1.5·i
```

The complex conjugate of a complex number is the reflection of the number $z = x + iy$ about the x axis, i.e., $\bar{z} = x - iy$. This is illustrated in the figure to the right:




```
//Powers of
//the imaginary
//unit:
```

$$\begin{pmatrix} i^2 \\ i^3 \\ i^4 \\ i^5 \end{pmatrix} = \begin{pmatrix} -1 \\ -i \\ 1 \\ i \end{pmatrix}$$

All other operations follow the rules of algebra with the caveat that $i^2 = -1$, etc. Other powers of the unit imaginary number are shown in the vector to the left.

Using the conjugate we can write: $z \cdot \bar{z} = r^2$.
This calculation is illustrated below:

```
z0:=-2.5+6.2·i   // number
z0c:=-2.5-6.2·i  // conjugate
z0·z0c=44.69     //number x conjugate
√(z0·z0c)=6.6851
|z0|=6.6851        // abs(z0)
```


The Gamma function

Most readers with courses in Algebra and Calculus I will be already familiar with most of the functions for real and complex numbers presented in this document. The *Gamma* function may be an exception, since it is an advanced mathematical function and probably would not have been introduced in those courses. The *Gamma* function is defined by an integral, namely,

$$\Gamma(x)=\int_0^{\infty} t^{z-1}e^{-t}dt \quad .$$

The *Gamma* function is related to the factorial operator as follows: $\Gamma(x+1)=x!$, if x is an integer.

The following examples use the *Gamma* function in some calculations:

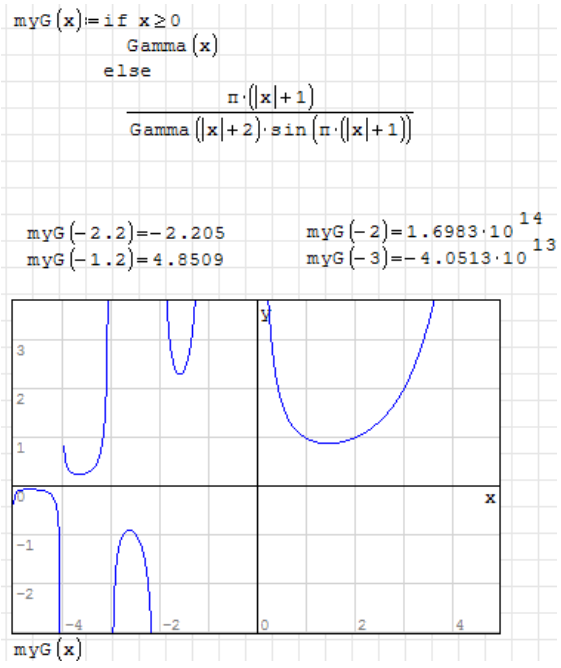
```
Gamma(5)=24        4!=24
Gamma(5.6)=61.554
Gamma(3+3·i)=-0.4401-0.0636·i
```

Note: the *Gamma* function currently defined in *SMath Studio 0.85* cannot handle negative arguments, or complex arguments whose real part is negative. For many applications this definition will be fine, but the full definition of the *Gamma* function should be able to handle negative arguments. Based on the paper **"***A note on the computation of the convergent Lanczos complex Gamma approximation*" by Paul Godfrey (2001), found in http://home.att.net/~numericana/answer/info/godfrey.htm#lanczoscoeffs , I redefined the *Gamma* function to include negative arguments, as follows:

The figure to the right also shows some calculations of the modified *Gamma* function, and a graph of the function.

Compare the graph with that shown in the one shown in the wikipedia entry:

http://en.wikipedia.org/wiki/Gamma_function



User-defined functions
You can define your own functions by using the expression

function_name(*argument*(*s*)): *function_expression*

Examples of definitions and evaluations of functions are shown below:

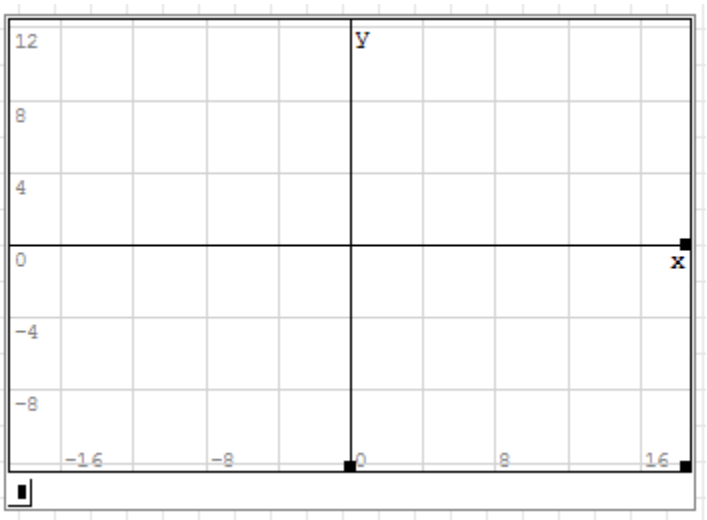
$f(x) := \frac{x^2+1}{x+2}$ $\quad f(2)=1.25$ $\quad f(3)=2$

$g(x, y) := x^2+y^2$ $\quad g(2, 3)=13$ $\quad g(5, -2)=29$

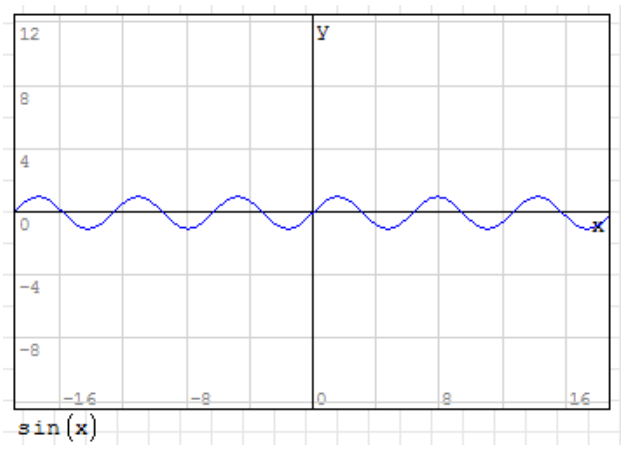
$h(x) := \sum_{k=1}^{x} \left(k^2\right)$ $\quad h(2)=5$ $\quad h(3)=14$

$r(x) := \int_0^x \frac{1}{\sqrt{1+t}}\, dt$ $\quad r(2)=1.4641$ $\quad r(3)=2$

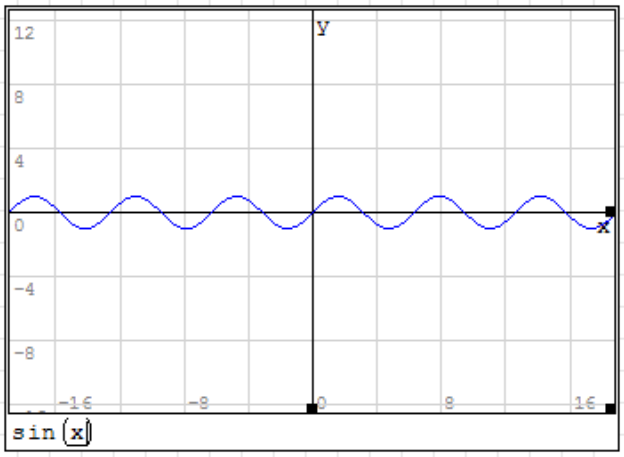
Plots of functions in 2D – a brief introduction
SMath Studio allows the user to insert two-dimensional (2D) plots by using the *2D* option in the *Plot* palette, or the menu option *Insert* > *Plot* > *2D*. This action will open a 2-D graph window, as illustrated below:



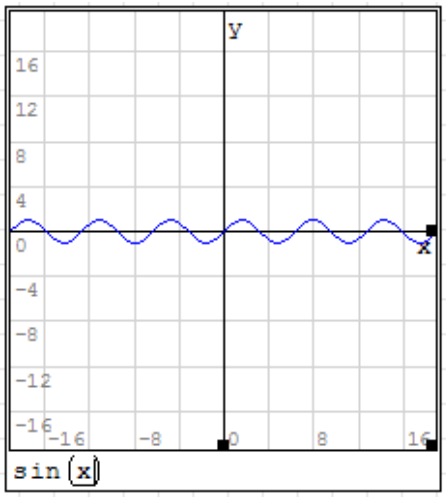
In the place holder on the lower left corner of the graph window you can enter the name of a function to plot, e.g.,

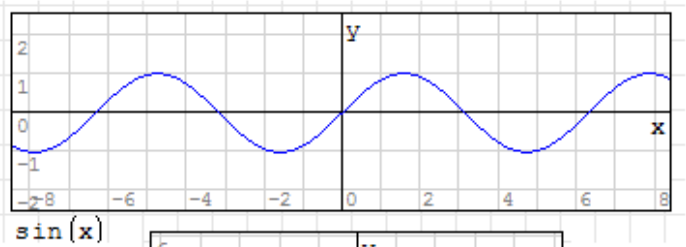


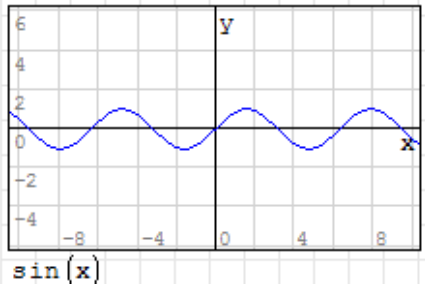
If you click on the graph window, you can drag one of the window handles shown below (right-hand side, bottom, or lower right corner) to modify the size of the window.



Some resizing examples are illustrated below:

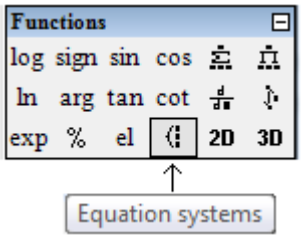






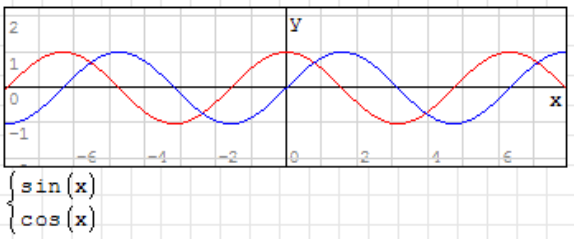
Plotting more than one function

To plot one than one function, use the *Equation systems* option in the *Functions* palette, namely,



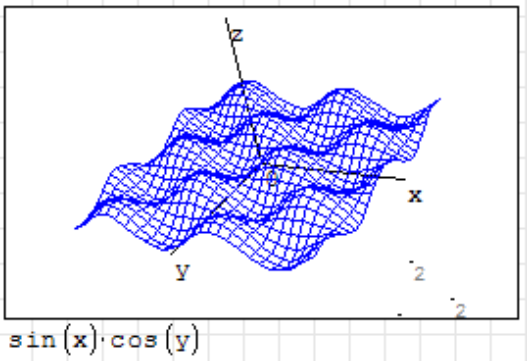
,

to insert more than one function in the function placeholder for the graph window. Here is an example:

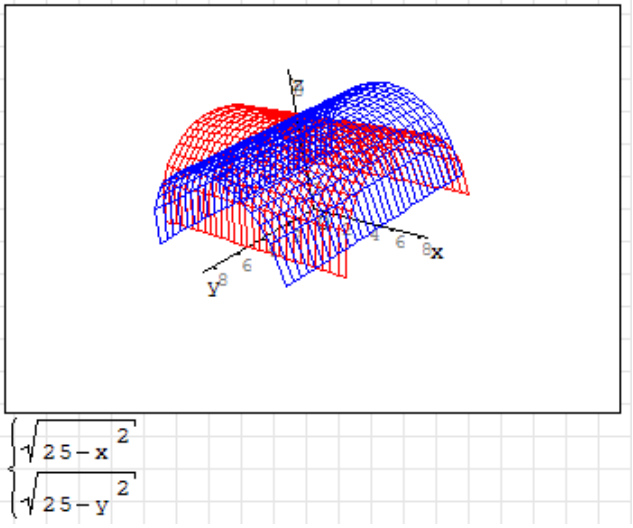


Plots of functions in 3D – a brief introduction

Use the *3D* option in the *Plots* palette, or the menu option *Insert* > *Plots* > *3D*. Type the equation *f*(*x*,*y*) in the place holder located in the lower left corner of the graph window. You can drag the mouse pointer in the graph window to rotate the graph for a better view of the mesh surface, as illustrated here:



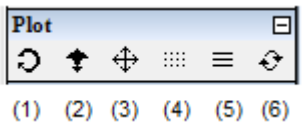
Resizing of the graph is also possible as done for 2D graphs.



As we did with 2D plots, you can plot more than one 3D plot as illustrated in the graph to the left.

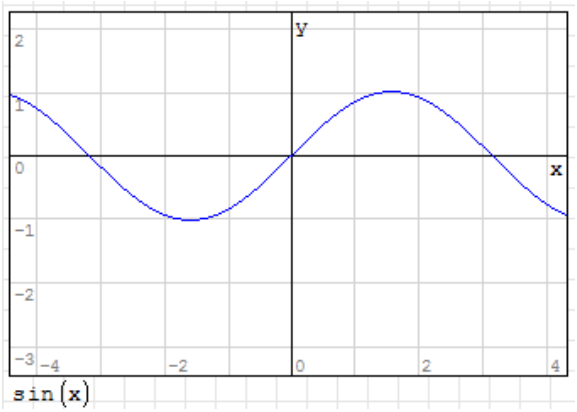
Additional manipulation of graphs

Additional manipulation of graphs is possible using the *Plot* palette. The *Plot* palette includes the following buttons:



(1) Rotate – for 3D graphs, default setting
(2) Scale – With *Ctrl* or *Shift*, while dragging mouse over the graph, for proportional scaling – play around with this command to understand its operation
(3) Move – move the graph around in existing graph window
(4) Graph by points – show graph made of discrete points
(5) Graph by lines – show graphs made of continuous lines
(6) Refresh – refresh graph window to its default settings

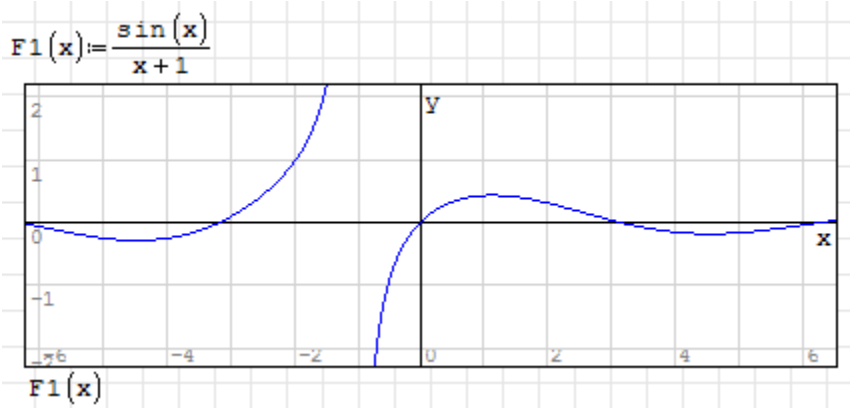
Try these options on your own to learn more about the different *Plot* palette actions. The example below shows a proportional scaling using item (3) above, while holding down the *Shift* key.



NOTES: (1) For additional examples of 2D and 3D graphics see Appendix 1.
(2) For examples of 2D geometric figures see Appendix 2.

Plotting your own functions

Define a function as indicated earlier, and replace the function name in the graph window place holder. Here is an example:

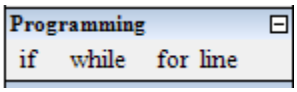


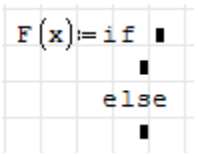
Defining functions with the *if* function

Some functions require more than one expression for their definition. For example,

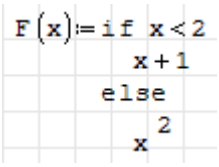
$$F(x)=\begin{cases} x+1, & x<2 \\ x^2, & x\geq 2 \end{cases}.$$

This function can be defined using the *if* function in the *Programming* palette. When selecting this option, the following programming structure, with placeholders, is produced:





To program the function shown above, we can use:

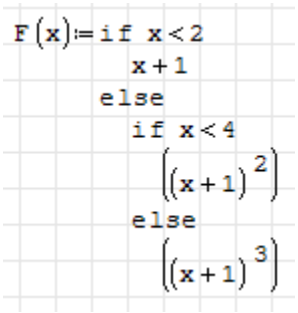


If the definition of the function requires more than two expressions, it will be necessary to use nested *if* programming structures. Consider the following example:

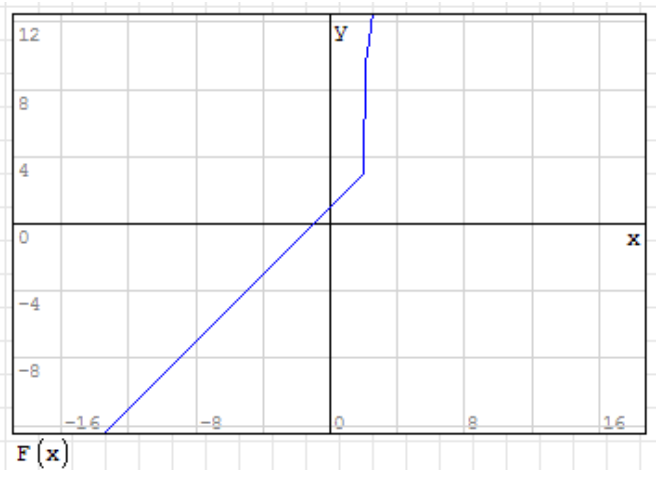
Function defined as:

$$F(x)=\begin{cases} x+1, & x<2 \\ (x+1)^2, & 2\leq x\leq 4 \\ (x+1)^3, & x>4 \end{cases}$$

Function programmed as:



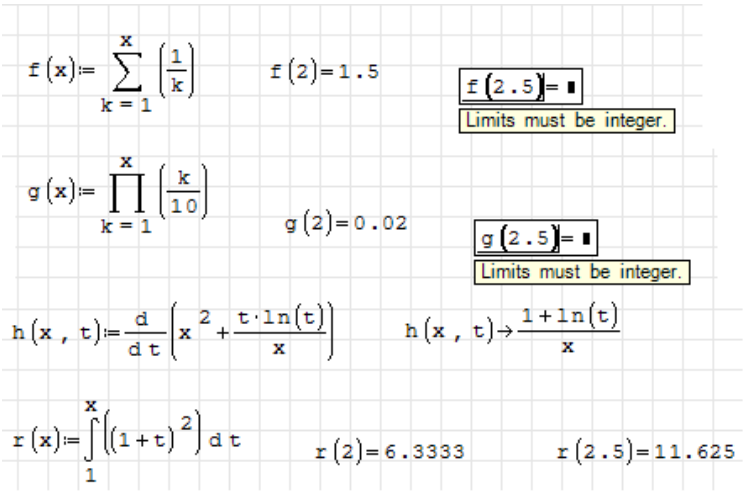
A plot of this function is shown below:



More details on programming in *SMath Suite* will be provided in a later section.

Functions defined by calculus operations

The *Functions* palette include four operations that are proper from calculus: summation, product, derivative, and integrals. These functions can be incorporated in the definition of functions as illustrated in the following examples:



Notice that the functions $f(x)$ and $g(x)$, whose argument defines the upper limit of the summation and product, are only defined for integer values.

Solving single equations with *solve*

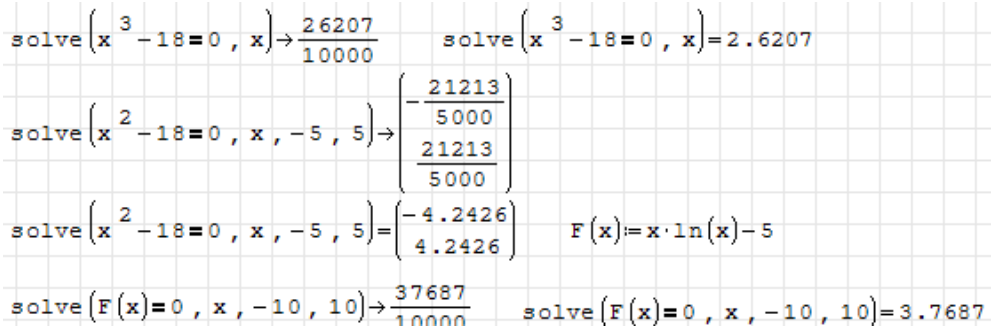
Function *solve* can be used to solve single equations. The function can be called using either of the following calls:

solve(*equation*, *variable*)

or

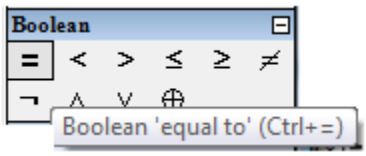
solve(*equation*, *variable, lower limit, upper limit*)

Examples using both symbolic and numeric results are shown below:



Function *solve* provides only real roots. Polynomial equations can be solved using *polyroots*, as described below.

Note: to enter the Boolean equal sign (bold =) use *Ctrl*+=, or the symbol in the *Boolean palette.*



Application: Solving for the flow depth in an open channel flow

The following example shows how to use function *solve* to calculate the normal depth of flow in an open channel flow of circular cross-section. The problem statement and solution commentaries were all entered using *SMath Studio*. Also, a figure was created using *Window's Paint* and pasted into the *SMath Studio* worksheet. To facilitate the solution, units were not used. The solution shown is *y = 1.7077 ft*.

Problem from open-channel hydraulics:

An open channel with a circular cross-section of diameter D = 3 ft is laid on a slope So = 0.0005 and carries a discharge Q = 10 cfs. If the channel is made of concrete, with a Manning's coefficient n = 0.012, find the normal depth of flow, y = ?

Solution: Use Manning's equation:

Q = discharge (cfs),
n = Manning's coefficient,
A = area (ft^2),
P = wetted perimenter (ft),
So = bed slope.

$$Q = \frac{Cu}{n} \cdot \frac{A^{\frac{5}{3}}}{P^{\frac{2}{3}}} \cdot \sqrt{So}$$

Functions for A and P are defined below. These functions use the auxiliary variable θ(y), that represents the half center angle in the flow.

To facilitate solution, we don't use units, but the values used in units of the English system are consistent:

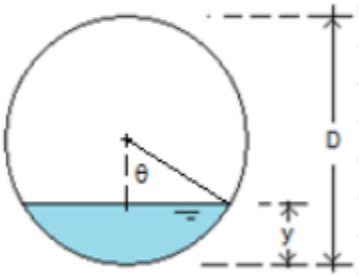
$D := 3 \quad Q := 10 \quad So := 0.0005$

$Cu := 1.486 \quad n := 0.012$

$\theta(y) := \operatorname{acos}\left(1 - 2 \cdot \frac{y}{D}\right)$

$P(y) := \theta(y) \cdot D$

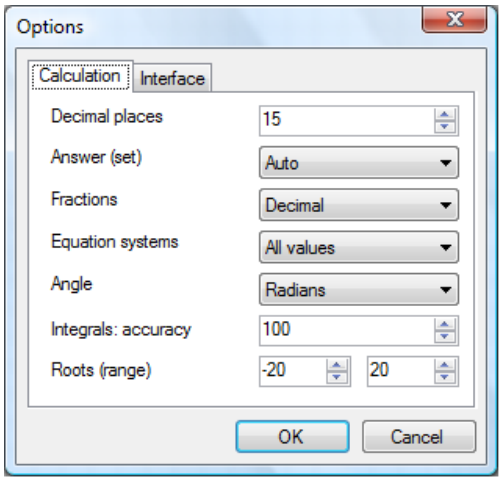
$A(y) := \frac{D^2}{4} \cdot \left(\theta(y) - \sin(\theta(y)) \cdot \cos(\theta(y))\right)$



$$\text{solve}\left(Q = \frac{Cu}{n} \cdot \frac{A(y)^{\frac{5}{3}}}{P(y)^{\frac{2}{3}}} \cdot \sqrt{So},\ y,\ 0,\ 3\right) = \begin{pmatrix} 1.7077 \\ 1.7077 \end{pmatrix}$$

Changing the number of decimals in the output

Most of the numeric calculations and equation solutions shown so far in this document use four decimals (default value). Using the *Tools* > *Options* menu item you can change the number of decimals in the output. For example, changing to 15 decimals for output, we solve the equation presented above to obtain the following results:



$$\text{solve}\left(x^2 - 18 = 0,\ x\right) = \begin{pmatrix} -4.24264068711929 \\ 4.24264068711928 \end{pmatrix}$$

For the rest of the examples in this document we will use the default number of decimal figures, i.e., 4 decimals.

Solving polynomial equations with *polyroots*

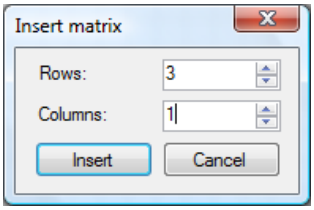
Function *polyroots* returns the roots of a polynomial using as argument a column vector with the polynomial coefficients. The coefficients must be entered in the vector in increasing order of the power of the independent variable. For example, the vector corresponding to the polynomial $1 + 2x - 5x^2$ is:

$$v = \begin{bmatrix} 1 \\ 2 \\ -5 \end{bmatrix}.$$

Using *SMath Studio*, the roots of this polynomial are calculated as follows:

$$v := \begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix} \qquad \text{polyroots}(v) = \begin{pmatrix} -0.2899 \\ 0.6899 \end{pmatrix}$$

To enter a column vector, a row vector, or a matrix, use the first icon in the *Matrices* palette. This icon opens a dialogue box where the user can select the number of rows and columns for defining a vector or matrix. By default, the dialogue box uses a 3×3 matrix. For the column vector *v*, shown above, we used, of course, 3 rows and 1 column:



Unlike function *solve*, which provides only real functions, function *polyroots* provides either real or complex roots, or both. For example:

$$w := \begin{pmatrix} 2 \\ -3 \\ 5 \\ 7 \end{pmatrix} \qquad \text{polyroots}(w) = \begin{pmatrix} 0.2647 + 0.3996 \cdot i \\ 0.2647 - 0.3996 \cdot i \\ -1.2436 \end{pmatrix}$$

Application: Solving homogeneous, linear, constant-coefficients ODEs

A *n*-th order, homogeneous, linear ordinary differential equation (ODE) with constant coefficients is represented by the expression:

$$\frac{d^n x}{dt^n} + a_{n-1}\frac{d^{n-1}x}{dt^{n-1}} + \ldots + a_2\frac{d^2x}{dt^2} + a_1\frac{dx}{dt} + a_0 x = 0$$

Introducing the notation $D^n = d^n(\,)/dt^n$, the general *n*-th order equation can be written as:

$$D^n x + a_{n-1}D x + \ldots + a_2 D^2 x + a_1 D x + a_0 x = 0$$

This expression can be "factored" in terms of the *D* operators to read:

$$(D^n + a_{n-1}D^{n-1} + \ldots + a_2 D^2 + a_1 D + a_0)\, x = 0$$

If we take the expression between parentheses and replace the *D* operators with the variable λ, and make it equal to zero, we would have produced the so-called *characteristic equation* of the ODE, i.e.,

$$\lambda^n + a_{n-1}\lambda^{n-1} + \ldots + a_2\lambda^2 + a_1\lambda + a_0 = 0$$

There are, in principle, *n* solutions of this polynomial equation, namely, $\lambda_1, \lambda_2, \ldots, \lambda_n$. Some of these solutions could be complex (or imaginary), or even repeated. In the case that all solutions are real and non-repeating, the general solution to the original homogeneous ODE is given by:

$$x(t) = A_1 e^{(\lambda_1 t)} + A_2 e^{(\lambda_2 t)} + \ldots + A_n e^{(\lambda_n t)}$$

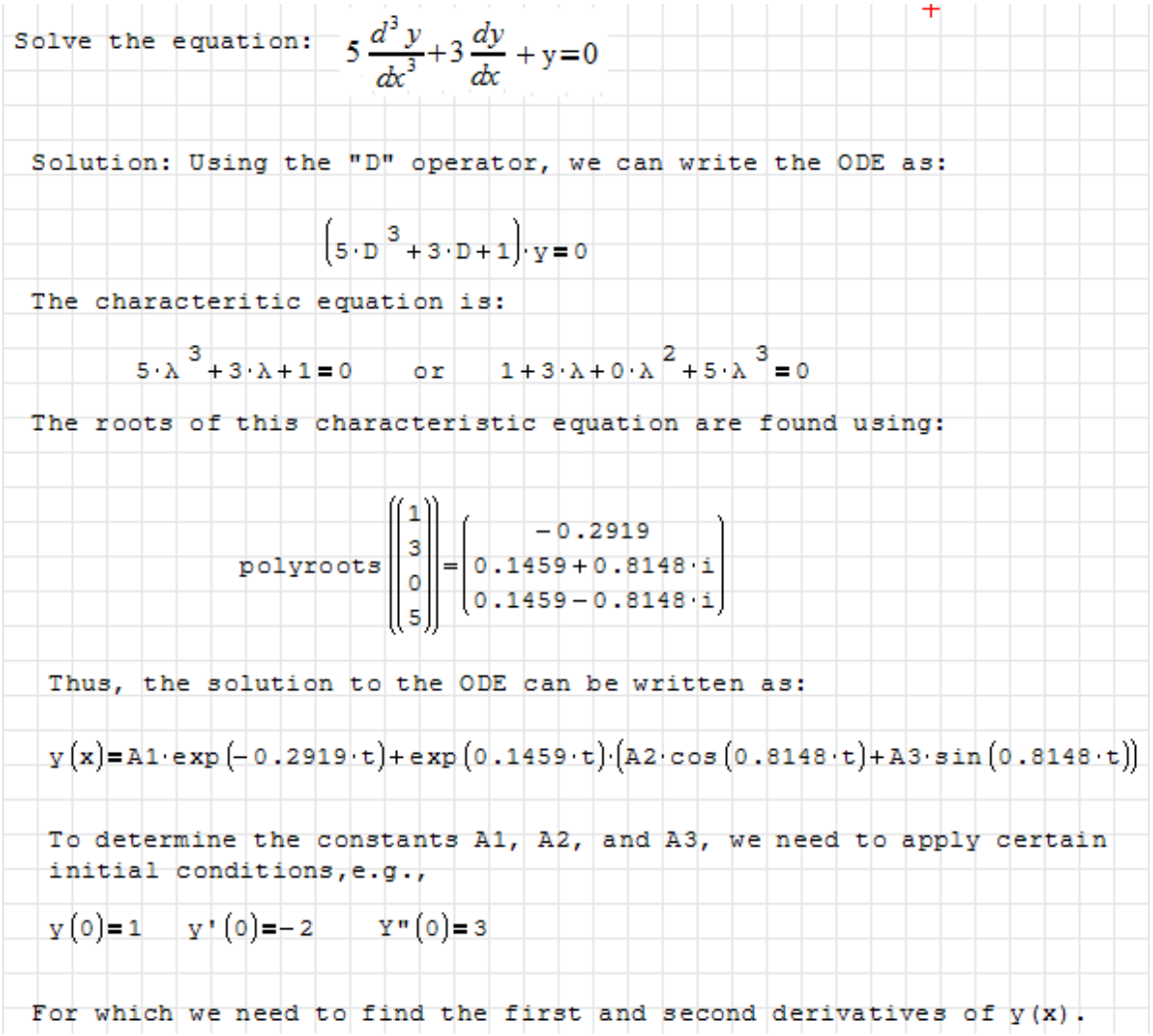
If a root λ_m, repeats *k* times, it will result in *k* terms in the solution given by

$$e^{\lambda_m t} \cdot (A_1 + A_2 t + A_3 t^2 + \cdots + A_k t^{k-1})$$

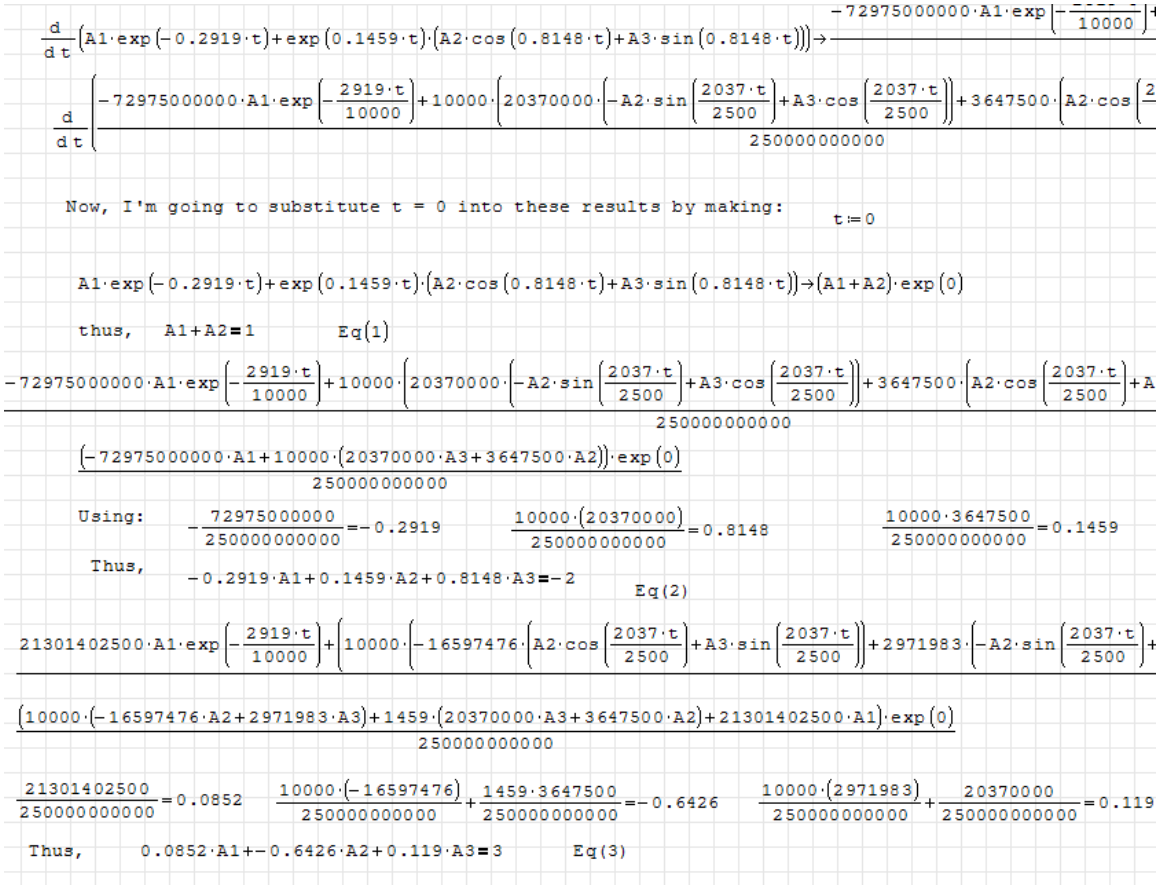
If there are complex roots, they will be an even number of them, i.e., pairs of complex conjugates. Thus, if one of those pairs of complex conjugate roots is $\lambda_1 = \alpha + \beta$ i and $\lambda_2 = \alpha - \beta$ i, then the following terms will appear in the solution:

$$A_1 e^{\alpha t} \cos(\beta t) + A_2 e^{\alpha t} \sin(\beta t)$$

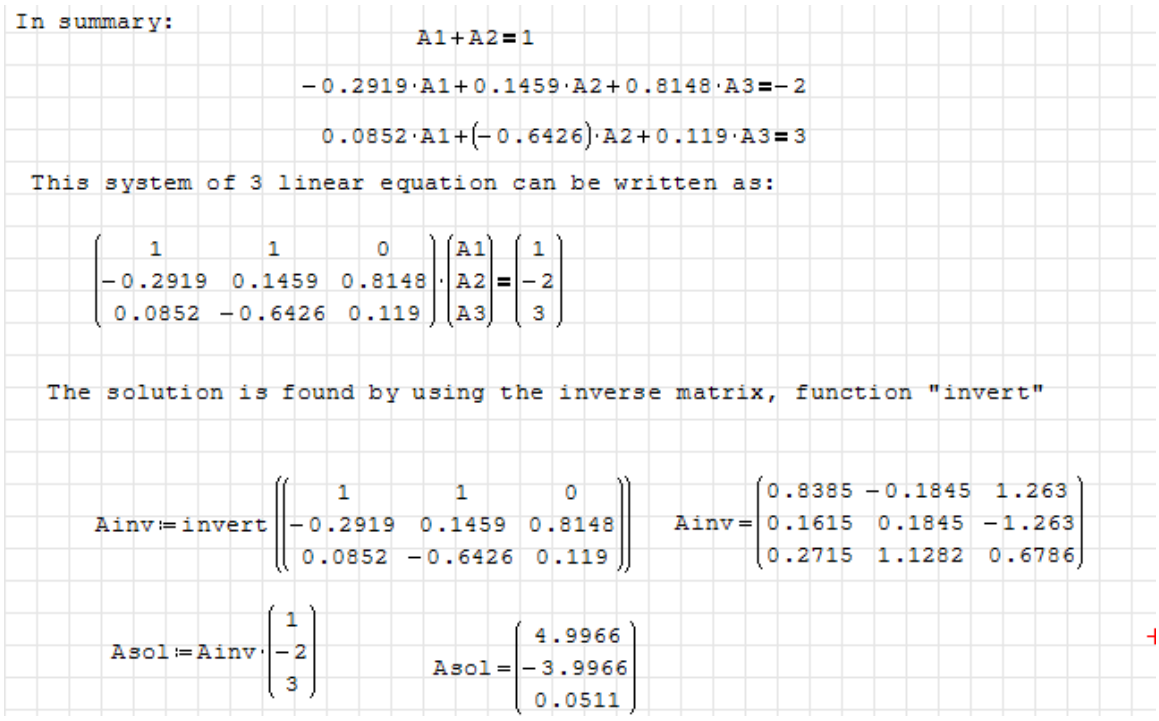
Function *polyroots* can be used to solve the characteristic equation, or characteristic polynomial, for an *n*-th order, homogeneous, linear ODE. The roots of the characteristic polynomial can then be used to build the solution. One example is shown below.

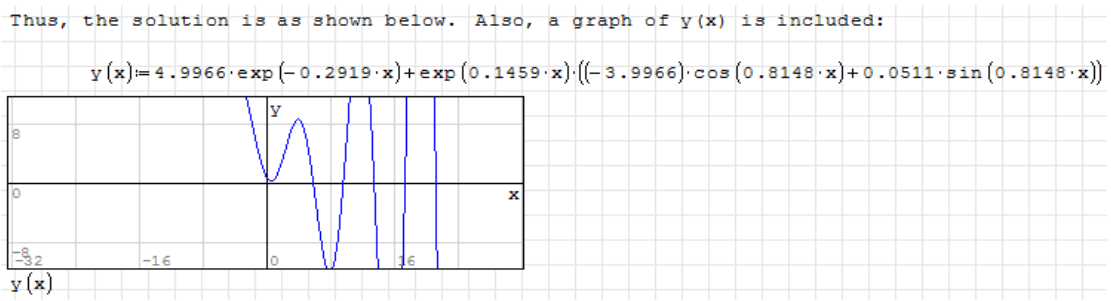


The calculation of derivatives is all performed using symbolic calculations, and the results are very long. Also, the current version of *SMath Studio* (version 0.85, 09/2009) doesn't simplify results and keeps the value *exp*(*0*) instead of 1. Helping *SMath Studio* with these shortcomings, and with a lot of patience, you can get to put together the system of linear equations needed to solve for the coefficients A1, A2, and A3, in the solution to this ODE. (The figure below shows only parts of some calculations).



The resulting equations can be summarized as follows:





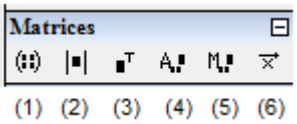
And, since we are dealing with matrices already, we introduce their use in *SMath Studio* next.

Operations with matrices

The functions available under the *Insert* > *Function* menu under the heading *Matrix and vector* can be roughly classified into the following categories:

- Creating matrices: *augment, diag, identity, mat, matrix, stack*
- Extracting rows, columns, elements: *col, el, row, submatrix, vminor, minor*
- Characterizing matrices: *cols, det, length, max, min, norm1, norme, normi, rank, rows, tr*
- Sorting functions: *csort, reverse, rsort, sort*
- Matrix operations: *alg, invert, transpose*

Some of these functions are also available in the *Matrices* palette:



(1) Matrix 3x3 (Ctrl+M) (*mat*)
(2) Determinant (*det*)
(3) Matrix transpose (Ctrl+1) (*transpose*)
(4) Algebraic addition to matrix (*alg*)
(5) Minor (*minor*)
(6) Cross product

The only function from the *Matrices* palette not available from the *Insert* > *Function* > *Matrix and vector* functions is (6) Cross product, which applies to two vectors of three elements.

It is important to indicate that even though you can form row vectors, many of the matrix and vector functions defined in *SMath Studio* apply only to column vectors. Thus, column vectors are considered true vectors for the purpose of matrix operations in *SMath Studio*. For example, to calculate a cross-product you will need two column vectors of three elements. Dot products, on the other hand, can be performed on column vectors of any length. The dot product of two column vectors is calculated as the matrix product of the transpose of one of the vectors times the second vector. Examples of matrix and vector operations follow. The vector and matrix names used in the examples suggest the dimensions of the vector or matrix. For example, u2x1 suggest the vector $\mathbf{u}_{2\times1}$, and A3x3 suggests the matrix $\mathbf{A}_{3\times3}$.

Examples of matrix creation:

$$u2x1 := \begin{pmatrix} 3 \\ 1 \end{pmatrix} \quad u3x1 := \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix} \quad v3x1 := \begin{pmatrix} 0 \\ -1 \\ -5 \end{pmatrix} \quad A3x3 := \begin{pmatrix} -2 & 3 & 2 \\ 0 & 1 & -5 \\ -3 & 5 & 2 \end{pmatrix}$$

Matrix creation examples:

$M3 := augment(u3x1, A3x3)$

$$M3 = \begin{pmatrix} 1 & -2 & 3 & 2 \\ -3 & 0 & 1 & -5 \\ 2 & -3 & 5 & 2 \end{pmatrix}$$

$M1 := diag(u2x1)$

$$M1 = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$$

$M2 := diag(v3x1)$

$$M2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -5 \end{pmatrix}$$

$I3x3 := identity(3)$

$$I3x3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$M0 := matrix(3, 3)$

$$M0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Examples of extracting rows, columns, or elements from matrices:

$vec01 := col(A3x3, 2)$

$$vec01 = \begin{pmatrix} 3 \\ 1 \\ 5 \end{pmatrix}$$

$x := A3x3_{2\,2}$

$x = 1$

$vec02 := row(A3x3, 3)$

$$vec02 = \begin{pmatrix} -3 & 5 & 2 \end{pmatrix}$$

$MM1 := submatrix(A3x3, 2, 3, 2, 3)$

$$MM1 = \begin{pmatrix} 1 & -5 \\ 5 & 2 \end{pmatrix}$$

Function "vminor":

$MM2 := M_{2\,2}(A3x3)$

$$MM2 = \begin{pmatrix} -2 & 2 \\ -3 & 2 \end{pmatrix}$$

Function "minor":

$MMin := M_{2\,2}(A3x3)$

$MMin = 2$

Examples of matrix characterization:

$$A3x3 := \begin{pmatrix} -2 & 3 & 2 \\ 0 & 1 & -5 \\ -3 & 5 & 2 \end{pmatrix}$$

Characterizing matrices:

$cols(A3x3) = 3$	$rows(A3x3) = 3$
$\lvert A3x3 \rvert = -3$	$length(A3x3) = 9$
$length(A3x3) = 9$	$max(A3x3) = 5$
$min(A3x3) = -5$	$norm1(A3x3) = 9$
$norme(A3x3) = 9$	$normi(A3x3) = 10$
$rank(A3x3) = 3$	$tr(A3x3) = 1$

Examples of matrix/vector sorting functions:

$$A3x3 := \begin{pmatrix} -2 & 3 & 2 \\ 0 & 1 & -5 \\ -3 & 5 & 2 \end{pmatrix} \qquad u3x1 := \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix}$$

Sorting:

$$csort(A3x3, 1) = \begin{pmatrix} -3 & 5 & 2 \\ -2 & 3 & 2 \\ 0 & 1 & -5 \end{pmatrix} \qquad rsort(A3x3, 2) = \begin{pmatrix} 2 & -2 & 3 \\ -5 & 0 & 1 \\ 2 & -3 & 5 \end{pmatrix}$$

$$reverse(A3x3) = \begin{pmatrix} -3 & 5 & 2 \\ 0 & 1 & -5 \\ -2 & 3 & 2 \end{pmatrix} \qquad reverse(u3x1) = \begin{pmatrix} 2 \\ -3 \\ 1 \end{pmatrix}$$

$$sort(u3x1) = \begin{pmatrix} -3 \\ 1 \\ 2 \end{pmatrix}$$

Examples of matrix operations:

$$A3x3 := \begin{pmatrix} -2 & 3 & 2 \\ 0 & 1 & -5 \\ -3 & 5 & 2 \end{pmatrix} \qquad B3x3 := \begin{pmatrix} 0 & -1 & 2 \\ -2 & 8 & 3 \\ 4 & -4 & -8 \end{pmatrix} \qquad v3x1 := \begin{pmatrix} 0 \\ -1 \\ -5 \end{pmatrix}$$

Matrix operations:

$$invert(A3x3) = \begin{pmatrix} -9 & -1.3333 & 5.6667 \\ -5 & -0.6667 & 3.3333 \\ -1 & -0.3333 & 0.6667 \end{pmatrix} \qquad A3x3 + B3x3 = \begin{pmatrix} -2 & 2 & 4 \\ -2 & 9 & -2 \\ 1 & 1 & -6 \end{pmatrix}$$

Function "transpose":

$$A3x3^{T} = \begin{pmatrix} -2 & 0 & -3 \\ 3 & 1 & 5 \\ 2 & -5 & 2 \end{pmatrix} \qquad A3x3 \cdot v3x1 = \begin{pmatrix} -13 \\ 24 \\ -15 \end{pmatrix}$$

Examples of vector products:

Vector products:

$$u3x1 := \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix} \qquad v3x1 := \begin{pmatrix} 0 \\ -1 \\ -5 \end{pmatrix}$$

Dot product: $u3x1^{T} \cdot v3x1 = (-7)$

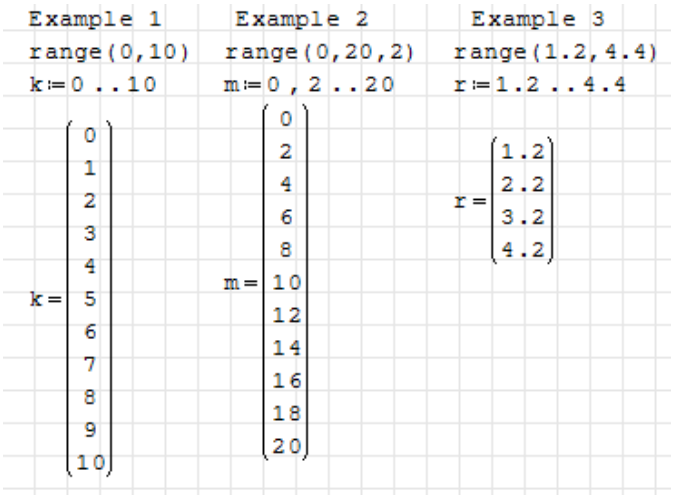
Cross product: $u3x1 \times v3x1 = \begin{pmatrix} 17 \\ 5 \\ -1 \end{pmatrix}$

Ranges and vectors

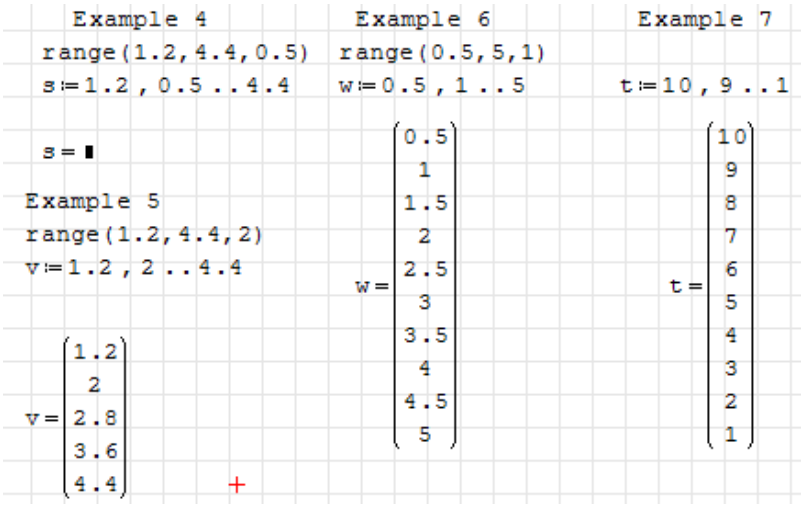
Function *range* is used to generate a vector (column vector) with elements equally spaced. The general form of the *range* function is as follows:

- *range*(*start*, *end*): shows up in the worksheet as *start..end*, and produces a vector whose elements are *start*, *start* + 1, *start* + 2, etc. The last element is the lowest value smaller than *last* by less than one.
- *range*(*start, end, start+increment*): shows up in the worksheet as *start, start+increment .. end*, and produces a vectors whose elements are *start, start + increment, start + 2 increment,* etc. The last element is the lowest value smaller than *end* by less than *increment*.

The following examples illustrate the use of function *range*. The examples are numbered 1 through 7. Immediately below the Example number the *range* function call that produces the range is shown. For instance, in *Example 1*, typing the expression *range(0,10)* produces the range *0..10*, assigned to *k*.



- Example 1 shows a vector from *0* to *10* with default increment of *1*.
- Example 2 shows a vector from *0* to *20* with increment of *2*.
- Example 3 shows a vector from *1.2* to *4.4* with default increment of *1*.



- Example 4 shows an impossibility since the value of *start* + *increment* < *start*, but *end* > *start*
- Example 5 shows a vector from *1.2* to *4.4* in increments of *2*
- Example 6 shows a vector from *0.5* to *5* in increments of *0.5*
- Example 7 shows a vector from 10 down to 1 in negative increments of *-1*

Vectors can be combined linearly to produce other vectors, as illustrated here:

$$x := 0..5 \qquad y := 2 \cdot x + 3 \qquad z := 0.8 \cdot x + 0.7 \cdot y$$

$$x = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \quad y = \begin{pmatrix} 3 \\ 5 \\ 7 \\ 9 \\ 11 \\ 13 \end{pmatrix} \quad z = \begin{pmatrix} 2.1 \\ 4.3 \\ 6.5 \\ 8.7 \\ 10.9 \\ 13.1 \end{pmatrix}$$

The following examples illustrate three different ways to calculate products of vectors of the same lengths:

1. Product of the vectors produces the scalar (dot) product
2. Product of a vector transpose with another vector produces the same scalar (dot) product as the only element of a 1×1 matrix
3. Product of a vector with the transpose of another vector produces a matrix

$$x \cdot x = 55 \qquad x \cdot y = 155$$

$$x^T \cdot x = (55) \qquad x^T \cdot y = (155)$$

$$x \cdot x^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 4 & 6 & 8 & 10 \\ 0 & 3 & 6 & 9 & 12 & 15 \\ 0 & 4 & 8 & 12 & 16 & 20 \\ 0 & 5 & 10 & 15 & 20 & 25 \end{pmatrix}$$

Individual elements of a vector can be extracted by using the name of the vector with a sub-index. To enter a sub-index, type the name of the vector and press the left square bracket key ([). Press the right arrow key [→] a couple of times to select the indexed variable. Examples:

$$x := 0..5$$
$$x_3 = 2$$

Vectors of the same length can be put together in a matrix through function *augment*:

$$\text{augment}(x, y, z) = \begin{pmatrix} 0 & 3 & 2.1 \\ 1 & 5 & 4.3 \\ 2 & 7 & 6.5 \\ 3 & 9 & 8.7 \\ 4 & 11 & 10.9 \\ 5 & 13 & 13.1 \end{pmatrix}$$

This can be used to build tables of data.

for loops and parametric plots

A parametric plot is defined by points $(x(t), y(t))$, where t is referred to as the parameter. In this example we generate data (x,y) and (y,x) using *for* loops, and plotting the resulting matrices.

First create column vectors x and y with 21 elements:

$x := \text{matrix}(21, 1)$ $n := \text{length}(x)$ $n = 21$ $y := x$

Use a "for" loop to generate parametric data in x,y:

for $k \in 1..n$

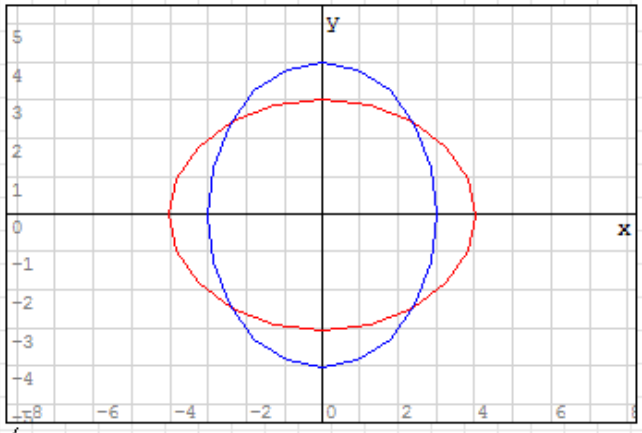
$x_k := 3 \cdot \sin\left(k \cdot \frac{\pi}{10}\right)$

$y_k := 4 \cdot \cos\left(k \cdot \frac{\pi}{10}\right)$

Note: in this program we use the "for" and the "line" commands from the "Programming" palette.

Form matrices with data points (x,y) and (y,x), and produce a plot of the matrices M1 and M2:

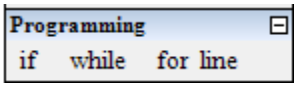
$M1 := \text{augment}(x, y)$ $M2 := \text{augment}(y, x)$



M1
M2

Commands in the *Programming* palette

The *Programming* palette includes four commands, namely, *if, while, for*, and *line*. We have presented examples of the commands *if*, *for*, and *line*, in this document. Here is an example of the *while* command:



$k := 10$ $b := 0$ (initial values)

while $k \geq 5$

$b := b + k$

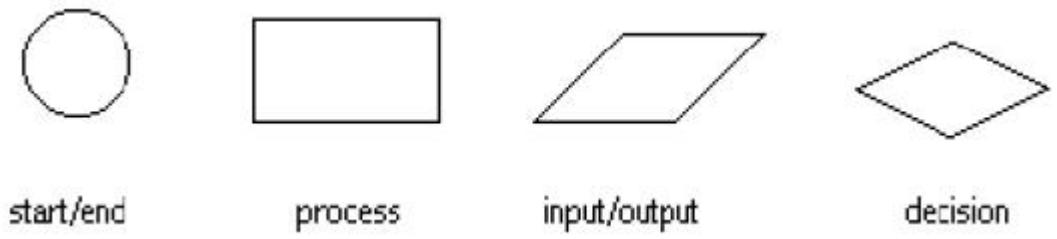
$k := k - 1$

$k = 4$ $b = 45$ (final values)

Introduction to Programming with SMath Studio
In this section we introduce basic concepts of programming for numerical solutions in *SMath Studio.*

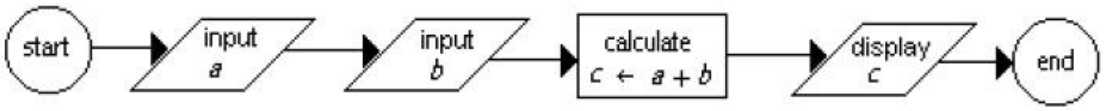
Programming structures and flowcharts
Programming, in the context of numerical applications, simply means controlling a computer, o other calculating device, to produce a certain numerical output. In this context, we recognize three main programming structures, namely, (a) sequential structures; (b) decision structures; and (c) loop structures. Most numerical calculations with the aid of computers or other programmable calculating devices (e.g., calculators) can be accomplished by using one of more of these structures.

The operation of these programming structures will be illustrated by the use of flow charts. A flow chart is just a graphical representation of the process being programmed. It charts the flow of the programming process, thus its name. The figure below shows some of the most commonly used symbols in flowcharts:

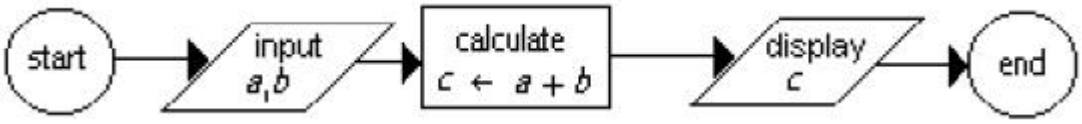


In a flowcharts, these symbols would be connected by arrows pointing in the direction of the process flow.

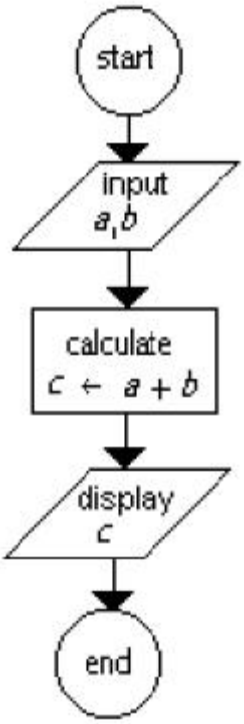
Sequential structures
A complete programming flowchart would have *start* and *end* points, and at least one *process* block in between. That would constitute the simplest case of a *sequential structure.* The following figure shows a sequential structure for the calculation of a sum:



This flowchart can be simplified slightly by using a single input block to enter both *a* and *b*:



Typically, a sequential structure is shown following a vertical direction:



The sequential structure shown in this flowchart can also be represented using *pseudo-code*. Pseudo-code is simply writing the program process in a manner resembling common language, e.g.,

```
Start
Input a, b
c ← a + b²
Display c
End
```

A flowchart or pseudo-code can be translated into code in different ways, depending on the programming language used. In *SMath Studio*, this sequential structure could be translated into the following commands:

```
a:=2  b:=3  // Input a,b
c:=a+b      // c <- a + b
 c=5        // Display c
```

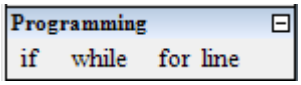
Here is another example of a sequential structure in *SMath Studio* showing more than one calculation step. The sequential structure in *SMath Studio* doesn't have to follow a strict vertical direction, as illustrated below.

```
SEQUENTIAL STRUCTURE:
 x1:=-10          y1:=2
 x2:=5            y2:=-3
 Δx:=x2-x1        Δx=15
 Δy:=y2-y1        Δy=-5

 d1:=√(Δx^2+Δy^2)  d1=15.8114
```

```
Sequential structure
in a program "line":
 xA:=-10   yA:=2
 xB:=5     yB:=-3
 Δxx:=xA-xB
 Δyy:=yA-yB
 d2:=√(Δxx^2+Δyy^2)
 d2=15.8114
```

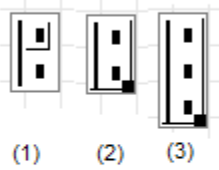
The *line* command and the *Programming* palette

The sequential structure is shown in the right-hand side of the figure above collecting the calculation steps under a programming line. The figure below illustrate the instructions to insert a programming line in a *SMath Studio* worksheet. The *line* command, together with other programming commands, is listed in the *Programming* palette shown here:



[2]The algorithmic statement $c \leftarrow a+b$ represents the assignment $c := a + b$ in *SMath Studio*


```
The "line" command can be entered:
(a) Using "line" in the "Programming" palette
(b) Typing "line(" in an entry region
```




```
The line command produces 2 entry points.
To add more entry points:
(1) Click between the two entry points
(2) Drag down lower right corner button
(3) A new entry point is added
Add more entry points as needed
```

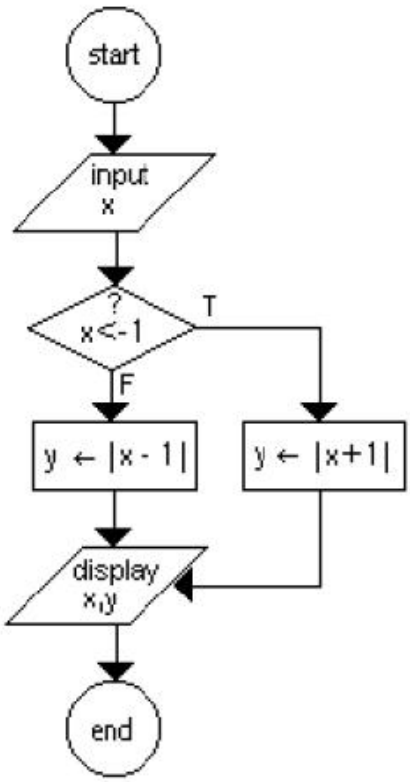
The *line* command can be used to add sequential structures to entry points in other programming instructions as will be illustrated below.

Decision structure

A decision structure provides for an alternative path to the program process flow based on whether a logical statement is true or false. As an example of a decision structure, consider the flowchart for the function listed below:

$$f(x)=\begin{cases}|x+1|, if\ x<-1\\ |x-1|, if\ x\geq -1\end{cases}.$$

The flowchart is shown on the right. The corresponding pseudo-code is shown below:




```
start
input x
if  x < -1  then
   y ← |x+1|
else
   y ← |x-1|
display x,y
end
```

In *SMath Studio* a decision structure is entered using the *if* command. Instructions on entering the *if* command are shown below:


```
The "if" command can be entered:
(1) Using "if" in the "Programming" palette
(2) Typing "if(condition, true, false)"

Using "if" in the "Programming" palette
produces these entry forms in the structure:
```

if ▮
▮
else
▮

To illustrate the use of the *if* command within *SMath Studio*, we enter the function $f(x)$, defined above, as shown here: → → → → → →

$$f(x) := \text{if } x < 1 \;\; |x+1| \;\; \text{else} \;\; |x-1|$$

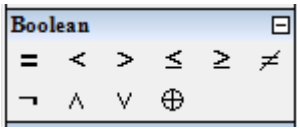
Logical statements and logical operations

Decision structures require a *condition* to trigger a decision on the program flow. Such condition is represented by a logical statement. Within the context of programming numerical calculations, a *logical statement* is a mathematical statement that can be either true or false, e.g., *3>2*, *5<2*, etc. In *SMath Studio* the logical outcomes *true* and *false* are represented by the integer values *1* and *0*, respectively.

The following figure illustrates examples of logical statements. It also includes examples of the four *logical operations*: (1) negation (*not* ¬); (2) conjunction (*and* ∧); (3) disjunction (*or* ∨); and, (4) exclusive or (*xor* ⊗). The figure also includes the truth tables for these four operations. A *truth table* is a table showing the outcome of all possible combinations of *true* and *false* statements.

```
Logical statements (use symbols in the "Boolean" palette):
//inequalities
3>2=1  // true        3<2=0
//Boolean equality & non-equality:
3=2=0  // false       3≠2=1   // true
Less-than-or-equal & Greater-than-or-equal:
 5≥π=1   // true      5≤π=0   // false

 Logical operations:
 // negation (not):          ¬(3>2)=0
 // conjunction (and):       (3>2)∧(4>3)=1
 // dijunction (or):         (3>2)∨(5<2)=1
 // exclusive or (xor):      (3<2)⊕(2>1)=1

Truth tables:
//Negation (not):
    ¬1=0
    ¬0=1

//Conjunction (and):
    1∧1=1
    1∧0=0
    0∧1=0
    0∧0=0

//Disjunction (or):
    1∨1=1
    1∨0=1
    0∨1=1
    0∨0=0

//Exclusive or (xor):
    1⊕1=0
    1⊕0=1
    0⊕1=1
    0⊕0=0
```

The symbols for the comparison operators (=, <, >, ≤ , ≥ , ≠) and logical operations (¬ , ∧ , ∨, ⊗) are available in the *Boolean* palette shown here:

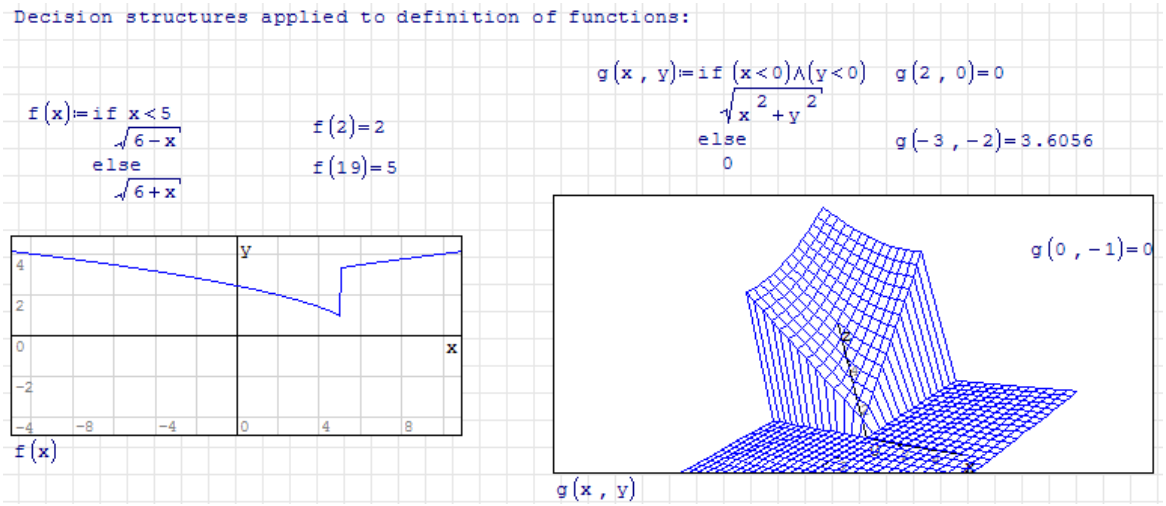


Notice the presence of the *Boolean equal* (bold =) which represents a comparison, rather than an evaluation. The Boolean equal sign can be used to define equations in *SMath Studio*, e.g., in this *solve* command:

$$\text{solve}\left(\sin(x) = x - \frac{\pi}{3}, x\right) = 1.0658$$

Examples of *if* statements used to define functions

The following figure illustrates other examples of *if* statements used in the definitions of functions:



Nested *if* statements

Decision structures, i.e., *if* statements, can be nested as illustrated below:

```
//Nested "if" statements:

s(x , y):=if x>0          s(2 , 2)=2
            if y>0
              √(x+y)      s(2 , −2)=1.4142
            else
              √(2·x+y)    s(−2 , −2)=2.8284·i
          else
            √(3·x+y)
```

Combination of *if* statements with *line* statements

The following two examples illustrate an *if* statement that exchanges the values of two variables *x* and *y* if *x<y,* or changes the signs of both variables otherwise. In the first case, *x<y,* thus, the values get exchanged:

```
case (a): x<y, exchange x and y:
x:=3    y:=4

if x<y
  | temp:=x
  | x:=y
  | y:=temp
else
  | x:=−x
  | y:=−y

x=4   y=3
```


In the second case, $x<y$, thus the *else* clause is activated and both x and y have their signs changed:

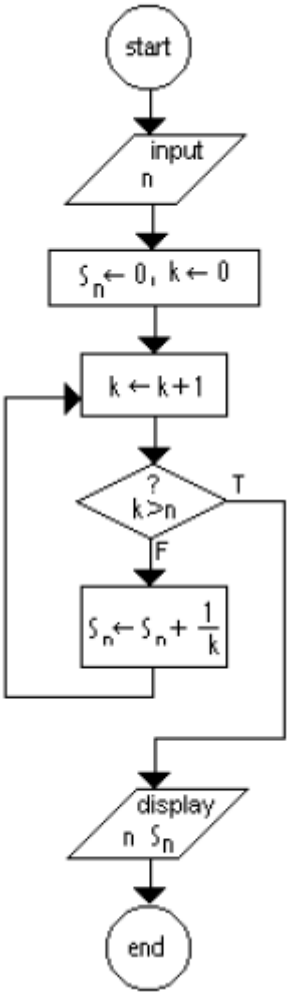
```
case(b): x>y, change signs of x and y:

x:=4   y:=3

if x<y
   |t:=x
   |x:=y
   |y:=t
else
   |x:=-x
   |y:=-y

x=-4 y=-3
```

Loop structures

In a loop structure the process flow is repeated a finite number of times before being send out of the loop. The middle part of the flowchart to the right illustrates a loop structure. The flowchart shown represents the calculation of a sum, namely,

$$S_n = \sum_{k=1}^{n} \frac{1}{k} \quad .$$

The sum S_n is initialized as $S_n \leftarrow 0$, and an index, k, is initialized as $k \leftarrow 0$ before the control is passed on to the loop. The loop starts incrementing k and then it checks if the index k is larger than its maximum value n. The sum is incremented within the loop, $S_n \leftarrow S_n + 1/k$, and the process is repeated until the condition $k>n$ is satisfied. After the condition in the decision block is satisfied (T = true), the control is sent out of the loop to the *display* block.



In terms of programming statements, there are two possible commands in *SMath Studio* to produce a loop: *while* and *for*. The *pseudo-code* for a *while* loop interpreting the flowchart to the right is shown below:

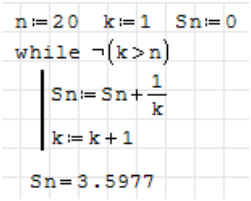
```
start
input n
Sn ← 0
k ← 0
do while ~(k>n)
   k ← k + 1
   Sn ← Sn + 1/k
end loop
display n, Sn
end
```

Since a *while* loop checks the condition at the top of the loop, the condition was converted to ~ ($k > n$), i.e., *not*($k>n$) = $k \leq n$.

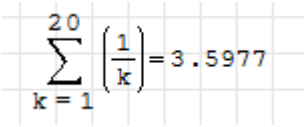
The *while* command in *SMath Studio*

```
//The "while" command can be entered by using:
(1) Using "while" in the "Programming" palette        while ▮
(2) Typing "while(condition,body)"                      ▮

 The expression defining the "condition" must be modified
 within the "while" loop so that an exit can be provided.
```

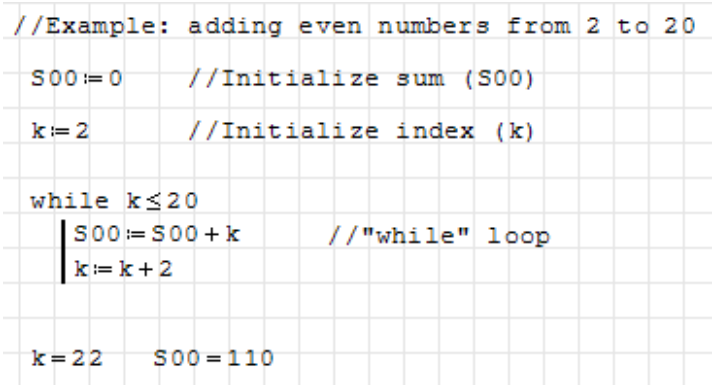
The pseudo-code listed above is translated as follows in *SMath Studio*:

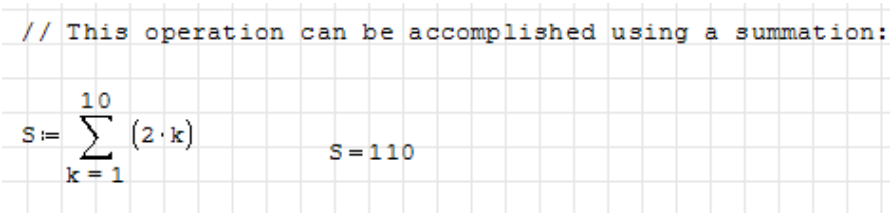


This summation can also be calculated using *SMath Studio's* summation command:



Here is another example of a *while* loop in *SMath Studio*:





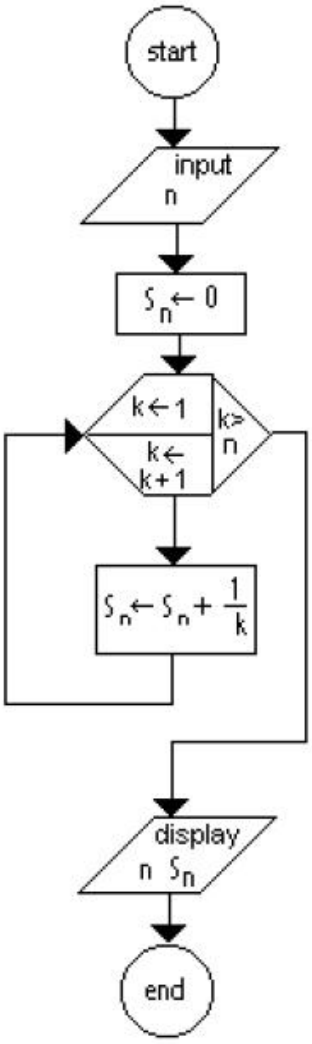
While loops can be nested as illustrated in the left-hand side of the figure shown below. The corresponding double summation is shown on the right-hand side of the figure below.


```
// Nested "while" loops:
S01 := 0    k := 1    j := 1

 while k ≤ 5
    j := 1
    while j ≤ 5
       S01 := S01 + k·j
       j := j + 1
    k := k + 1
S01 = 225
```

$$S05 := \sum_{k=1}^{5} \sum_{j=1}^{5} (k \cdot j) \qquad S05 = 225$$

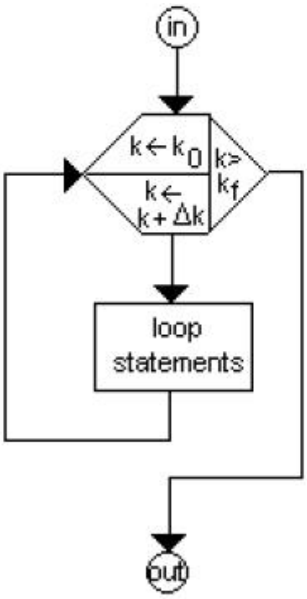
The figure to the right shows an alternative flowchart for calculating the summation:

$$S_n = \sum_{k=1}^{n} \frac{1}{k}$$



The hexagonal symbol in the flowchart shows three elements: (1) the initialization of the index, $k \leftarrow 1$; (2) the index being incremented, $k \leftarrow k + 1$; and, (3) the condition checked to exit the loop, $k > n$. This hexagonal symbol represents the *for* command for this summation.

A more general symbol for a *for* command is shown below. This symbol shows a *for* loop with index k, starting at value k_0, and ending at value k_f, with increment Δk:



The index, therefore, takes values $k = k_0, k_0+\Delta k, k_0+2\Delta k, \ldots, k_{end}$, such that $k_{end} \le k_f$, within one Δk.

The *for* and *range* commands in *SMath Studio*

```
//The "for" command can be entered by using:
(1) Using "for" in the "Programming" palette
(2) Typing "for(index,range,body)"
```

for ▮∈▮
▮

The *for* command in *SMath Studio* uses a *range* of values to indicate the values that the index *k* takes to complete the loop.

```
Using ranges:
-------------
Ranges are needed for the "for" statement.
A range represents a vector whose elements follow a certain pattern.
Ranges can be entered as:
(1) range(start,end)                  becomes:  start..end (increment = 1)
(2) range(start,end,start+increment)  becomes:  start, start+increment..end

A range represents a column vector.  Below, we use transposed vectors to
show ranges as row vectors:
```

```
//Examples of ranges with increment of 1:

//Type "range(2,5)" to produce:
  r1:=2..5          r1^T=(2 3 4 5)

//Type "range(10,18)" to produce:
  r2:=10..18        r2^T=(10 11 12 13 14 15 16 17 18)
```

```
// Examples of ranges with positive increment:

// Type "range(2,18,4)" to produce:
 r3:=2,4..18          r3^T=(2 4 6 8 10 12 14 16 18)

// Type "range(20,300,80)" to produce:
 r4:=20,80..300       r4^T=(20 80 140 200 260)
```

```
// Examples of ranges with negative increment:

// Type "range(100,20,80)" to produce:
 r5:=100,80..20          r5^T=(100 80 60 40 20)

// Type "range(5,1,4)" to produce:
 r6:=5,4..1              r6^T=(5 4 3 2 1)
```


Here is an example of the *for* command in *SMath Studio* using the range *1..10*:

```
// Example sum of even numbers using "for":

S03≔0            // initialize a sum (S03)

for k∈1..10      //"for" loop, enter the range as:
  S03≔S03+2·k    //'range(1,10)'

S03=110          // final value of S03
```

The double summation programmed earlier using *while* loops, can be programmed also using *for* loops:

```
// Nested "for" loops:

S04≔0

for j∈1..5
  for k∈1..5
    S04≔S04+k·j

S04=225
```

A programming example using sequential, decision, and loop structures

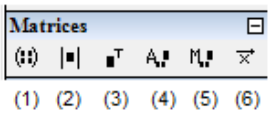
This example illustrates a program in *SMath Studio* that uses all three programming structures. This is the classic "bubble" sort algorithm in which, given a vector of values, the smallest ("lighter") value bubbles up to the top. The program shown uses a row vector *rS*, and refers to its elements using sub-indices, e.g., S_{1k}, etc. The example shows how to enter sub-indices. The output from the program is the sorted vector *rS*.


```
// Example using "for" and "if": Classical bubble sort
rS:=(5.4 1.2 3.5 10.2 -2.5 4.1)    // Given a vector "rS"
nS:=length(rS)      nS=6           // First, find length of vector
                                   // Double loop that re-arranges
    for k∈1..nS-1                  // order of elements in vector rS
      for j∈k+1..nS
        if rS1,k>rS1,j             // To enter sub-indices use,
          temp:=rS1,j              // for example, rS[1,k
          rS1,j:=rS1,k
          rS1,k:=temp
        else
          0
rS=(-2.5 1.2 3.5 4.1 5.4 10.2)     // Result: vector sorted

// This sorting can be accomplished using function "sort":

rT:=(5.4 1.2 3.5 10.2 -2.5 4.1)
```

$$\text{sort}\left(rT^{T}\right)=\begin{pmatrix}-2.5\\1.2\\3.5\\4.1\\5.4\\10.2\end{pmatrix}$$

Many programming applications, such as the one shown above, use vectors and matrices. Luckily, *SMath Studio* already includes a good number of functions that apply to matrices, e.g,:

- Creating matrices: *augment, diag, identity, mat, matrix, stack*
- Extracting rows, columns, elements: *col, el, row, submatrix, vminor, minor*
- Characterizing matrices: *cols, det, length, max, min, norm1, norme, normi, rank, rows, tr*
- Sorting functions: *csort, reverse, rsort, sort*
- Matrix operations: *alg, invert, transpose*



Some of these functions are also available in the *Matrices* palette:
(1) Matrix 3x3 (Ctrl+M) (*mat*)
(2) Determinant (*det*)
(3) Matrix transpose (Ctrl+1) (*transpose*)
(4) Algebraic addition to matrix (*alg*)
(5) Minor (*minor*)
(6) Cross product

Examples of matrix operations were presented earlier.

Steps in programming
The following are recommended steps to produce efficient programs:

(1) Clearly define problem being solved
(2) Define inputs and outputs of the program
(3) Design the algorithm using flowcharts or pseudo-code
(4) Program algorithm in a programming language
(e.g., *SMath Studio* programming commands)
(5) Test code with a set of known values

Errors in programming
Typically there are three main types of errors in developing a program:

(1) *Syntax errors*: when the command doesn't follow the programming language syntax. These are easy to detect as the program itself will let you know of the syntax violations.

(2) *Run-time errors*: errors due to mathematical inconsistencies, e.g., division by zero. These may be detected by the program also.

(3) *Logical errors*: these are errors in the algorithm itself. These are more difficult to detect, thus the need to test your programs with known values. Check every step of your algorithm to make sure that it is doing what you intend to do.

Notes: Appendices 3 through 6 show examples of programs used to solve selected problems in numerical methods, namely:

Appendix 3 - The Newton-Raphson for solving equations
Appendix 4 - The 4th-order Runge-Kutta method for a single ODE
Appendix 5 - The 4th-order Runge-Kutta method for a system of ODEs
Appendix 6 - The 4th-order Runge-Kutta method for a 2nd order ODE
Appendix 7 – Finding eigenvalues and eigenvectors of a matrix

Input/Output of data and file manipulation in *SMath Studio*
SMath Studio provides functions *wfile*, *rfile,* and *dfile* for output into a file, input from a file, and deleting an existing file. To illustrate the use of these functions we could start by writing some data to a file and exploring the contents of the file. Before we do that, however, I want to address an issue related to the location of the *SMath Studio* installation folder.

The location of the SMath Studio installation folder
When I first attempted to use I/O functions` after installing *SMath Studio* in my *Windows Vista Ultimate 64-bit* machine I kept getting a message indicating that access to the file location was not permitted. This was so because *SMath Studio*, by default, sends output

to and reads input from the folder */*SMath Studio Installation Folder*/user/*. In my computer, the **SMath Studio Installation Folder** corresponds to the directory *C:\Program Files(x86)\SMath\SMath Studio* and it requires *Administrator* access to modify its contents. To be able to write data from and read data into *SMath Studio*, therefore, I copied the folder *C:\Program Files(x86)\SMath* to a different location. In my case the location chosen was:

C:\Users\Gilberto E. Urroz\Documents\NUMERICAL_APPLICATIONS\SMath

In your computer it could be any location where you don't need *Administrator* rights to modify content. I then proceeded to create the folder *user*, where the data files would reside. Thus, the full address of this new folder, in my case, is:

C:\Users\Gilberto E. Urroz\Documents\NUMERICAL_APPLICATIONS\SMath\SMath Studio\user

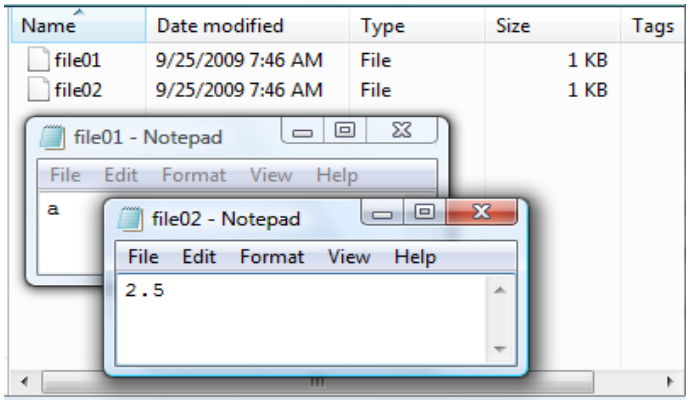
To facilitate access to the program, at this point, I created a new shortcut to *SMath Studio* in my desktop. Now we proceed to illustrate the use of the *wfile* command.

Writing data to a file

You can use function *wfile* to write a single item of data into a file. Consider the following examples[3]:

```
a := 2.5
wfile(a , file01) = 1     Stores the letter 'a' in file "file01"
wfile(2.5 , file02) = 1     Stores the value "2.5" in file "file02"
```

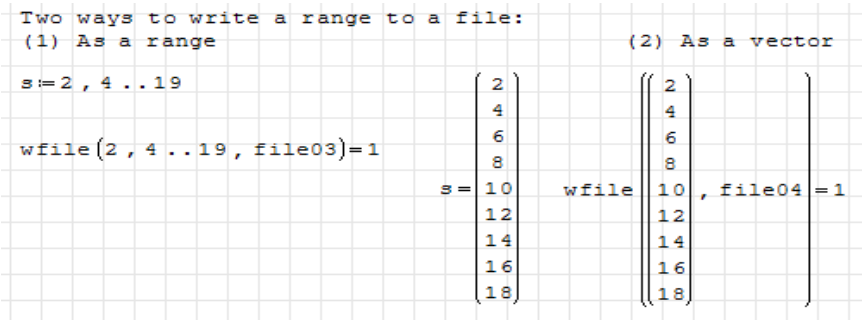
You can open the files using, for example, *Notepad* in a *Windows* system to check the contents of the file:



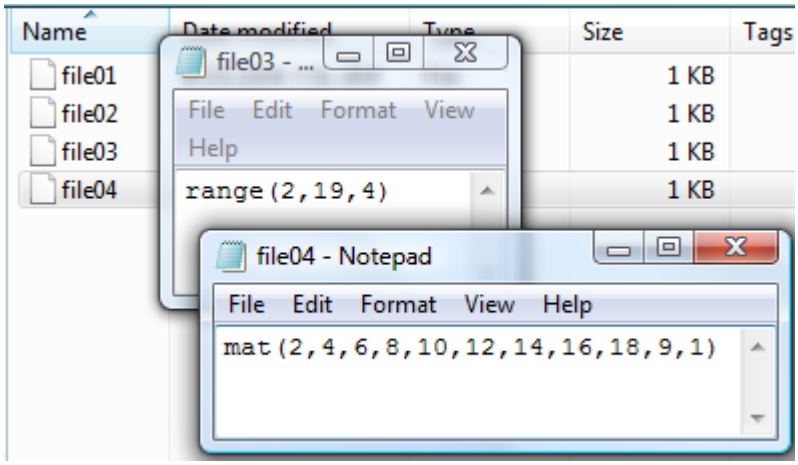
[3]I recommend you use only lowercase letters in the filenames used to store data. I had difficulties getting *SMath Studio* to read data out of files with uppercase letters in their names.

Thus, you'll need to use the numerical result that you want to write out to a file as the first argument of *wfile*. Using the reference to a variable will only write the variable name. The second argument to *wfile* is the name of the file, entered without quotes. The file name can only use letters and numbers. No other characters are allowed in the name of a file.

You can store a range or the resulting vector to a file, e.g.,



The contents of the corresponding files are shown below:



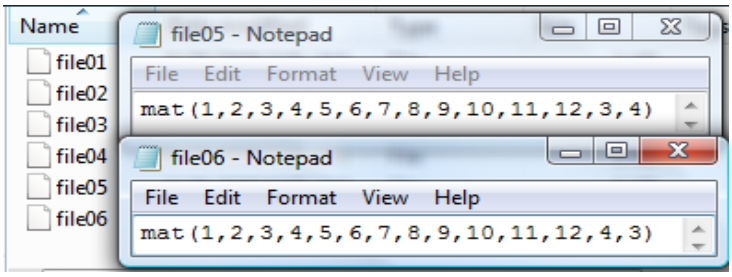
Regarding the notation *mat*(*2,4,6,10,12,14,16,18,9,1*) written in file *file04*: this specification contains the 9 elements of the vector shown as *s*, above, plus the number of rows (*9*) and columns (*1*) defining the vector. Consider, for example, the writing of matrices to files. In the following two examples, the two matrices have 12 elements, each, but arranged differently. Matrix A has 3 rows and 4 columns, while matrix B has 3 rows and four columns. They get stored into files *file05* and *file06*, respectively, as shown below:

Writing a matrix to a file:

$$A := \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \qquad wfile\left(\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}, file05\right) = 1$$

$$B := \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} \qquad wfile\left(\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}, file06\right) = 1$$

Next, we show the contents of the two files. Notice the specification of rows and columns as the last two values in the list of data:



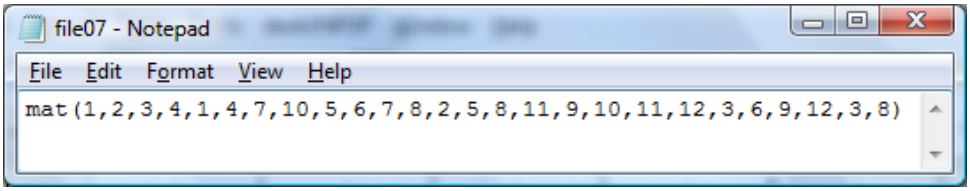
Here is another example of using function *wfile*:

Another example of writing a matrix:

$$C := augment\left(A, B^{T}\right) \qquad C = \begin{pmatrix} 1 & 2 & 3 & 4 & 1 & 4 & 7 & 10 \\ 5 & 6 & 7 & 8 & 2 & 5 & 8 & 11 \\ 9 & 10 & 11 & 12 & 3 & 6 & 9 & 12 \end{pmatrix}$$

$$wfile\left(\begin{pmatrix} 1 & 2 & 3 & 4 & 1 & 4 & 7 & 10 \\ 5 & 6 & 7 & 8 & 2 & 5 & 8 & 11 \\ 9 & 10 & 11 & 12 & 3 & 6 & 9 & 12 \end{pmatrix}, file07\right) = 1$$

The contents of the file are:



Reading data from a file

To read data from a file use function *rfile*. The following examples use the files we created above (i.e., *file01, file02, ..., file07*) to illustrate the use of function *rfile*. The result from reading data can be simply shown by using an equal sign(=), or assigned to a variable by using the assignment operator (:=). Try the following examples:


```
Reading files: (1) Showing result only:
rfile(file02)=2.5      Use = after command
rfile(file02)→5/2      Use Crtl+. (symbolic)
Reading files: (2) Assigning to a variable:
a:=rfile(file02)       a=2.5      a→5/2
```

Additional examples of reading from a file are shown below:

$$A:=\text{rfile}(\text{file06}) \qquad A=\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\\10&11&12\end{pmatrix} \qquad B:=\text{rfile}(\text{file05}) \qquad B=\begin{pmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\end{pmatrix}$$

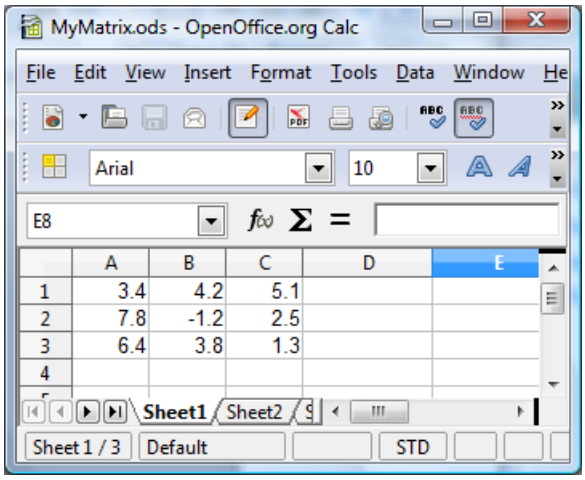
$$\text{rfile}(\text{file07})=\begin{pmatrix}1&2&3&4&1&4&7&10\\5&6&7&8&2&5&8&11\\9&10&11&12&3&6&9&12\end{pmatrix} \qquad \text{rfile}(\text{file03})=\begin{pmatrix}2\\4\\6\\8\\10\\12\\14\\16\\18\end{pmatrix}$$

If you attempt to read from a non-existing file you'll get an error message:

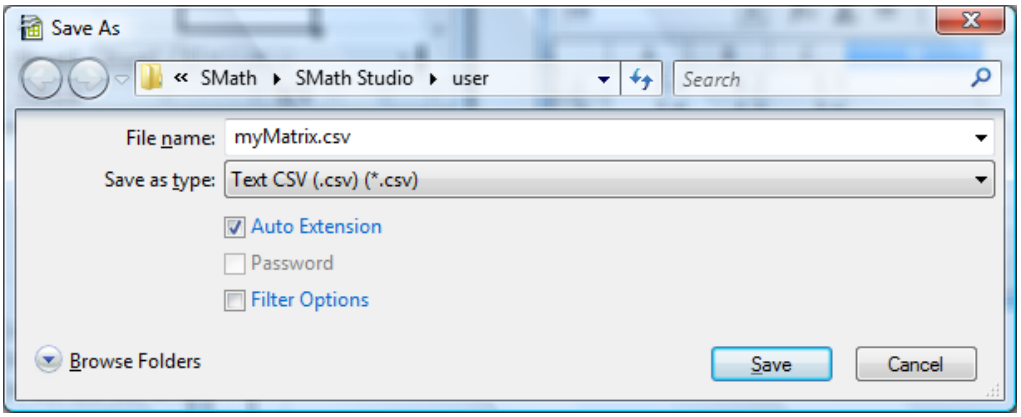
```
rfile(file08)= ∎
Could not find file 'C:\Users\Gilberto
E. Urroz\Documents\NUMERICAL_APPLICATIONS\SMath\SMath
Studio\user\file08'.
```

Copying a matrix from a spreadsheet

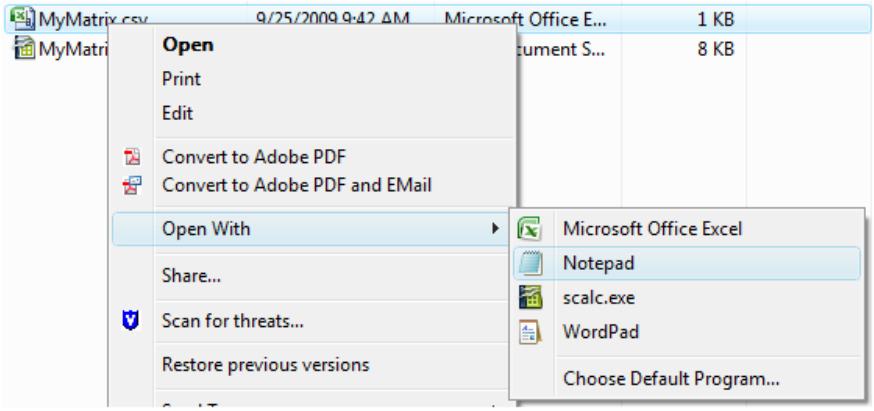
Suppose you have the following 3x3 matrix (only the numbers, no labels, etc.) in an OpenOffice.org Calc spreadsheet named *myMatrix.ods:*



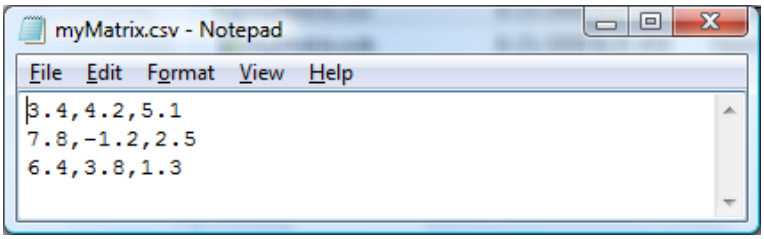
Save the file in *csv* (*c*omma-*s*eparated *v*alues) format into your *user* folder in your *SMath Studio* installation:



Then, close the newly saved file *myMatrix.csv,* and open it using *Notepad* (right-click on the icon, and select *Open with* ...):

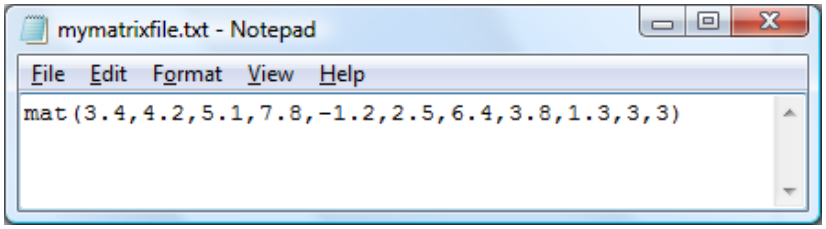


to see the contents of the file:



Copy the file contents into another text file, let's call it *mymatrixfile*[4], and edit it to read:

[4]A few notes: (1) Use only lowercase letters in the name of the file. I couldn't get *SMath Studio* to recognize filenames that include uppercase letters. (2) Make sure the filename to be read has no extension, therefore, the file should be named, in this case, *mymatrixfile*, and not *mymatrixfile.txt* or *mymatrixfile.dat*, etc. (3) Do not try to simply use the same *.csv* file without an extension as your input file to *SMath Studio.* It won't work. You need to create a new file.



Thus, the editing consisted in:

1. adding "mat(" at the beginning of the line
2. placing all the data into a single line
3. replacing the missing commas (between 5.1 and 7.8, and between 2.5 and 6.4)
4. adding "3,3)" at the end of the line.

Save this file and then try the command:

$$\text{rfile}(\text{mymatrixfile}) = \begin{pmatrix} 3.4 & 4.2 & 5.1 \\ 7.8 & -1.2 & 2.5 \\ 6.4 & 3.8 & 1.3 \end{pmatrix}$$

Of course, to get this simple 3x3 matrix into *SMath Studio* it will be easier to type it in into the worksheet. The procedure illustrated above would be more practical for very large spreadsheet data sets.

<u>Writing and reading symbolic expressions</u>
The following examples illustrate the writing of expressions to a file, and the reading of the same expressions from files.

Writing and reading symbolic expressions:

Example 1:

Writing to a file:

$$\text{wfile}\left(\sqrt{2}+\tan\left(\frac{\pi}{6}\right), \text{file10}\right)=1$$

Two ways of reading from the file:

$$\text{rfile}(\text{file10})=1.9916$$

$$\text{rfile}(\text{file10})\rightarrow\frac{\sqrt{2}\cdot\sqrt{3}+1}{\sqrt{3}}$$

Example 2:

Writing to a file:

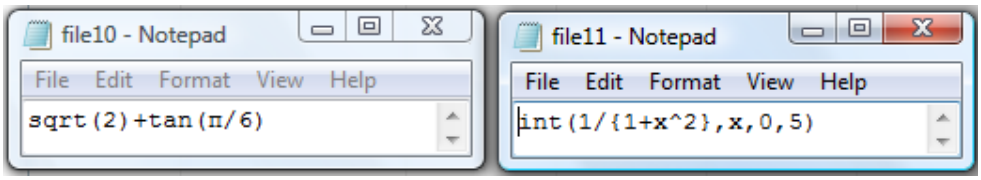
$$\text{wfile}\left(\int_0^5 \frac{1}{1+x^2}\,dx, \text{file11}\right)=1$$

Two ways of reading from the file:

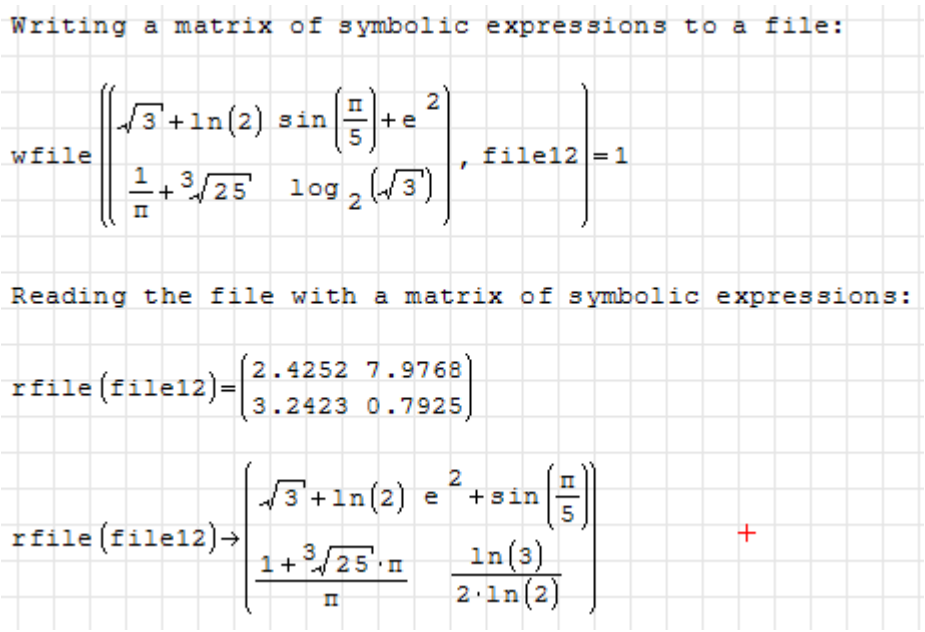
$$\text{rfile}(\text{file11})=1.3734$$

$$\text{rfile}(\text{file11})\rightarrow\frac{68670038346567}{50000000000000}$$

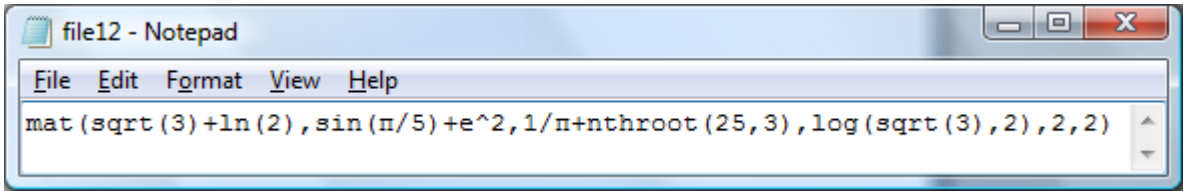
The contents of the files are shown below:



The following example shows writing and reading of a matrix of symbolic expressions:

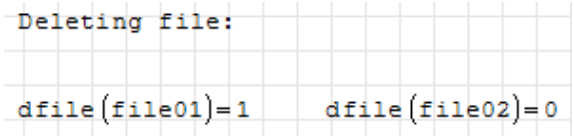


The content of the file is shown below:

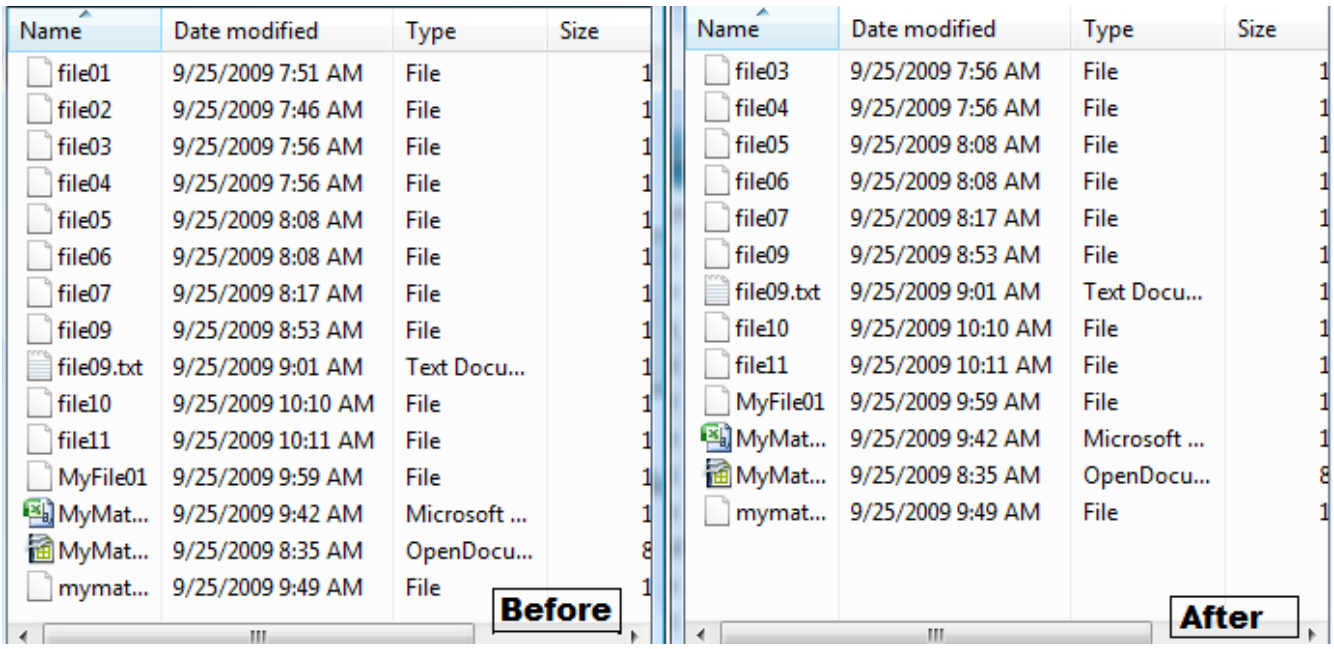


Deleting files

Use function *dfile*(*filename*) to delete files in the folder */*SMath Studio Installation Folder*/user*. Examples:



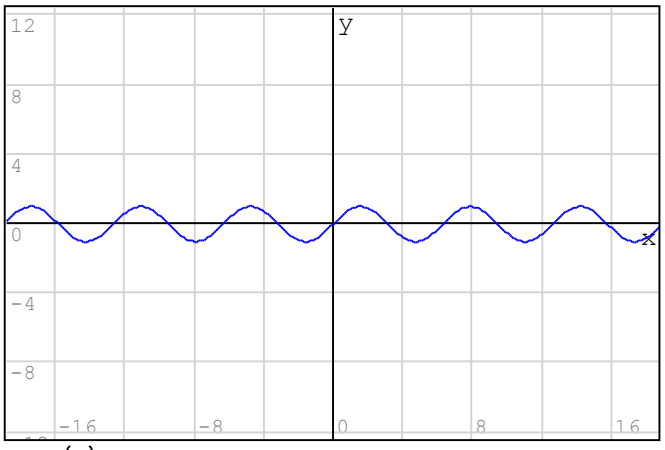
The directory */*SMath Studio Installation Folder*/user* is shown below before and after the execution of the *dfile* commands shown above:



Application of functions *wfile, rfile,* and *dfile* could be useful in programming if there is a need to store data temporarily in a file.

APPENDIX 1 - Examples of two- and three-dimensional graphics in Smath Studio

Plotting a single function of x:



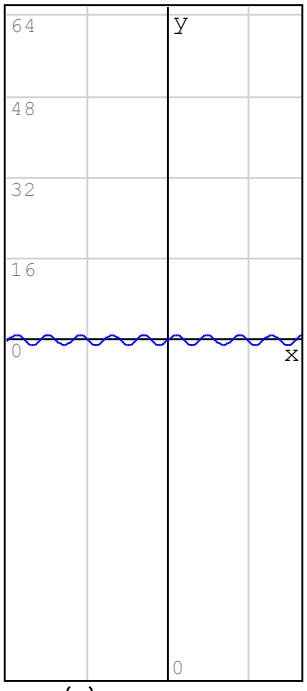
sin(x)

1 - Click on point in your worksheet where upper left corner of graph will go
2 - Click on the "2D" option in the "Functions" palette or use the "Insert > Graph > 2D" menu option
3 - Type the function name in the placeholder below the graph

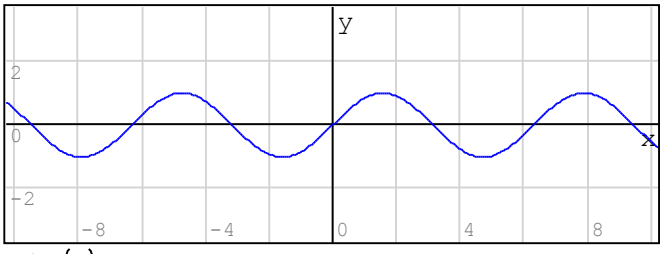
In this example we plot the function: f(x) = sin(x)

Changing the size of the graph window:

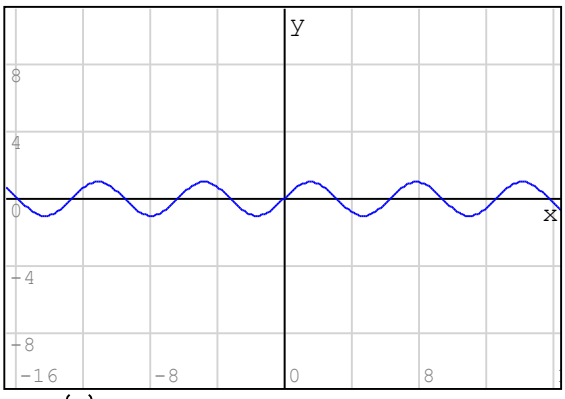
Click on the graph window, then drag one of the three black handlers in the graph window to adjust its size



sin(x)

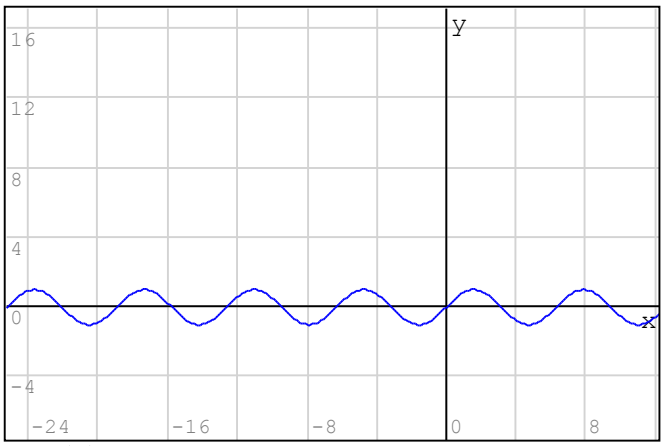


sin(x)

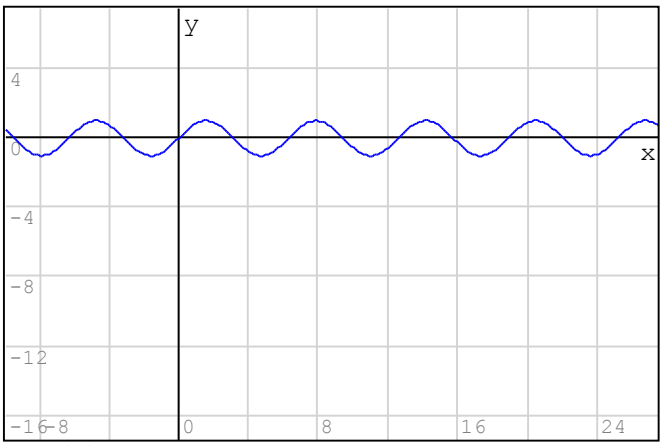


sin(x)

Moving the axes about the graph window:



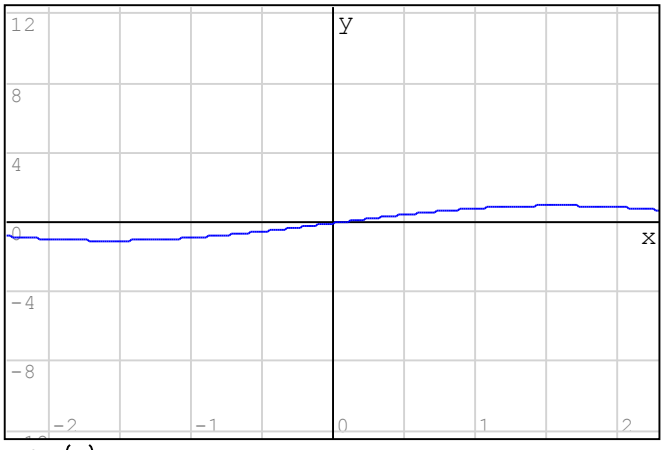
sin(x)



sin(x)

1 - Click on the "Move" option in the "Plot" palette
2 - Click on the graph window and drag the mouse in the direction where you want to move the axes.

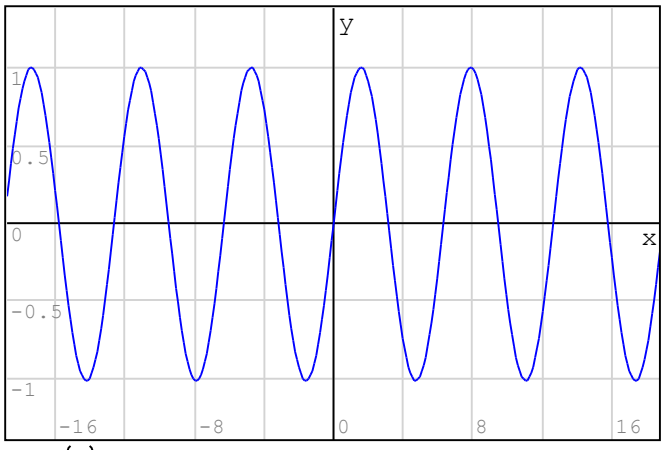
Scaling (zooming) the graph:



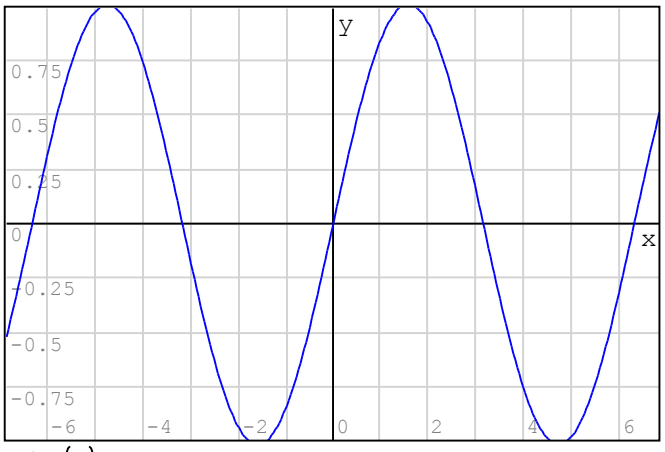
sin(x)

To zoom the x-axis only: ---->
1 - Click on "Scale" in the "Plot" palette
2 - Click on the graph window
3 - Hold down the "Control" key
4 - Roll the mouse wheel up or down

<--- To zoom the x-axis only:
1 - Click on "Scale" in the "Plot" palette
2 - Click on the graph window
3 - Hold down the "Shift" key
4 - Roll the mouse wheel up or down



sin(x)

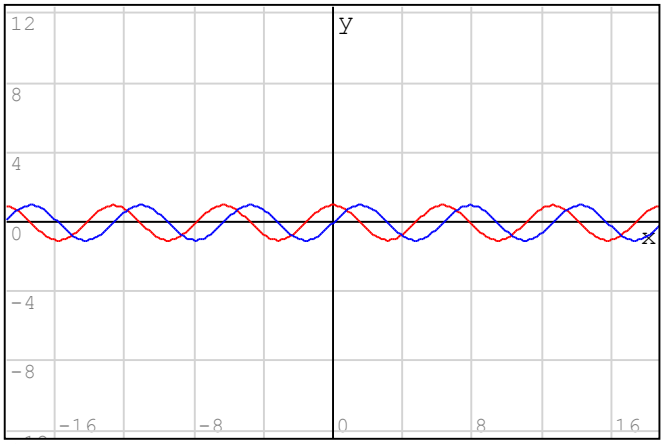


sin(x)

<--- You can zoom both axes by zooming one axis at a time. In this case, I zoomed the x axis first, and then the y axis.

Note: Use the "Refresh" option in the "Plot" menu to recover the original version of any plot.

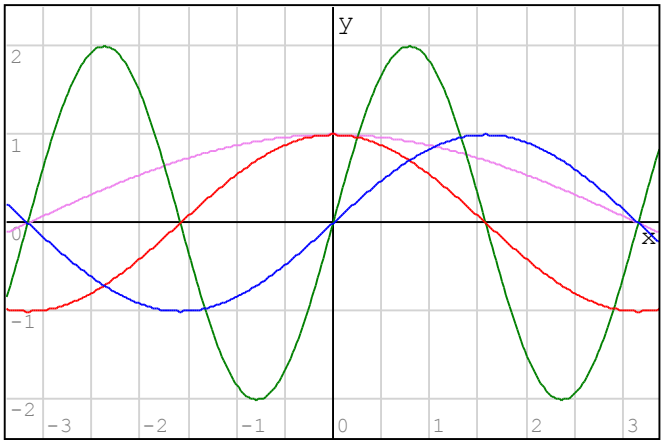
Plotting various functions simultaneously:



$$\begin{cases} \sin(x) \\ \cos(x) \end{cases}$$

1 - create a 2D graph
2 - click on the function placeholder (lower left corner) to select it
3 - click on the "Equation System" option in the "Functions" palette to produce a minimum of two function entries
4 - Type the two functions to be plotted

Note: the first function listed is plotted using a blue line, the second one uses a red line



$$\begin{cases} \sin(x) \\ \cos(x) \\ 2\cdot\sin(2\cdot x) \\ \cos\left(\frac{x}{2}\right) \end{cases}$$

1 - To add more than two functions to plot, use the "Equation System" option in the "Functions" palette as above
2 - Click on the "Equation System" cell, and drag down corner of cell adding as many placeholders as you want.
3 - Type the functions to be plotted, one at a time, clicking outside of the graph after you enter each one of them (this will allow you to see each function plota as they are added to the graph)

Notes:
1 - As you add functions to plot, SMath Studio uses the following colors for the plots:

1 - blue 2 - red
3 - black 4 - magenta, etc.

2 - In this example the graph has been zoomed using the approach described above

Plotting a function using vectors:

$$x := -\pi, -\pi + \frac{\pi}{20} .. \pi$$

$$n := \text{length}(x)$$
$$n = 41$$

for $k \in 1 .. n$

$$y_k := \sin\left(x_k\right)^2 + \sin\left(2\cdot x_k\right)$$

Vectors of x and y data are created using ranges, example:

Create x vector as follows
Type: x : range - p cntl-G , p cntl-G ,
- p cntl-G + p cntl-G / 20

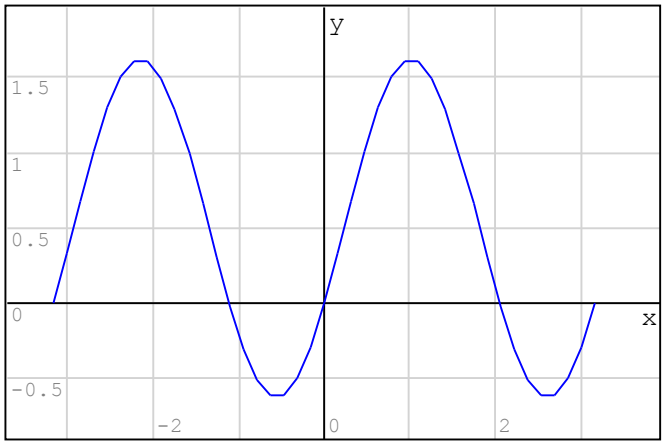
Calculate the length of vector = n

Fill out y vector using a for loop. Click "for" in the "Programming" palette, then use:
range 1 , n

Use sub-indices, e.g., y [k ... etc.

$M := \text{augment}(x, y)$

Form augmented matrix M with vectors x and y, place M in graph as a function name:

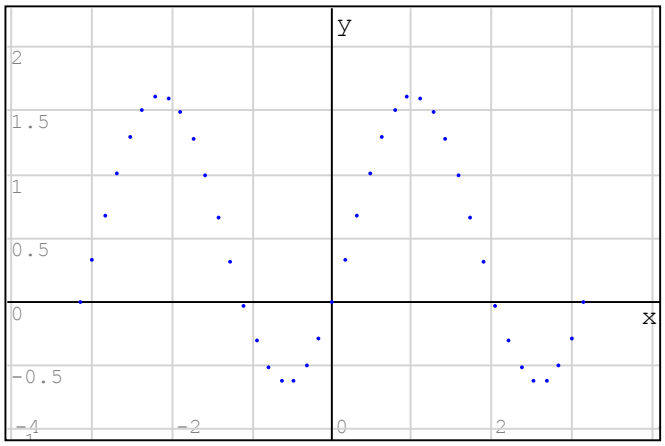


M

< --- The graph was zoomed in and the axes moved by using the following procedures:

1 - To zoom x-axis only: click on "Scale" in the "Plot" palette, hold the "Control" key, and use the mouse wheel
2 - To zoom y-axis only: click on "Scale" in the "Plot" paletted, hold the "Shift" key and use the mouse wheel
3 - To move axes, drag mouse across graph window

Using points or lines for a plot:



M

Using the sparse data in matrix M we reproduce the graph above, but then we selected the "Graph by points" option in the "Plot" palette to produce the graph shown to the left.

You can click the option "Graph by lines" option in the "Plot" palette to return to the default graph format of continuous lines.

Matrices can be used for plotting parametric plots:

$t := -\pi, -\pi + \frac{\pi}{50} .. \pi$

Define the vector of the parameter t

$n := \text{length}(t)$

Determine length of vector t = n

for $k \in 1 .. n$

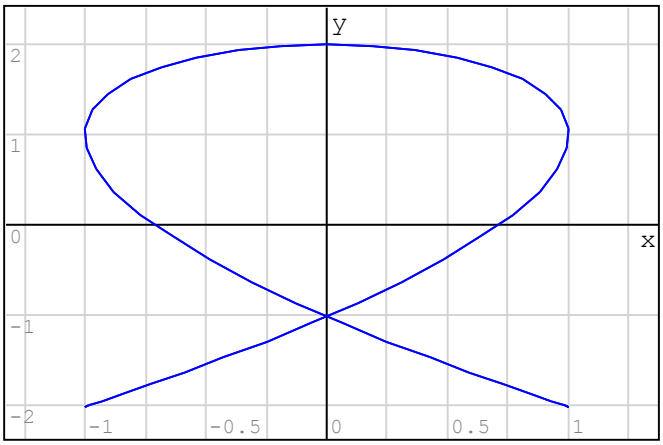
$x_k := \sin(3 \cdot t_k)$

$y_k := 2 \cdot \cos(2 \cdot t_k)$

Calculate vectors of x = x(t) and y = y(t)

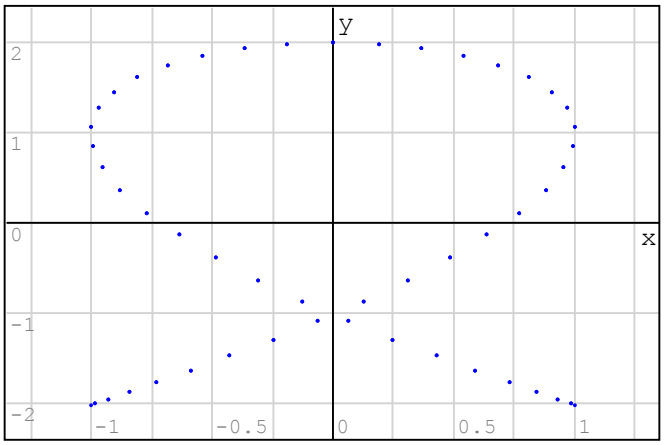
$M := \text{augment}(x, y)$

Produce matrix of (x,y) and plot it



M

1 - Using "Plot by Lines" option
in the "Plot" palette



M

2 - Using "Plot by Lines" option
in the "Plot" palette

Polar plots can be produced using vectors and matrices:

$\theta := 0, \frac{\pi}{50} .. 2\cdot\pi$ Generate vector of θ between 0 and 2π

$n := \text{length}(\theta)$ Determine lenght of vector θ

for $k \in 1..n$ Generate values of r = f(θ)

$r_k := 2\cdot(1 + 2\cdot\sin(\theta_k))$

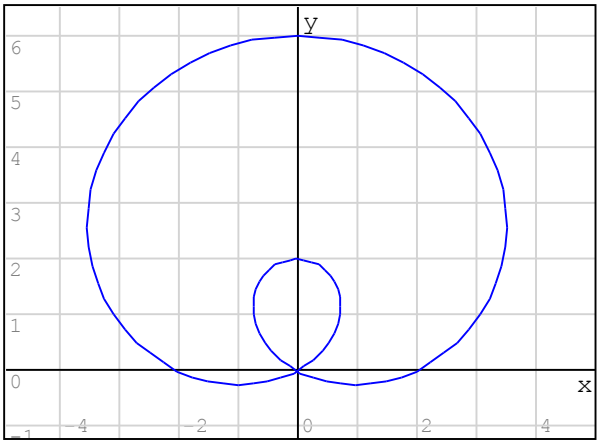
for $k \in 1..n$ Generate coordinates:

$xx_k := r_k\cdot\cos(\theta_k)$

$yy_k := r_k\cdot\sin(\theta_k)$

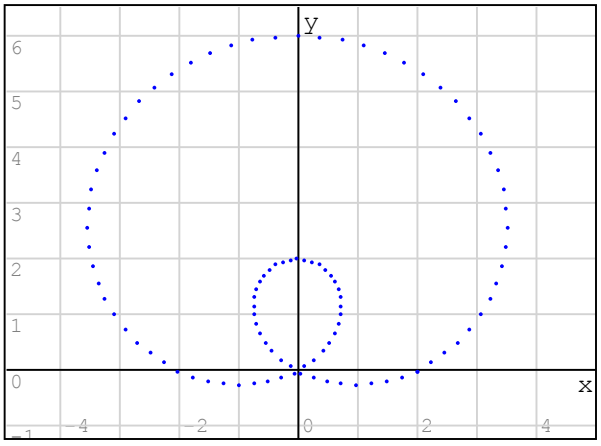
x = r cos(θ)
y = r sin(θ)

$P := \text{augment}(xx, yy)$ Produce matrix of (x,y) and plot it



P

1 - Using "Plot by Lines" option
in the "Plot" palette



P

2 - Using "Plot by Lines" option
in the "Plot" palette

Using graphs in solving equations:

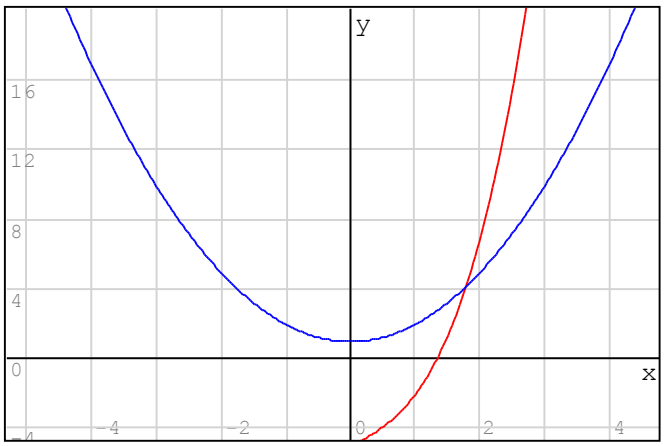
In this example we seek the solution(s) for the equation:

$$x^2 + 1 = x^3 + 2\cdot x - 5$$

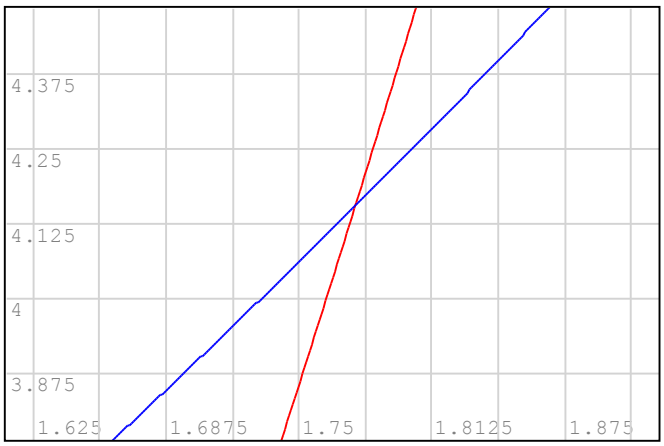
A solution can be found by determining the intersection of the functions:

$f(x) := x^2 + 1$ $\quad$ $g(x) := x^3 + 2\cdot x - 5$

Using graphics and zooming the intersection we estimate the solution to be close to x = 1.80



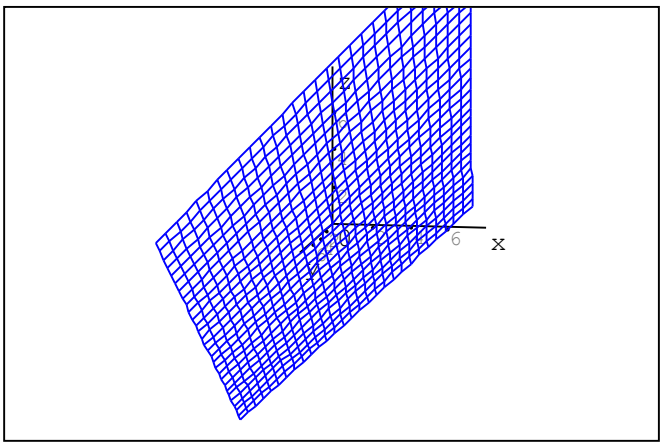
$\begin{cases} f(x) \\ g(x) \end{cases}$



$\begin{cases} f(x) \\ g(x) \end{cases}$

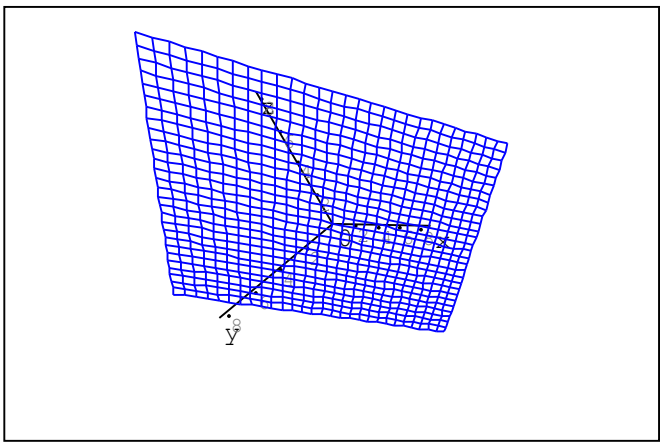
The exact solution can be found using: $\text{solve}\left(x^2 + 1 = x^3 + 2\cdot x - 5,\ x\right) = 1.776$

Three-dimensional graphs - surfaces:



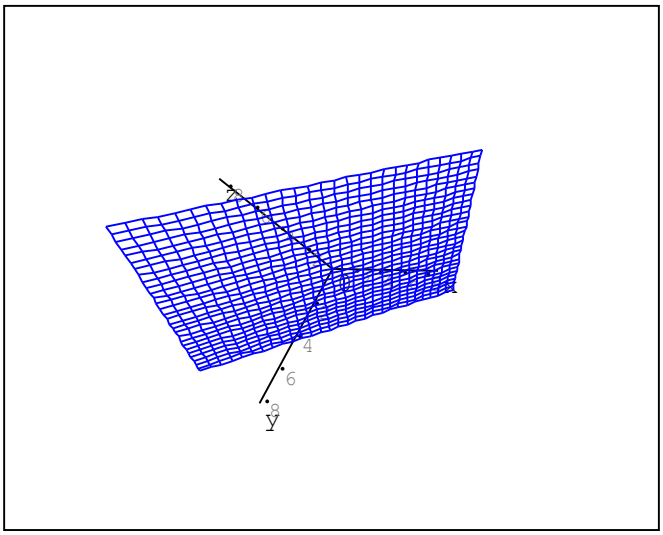
x + y

Use the option "3D" in the "Functions" palette, and enter the function f(x,y) in the place-holder. The result is a 3D surface, in this case, a plane. The original plot is shown above.



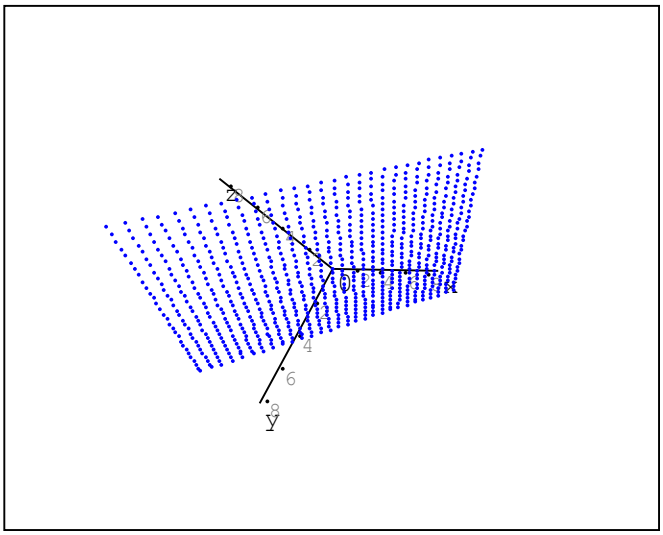
x + y

Use the "Rotate" option in the "Plot" palette to change the surface view.



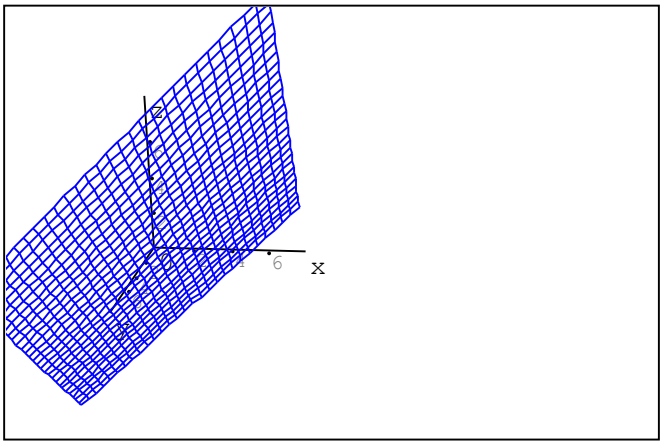
x + y

This figure uses the option "Graph by Lines" in the "Plot" palette (default).



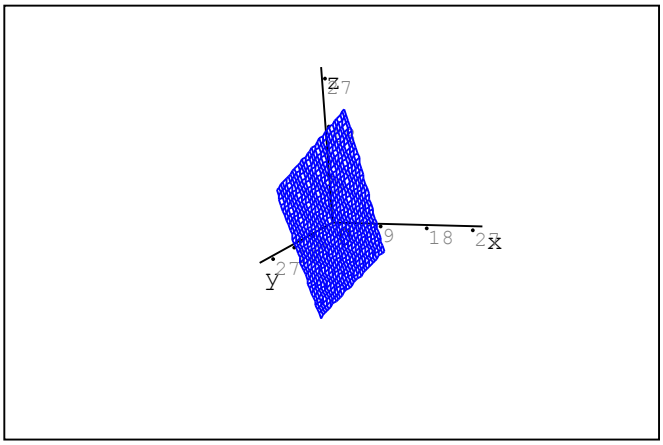
x + y

This figure uses the option "Graph by Points" in the "Plot" palette.



x + y

Use the "Move" option in the "Plot" palette to move the location of the origin in the graphics window.

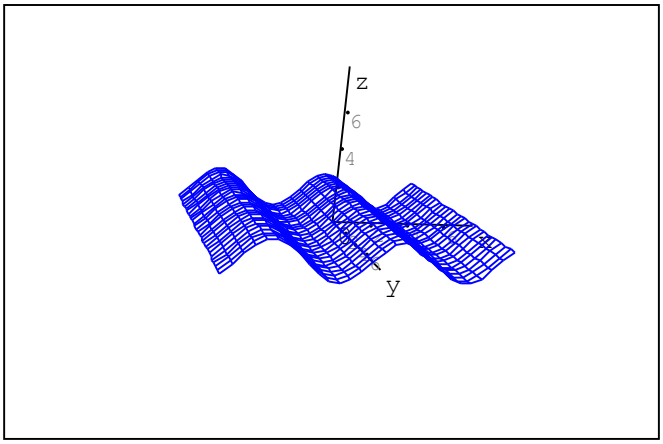


x + 2·y

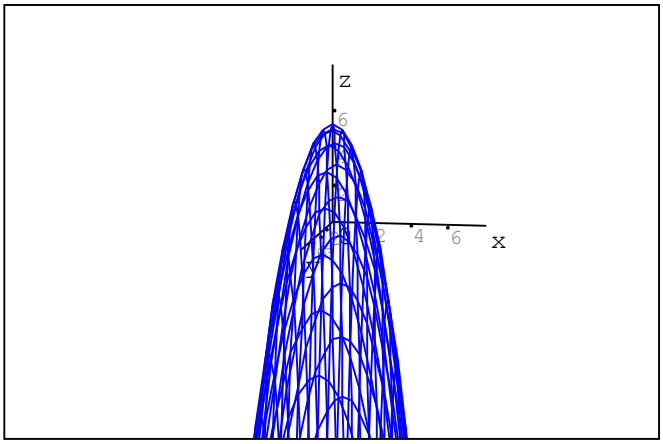
Use the "Scale" option in the "Plot" palette, click on the graph, and drag the mouse over it to zoom in or out.

Note: Use the "Refresh" option in the "Plot" menu to recover the original version of any plot.

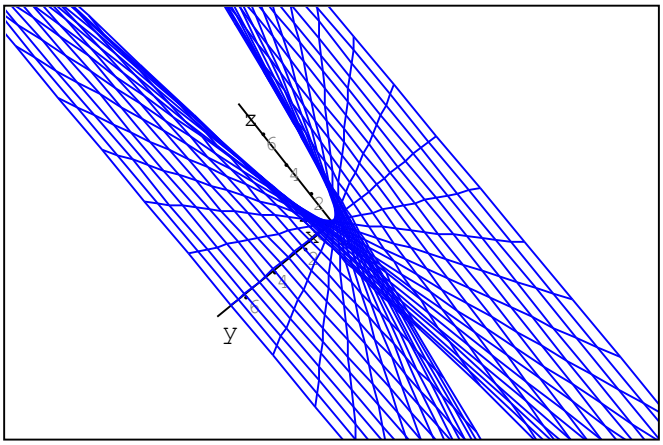
The following examples use more complex surfaces:



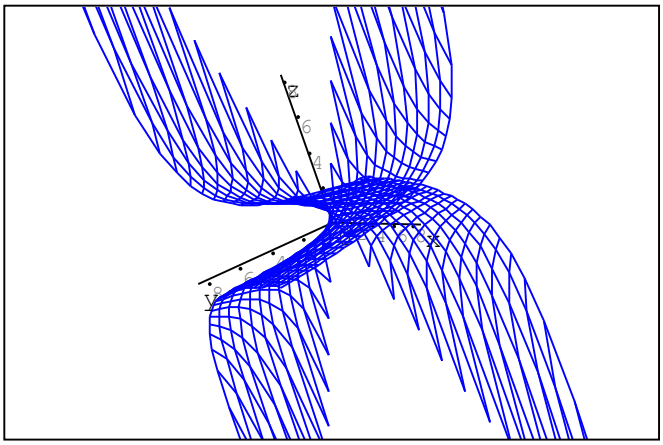
$\sin(x)$



$5-\left(x^2+y^2\right)$



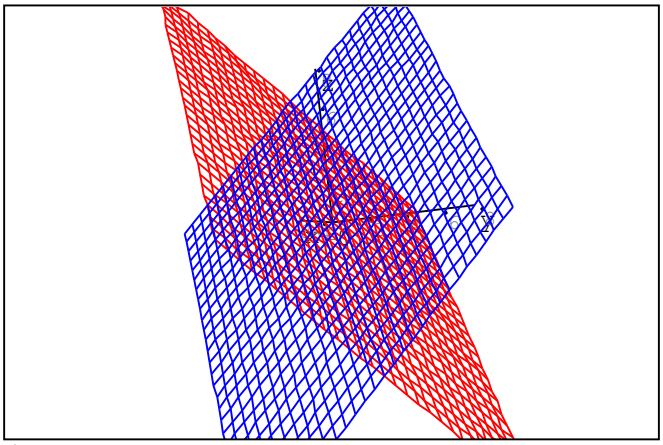
$x\cdot y$



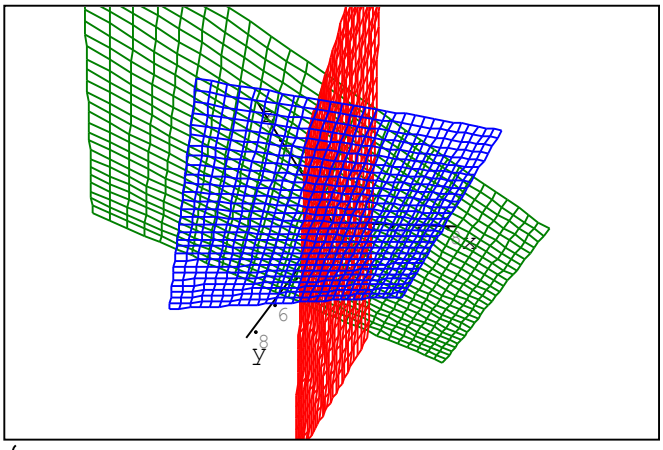
$\frac{x-y}{x+y}$

The type of 3D graphs of surfaces produced by SMath Studio are referred to as wireframe plots.

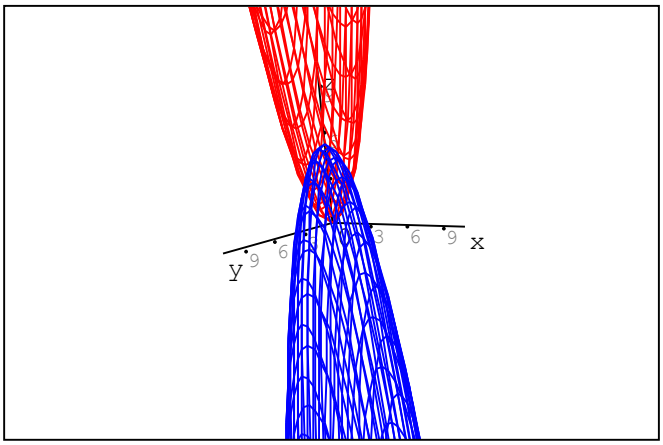
The following examples show more than one surface plot together in 3D:



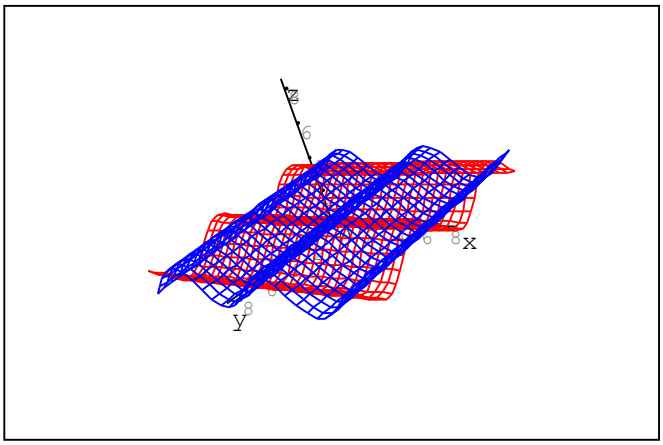
$\begin{cases} x+y \\ x-y \end{cases}$



$\begin{cases} x+y \\ x-y \\ x+2\cdot y \end{cases}$



$$\begin{cases} 5-\left(x^{2}+y^{2}\right) \\ x^{2}+y^{2} \end{cases}$$



$$\begin{cases} \sin(x) \\ \sin(y) \end{cases}$$

Drawing a space curve in 3D:

A space curve is defined by a matrix of three columns corresponding to coordinates x, y, and z of the curve.

$t := 0, 0.1 .. 10$

Create a vector t with values of the parameter that will produce x = x(t), y = y(t), and z = z(t).

$n := \text{length}(t)$

Determine the length of vector t

$n = 101$

for $k \in 1 .. n$

$x_k := \sin(t_k)$

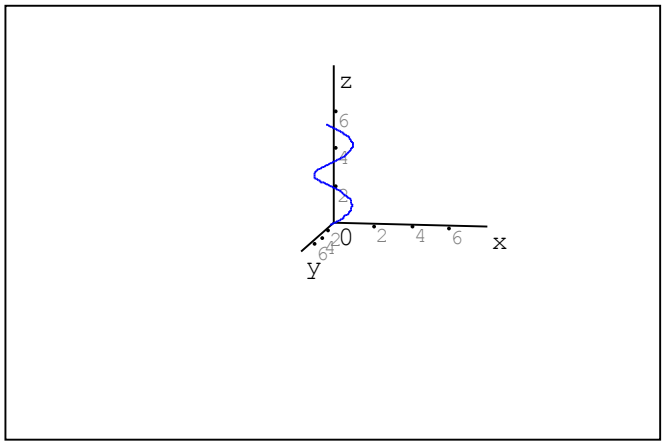
$y_k := \cos(t_k)$

$z_k := \frac{t_k}{2}$

Generate vectors x, y, and z using a "for" loop

$M := \text{augment}(x, y, z)$

Build matrix M with coordinates (x,y,z)

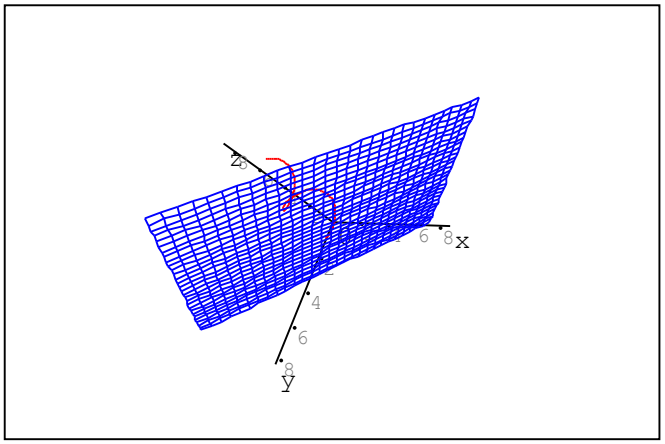


M

Plot matrix M in a 3D plot

Drawing a surface and a space curve:

Use the "Equation System" option in the "Plot" palette, and enter the equation of the surface (e.g., x+y) and the matrix that represents the space curve (e.g., M)



$$\begin{cases} x+y \\ M \end{cases}$$

Plotting 2 space curves:

In this example two straight lines in 3D are produced by using linear parametric equations.

$t := -2, -1.9 .. 2$

Vector t is the parameter, and the coordinates of the two curves are given by (x1,y1,z1) and (x2,y2,z2). These lines are represented by the matrices P and Q, respectively.

$n := \text{length}(t)$

$n = 40$

for $k \in 1 .. n$

$x1_k := 1 + 5 \cdot t_k$

$y1_k := -1 + 4 \cdot t_k$

$z1_k := 1 + 8 \cdot t_k$

for $k \in 1 .. n$

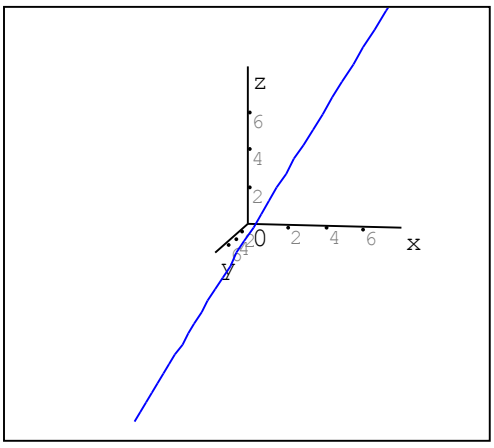
$x2_k := 1 + 3 \cdot t_k$

$y2_k := -1 + t_k$

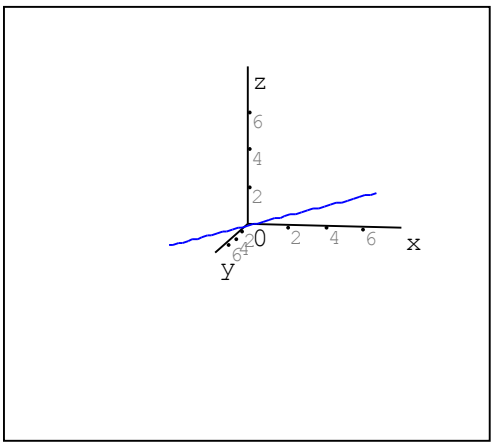
$z2_k := t_k$

$P := \text{augment}(x1, y1, z1)$

$Q := \text{augment}(x2, y2, z2)$

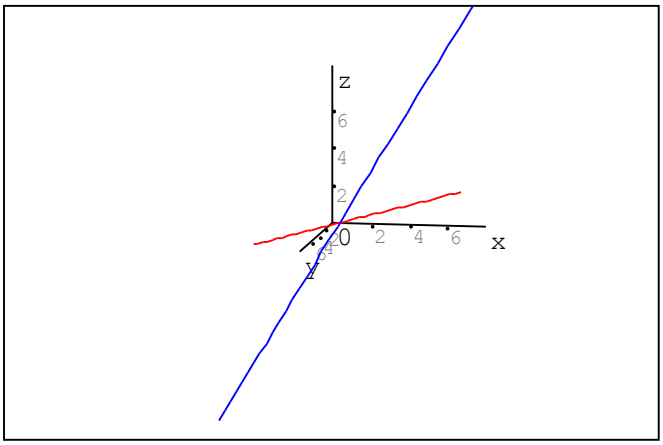


P



Q

Individual plots of curves given by matrices P and Q.



<--- Join plot of
lines given
by matrices
P and Q

P
Q

Appendix 2 - Examples of 2D geometric figures in SMath Studio

This worksheet illustrates the use of SMath Studio 2D graphs to produce selected regular and irregular geometric figures.

Straight-line segment:

Given the two points, A and B, representing the extremes of the straight-line segment, plot the segment.

* Extreme points: $A := \begin{pmatrix} -5 \\ -2 \end{pmatrix}$ $B := \begin{pmatrix} 3 \\ 10 \end{pmatrix}$

* Generate matrix of points:

  * Slope: $m := \frac{B_2 - A_2}{B_1 - A_1}$ $m = 1.5$

  * Intercept: $b := A_2 - m \cdot A_1$ $b = 5.5$

  * Number of points: $n := 5$

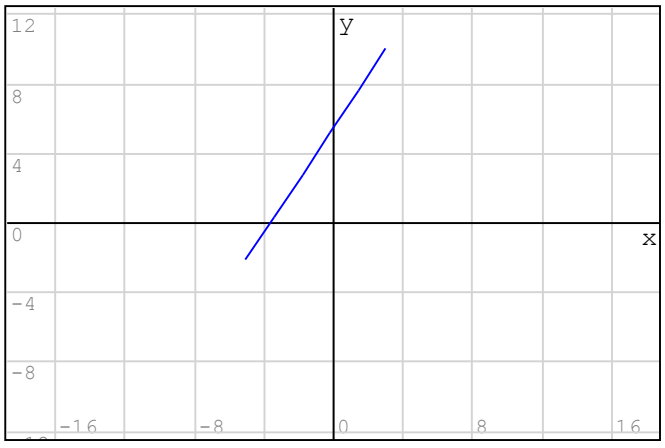
  * Increment: $\Delta x := \frac{B_1 - A_1}{n}$ $\Delta x = 1.6$

  * x-series: $xS := A_1 , A_1 + \Delta x .. B_1$

  * y-series:

$$\text{for } k \in 1 .. n+1$$
$$yS_k := m \cdot xS_k + b$$

  * Matrix of points: $MS := \text{augment}(xS , yS)$



MS

* Mid-point: $M := \frac{1}{2} \cdot (A + B)$ $M = \begin{pmatrix} -1 \\ 4 \end{pmatrix}$

* Normal slope: $mP := -\frac{1}{m}$ $mP = -0.6667$

* Normal intercept: $bP := M_2 - mP \cdot M_1$ $bP = 3.3333$

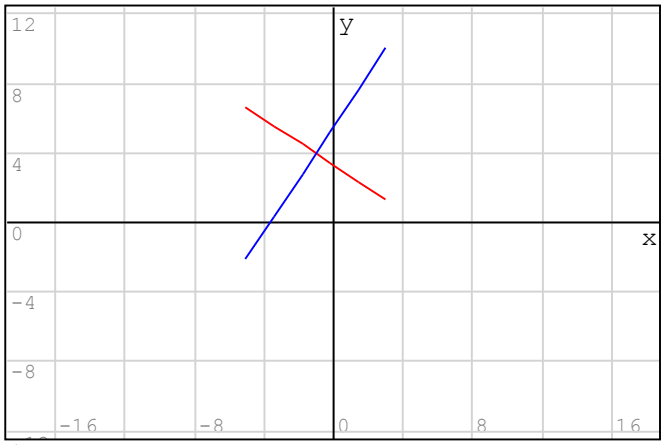
* Normal increment: $\Delta xP := \frac{B_2 - A_2}{n}$ $\Delta xP = 2.4$

* Normal x series: $xP := A_2 , A_2 + \Delta x .. B_2$

* Normal y-series:

```
for k ∈ 1 .. n+1
  yP_k := mP · xS_k + bP
```

* Matrix of points (normal): $MP := augment(xS , yP)$



MS
MP

Circle:

Given the center, C, and radius, r, of a circle, plot the circle.

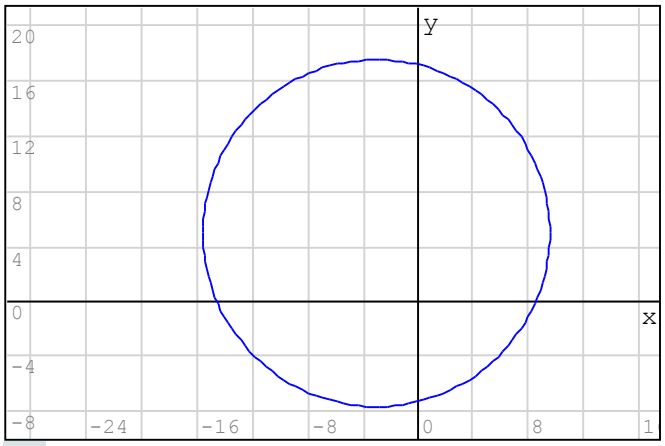
* Center: $C := \begin{pmatrix} -3 \\ 5 \end{pmatrix}$ * Radius: $r := 12.5$

* Generate matrix of data: $n := 150$

$\Delta\theta := \frac{2 \cdot \pi}{n}$ $\Delta\theta = 0.0419$

```
for k ∈ 0 .. n
  xC_{k+1} := C_1 + r · cos(k · Δθ)
  yC_{k+1} := C_2 + r · sin(k · Δθ)
```

$MC := augment(xC , yC)$



MC

Regular polygon of n sides:

Given the number of sides, n, the center of the circumscribed circle, C, and its radius, plot the regular polygon of n sides. This example shows a regular pentagon.

* Number of sides: $n := 5$

* Center of polygon: $C := \begin{pmatrix} 3 \\ -2 \end{pmatrix}$

* Radius of circumscribed circle: $r := 12$

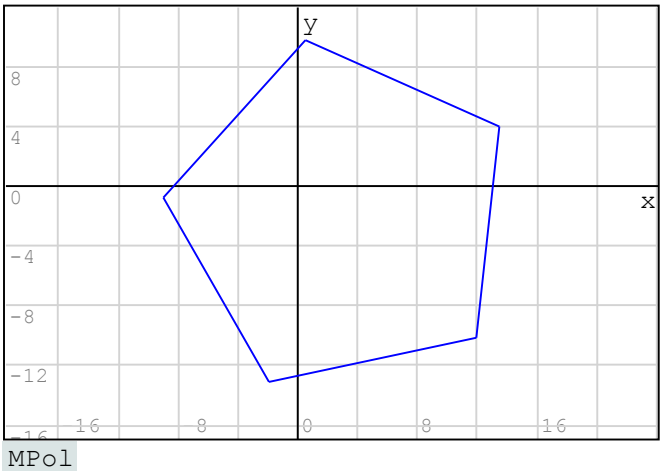
* Initial angle: $\theta 0 := \frac{\pi}{6}$

* Incremental angle: $\Delta\theta := \frac{2 \cdot \pi}{n}$ $\Delta\theta = 1.2566$

* Generate matrix of points:

$$\text{for } k \in 0 \ldots n$$
$$xPol_{k+1} := C_1 + r \cdot \cos(\theta 0 + k \cdot \Delta\theta)$$
$$yPol_{k+1} := C_2 + r \cdot \sin(\theta 0 + k \cdot \Delta\theta)$$

$MPol := \text{augment}(xPol, yPol)$



MPol

Calculating the area and perimeter of the regular polygon:

The matrices representing the vertices of a polygon, whether regular or not, include an extra repetition of the first vertex:

$$MPol = \begin{pmatrix} 13.3923 & 4 \\ 0.5051 & 9.7378 \\ -8.9343 & -0.7457 \\ -1.8808 & -12.9625 \\ 11.9177 & -10.0296 \\ 13.3923 & 4 \end{pmatrix}$$

With such a matrix available, the area and perimeter of the polygon are calculated as follows.

$M := MPol$

$n := rows(M) - 1$ $\quad n = 5$

$Area := 0$

$Perim := 0$

for $k \in 1 .. n$

$Area := eval\left(Area + \frac{1}{2} \cdot \left(M_{k,1} \cdot M_{k+1,2}\right)\right)$

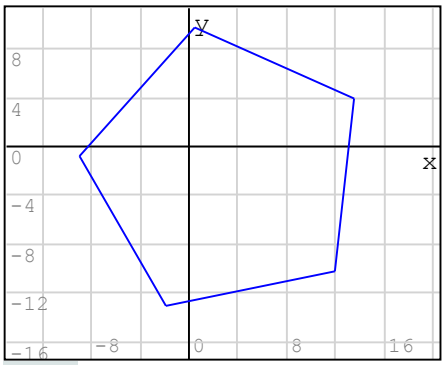
$Area := eval\left(Area - \frac{1}{2} \cdot \left(M_{k+1,1} \cdot M_{k,2}\right)\right)$

$d2 := eval\left(\left(M_{k+1,1} - M_{k,1}\right)^2\right)$

$d2 := eval\left(d2 + \left(M_{k+1,2} - M_{k,2}\right)^2\right)$

$Perim := eval\left(Perim + \sqrt{d2}\right)$

$Area := |Area|$



MPol

$Area = 342.3803$ $\quad Perim = 70.5342$

Irregular polygon:

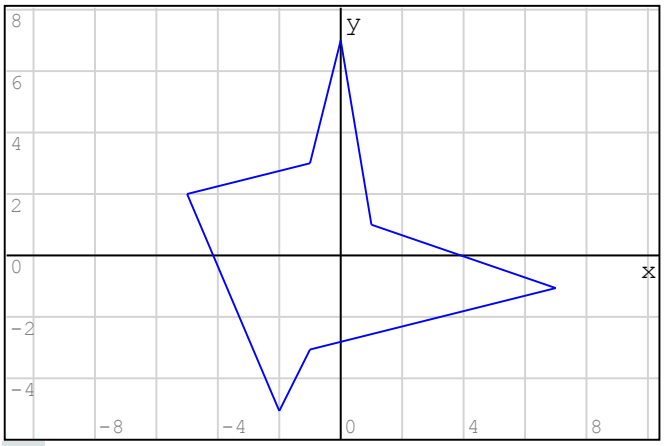
Enter the (x,y) coordinates of the n polygon vertices in a nx2 matrix, e.g.,

$$MI := \begin{pmatrix} -5 & 2 \\ -1 & 3 \\ 0 & 7 \\ 1 & 1 \\ 7 & -1 \\ -1 & -3 \\ -2 & -5 \end{pmatrix}$$

Typically, the initial point needs to be repeated for the polygon to be completed. This is accomplished by using:

$$MN := augment\left(MI^T, row(MI, 1)^T\right)^T$$

$$MN = \begin{pmatrix} -5 & 2 \\ -1 & 3 \\ 0 & 7 \\ 1 & 1 \\ 7 & -1 \\ -1 & -3 \\ -2 & -5 \\ -5 & 2 \end{pmatrix}$$



MN

Calculating the area and perimeter of the irregular polygon:

As with the case of the regular polygon, the matrix representing the vertices of the irregular polygon include an extra repetition of the first vertex:

$$MN = \begin{pmatrix} -5 & 2 \\ -1 & 3 \\ 0 & 7 \\ 1 & 1 \\ 7 & -1 \\ -1 & -3 \\ -2 & -5 \\ -5 & 2 \end{pmatrix}$$

With such a matrix available, the area and perimeter of the polygon is calculated as follows.

Calculating area and perimeter for MN:

$M := MN$

$n := rows(M) - 1$ $\quad n = 7$

$Area := 0$

$Perim := 0$

for $k \in 1..n$

$Area := eval\left(Area + \frac{1}{2} \cdot \left(M_{k,1} \cdot M_{k+1,2}\right)\right)$

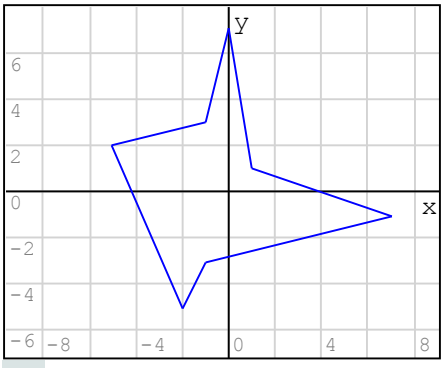
$Area := eval\left(Area - \frac{1}{2} \cdot \left(M_{k+1,1} \cdot M_{k,2}\right)\right)$

$d2 := eval\left(\left(M_{k+1,1} - M_{k,1}\right)^2\right)$

$d2 := eval\left(d2 + \left(M_{k+1,2} - M_{k,2}\right)^2\right)$

$Perim := eval\left(Perim + \sqrt{d2}\right)$

$Area := |Area|$



MN

$Area = 7.5$ $\quad Perim = 38.7516$

Appendix 3 - The Newton-Raphson method for solving equations

In this appendix we present examples of the Newton-Raphson method.

1 - The Newton-Raphson for solving single equations:

The Newton-Raphson method used for solving an equation of the form

$$f(x) = 0$$

requires the knowledge of the derivative f'(x). This can be easily accomplished in SMath Studio using the "Derivative" option in the "Functions" palette:

$$fp(x) = \frac{d}{dx} f(x)$$

Given an initial guess of the solution, $x = x_0$, the solution can be approximated by the iterative calculation:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

for $k = 0, 1, \ldots$

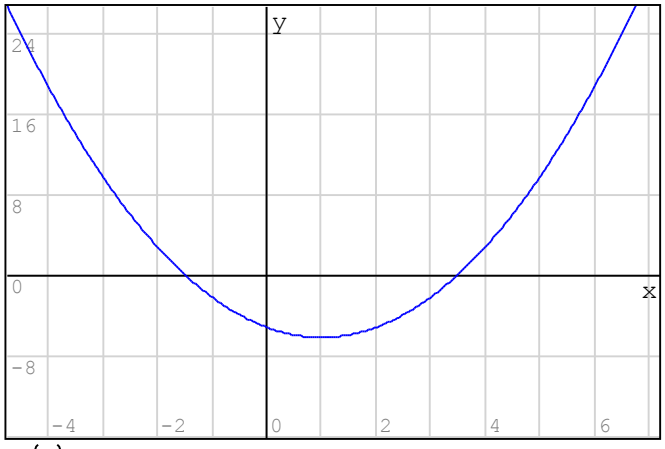
The iteration continues until either the solution converges, i.e., $|f(x_{k+1})| < \varepsilon$, or a certain large number of iterations are performed without convergence, i.e., $k > n_{max}$

Example: Solve the equation: $x^2 - 2 \cdot x - 5 = 0$

Solution: A graph of the function can help us find where the solutions may be located:

Define the function: $f(x) := x^2 - 2 \cdot x - 5$

Produce a graph of f(x):



f(x)

The graph shows solutions near x = -2 and x = 3. We can implement the solution using the Newton-Raphson method as follows:

$$fp(x) := \frac{d}{dx} f(x) \qquad fp(x) \rightarrow 2\cdot(-1+x)$$

Parameters of the solution are: $\varepsilon := 1.0\cdot 10^{-6}$ $nmax := 100$

First solution:

Starting with a guess of $xG := -2.5$

we find a solution by using the following iterative procedure:

$k := 0$

while $\left((k \le nmax) \wedge \left(|f(xG)| > \varepsilon\right)\right)$

$xGp1 := xG - \frac{f(xG)}{fp(xG)}$

$k := k + 1$

$xG := xGp1$

$xG = -1.4495$ This is the solution found

$k = 4$ After this many iterations

$f(xG) = 2.1427\cdot 10^{-11}$ The function at the solution point

Second solution:

Starting with a guess of $xG := 4.2$

we find a solution by using the following iterative procedure:

$k := 0$

while $\left((k \le nmax) \wedge \left(|f(xG)| > \varepsilon\right)\right)$

$xGp1 := xG - \frac{f(xG)}{fp(xG)}$

$k := k + 1$

$xG := xGp1$

$xG = 3.4495$ This is the solution found

$k = 4$ After this many iterations

$f(xG) = 2.2529\cdot 10^{-13}$ The function at the solution point

2 - Solution to equations with function "solve":

Most equations can be solved using function "solve" in SMath Studio. For the present case we'll have:

$$solve(f(x) = 0, x) = \begin{pmatrix} -1.4495 \\ 3.4495 \end{pmatrix}$$

Alternatively, you can use:

$$solve(x^2 - 2\cdot x - 5 = 0, x) = \begin{pmatrix} -1.4495 \\ 3.4495 \end{pmatrix}$$

3 - The Newton-Raphson method for a system of equations:

A system of n equations in n unknowns can be represented as:

$$f1(x_1, x_2 .. x_n) = 0$$
$$f2(x_1, x_2 .. x_n) = 0$$
$$\ldots$$
$$fn(x_1, x_2 .. x_n) = 0$$

or simply, $f(x) = 0$, with

$$f(x) = \begin{pmatrix} f1(x_1, x_2 .. x_n) \\ f2(x_1, x_2 .. x_n) \\ \cdot \\ fn(x_1, x_2 .. x_n) \end{pmatrix} = \begin{pmatrix} f1(x) \\ f2(x) \\ \cdot \\ fn(x) \end{pmatrix}$$

The variable x is defined as the vector:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ x_n \end{pmatrix}$$

We can provide an initial guess for the solution, x_0 , and proceed with an iterative process defined by the formula:

$$x_{k+1} = x_k - J(x_k)^{-1} \cdot f(x_k)$$

for $k = 0, 1, \ldots$ In this formula, $J(x_k)$, is the Jacobian matrix of the function defined as [to be 100% correct the derivatives in this matrix should be partial derivatives]:

$$J(x_k) = \begin{pmatrix} \frac{dy_1}{dx_1} & \frac{dy_1}{dx_2} & \cdot & \frac{dy_1}{dx_n} \\ \frac{dy_2}{dx_1} & \frac{dy_2}{dx_2} & \cdot & \frac{dy_2}{dx_n} \\ \cdot & \cdot & \cdot & \cdot \\ \frac{dy_n}{dx_1} & \frac{dy_n}{dx_2} & \cdot & \frac{dy_n}{dx_n} \end{pmatrix}$$

Example:

How to calculate the Jacobian matrix of a system of three equations. Given the system of three equations:

$$f(x) := \begin{pmatrix} x_1 + x_2 + x_3 - 6 \\ x_1 \cdot x_2 \cdot x_3 - 6 \\ x_1^2 + x_2^2 + x_3^3 - 14 \end{pmatrix}$$

This is obvious, but could useful for larger functions:

$n := \text{length}(f(x))$

$n = 3$

The following nested "for" loops calculate the elements of the jacobian matrix as the elements "jac[i,j]":

```
for i ∈ 1..n
  for j ∈ 1..n
    jac_{i,j} := d/dx_j f(x)_i
```

The following definition creates the function "Jacobi" that represents the Jacobian matrix of the function f(x) shown earlier:

$Jacobi(x) := jac$

$$Jacobi(x) \rightarrow \begin{pmatrix} 1 & 1 & 1 \\ x_2 \cdot x_3 & x_1 \cdot x_3 & x_1 \cdot x_2 \\ 2 \cdot x_1 & 2 \cdot x_2 & 3 \cdot x_3^2 \end{pmatrix}$$

Note: This approach for calculating the Jacobian matrix of a vector function was made available by Radovan Omorjan (omorr) in the SMath Studio wiki page: http://smath.info/wiki/diff.ashx

---

Generalized Newton-Raphson method for solving a system of equations:

The parameters of the solution are: $nmax := 100$ $\varepsilon := 1 \cdot 10^{-20}$

An initial guess is: $xG := \begin{pmatrix} 5 \\ -1 \\ 4 \end{pmatrix}$

The iterative process for the solution is expressed as:

$k := 0$

```
while ((k ≤ nmax) ∧ (max(f(xG)) > ε))
    xGp1 := xG - Jacobi(xG)^-1 · f(xG)
    k := k + 1
    xG := xGp1
```

A solution is found after these many iterations:

$k = 12$

Here's a solution: $xG = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$

And the function at that point: $f(xG) = \begin{pmatrix} -3.9328 \cdot 10^{-15} \\ -1.0068 \cdot 10^{-14} \\ -3.253 \cdot 10^{-14} \end{pmatrix}$

Note: The function representing the system of equations solved above, namely,

$$f(x) := \begin{pmatrix} x_1 + x_2 + x_3 - 6 \\ x_1 \cdot x_2 \cdot x_3 - 6 \\ x_1^2 + x_2^2 + x_3^3 - 14 \end{pmatrix}$$

can be thought of representing the system of equations:

$$x + y + z - 6 = 0 \qquad x + y + z = 6$$

$$x \cdot y \cdot z - 6 = 0 \quad \text{or} \quad x \cdot y \cdot z = 6$$

$$x^2 + y^2 + z^2 - 14 = 0 \qquad x^2 + y^2 + z^2 = 14$$

with the variable substitution: $x_1 = x$, $x_2 = y$, and $x_3 = z$.

Appendix 4 - The 4th -order Runge-Kutta method for a single ODE

Problem description

This worksheet provides two different versions of the 4th-order Runge-Kutta method for solving a first-order ordinary differential equation (ODE) of the form:

$$\frac{d}{dx} y = f(x, y)$$

subject to the initial condition: $y(xs) = ys$

The solution will be provided in the range: $xs \le x \le xe$ which will be divided into n subintervals to produce solution vectors "xsol" and "ysol", as detailed below.

The increment in the x solution vector, Δx, is calculated as:

$$\Delta x = \frac{xe - xs}{n}$$

Thus, the solution vector will be calculated using the uniformly-distributed values: [xs,xs+Δx,xs+2*Δx,...,xs+n*Δx]. Using SMath Studio the x solution vector can be easily created using the command:

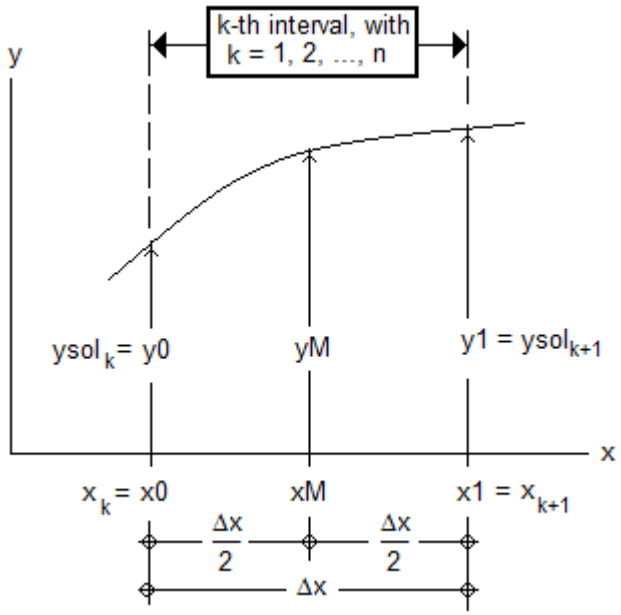
xsol: range(xs,xe,xs+Δx)

The y solution vector will be calculated using the Runge-Kutta algorithm, which is detailed below. The first value in the y solution will be the initial condition ys, i.e.,

$$ysol_1 = ys$$

Iterative process for solution - version 1

Next, we start an iterative procedure with the index k varying between 1 and n, i.e., k = 1, 2, ..., n. To illustrate the calculations involved, we illustrate the k-th interval in the following figure:



In this figure the lower x limit of the interval is x[k], which is represented by x0 in the calculations. On the other hand, the upper x limit of the interval is x[k+1], which is represented by x1. The corresponding values of the solution are y0 = y(x0) and y1 = y(x1). The Runge-Kutta algorithm uses the mid-point (xM,yM) illustrated also in the figure.

Runge-Kutta calculations - version 1

The calculations involved in the 4th-order Runge-Kutta algorithm to calculate y1 = ysol[k+1] are listed below:

$$xM = x0 + \frac{1}{2}\cdot\Delta x \qquad K1 = \Delta x\cdot f(x0, y0)$$

$$yM = y0 + \frac{1}{2}\cdot K1 \qquad K2 = \Delta x\cdot f(xM, yM)$$

$$yM = y0 + \frac{1}{2}\cdot K2 \qquad K3 = \Delta x\cdot f(xM, yM)$$

$$y1 = y0 + K3 \qquad K4 = \Delta x\cdot f(x1, y1)$$

$$ysol_{k+1} = y0 + \frac{1}{6}\cdot(K1 + 2\cdot K2 + 2\cdot K3 + K4)$$

Solution summary and graph

Once the iterative process has been completed, the solution will be contained in the column vectors "xsol" and "ysol". At this point, the user may decide to show the solution as vectors, or put them together into a matrix, say,

$$M = augment(xsol, ysol)$$

This matrix can be stored in a file, or plotted in a 2D graph to show the solution graphically.

Example of 4th-order Runge-Kutta solution -version 1

Solve the ODE: $\frac{dy}{dx} = \sin(x)$, subject to the initial condition: $y(0) = 1$ in the interval: $0 \le x \le 20$

using 100 intervals in the solution.

Solution:

First, define the function f(x,y): $f(x, y) := \sin(x) + \cos(y)$

The initial conditions are: $xs := 0$ $ys := 1$

The end of the solution interval is: $xe := 20$

Use 100 intervals: $n := 100$

Calculate the increment size, Δx:

$\Delta x := \text{eval}\left(\frac{xe - xs}{n}\right)$ $\Delta x = 0.2$

Create the x solution vector: $xsol := \text{eval}(xs, xs + \Delta x .. xe)$

Create the y solution vector as a zero matrix of n rows and 1 column, and initialize the y solution vector with the initial condition ys:

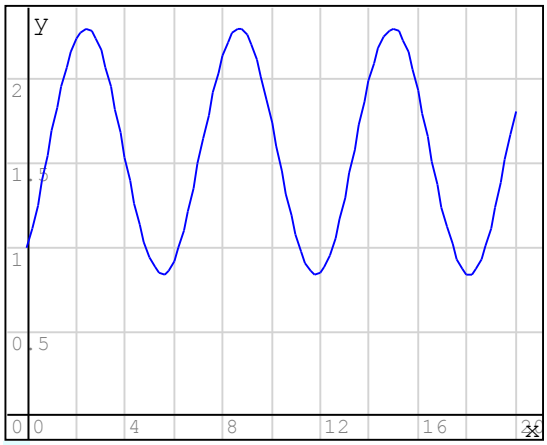
$ysol := \text{matrix}(n, 1)$ $ysol_1 := ys$

The following "for" loop calculates the Runge-Kutta algorithm (version 1) to produce the solution:

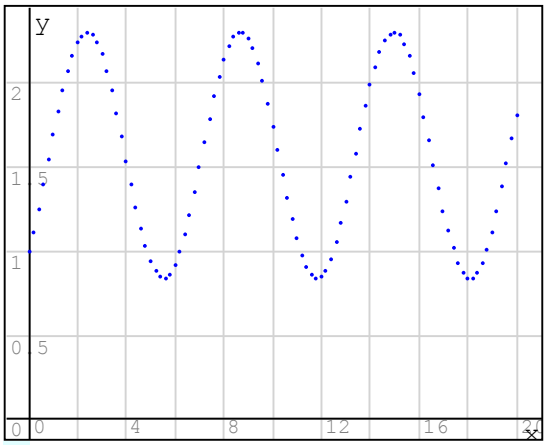
```
for k ∈ 1 .. n
    x0 := eval(xsol_k)
    y0 := eval(ysol_k)
    xM := eval(x0 + (1/2)·Δx)
    K1 := eval(Δx·f(x0, y0))
    yM := eval(y0 + (1/2)·K1)
    K2 := eval(Δx·f(xM, yM))
    yM := eval(y0 + (1/2)·K2)
    K3 := eval(Δx·f(xM, yM))
    y1 := eval(y0 + K3)
    x1 := eval(xsol_(k+1))
    K4 := eval(Δx·f(x1, y1))
    ysol_(k+1) := eval(y0 + (1/6)·(K1 + 2·K2 + 2·K3 + K4))
```

The solution is summarized into matrix M and shown as a x-y plot:

$M := \text{augment}(xsol, ysol)$



M

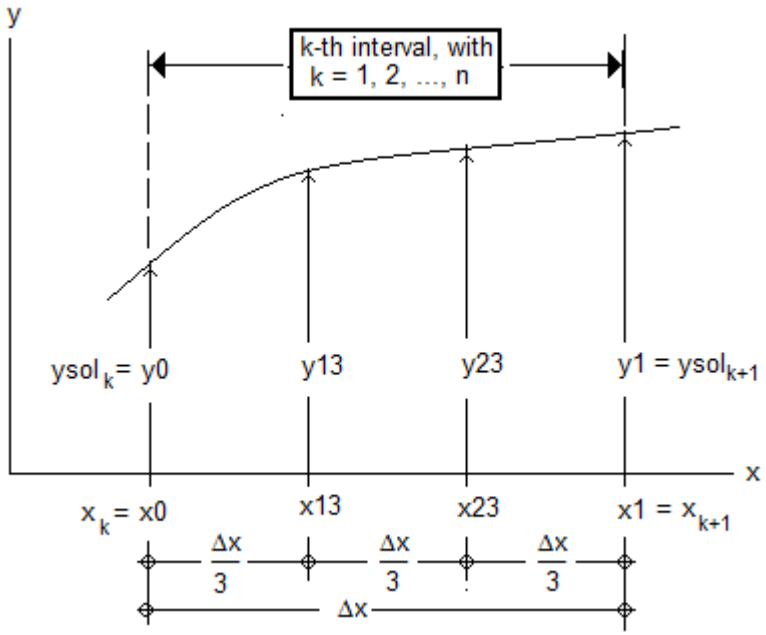


M

The plot to the left uses the "Graph by lines" option in the "Plot" palette, while the plot to the right uses the "Graph by points" option in the "Plot" palette.

Iterative process for solution - version 2

The second version of the Runge-Kutta solution divides the k-th interval into three equal parts are illustrated in the figure below:



In this figure the lower x limit of the interval is x[k], which is represented by x0 in the calculations. On the other hand, the upper x limit of the interval is x[k+1], which is represented by x1. The corresponding values of the solution are y0 = y(x0) and y1 = y(x1). The Runge-Kutta algorithm uses two mid-points (x13,y13) and (x23, y23) illustrated also in the figure.

Runge-Kutta calculations - version 2

The calculations involved in the 4th-order Runge-Kutta algorithm to calculate y1 = ysol[k+1] are listed below:

$x13 = x0 + \frac{1}{3}\cdot\Delta x$ $\quad x23 = x0 + \frac{2}{3}\cdot\Delta x$ $\quad K1 = \Delta x\cdot f(x0, y0)$

$y13 = y0 + \frac{1}{3}\cdot K1$ $\quad K2 = \Delta x\cdot f(x13, y13)$

$y23 = y13 + \frac{1}{3}\cdot K2$ $\quad K3 = \Delta x\cdot f(x23, y23)$

$y1 = y0 + K1 - K2 + K3$ $\quad K4 = \Delta x\cdot f(x1, y1)$

$$ysol_{k+1} = y0 + \frac{1}{8}\cdot(K1 + 3\cdot K2 + 3\cdot K3 + K4)$$

Example of 4th-order Runge-Kutta solution -version 2

Solve the ODE: $\frac{dy}{dx} = \sin(x)$, subject to the initial condition: $y(0) = 1$ in the interval: $0 \le x \le 20$ using 100 intervals in the solution.

Solution:

First, define the function f(x,y): $f(x, y) := \sin(x) + \cos(y)$

The initial conditions are: $xs := 0$ $\quad ys := 1$

The end of the solution interval is: $xe := 20$

Use 100 intervals: $n := 100$

Calculate the increment size, Δx:

$\Delta x := \text{eval}\left(\frac{xe - xs}{n}\right)$ $\quad \Delta x = 0.2$

Create the x solution vector: $xsol := \text{eval}(xs, xs + \Delta x .. xe)$

Create the y solution vector as a zero matrix of n rows and 1 column, and initialize the y solution vector with the initial condition ys:

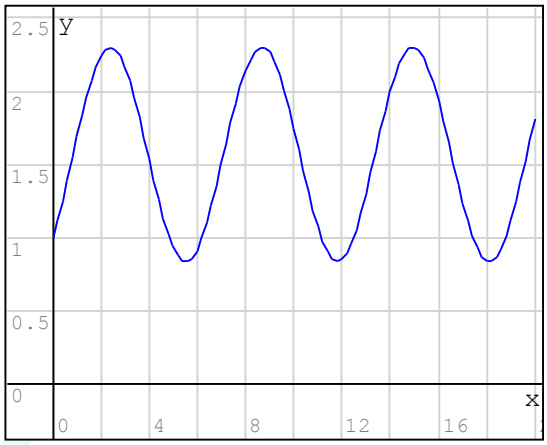
$ysol := \text{matrix}(n, 1)$ $\quad ysol_1 := ys$

The following "for" loop calculates the Runge-Kutta algorithm (version 2) to produce the solution:

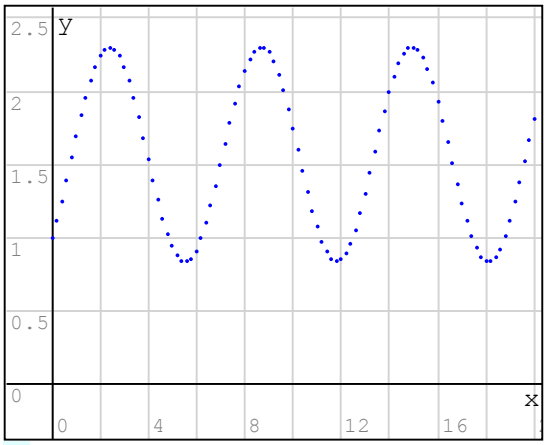
```
for k ∈ 1 .. n
    x0 := eval(xsol_k)
    y0 := eval(ysol_k)
    x13 := eval(x0 + (1/3)·Δx)
    x23 := eval(x0 + (2/3)·Δx)
    K1 := eval(Δx·f(x0, y0))
    y13 := eval(y0 + (1/3)·K1)
    K2 := eval(Δx·f(x13, y13))
    y23 := eval(y13 + (1/3)·K2)
    K3 := eval(Δx·f(x23, y23))
    y1 := eval(y0 + K1 − K2 + K3)
    x1 := eval(xsol_(k+1))
    K4 := eval(Δx·f(x1, y1))
    ysol_(k+1) := eval(y0 + (1/8)·(K1 + 3·K2 + 3·K3 + K4))
```

The solution is summarized into matrix N and shown as a x-y plot:

```
N := augment(xsol, ysol)
```



N



N

The plot to the left uses the "Graph by lines" option in the "Plot" palette, while the plot to the right uses the "Graph by points" option in the "Plot" palette.

Appendix 5 - The 4th -order Runge-Kutta method for a system of ODEs

Problem description

Consider the case of a system of two first-order ODEs given by:

$$\frac{dy_1}{dx} = f1\left(x, \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right) = f1(x, y)$$

$$\frac{dy_2}{dx} = f2\left(x, \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right) = f2(x, y)$$

subject to the initial conditions:

$$ys_1 = y_1(xs_1) \quad \text{and} \quad ys_2 = y_2(xs_2)$$

This system of equations can be re-written as a single ODE in which y and f are column vectors, i.e.,

$$\frac{dy}{dx} = f(x, y) \quad , \text{ with } \quad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \quad \text{and} \quad f(x, y) = \begin{pmatrix} f1(x, y) \\ f2(x, y) \end{pmatrix}$$

The initial conditions are given by the vector: $ys = \begin{pmatrix} ys1 \\ ys2 \end{pmatrix}$

Once the system of equations is written as a single ODE, the Runge-Kutta algorithms presented for a single ODE can be used to solve the equation. This illustrated in the following example.

Example

Solve the system of first-order ODEs:

$$\frac{dy_1}{dx} = \sin(x) + \cos(y_1) + \sin(y_2)$$

$$\frac{dy_2}{dx} = \cos(x) + \sin(y_2)$$

Subject to the initial conditions:

$$y1(0) = -1 \quad \text{and} \quad y2(0) = 1$$

Solve the ODEs in the interval: $0 \le x \le 20$ using 100 intervals.

Solution (version 1):

First, define the vector function f(x,y):

$$f(x, y) := \begin{pmatrix} \sin(x) + \cos(y_1) + \sin(y_2) \\ \cos(x) + \sin(y_2) \end{pmatrix}$$

The initial conditions are: $xs := 0$ $ys := \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The end of the solution interval is: $xe := 20$

Use 100 intervals: $n := 100$

Calculate the increment size, Δx:

$\Delta x := \text{eval}\left(\frac{xe - xs}{n}\right)$ $\Delta x = 0.2$

Create the x solution vector: $xsol := \text{eval}(xs, xs + \Delta x .. xe)$

The y-solution vector gets initialized as follows:

$ysol := ys$ $ysol = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The following "for" loop calculates the Runge-Kutta algorithm (version 1) to produce the solution:

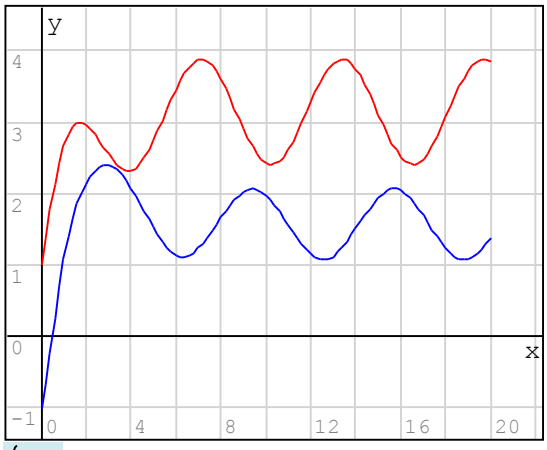
```
for k ∈ 1 .. n
    x0 := eval(xsol_k)
    y0 := eval(col(ysol, k))
    xM := eval(x0 + (1/2)·Δx)
    K1 := eval(Δx·f(x0, y0))
    yM := eval(y0 + (1/2)·K1)
    K2 := eval(Δx·f(xM, yM))
    yM := eval(y0 + (1/2)·K2)
    K3 := eval(Δx·f(xM, yM))
    y1 := eval(y0 + K3)
    x1 := eval(xsol_(k+1))
    K4 := eval(Δx·f(x1, y1))
    y1 := eval(y0 + (1/6)·(K1 + 2·K2 + 2·K3 + K4))
    ysol := augment(ysol, y1)
```

After completing the iterative process, the solution is stored in a row vector called "ysol". This vector can be transposed to put together the graph of the two solutions as illustrated here:

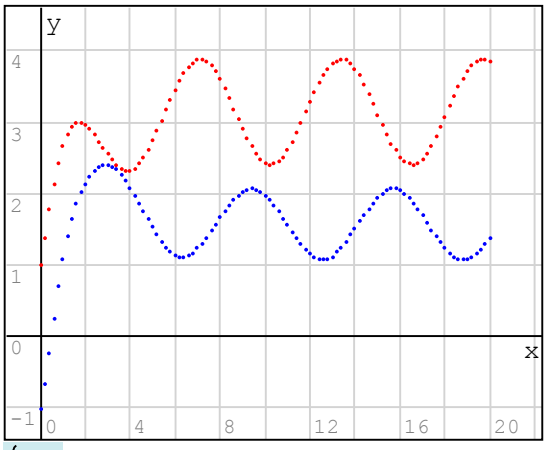
$ysol := ysol^T$

M1 := augment(xsol, col(ysol, 1))

M2 := augment(xsol, col(ysol, 2))



{M1
M2



{M1
M2

The plot to the left uses the "Graph by lines" option in the "Plot" palette, while the plot to the right uses the "Graph by points" option in the "Plot" palette.

Solution (version 2):

First, define the vector function f(x,y):

$$f(x, y) := \begin{pmatrix} \sin(x) + \cos(y_1) + \sin(y_2) \\ \cos(x) + \sin(y_2) \end{pmatrix}$$

The initial conditions are: xs := 0 $ys := \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The end of the solution interval is: xe := 20

Use 100 intervals: n := 100

Calculate the increment size, Δx:

$\Delta x := \text{eval}\left(\frac{xe - xs}{n}\right)$ Δx = 0.2

Create the x solution vector: xsol := eval(xs, xs + Δx .. xe)

The y-solution vector gets initialized as follows:

ysol := ys $ysol = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The following "for" loop calculates the Runge-Kutta algorithm (version 1) to produce the solution:

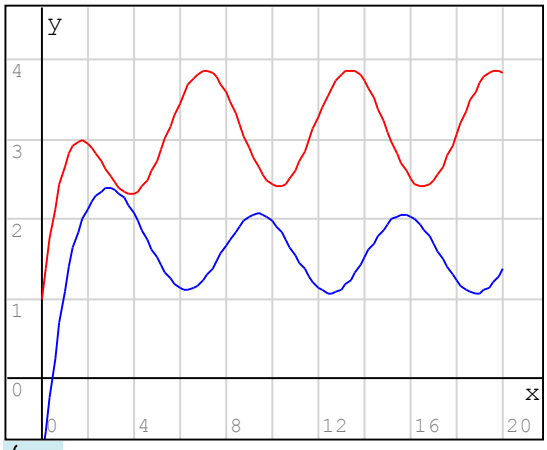
```
for k ∈ 1 .. n
    x0 := eval(xsol_k)
    y0 := eval(col(ysol, k))
    x13 := eval(x0 + (1/3)·Δx)
    x23 := eval(x0 + (2/3)·Δx)
    K1 := eval(Δx·f(x0, y0))
    y13 := eval(y0 + (1/3)·K1)
    K2 := eval(Δx·f(x13, y13))
    y23 := eval(y13 + (1/3)·K2)
    K3 := eval(Δx·f(x23, y23))
    y1 := eval(y0 + K1 − K2 + K3)
    x1 := eval(xsol_(k+1))
    K4 := eval(Δx·f(x1, y1))
    y1 := eval(y0 + (1/8)·(K1 + 3·K2 + 3·K3 + K4))
    ysol := augment(ysol, y1)
```

After completing the iterative process, the solution is stored in a row vector called "ysol". This vector can be transposed to put together the graph of the two solutions as illustrated here:

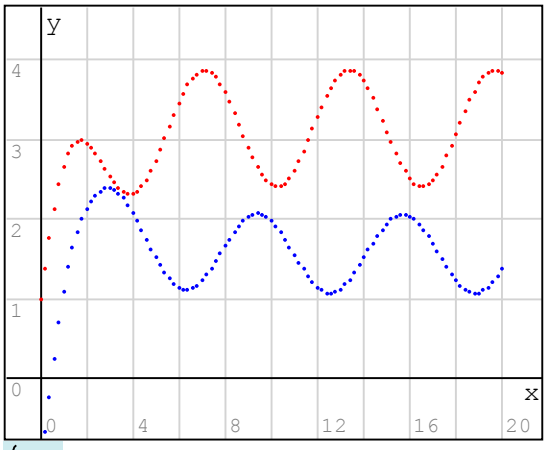
```
ysol := ysol^T
```

```
N1 := augment(xsol, col(ysol, 1))
```

```
N2 := augment(xsol, col(ysol, 2))
```




```
N1
N2
```




```
N1
N2
```

The plot to the left uses the "Graph by lines" option in the "Plot" palette, while the plot to the right uses the "Graph by points" option in the "Plot" palette.

Appendix 6 - The 4th -order Runge-Kutta method for a 2nd order ODE

By Gilberto E. Urroz, Ph.D., P.E.
January 2010

Problem description

Consider the 2nd-order ODE: $y''+y\cdot y'+3\cdot y=\sin(x)$

subject to the initial conditions: $y(0)=-1$ $y'(0)=1$

Variable substitution to form a system of ODEs:

This 2nd-order ODE can be converted into a system of two 1st-order ODEs by using the following variable substitution:

$$u_1=y \quad u_2=y'$$

with initial conditions:

$u_1=-1$ and $u_2=1$ at $x=0$.

The variable substitution $u_2=y'$ is equivalent to:

$$\frac{d}{dx}u_1=u_2 \qquad \text{[Eq. 1]}$$

while the ODE is re-written as: $y''=-y\cdot y'-3\cdot y+\sin(x)$

or: $$\frac{d}{dx}u_2=-u_1\cdot u_2-3\cdot u_1+\sin(x) \qquad \text{[Eq. 2]}$$

The system of equations [Eq. 1] and [Eq. 2] is transformed into the vector ODE:

$$\frac{d}{dx}\begin{pmatrix}u_1\\u_2\end{pmatrix}=\begin{pmatrix}u_2\\-u_1\cdot u_2-3\cdot u_1+\sin(x)\end{pmatrix}$$

or, $$\frac{d}{dx}u=f(x,u)\ , \text{ where } u=\begin{pmatrix}u_1\\u_2\end{pmatrix} \text{ and}$$

$$f(x,u)=\begin{pmatrix}u_2\\-u_1\cdot u_2-3\cdot u_1+\sin(x)\end{pmatrix}$$

The initial conditions are $us=\begin{pmatrix}-1\\1\end{pmatrix}$ at $xs=0$

We'll solve the ODEs in the interval: $0\le x\le 20$ using 100 intervals.

Solution (version 1):

First, define the vector function f(x,u):

$$f(x, u) := \begin{pmatrix} u_2 \\ -u_1 \cdot u_2 - 3 \cdot u_1 + \sin(x) \end{pmatrix}$$

The initial conditions are: $xs := 0$ $us := \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The end of the solution interval is: $xe := 20$

Use 100 intervals: $n := 100$

Calculate the increment size, Δx:

$\Delta x := \text{eval}\left(\frac{xe - xs}{n}\right)$ $\Delta x = 0.2$

Create the x solution vector: $xsol := \text{eval}(xs, xs + \Delta x .. xe)$

The y-solution vector gets initialized as follows:

$usol := us$ $usol = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The following "for" loop calculates the Runge-Kutta algorithm (version 1) to produce the solution:

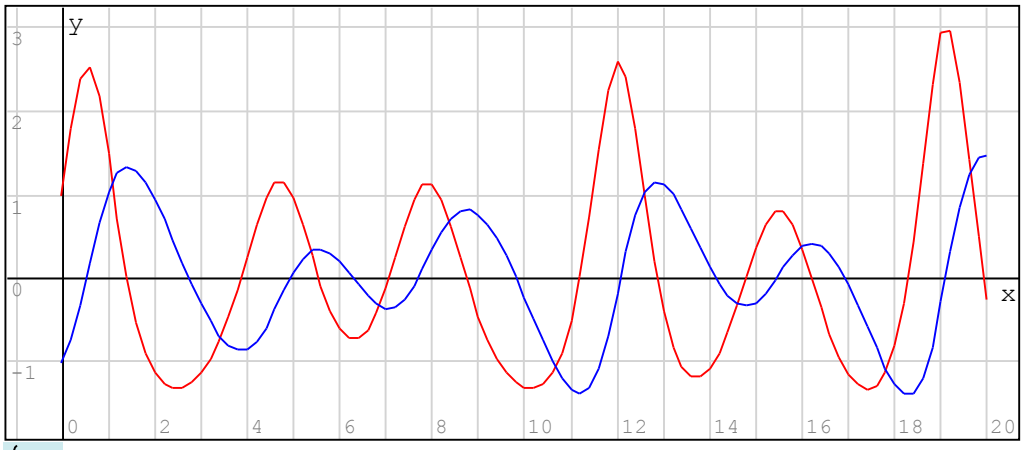
```
for k ∈ 1 .. n
    x0 := eval(xsol_k)
    u0 := eval(col(usol, k))
    xM := eval(x0 + (1/2)·Δx)
    K1 := eval(Δx·f(x0, u0))
    uM := eval(u0 + (1/2)·K1)
    K2 := eval(Δx·f(xM, uM))
    uM := eval(u0 + (1/2)·K2)
    K3 := eval(Δx·f(xM, uM))
    u1 := eval(u0 + K3)
    x1 := eval(xsol_(k+1))
    K4 := eval(Δx·f(x1, u1))
    u1 := eval(u0 + (1/6)·(K1 + 2·K2 + 2·K3 + K4))
    usol := augment(usol, u1)
```

After completing the iterative process, the solution is stored in a row vector called "ysol". This vector can be transposed to put together the graph of the two solutions as illustrated here:

$usol := usol^T$

M1 := augment(xsol, col(usol, 1))

M2 := augment(xsol, col(usol, 2))



$\begin{cases} M1 \\ M2 \end{cases}$

The blue line represents u[1]=y while the red line represents u[2] = dy/dx.

Solution (version 2):

First, define the vector function f(x,y):

$$f(x, u) := \begin{pmatrix} u_2 \\ -u_1 \cdot u_2 - 3 \cdot u_1 + \sin(x) \end{pmatrix}$$

The initial conditions are: $xs := 0$ $\quad us := \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The end of the solution interval is: $xe := 20$

Use 100 intervals: $n := 100$

Calculate the increment size, Δx:

$$\Delta x := \text{eval}\left(\frac{xe - xs}{n}\right) \qquad \Delta x = 0.2$$

Create the x solution vector: $xsol := \text{eval}(xs, xs + \Delta x .. xe)$

The y-solution vector gets initialized as follows:

$usol := us$ $\quad usol = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The following "for" loop calculates the Runge-Kutta algorithm (version 1) to produce the solution:

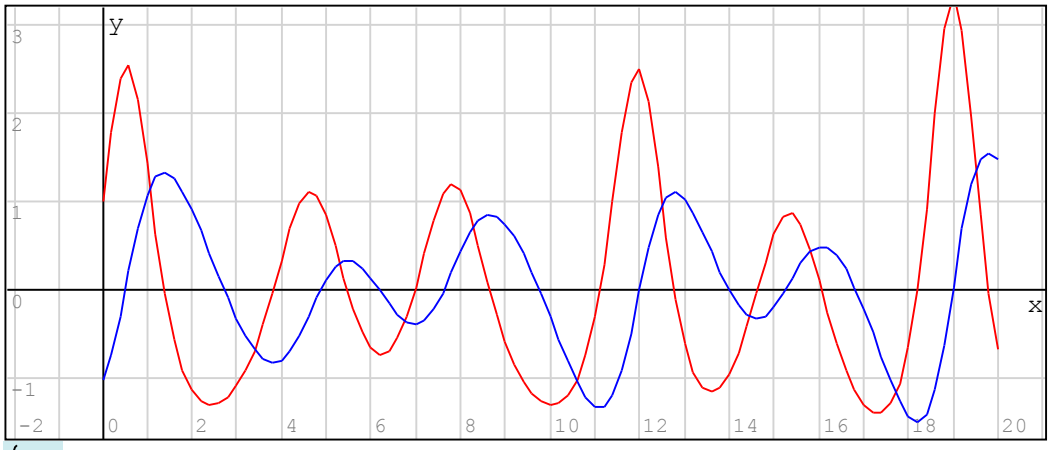
```
for k ∈ 1 .. n
    x0 := eval(xsol_k)
    u0 := eval(col(usol, k))
    x13 := eval(x0 + (1/3)·Δx)
    x23 := eval(x0 + (2/3)·Δx)
    K1 := eval(Δx·f(x0, u0))
    u13 := eval(u0 + (1/3)·K1)
    K2 := eval(Δx·f(x13, u13))
    u23 := eval(u13 + (1/3)·K2)
    K3 := eval(Δx·f(x23, u23))
    u1 := eval(u0 + K1 − K2 + K3)
    x1 := eval(xsol_(k+1))
    K4 := eval(Δx·f(x1, u1))
    u1 := eval(u0 + (1/8)·(K1 + 3·K2 + 3·K3 + K4))
    usol := augment(usol, u1)
```

After completing the iterative process, the solution is stored in a row vector called "ysol". This vector can be transposed to put together the graph of the two solutions as illustrated here:

```
usol := usol^T
```

```
N1 := augment(xsol, col(usol, 1))
```

```
N2 := augment(xsol, col(usol, 2))
```




```
{N1
 N2
```

The blue line represents u[1]=y while the red line represents u[2] = dy/dx.

Appendix 7 - Finding eigenvalues and eigenvectors of a matrix

Given a matrix, say the 4x4 matrix symmetric matrix A:

$$A := \begin{pmatrix} 1 & -2 & 4 & 2 \\ -2 & 2 & -2 & 6 \\ 4 & -2 & 3 & 8 \\ 2 & 6 & 8 & 4 \end{pmatrix}$$

The following program calculates the coefficients of the characteristic equation: $|A - \lambda \cdot I| = 0$

The resulting equation is the polynomial:

$$p_1 + p_2 \cdot \lambda + p_3 \cdot \lambda^2 + \ldots + p_n \cdot \lambda^{n-1} + p_{n+1} \cdot \lambda^n = 0$$

which is solved uising function "polyroots."

The programming steps following the line with function "polyroots," below, are used to calculate a matrix, V, whose columns are the eigenvectors of the matrix A.

The SMath Studio program that calculates eigenvalues and eigenvectors of A is the following:

```
n := rows(A)
p := matrix(n , 1)
I := identity(n)
p_{n+1 1} := -1.0
B := A
p_{n 1} := tr(B)
for j ∈ n-1 , n-2 .. 1
    B := A·(B - p_{j+1 1}·I)
    p_{j 1} := tr(B)/(n-j+1)
p := (-1)^n·p
λ := polyroots(p)
V := matrix(n-1 , 1)
for j ∈ 1 .. n
    Aλ := A - λ_j·I
    Aλm := submatrix(Aλ , 1 , n-1 , 1 , n-1)
    bλ := -submatrix(Aλ , 1 , n-1 , n , n)
    xλ := Aλm^-1·bλ
    V := augment(xλ , V)
V := submatrix(V , 1 , n-1 , 1 , n)
Vb := matrix(1 , n)
for j ∈ 1 .. n
    Vb_{1 j} := 1
V := stack(V , Vb)
for j ∈ 1 .. n
    xλ := col(V , j)
    xλ := xλ/norme(xλ)
    V := augment(V , xλ)
V := submatrix(V , 1 , n , n+1 , 2·n)
```

Coefficients: Eigenvalues: Eigenvectors:

$$p=\begin{pmatrix}-1300\\-530\\93\\10\\-1\end{pmatrix}\qquad \lambda=\begin{pmatrix}-1.952\\6.3272\\-7.8257\\13.4505\end{pmatrix}\qquad V=\begin{pmatrix}-0.2746 & 0.0037 & 0.4651 & 0.8416\\-0.2188 & 0.5103 & -0.7569 & 0.3446\\-0.6082 & 0.5693 & 0.374 & -0.4076\\0.7119 & 0.6446 & 0.2662 & 0.0823\end{pmatrix}$$

Extracting the individual eigenvectors:

$x1:=\mathrm{col}(V,1)\quad x2:=\mathrm{col}(V,2)\quad x3:=\mathrm{col}(V,3)\quad x4:=\mathrm{col}(V,4)$

$$x1=\begin{pmatrix}-0.2746\\-0.2188\\-0.6082\\0.7119\end{pmatrix}\quad x2=\begin{pmatrix}0.0037\\0.5103\\0.5693\\0.6446\end{pmatrix}\quad x3=\begin{pmatrix}0.4651\\-0.7569\\0.374\\0.2662\end{pmatrix}\quad x4=\begin{pmatrix}0.8416\\0.3446\\-0.4076\\0.0823\end{pmatrix}$$

Checking that the eigenvectors are unit and orthogonal:

$x1^{T}\cdot x2=\left(2.8843\cdot 10^{-16}\right)$

$x1^{T}\cdot x3=\left(2.2924\cdot 10^{-14}\right)$

$x1^{T}\cdot x4=\left(-3.5213\cdot 10^{-14}\right)$

$x2^{T}\cdot x3=\left(-1.0401\cdot 10^{-14}\right)$

$x2^{T}\cdot x4=\left(8.4221\cdot 10^{-14}\right)$

$x3^{T}\cdot x4=\left(-2.6966\cdot 10^{-14}\right)$

$\mathrm{norme}(x1)=1$

$\mathrm{norme}(x2)=1$

$\mathrm{norme}(x3)=1$

$\mathrm{norme}(x4)=1$