Chapter 7 - Special topics and additional applications

7.1 MuPAD

1. General concepts

   (1) For using the MuPAD, the *Symbolic Math Toolbox* of MATLAB (version R2008b and later) is needed to be installed in your programming environments.

   (2) Since MATLAB R2020a, MATLAB's Symbolic Math Toolbox still uses the MuPAD language as part of its underlying computational engine. MATLAB *Live Editor* is the recommended environment for performing symbolic math operations.

   (3) MuPAD is a Graphical user interface driven by MATLAB that helps you do algebra, calculus, as well as to graph and visualize functions. As you know, MATLAB is good for writing compact programs and working with numbers. In contrast, MuPAD works with symbols by default, and has a nice menu-aided interface.

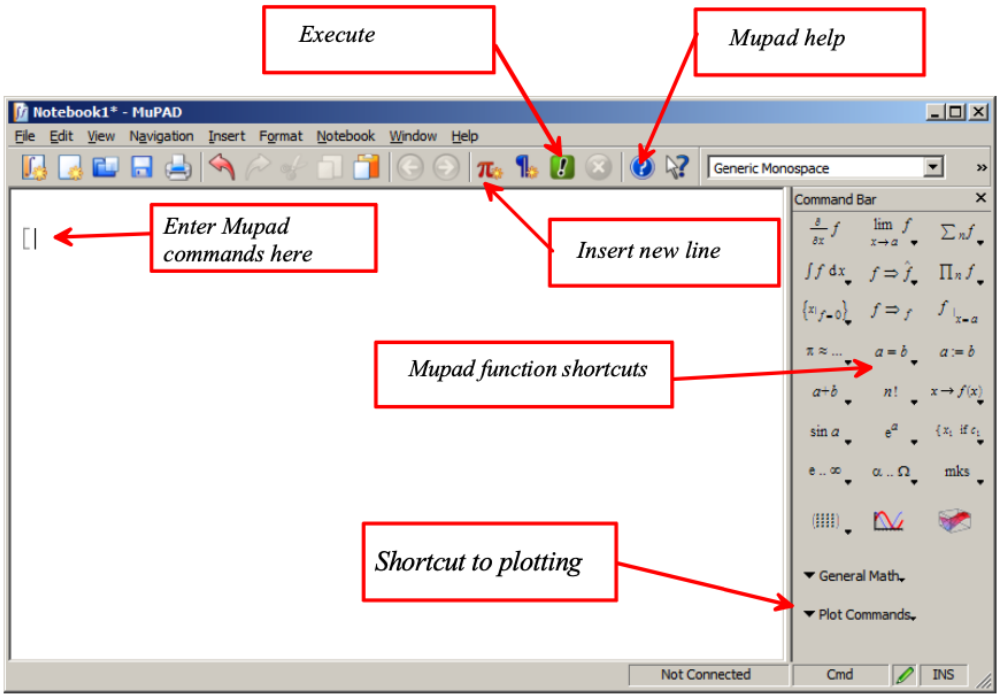
   (4) You could run Mupad by first starting MATLAB, then typing

   ```
   mupad
   ```

   in the MATLAB command window. Now, MuPAD is started, and try to apply the commands introduced in this Section to explore the various functions on doing algebra.

2. First Steps in MuPAD

   (1) You should see the GUI shown in the figure below. Most of the buttons are quite self-explanatory.



   (2) MuPAD functions and operators for numbers are listed in the table below:

`+, -, *, /, ^`	: basic arithmetic operations
`abs`	: absolute value
`ceil`	: rounding up
`div`	: quotient “modulo”
`fact`	: factorial
`float`	: approximation by floating-point numbers
`floor`	: rounding down
`frac`	: fractional part
`ifactor, factor`	: prime factorization
`isprime`	: primality test
`mod`	: remainder “modulo”
`round`	: rounding to nearest
`sign`	: sign
`sqrt`	: square root
`trunc`	: integral part

(3) Simple arithmetic calculations are demonstrated as follows:

The *gamma* function and *besselJ* are special functions in Mathematics - the gamma function is the generalization of the factorial to non-integers, and the Bessel function is the solution to a common differential equation. MuPAD has lots of built in special functions, which can be very useful.

```
2+2
```
$$4$$

```
sin(PI/3)
```
$$\frac{\sqrt{3}}{2}$$

```
30!
```
$$265252859812191058636308480000000$$

```
sin(PI)
```
$$0$$

```
gamma(0.1)
```
$$9.513507699$$

```
besselJ(0.1,1)
```
$$0.770765187$$

(4) For example, in the second case below, MuPAD gives a floating point number because you typed in a floating point number `0.5` as the argument to the Gamma function. You can also ask MuPAD to compute a numerical value for an expression with the `float` function. Note the use of the `%` character - this always refers to the result of the last calculation that Mupad has done.

```
gamma(1/2)
```
$$\sqrt{\pi}$$

```
gamma(0.5)
```
$$1.772453851$$

```
gamma(PI)
```
$$\Gamma(\pi)$$

```
float(%)
```
$$2.288037795$$

(5) Help documentation

a. MuPAD will automatically link the help page for the MATLAB symbolic math toolbox when the the question mark icon is pressed.

b. In the ducumentation of the Symbolic Math Toolbox, the MuPAD section is recommended to be checked.

(6) Save

a. You can save your work in a MuPAD "Notebook" (a bit like a MATLAB script) by going the the *File -> Save menu*. The file should be saved with the default `.mn` extension.

b. Mupad notebooks can also include detailed comments and annotations that help readers follow what the calculations are doing. To insert a text paragraph, just use *Insert -> Text Paragraph*. Use *Insert -> Calculation* to go back to typing in math.

c. Also, you can also export a Mupad notebook to html or pdf format, if you want to publish your work.

3. Basic algebra

(1) Solve equations is demonstrated as:

```
eq1 := a*x^2 + b*x+c = 0
```

$$a x^2 + b x + c = 0$$

```
solve(eq1,x)
```

$$\begin{cases} \left\{ -\frac{b+\sqrt{b^2-4ac}}{2a}, -\frac{b-\sqrt{b^2-4ac}}{2a} \right\} & \text{if } a \neq 0 \\ \left\{ -\frac{c}{b} \right\} & \text{if } a = 0 \wedge b \neq 0 \\ \mathbb{C} & \text{if } a = 0 \wedge b = 0 \wedge c = 0 \\ \varnothing & \text{if } a = 0 \wedge b = 0 \wedge c \neq 0 \end{cases}$$

```
solution := solve(eq1, x, IgnoreSpecialCases)
```

$$\left\{ -\frac{b+\sqrt{b^2-4ac}}{2a}, -\frac{b-\sqrt{b^2-4ac}}{2a} \right\}$$

```
solution[1]
```

$$-\frac{b+\sqrt{b^2-4ac}}{2a}$$

```
solution[2]
```

$$-\frac{b-\sqrt{b^2-4ac}}{2a}$$

a. Because there are two solutions, they are returned in a set (enclosed by `{}`). You can extract each one by using the `[number]` convention.

b. Notice that the "equals" sign is used in two different ways. If you just type `a=b`, you have created an *equation object* that you can use in later manipulations (e.g. solve the equation object). On the other hand, if you type `a:=b^2` (with a colon) then you have assigned the value `b^2` (a symbol) to a variable called `a`. MuPAD will substitute `b^2` for a any time it is used later. For example try this:

```
a := b^2
```

$$b^2$$

```
eq1
```

$$b^2 x^2 + b x + c = 0$$

```
solution[1]
```

$$-\frac{b-\sqrt{b^2-4b^2c}}{2b^2}$$

c. Notice that a in the `eq1` object has been replaced by `b^2`. You can clear the value of `a` variable using the `delete` function.

```
delete(a)
```

```
a
```

$$a$$

```
solution[1]
```

$$-\frac{b+\sqrt{b^2-4ac}}{2a}$$

d. If you want to clear all variables, you can use the `reset` function. This completely restarts MuPAD from the beginning. This is often useful for starting a new problem.

(2) More algebra:

```
(x+2*y)^2
```
$$(x+2\,y)^2$$

```
expand(%)
```
$$x^2+4\,x\,y+4\,y^2$$

```
(x^2-y^2)/(x+y)
```
$$\frac{x^2-y^2}{x+y}$$

MuPAD does not simplify expressions by default. But it can do so if you ask it to

$$\frac{x^2-y^2}{x+y}$$

```
simplify(%)
```
$$x-y$$

This sort of thing is especially handy for trigonometric functions like:

```
sin(a+b+c)
```
$$\sin(a+b+c)$$

```
expand(%)
```
$$\cos(a)\cos(b)\sin(c)+\cos(a)\cos(c)\sin(b)+\cos(b)\cos(c)\sin(a)-\sin(a)\sin(b)\sin(c)$$

```
simplify(%)
```
$$\sin(a+b+c)$$

(3) MuPAD can solve systems of equations:

```
reset
```
reset

```
eq1 := a*x + b*y = c
```
$$a\,x+b\,y=c$$

```
eq2 := c*x+d*y=e
```
$$c\,x+d\,y=e$$

```
sol := solve({eq1,eq2},{x,y}, IgnoreSpecialCases)
```
$$\left\{\left[x=-\frac{b\,e-c\,d}{a\,d-b\,c},\ y=\frac{a\,e-c^2}{a\,d-b\,c}\right]\right\}$$

You often want to solve an equation or system of equations, and then substitute that solution into a third equation. You can use the `subs` function to do this:

```
eq3 := x^3-y
```
$$x^3 - y$$

```
subs(eq3,sol[1])
```
$$-\frac{a\,e-c^2}{a\,d-b\,c}-\frac{(b\,e-c\,d)^3}{(a\,d-b\,c)^3}$$

```
subs(eq3,sol[1][1])
```
$$-y-\frac{(b\,e-c\,d)^3}{(a\,d-b\,c)^3}$$

```
subs(eq3,sol[1][2])
```
$$x^3-\frac{a\,e-c^2}{a\,d-b\,c}$$

In the first example, `sol[1]` extracts the first (and only) element from the set. Both the solutions for `x` and `y` are extracted and substituted into `eq3`. In the second example, `sol[1][1]` substitutes only `x`. In the third, `sol[1][2]` substitutes only `y`.

(4) Of course not all equations can be solved exactly.

```
solve(cosh(x)=2*x,x)
```
$$\text{solve}(\cosh(x)-2\,x=0,\,x)$$

But you can get an approximate, and the numerical solution is:

```
numeric::solve(cosh(x)=2*x,x)
```
$$\{0.5893877635\}$$

```
eqs := [x^2 = sin(y), y^2 = cos(x)]:
solution := numeric::fsolve(eqs, [x, y])
```
$$[x=-0.8517004887,\,y=0.8116062151]$$

4. Calculus

(1) First, try:

```
diff(x^2,x)
```
$$2\,x$$

```
int(%,x)
```
$$x^2$$

```
diff(x*sin(x)^2*cos(x)^2,x)
```
$$2\,x\cos(x)^3\sin(x)+\cos(x)^2\sin(x)^2-2\,x\cos(x)\sin(x)^3$$

```
simplify(%)
```
$$\frac{x\sin(4\,x)}{2}-\frac{\cos(4\,x)}{8}+\frac{1}{8}$$

```
int(%,x)
```
$$\frac{x\sin(2\,x)^2}{4}$$

```
expand(%)
```
$$x\cos(x)^2\sin(x)^2$$

(2) MuPAD can do partial derivatives as well:

```
f:=sqrt(x^2+y^2+z^2)
```

$$\sqrt{x^2+y^2+z^2}$$

```
diff(f,x)
```

$$\frac{x}{\sqrt{x^2+y^2+z^2}}$$

(3) It can also do definite integrals:

```
assume(`&sigma;`>0)
```

```
f := exp(-x^2/`&sigma;`^2)
```

$$e^{-\frac{x^2}{\sigma^2}}$$

```
int(f,x=-infinity..infinity)
```

$$\sigma\sqrt{\pi}$$

```
f:= x*log(x)
```

$$x\ln(x)$$

```
int(f,x=a..b)
```

$$\frac{b^2\left(\ln(b)-\frac{1}{2}\right)}{2}-\frac{a^2\left(\ln(a)-\frac{1}{2}\right)}{2}$$

```
simplify(%)
```

$$\frac{a^2}{4}-\frac{b^2}{4}-\frac{a^2\ln(a)}{2}+\frac{b^2\ln(b)}{2}$$

(4) Of course not all integrals can be evaluated. But you can get an approximate of the integral.

```
f := sin(sin(x))
```

$$\sin(\sin(x))$$

```
int(f,x=0..PI/2)
```

$$\int_0^{\frac{\pi}{2}}\sin(\sin(x))\,dx$$

```
float(%)
```

$$0.893243741$$

(5) Another very useful application is to take limits and Taylor series expansions of functions:

```
limit(sin(x)/x, x=0)
```
$$1$$

```
taylor(sin(x)/x,x)
```
$$1-\frac{x^2}{6}+\frac{x^4}{120}+O(x^6)$$

```
expr(%)
```
$$\frac{x^4}{120}-\frac{x^2}{6}+1$$

```
taylor(cos(x)/(1-x),x=0,8)
```
$$1+x+\frac{x^2}{2}+\frac{x^3}{2}+\frac{13\,x^4}{24}+\frac{13\,x^5}{24}+\frac{389\,x^6}{720}+\frac{389\,x^7}{720}+O(x^8)$$

Here, the `expr(%)` gets rid of the funny O(x6) that denotes how many terms were included in the series – this can be useful if you want to substitute the Taylor expansion into another equation later.

(6) You can use MuPAD to do the usual calculus to maximize or minimize a function. For example, let's try to find the maximum value of $f(x) = x^4\,e^{-x}$:

To do this by hand, you would find the values of x that make f stationary, i.e., solve $\mathrm{df}/\mathrm{dx} = 0$ and then substitute the solution back into the function. Here is the Mupad version of this:

```
reset():
f := x^4*exp(-x):
xatmax := solve(diff(f,x)=0,x)
```
$$\{0, 4\}$$

```
subs(f,x=xatmax[2])
```
$$256\,e^{-4}$$

5. Solving differential equations

(1) MuPAD can also solve differential equations - both analytically and numerically. For example, let's solve the differential equation as follows:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = 10\,y + \sin(t)$$

given that $y = 0$ at time $t = 0$.

```
diff_equation := diff(y(t),t) = -10*y(t)+sin(t)
```
$$\frac{\partial}{\partial t}\,y(t) = \sin(t) - 10\,y(t)$$

```
init_condition := y(0)=0
```
$$y(0) = 0$$

```
solve(ode({diff_equation, init_condition}, y(t) ))
```
$$\left\{\frac{e^{-10\,t}}{101}-\frac{\cos(t)}{101}+\frac{10\,\sin(t)}{101}\right\}$$

Notice that MuPAD gives a formula for the solution.

(2) Another example: This is the differential equation governing the free vibration of a damped spring-mass system:

$$\frac{\mathrm{d}^2y}{\mathrm{d}t^2} + 2\zeta\omega\frac{\mathrm{d}y}{\mathrm{d}t} + \omega^2\,y(t) = 0$$

```
diff_equation := y''(t) + 2*`&zeta;`*`&omega;`*y'(t) + `&omega;`^2*y(t)
```

$$y''(t) + \zeta\, y'(t)\, \omega\, 2 + y(t)\, \omega^2 = 0$$

```
init_condition := y(0)=y0, y'(0)=v0
```

$$y(0) = y0,\ y'(0) = v0$$

```
solve(ode({diff_equation, init_condition}, y(t) ),IgnoreSpecialCases)
```

$$\left\{ \frac{e^{t\left(\omega\,\sigma_1 - \omega\,\zeta\right)}\left(v0 + \omega\,\zeta\,y0 + \omega\,y0\,\sigma_1\right)}{2\,\omega\,\sigma_1} - \frac{e^{-t\left(\omega\,\sigma_1 + \omega\,\zeta\right)}\left(v0 + \omega\,\zeta\,y0 - \omega\,y0\,\sigma_1\right)}{2\,\omega\,\sigma_1} \right\}$$

where

$$\sigma_1 = \sqrt{(\zeta - 1)\,(\zeta + 1)}$$

```
simplify(%)
```

$$\left\{ \frac{e^{-\omega\,t\left(\zeta - \sigma_1\right)}\left(v0 + \omega\,\zeta\,y0 + \omega\,y0\,\sigma_1\right)}{2\,\omega\,\sigma_1} - \frac{e^{-\omega\,t\left(\zeta + \sigma_1\right)}\left(v0 + \omega\,\zeta\,y0 - \omega\,y0\,\sigma_1\right)}{2\,\omega\,\sigma_1} \right\}$$

where

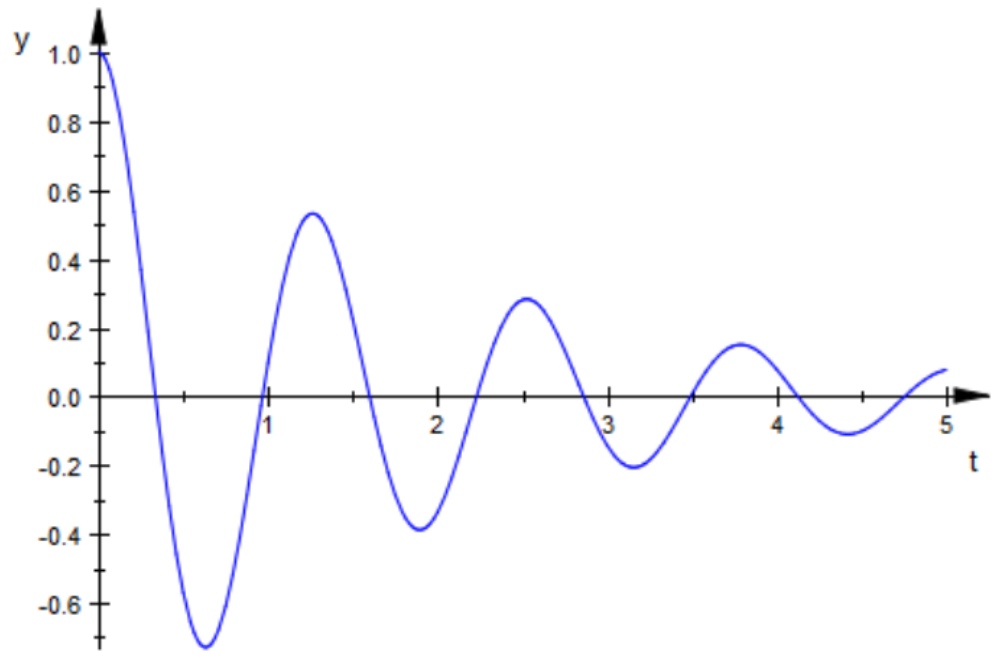
$$\sigma_1 = \sqrt{\zeta^2 - 1}$$

(3) Again, MuPAD gives an exact solution, but it is usually not easy to realize the solution. If we substitute numbers we can plot it as:

```
subs(%, `&omega;`=5, `&zeta;`=0.1, v0=0, y0=1)
```

$$\left\{ e^{t\,(-0.5 + 4.974937186\,i)}\,(0.5 - 0.05025189076\,i) + e^{t\,(-0.5 - 4.974937186\,i)}\,(0.5 + 0.05025189076\,i) \right\}$$

```
plot(%,t=0..5)
```



(4) Of course, not all differential equations can be solved exactly.

```
diff_equation := y(t)*y''(t) + 2*`&zeta;`*`&omega;`*y'(t) + `&omega;`^2*y(t)=0
```
$$y''(t)\,y(t)+\zeta\,y'(t)\,\omega\,2+y(t)\,\omega^2=0$$

```
init_condition := y(0)=y0, y'(0)=v0
```
$$y(0)=y0,\ y'(0)=v0$$

```
solve(ode({diff_equation, init_condition}, y(t) ),IgnoreSpecialCases)
```
$$\text{solve}(\text{ode}(\{y(0)=y0,\ y'(0)=v0,\ y''(t)\,y(t)+\zeta\,y'(t)\,\omega\,2+y(t)\,\omega^2\},\ y(t)),\ \text{IgnoreSpecialCases})$$

Note that MuPAD now gives no solution. It is possible for MuPAD to compute a numerical approximation, but in most cases it is more straightforward to use MATLAB if an analytical solution cannot be found in MuPAD.

6. Vectors and matrices

(1) We will take a look at MuPAD's functions that deal with vectors and matrices. Here are some basic vector and matrix manipulations.

```
reset
```
reset

```
vcol := matrix([x,y,z])
```
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

```
vrow := matrix([[w,s,t]])
```
$$\begin{pmatrix} w & s & t \end{pmatrix}$$

```
Mx := matrix([[a,b,c],[d,e,f],[g,h,i]])
```
$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

```
Mx*vcol
```
$$\begin{pmatrix} a\,x+b\,y+c\,z \\ d\,x+e\,y+f\,z \\ g\,x+h\,y+i\,z \end{pmatrix}$$

```
vrow*%
```
$$(w\,(a\,x+b\,y+c\,z)+s\,(d\,x+e\,y+f\,z)+t\,(g\,x+h\,y+i\,z))$$

```
linalg::scalarProduct(vrow,vcol,Real)
```
$$s\,y+t\,z+w\,x$$

```
linalg::scalarProduct(vrow,vrow)
```
$$s\,\overline{s}+t\,\overline{t}+w\,\overline{w}$$

Here, we do a dot product (scalar product) of two vectors in two different ways. Note that by default MuPAD assumes that all variables could be complex numbers - hence the complex conjugates unless you specify `Real` in the `ScalarProduct` function.

(2) Cross products:

```
linalg::crossProduct(vrow,vcol)
```
$$(s\,z-t\,y\quad t\,x-w\,z\quad w\,y-s\,x)$$

(3) Vector calculus: Curl and divergence

```
vfun := matrix([vx(x,y,z),vy(x,y,z),vz(x,y,z)])
```

$$\begin{pmatrix} vx(x, y, z) \\ vy(x, y, z) \\ vz(x, y, z) \end{pmatrix}$$

```
divergence(vfun,[x,y,z])
```

$$\frac{\partial}{\partial x} vx(x, y, z) + \frac{\partial}{\partial y} vy(x, y, z) + \frac{\partial}{\partial z} vz(x, y, z)$$

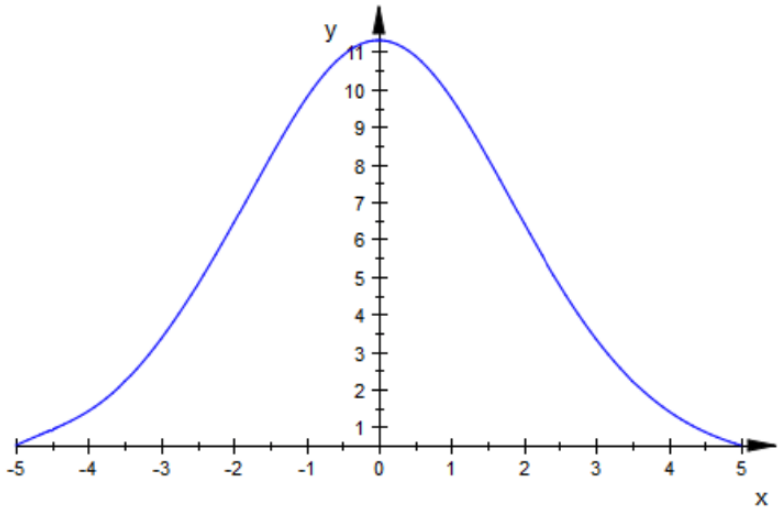
```
curl(vfun,[x,y,z])
```

$$\begin{pmatrix} \frac{\partial}{\partial y} vz(x, y, z) - \frac{\partial}{\partial z} vy(x, y, z) \\ \frac{\partial}{\partial z} vx(x, y, z) - \frac{\partial}{\partial x} vz(x, y, z) \\ \frac{\partial}{\partial x} vy(x, y, z) - \frac{\partial}{\partial y} vx(x, y, z) \end{pmatrix}$$

7. Visualization

(1) MuPAD is also good at plotting and graphics. For a basic plot, try:

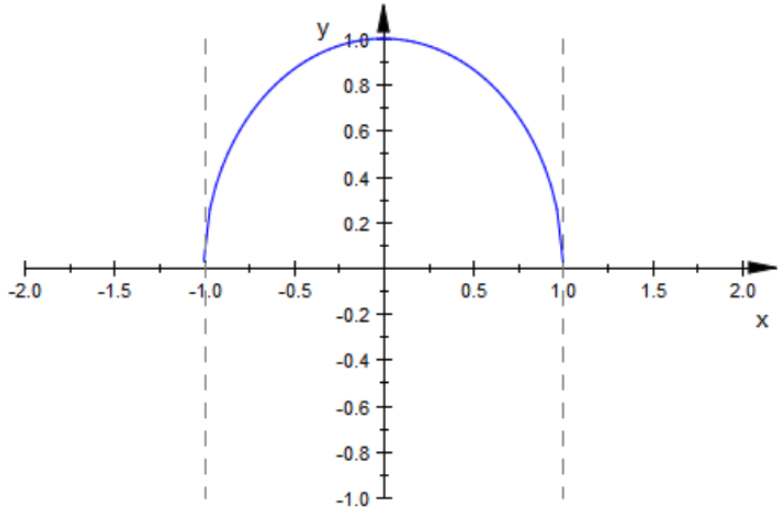
```
plot(besselI(x,4))
```



Note that the range of the plot can be controlled by `plot(jacobiZeta(x,1),x=-3..3)`

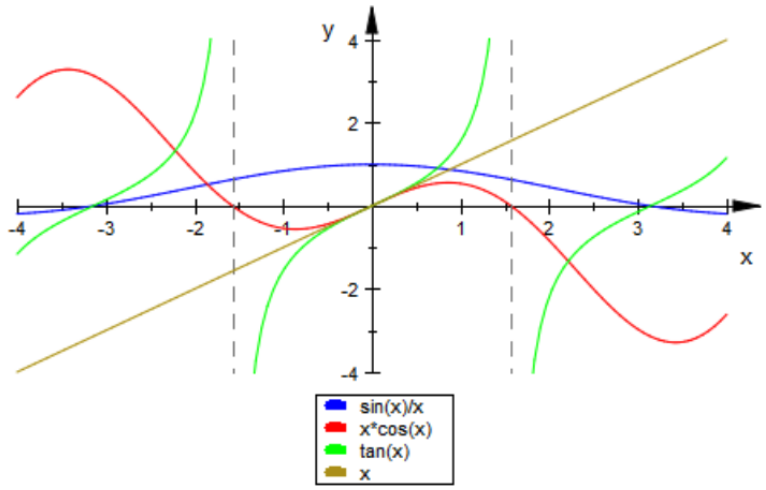
(2) The `plotfunc2d` command does the same thing as plot but has more options to control the appearance of the plot:

```
plotfunc2d(sqrt(1-x^2),x=-2..2,YRange=-1..1)
```



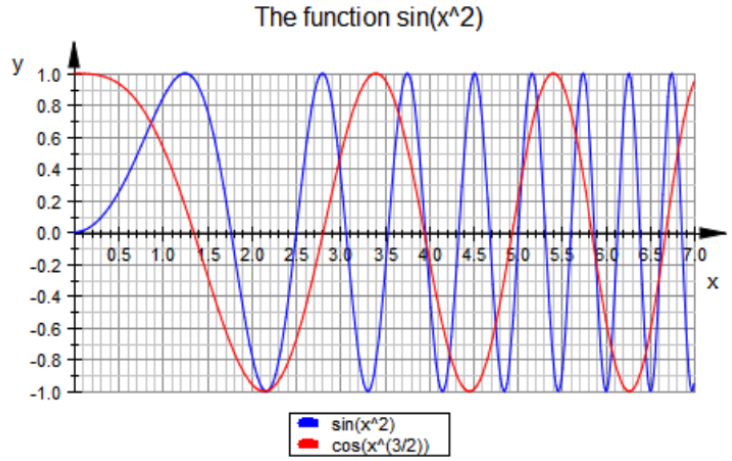
(3) You can display multiple plots on the same axes:

```
plotfunc2d(sin(x)/x, x*cos(x), tan(x),x, x = -4..4)
```



(4) You may make very fancy looking plots as:

```
plotfunc2d(sin(x^2),cos(x^(3/2)), x = 0..7,
           Header = "The function sin(x^2)",
           XTicksDistance = 0.5, YTicksDistance = 0.2,
           XTicksBetween = 4, YTicksBetween = 1,
           GridVisible = TRUE, SubgridVisible = TRUE):
```

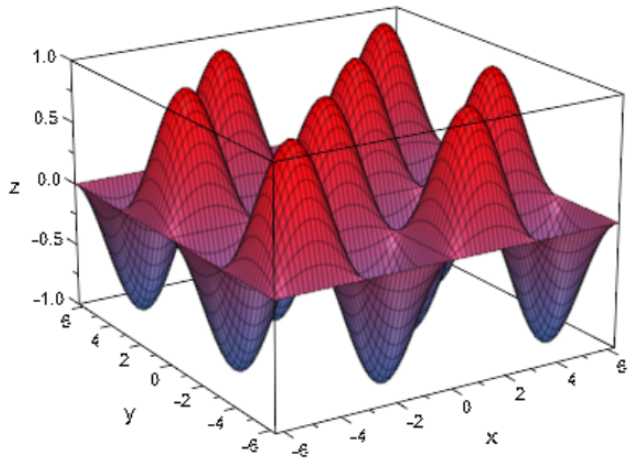


(5) You can do pretty 3D plots as well:

```
plot::Function3d(sin(x)*sin(y), x=-2*PI..2*PI, y=-2*PI..2*PI, Mesh=[75,75])
```

plot::Function3d(sin(x) sin(y), x = −2 π..2 π, y = −2 π..2 π)

```
display(%)
```



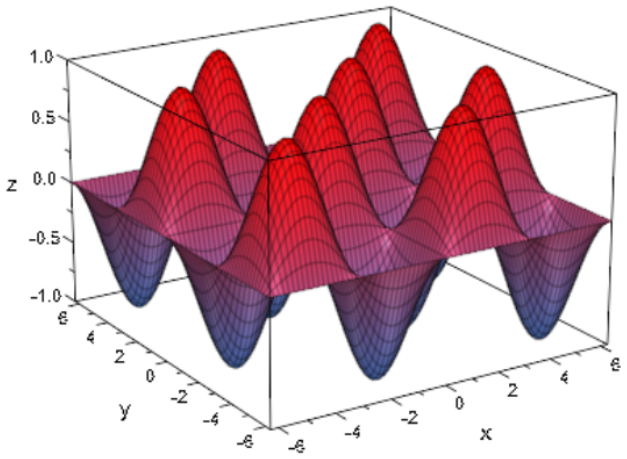
Click on the plot, and then try experimenting with some of the buttons on the toolbar window – you can rotate the plot around, zoom in, and so on.

(6) You can make animations as well, by adding a third parameter to a 3D plot (a in the example below). To play the animation, click on the picture, then press the big blue right pointing arrow.

```
plot::Function3d(sin(x-a)*sin(y-a),x=-2*PI..2*PI,y=-2*PI..2*PI, a=0..6*PI, Mesh=[75,75])
```

plot::Function3d(sin(a − x) sin(a − y), x = −2 π..2 π, y = −2 π..2 π)

```
display(%)
```

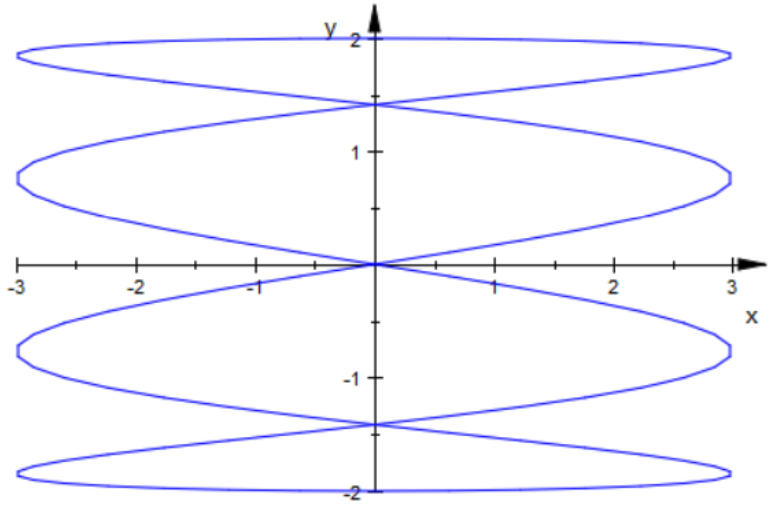


(7) MuPAD can do parametric plots as well, in both 2D and 3D:

```
plot::Curve2d([3*sin(4*q),2*cos(q)], q=0..2*PI)
```

plot::Curve2d([3 sin(4 q), 2 cos(q)], q = 0..2 π)

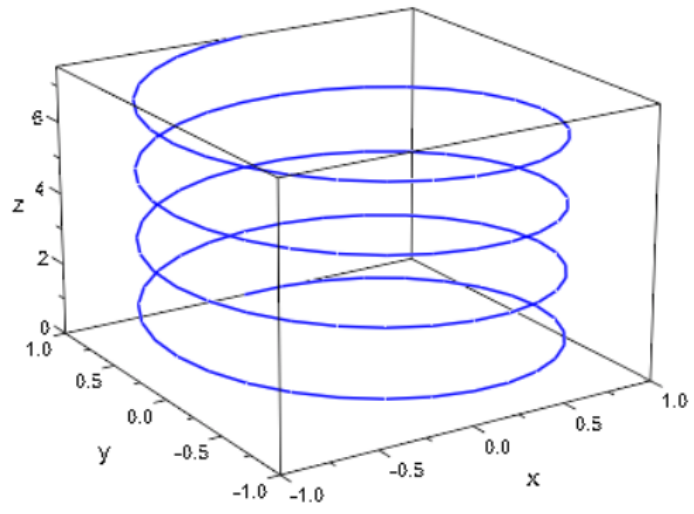
```
display(%)
```




```
plot::Curve3d([sin(t),cos(t),0.3*t], t=0..a, a=0..8*PI)
```

plot::Curve3d([sin(t), cos(t), 0.3 t], t = 0..a)

```
display(%)
```

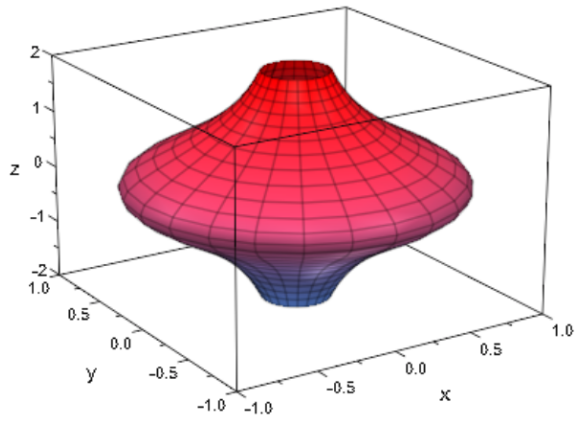


(8) You can plot 3D surfaces as well:

```
plot::Surface([sin(t)/(1+u^2),cos(t)/(1+u^2),u], t=0..2*PI,u=-2..2)
```

$$\text{plot::Surface}\left(\left[\frac{\sin(t)}{u^2+1}, \frac{\cos(t)}{u^2+1}, u\right], t = 0..2\,\pi, u = -2..2\right)$$

```
display(%)
```



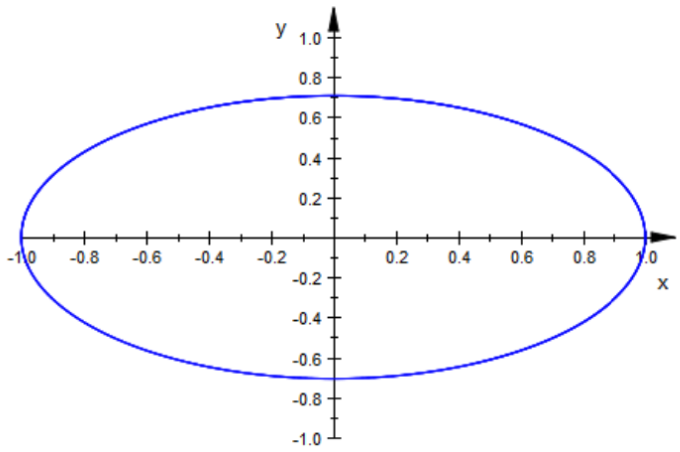
(9) The `implicitplot` is another very useful function. In 2D, it will plot a line or curve that satisfies an equation. In 3D, it will plot a plane or surface that satisfies a 3D equation. Here are two simple examples:

a.

```
plot::Implicit2d(x^2+2*y^2=1, x=-1..1, y=-1..1)
```

$$\text{plot::Implicit2d}(x^2+2\,y^2-1, x = -1..1, y = -1..1)$$

```
display(%)
```

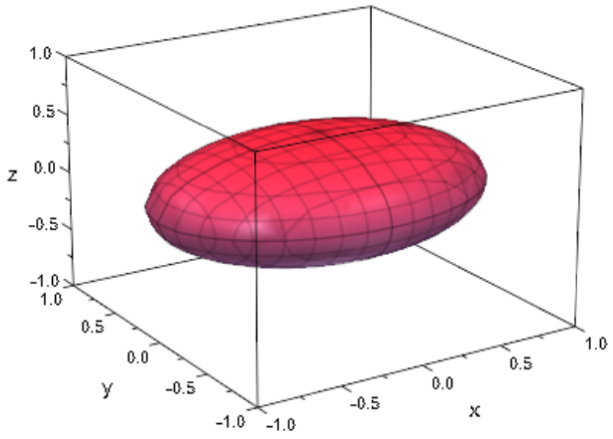


b.

```
plot::Implicit3d(x^2+2*y^2+4*z^2=1, x=-1..1, y=-1..1, z=-1..1)
```

$$\text{plot::Implicit3d}(x^2+2\,y^2+4\,z^2-1, x = -1..1, y = -1..1, z = -1..1)$$

```
display(%)
```



7.2 Alternative to perform finite element assembly in programming

1. General concepts

(1) As we all know, finite element matrices are *sparse* and many memory can be saved if their sparsity is exploited.

(2) Consequently, we are often provoked to declare the $n \times n$ stiffness matrix as a sparse one just before the element loop. However,

be careful of this method since the larger the matrix the slower the assembly operations.

(3) There is a better and more efficient way to code the assembly procedure in MATLAB. For the sake of simplicity, the stiffness matrix of the two-dimensional problem on the domain $(0,1) \times (0,1)$ is considered in this Section. A uniform mesh of $m \times m$ four-node linear quadrilateral elements is used, where m is the number of subdivisions per side.

2. Algorithms

(1) Method 1

In this method, a *sparse matrix* `K` is declared before the assembly loop over finite elements. However, it turns out that this method is the slowest among the three algorithms demonstrated in this Section and will be discussed later, but it can save significant amount of memory.

(2) Method 2

In this method, a *full matrix* `K` is declared before the assembly loop over finite elements. This scheme is faster than Method 1. However, it will rapidly run out of memory. Because of the later, its applicability is limited to matrix sizes.

(3) Method 3

In this method, three *vector arrays* `I`, `J`, and `X` such that:

```
K(I(a),J(a)) = X(a)
```

are applied, and thus the sparse matrix `K` of size `n x n` is built as:

```
K = sparse(I,J,X,n,n)
```

In the scheme of this method:

a. The `X` values of repeated pairs in `I` and `J` are added each other in MATLAB sparse function, which represents the standard assembly procedure of the finite element method.

b. This method can save as much memory as Method 1 but it is significantly faster.

c. By using MATLAB zeros function, the allocations of `I`, `J`, and `X` are done before the finite element loop. Note that the sizes of these arrays have to be known and declared before such assembly procedure.

d. For instance, consider the case of the two-dimensional problem. Since each node has 2 DOFs, each of the four-node quadrilateral elements contributes with $(4 \times 2) \times (4 \times 2)$ entries to the stiffness matrix. Hence, the sizes of `I`, `J`, and `X` are all 64 times the number of elements.

e. If 100×100 subdivisions (i.e., $m = 100$ and 101×101 nodes) are specified, that is 100×100 quadrilateral elements, the sizes of `I`, `J`, and `X` are all 640000.

f. In Method 2, since each node has 2 DOFs, the number of total DOFs in the system is $101 \times 101 \times 2$. Hence, the size of the *full matrix* `K` is $(101 \times 101 \times 2) \times (101 \times 101 \times 2)$, and total number of the entries in `K` is thus 416241604.

g. In Method 3, for `I`, `J`, and `X`, the total size that one needs to allocate for is then 1920000, which represents about only 0.46% of the amount needed to allocate `K` in Method 2.

h. Moreover, after the construction of `K` with `sparse(I,J,X,n,n)`, the arrays `I`, `J`, and `X` can be deleted as they are not further needed, which results in additional memory saving.

3. Discussions

(1) The assembly computations can significantly be improved if MATLAB sparse function is used appropriately.

(2) Method 1 is not suitable among the three methods discussed in this Section, since in the finite element loop, there are as many insertion operations as the number of `K(i,j)` evaluations. In other words, if a new non-zero entry is to be introduced in the sparse structure of `K`, then MATLAB kernel has to first make room for such non-zero numbers, and then insert those values into `K`.

(3) The computing time of an insertion operation into an array is proportional to the number of non-zero data in the array. Therefore, in the case discussed above, the assembly loop over `K` takes computing time proportional to the number of non-zero entries in the sparse matrix.

(4) Method 2 is the one that many of us would normally apply to perform the matrix assembly in a finite element code. Considering computing time, Method 2 is preferable over Method 1, but it has limited applicability due to matrix sizes of large-scale problems.

(5) Finally, Method 3 is the most efficinet way to perform matrix assembly considering both computation time and memory. Therefore, Method 3 is recommended, and Method 1 should be avoided among the three methods introduced in this Section.
