my Maple cheat sheet

Nasser M. Abbasi

October 16, 2025 Compiled on October 16, 2025 at 8:29pm [public]

Contents

1 Pattern matching **7**
1.1 Detect reduced Riccati ode $y' = ax^n + by^2$ 7
1.2 Detect general Riccati ode $y' = f_0(x) + f_1(x)y(x) + f_2(x)y^2$ 10
1.3 Detect ode of form $y' = (a + bx + cy(x))^n$ 14
1.4 Detect first order linear ode of form $A(t)y'(t) + B(t)y(t) = f(t)$ 16
1.5 Detect second order linear ode of form $A(t)y''(t) + B(t)y'(t) + C(t)y(t) = f(t)$ 19
1.6 pattern example 1 23
1.7 pattern example 2 24
1.8 pattern example 3 24
1.9 pattern example 4 25
1.10 pattern example 5 26
1.11 pattern example 6 27
1.12 pattern example 7 27
1.13 pattern example 8 28
1.14 pattern example 9 28
1.15 pattern example 10 29
1.16 pattern example 11 30
1.17 pattern example 12 30
1.18 pattern example 13 31
1.19 pattern example 14 31
1.20 pattern example 15 32
1.21 pattern example 16 33
1.22 pattern example 17 34
1.23 pattern example 18 35
1.24 pattern example 19 35

2 OOP in Maple **36**
2.1 How to use new object method calling in Maple 2021? 36

2.2 How to make a constructor for an Object? . . . 37
2.3 How to make different constructors for an Object? . . . 38
2.4 How to do OOP inheritance? . . . 40
2.5 How to extend a base class and override its method with different one? . . . 42
2.6 How to extend a class and call base class function from the extended class?? 43
2.7 How to use object as user defined record inside a proc? . . . 44
2.8 How to make copy of list of objects? . . . 45
2.9 How to use OOP to implement ode solver? . . . 45
2.10 How to make a complete OOP ode solver in Maple? . . . 52

3 Differential equations **60**
3.1 How to check if ode is of certain type? . . . 60
3.2 How to force dsolve to use specific method for solving? . . . 60
3.3 How to find a particular solution to ODE? . . . 67
3.4 How to find basis solutions for homogeneous ode? . . . 67
3.5 How to solve a differential equation with initial conditions? . . . 67
3.6 How to verify that the ODE solution given is correct? . . . 68
3.7 How to know the type of ODE? . . . 68
3.8 What packages to load for differential equations? . . . 69
3.9 How to plot solution of differential equations? . . . 69
3.10 On High precision. Using Taylor to solve ODE . . . 70
3.11 Obtain ODE in canonical coordinates (Lie symmetry) . . . 70
3.12 How to use hint with symgen? . . . 72
3.13 How to parse a single ode? . . . 72
3.14 How to check if single ODE is valid? . . . 76
3.15 How to check if an ode has y' in it? . . . 78
3.16 How to check if an ode is linear ode? . . . 79
3.17 How to find the order of an ode? . . . 79
3.18 How to find the coefficients of a linear ode? . . . 79
3.19 How to find the coefficients of any ode, linear or not and any order? . . . 80
3.20 find order and degree of highest derivative . . . 82
3.21 How to move all derivatives to one side in an equation? . . . 85
3.22 How to obtains list of all derivatives in expression? . . . 85
3.23 How to invert roles of dependent variable and independent variable in an ode? 87
3.24 How to find the indicial equation for an ODE? . . . 88
3.25 How to write derivative . . . 88
3.26 How to solve heat PDE in 1D in Maple 2017? . . . 88
3.27 How to make Maple display diff(y(x),x) as y'(x) or as y' ? . . . 90
3.28 How to set boundary conditions for dsolve or pdsolve? . . . 91
3.29 How force dsolve to use Lie? . . . 91
3.30 How to do change of variables on the dependent variable for an ODE? . . . 92

3.31 How to do change of variable on the independent variable for an ODE? . . . 92
3.32 ODE change of variable on both dependent and independent variable? . . . 93
3.33 How to make phase plot of first order ODE? . . . 94
3.34 How to make phase plot of second order ODE? . . . 96
3.35 How to move all terms with y to one side? . . . 97
3.36 How to find all constants of integration in an expression? . . . 98
3.37 How to find if an ode is missing the dependent variable? . . . 98
3.38 How to convert Riccati general ode to second order ode? . . . 99
3.39 How to reverse the role of the dependent and independent variables in ode? . 100
3.40 function to solve reduced Riccati ode in Maple? . . . 101
3.41 How to guess a solution to an ode? . . . 103

4 Symbolic manipulation, expression filtering, structured types 105
4.1 on maple data types . . . 105
4.2 how to extract elements from a list based on some selection? . . . 105
4.3 how to test if all elements of a matrix are integers? . . . 105
4.4 How to select terms with sqrt or radical . . . 106
4.5 How to find all csgn() and replace them by 1 . . . 106
4.6 How to find symbols inside csgn() in an expression? . . . 107
4.7 How to replace all abs(expr) by expr . . . 107
4.8 How to obtain list of all occurrences of some function in an expression? . . . 108
4.9 How to replace $\ln(|x|)$ with $\ln(x)$ in an expression? . . . 109
4.10 How to find all signum functions in expression and simplify it? . . . 109
4.11 How to replace all signum functions in expression by 1? . . . 110
4.12 How to find if some function is present in an expression . . . 110
4.13 How to find all functions in an expression? . . . 111
4.14 find all functions except builtin math functions . . . 111
4.15 How to obtain a list of all arguments of a single function? . . . 111
4.16 How to obtain a list of all arguments of a single function in an expressions? . 112
4.17 How to obtain a list of all arguments of the D function in an expressions? . . 112
4.18 Find functions whose first argument is z . . . 113
4.19 Find functions whose second argument is t? . . . 113
4.20 How to use select with own type to find sub expressions? . . . 113
4.21 How to write structured types to match some expressions? . . . 114
4.21.1 type for $\sin^m(x)\cos^n(x)$ . . . 114
4.22 select only indexed variables from an expression . . . 115
4.23 Given an expression, how to find all variables and functions in it? . . . 115
4.24 How to check if an expression is integer, when it has symbols in it? . . . 116
4.25 How to find parameters such as π in an expression? . . . 116
4.26 How to find all derivatives $y'(x)$ in an expression? . . . 116
4.27 How combine log terms? . . . 118

4.28 Find position in a list of items that are not numeric . . . 119
4.29 How to change arctan(y,x) to arctan $\left(\frac{y}{x}\right)$ in an expression . . . 119
4.30 How to find all symbols that represent variables in an expression? . . . 120
4.31 How to change first argument of function? . . . 120
4.32 How to change last argument of function? . . . 121
4.33 How to remove last argument of function? . . . 121
4.34 How to find a pattern inside an expression . . . 122
4.35 How to find parts of a Sum? . . . 122
4.36 find if sequence or list is inside another and the indices . . . 123
4.37 find if some type inside some expression and its location . . . 124
4.38 How to check that all entries in a list have the same type? . . . 124
4.39 How to replace generic function inside derivative? . . . 125
4.40 Examples how to match types . . . 126
4.41 On the order of terms when using indents . . . 129

5 General FAQ **130**
5.1 How to convert Mathematica expression to Maple? . . . 130
5.2 How to debug and stopat internal procedures, such as dsolve? . . . 131
5.3 How to display source code of a function? . . . 132
5.4 How to display trace of a function as it runs in maple? . . . 134
5.5 How to display a build in function code? . . . 135
5.6 How to build a LIST or a SET on the fly? . . . 136
5.7 make function display more information . . . 138
5.8 How to plot a function? . . . 139
5.9 How to run maple from command line? . . . 139
5.10 How to use matrices in maple? . . . 140
5.11 return more than value from a procedure . . . 142
5.12 How does maple handle procedure arguments? . . . 142
5.13 How to define your own data types? . . . 143
5.14 find max element and position in matrix . . . 143
5.15 How to create a package? . . . 144
5.16 How to convert from floating point to Hex? . . . 146
5.17 How to find Taylor series expansion of functions? . . . 146
5.18 How to print elements of a matrix? . . . 147
5.19 How to find determinant of matrix? . . . 147
5.20 How to generate Hilbert matrix? . . . 147
5.21 How to plot matrix data? . . . 148
5.22 How to catch an error from a proc()? . . . 148
5.23 How to convert 3456 to 3,456 ? . . . 148
5.24 How to use units ? . . . 149
5.25 How to evaluate Catalan number and other sums? . . . 149

5.26 write a text file that contains a package, and load it 150
5.27 How to find what packages are included in maple 151
5.28 How to plot the gradient vector field? 151
5.29 How to put the digits of Pi into a list? 151
5.30 Digits of PI in maple and mma 152
5.31 How to find where functions are? 154
5.32 how to use Laplace transform? 155
5.33 questions I have 156
5.34 3D plotting 156
5.35 How to raise each element in a list to a power? 157
5.36 How to generate a sequence with any increment? 157
5.37 What shortcuts are there for matrix manipulation? 157
5.38 How to solve a set of equations for the derivative? 157
5.39 How to solve a set of equations for differentials? 158
5.40 How to plot binary tree 159
5.41 Problem 12.4 chapter 4, Boas book 160
5.42 example of doing convergence test in maple 160
5.43 Problem ch 14, 3.18, Boas book. contour integration 161
5.44 How to find multiple roots to an equation such as $\sin(x) = 0$ 161
5.45 Dr Basti Associated Legendre 161
5.46 Understanding conformal mapping in maple 164
5.47 Hide tilde character when using assumption 167
5.48 Fourier series in maple 167
5.49 How to plot graphs next to each others in a grid like fashion 169
5.50 How to generate Pi on X-axis 170
5.51 How to make output from FunctionAdvisor look better? 170
5.52 How to do partial fractions? 171
5.53 How to generate sequence sum symbolically 171
5.54 Nice plot from Maple 171
5.55 How to check if 2 expressions are the same? 172
5.56 converting series to factorials 172
5.57 How to find what new additions made to Maple? 173
5.58 Maple can't solve laplace equation and numerically 173
5.59 Some Maple Matrix operations 173
5.60 How set diagonal elements to some value, say 1? 176
5.61 How to multiply roots of a polynomial? 176
5.62 How to plot a surface in 3D? 177
5.63 How to convert trig to sinc function in an expression 178
5.64 How to find NullSpace and ColumnSpace of a matrix? 178
5.65 How to fix the interface to using Maple notation for input? 179
5.66 How to find all solutions using allvalues ? 179

5.67 Adding only to diagonal of a matrix . . . 180
5.68 How to search help for updates on some package . . . 180
5.69 How to work with groups in worksheet . . . 180
5.70 How to read code into worksheet? . . . 180
5.71 Code editors for Maple . . . 180
5.72 How to find if package is module or table? . . . 181
5.73 How to replace a string? . . . 181
5.74 How to use geometry and plottools ? . . . 181
5.75 How to simplify log expressions ? . . . 181
5.76 How to simplify hyperbolic expression ? . . . 182
5.77 How to create text file and append string to it? . . . 182
5.78 How to search packages and libraries? . . . 183
5.79 How to numerically solve a BVP ode and plot the solution? . . . 185
5.80 How to display on screen for specific width? . . . 186
5.81 Maple IDE links . . . 187
5.82 loading, remove and finding what packages loaded . . . 187
5.83 some rules of thumbs when using Maple . . . 187
5.84 How to make multiple assumptions on a symbol? . . . 187
5.85 How to check if expression is an equation? . . . 188
5.86 How to check if expression is a set? . . . 188
5.87 How to export a plot to PDF? . . . 188
5.88 How to find all roots of complex number . . . 189
5.89 How to convert matrix of matrices to a matrix? . . . 189
5.90 How to do pattern matching in Maple? . . . 190
5.91 Example 1 . . . 190
5.92 Example 2 . . . 191
5.93 How to find trig identities? . . . 191
5.94 How to find directional derivative of scalar function? . . . 192
5.95 How to check if name is assigned a value? . . . 192
5.96 How to simplify $e^{\ln(x)+\ln(y)}$ . . . 193
5.97 Basis for Null space, Row space and column space of matrix . . . 194
5.98 How to do Gaussian elimination on a Matrix? . . . 195
5.99 How to find Reduced Echelon form of a Matrix? . . . 195
5.100 How add a new row to bottom of matrix? . . . 196
5.101 How to find the cofactor matrix of a matrix? . . . 197
5.102 How to normalize eigenvectors? . . . 197
5.103 How to typeset $\hbar$? . . . 198
5.104 How to find the Curl of a vector? . . . 198
5.105 See all steps in RREF of an augmented matrix . . . 199
5.106 How to find column space of matrix? . . . 200
5.107 Show step by step. Calculus problem and differential equations. . . . 200

5.108 How to obtain list of files with some extension in folder? 201
5.109 How to delete lines from text file that contains some string? 201
5.110 How to truncate a polynomial? 202
5.111 How to make a local declare like block inside a proc? 202
5.112 Using short name for a proc inside nested modules instead of long name . . 203
5.113 Remove duplicates objects in a list based on condition on a field 204
5.114 How to remove duplicates Vectors from a list? 205
5.115 How to find all poles and their order of a rational function? 206
5.116 find series of function with specific number of terms 207
5.117 How to call sibling's proc without making the sibling module exported? . . . 208
5.118 Convert time to use seconds instead of milliseconds 209
5.119 Change the summation index letter 210
5.120 How to use RootOf to solve an equation? 211

6 General settings issues and interface **211**
6.1 How to change zoom factor for worksheet to non standard value? 211

1 Pattern matching

This section shows some examples using Maple to detect some patterns. In all of these the main maple function used is **patmatch**. Some example also show the Mathematica command to compare results with.

Many of these can be done in Maple without using **patmatch** command, and even using shorter code. But these examples are meant to show how to use **patmatch**.

1.1 Detect reduced Riccati ode $y' = ax^n + by^2$

This detects Reduced Riccati ode $y' = ax^n + by^2$ where a, n, b are scalars and $y(x)$ is the dependent function. The input to the proc is the ode and the dependent function $y(x)$. The parsing function will return a, b, n if successful match is found, or FAIL if ode does not match the expected pattern.

```
reduced_riccati_parse:=proc(ode::`=`,func::function(name),$)
local RHS;
local y:=op(0,func);
local x:=op(1,func);
local la,a,b,n;

   try
      RHS:=timelimit(30,[solve(ode,diff(y(x),x))]);
      if nops(RHS)<>1 then RETURN(FAIL); fi;
```

```
      RHS:=expand(RHS[1]);
   catch:
      RETURN(FAIL);
   end try;

   if patmatch(RHS,a::anything*x^(n::anything)+b::anything*y(x)^2,'la') then
      assign(la);
      if has(a,x) or has(a,y(x)) then RETURN(FAIL); fi;
      if has(n,x) or has(n,y(x)) then RETURN(FAIL); fi;
      if has(b,x) or has(b,y(x)) then RETURN(FAIL); fi;
      RETURN(a,b,n);
   else
      RETURN(FAIL);
   fi;

end proc:
```

Example usage

```
ode:=diff(y(x),x)=x-1+y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);
                              FAIL
                          [_Riccati]

ode:=diff(y(x),x)=x-1+y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);
                              FAIL
                          [_Riccati]

ode:=diff(y(x),x)=-3*x^2+y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                           -3, 1, 2
                    [[_Riccati, _special]]

ode:=diff(y(x),x)=x^(-4)+y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);
```

```
                              1, 1, -4
                [_rational, [_Riccati, _special]]

ode:=diff(y(x),x)=sin(x)*x^(-4)+y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                                FAIL
                              [_Riccati]

ode:=diff(y(x),x$2)=x^(-4)+y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                                FAIL
             [[_2nd_order, _with_linear_symmetries]]

ode:=diff(y(x),x)-x^(-4)-9*y(x)^2=0:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                              1, 9, -4
                [_rational, [_Riccati, _special]]

ode:=diff(y(x),x)=1/x+x*y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                                FAIL
                      [_rational, _Riccati]

ode:=diff(y(x),x)=a*1/x+b*y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                              a, b, -1
                [_rational, [_Riccati, _special]]

ode:=diff(y(x),x)=a*x^n+b*y(x)^2:
reduced_riccati_parse(ode,y(x));
```

```
DEtools:-odeadvisor(ode);
                              a, b, n
                      [[_Riccati, _special]]

ode:=diff(y(x),x)=a*x^n+y(x)+b*y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);
                                 FAIL
                              [_Riccati]

ode:=diff(y(x),x)=y(x)*x^n+b*y(x)^2:
reduced_riccati_parse(ode,y(x));
DEtools:-odeadvisor(ode);
                                 FAIL
                            [_Bernoulli]
```

1.2 Detect general Riccati ode $y' = f_0(x) + f_1(x)y(x) + f_2(x)y^2$

This detects the general Riccati ode $y' = f_0(x) + f_1(x)y(x) + f_2(x)y^2$

The input to the proc is the ode and the dependent function $y(x)$. The parsing function will return f_0, f_1, f_2 if successful match is found, or FAIL if ode does not match the expected pattern.

f_0, f_2 can not be zero (i.e. missing), but f_1 could be missing.

In the case $f_1 = 0$, either f_0 or f_2 or both must be functions of x. Also in the case of $f_1 = 0$, if it is reduced Riccati, then it return FAIL.

In the case f_1 is present and also depends on x, then now f_0, f_2 are both allowed not to be functions of x. But if f_1 present and also does not depend on x, then now at least one of f_0, f_2 must be function of x.

```
general_riccati_parse:=proc(ode::`=`,func::function(name),$)
local RHS;
local y:=op(0,func);
local x:=op(1,func);
local stat;
local la,f0,f1,f2;

   stat:= reduced_riccati_parse(ode,func);
   if stat<>FAIL then #this is reduced riccati, hence not general
```

```
      RETURN(FAIL);
   fi;

   #now check if general riccati
   try
      RHS:=timelimit(30,[solve(ode,diff(y(x),x))]);
      if nops(RHS)<>1 then RETURN(FAIL); fi;
      RHS:=expand(RHS[1]);
      RHS:=collect(RHS,[y(x),y(x)^2]);#must collect to insure the right form
   catch:
      RETURN(FAIL);
   end try;

   if patmatch(RHS,f0::anything+f1::anything*y(x)+f2::anything*y(x)^2,'la') then
      assign(la);
      if f0=0 or f2=0 then RETURN(FAIL); fi;
      if has(f0,y(x)) or has(f1,y(x)) or has(f2,y(x)) then RETURN(FAIL); fi;

      if has(f1,x) then
         RETURN(f0,f1,f2);
      else
         if not has(f0,x) and not has(f2,x) then
            RETURN(FAIL);
         else
            RETURN(f0,f1,f2);
         fi;
      fi;
   else #check for f1 missing
      if patmatch(RHS,f0::anything+f2::anything*y(x)^2,'la') then
         assign(la);
         if has(f0,y(x)) or has(f2,y(x)) then RETURN(FAIL); fi;
         if not has(f0,x) and not has(f2,x) then RETURN(FAIL); fi;
         RETURN(f0,0,f2);
      else
         RETURN(FAIL);
      fi;
   fi;

end proc:
```

Example usage

```
ode:=diff(y(x),x)=f0(x)+f1*y(x)+f2*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                          f0(x), f1, f2
                            [_Riccati]

ode:=diff(y(x),x)=f0+f1(x)*y(x)+f2*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                          f0, f1(x), f2
                            [_Riccati]

ode:=diff(y(x),x)=f0(x)+f1(x)*y(x)+f2*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                         f0(x), f1(x), f2
                            [_Riccati]

ode:=diff(y(x),x)=f0(x)+f1(x)*y(x)+f2(x)*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                       f0(x), f1(x), f2(x)
                            [_Riccati]

ode:=diff(y(x),x)=f0(x)+f2(x)*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                         f0(x), 0, f2(x)
                            [_Riccati]

ode:=diff(y(x),x)=x+f2(x)*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                           x, 0, f2(x)
```

```
                          [_Riccati]

ode:=diff(y(x),x)=x+f2*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                             FAIL
                    [[_Riccati, _special]]

ode:=diff(y(x),x)=1/x+y(x)+y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                            1
                            -, 1, 1
                            x
                    [_rational, _Riccati]

ode:=diff(y(x),x)=1/x+y(x)+x*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                            1
                            -, 1, x
                            x
                    [_rational, _Riccati]

ode:=diff(y(x),x)=x*y(x)+c*y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                             FAIL
                         [_Bernoulli]

ode:=diff(y(x),x)=x-1+y(x)^2:
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

                         -1 + x, 0, 1
                          [_Riccati]
```

```
ode:=diff(y(x),x)=(3+9*x+8*y(x))^(2): #this ode needs the collect
general_riccati_parse(ode,y(x));
DETools:-odeadvisor(ode);

              81*x^2 + 54*x + 9, 144*x + 48, 64
              [[_homogeneous, class C], _Riccati]
```

1.3 Detect ode of form $y' = (a + bx + cy(x))^n$

Where n can not be 1. a can be missing but not b, c. And a, b, c are scalars.

The parsing routing returns either FAIL or a, b, c, b if pattern is matched.

```
homog_c_parse:=proc(ode::`=`,func::function(name),$)

local RHS;
local y:=op(0,func);
local x:=op(1,func);

local la,a,b,c,n,p,the_power;

   try
      RHS:=timelimit(30,[solve(ode,diff(y(x),x))]);
      if nops(RHS)<>1 then RETURN(FAIL); fi;
      RHS:=simplify(RHS[1]);
   catch:
      RETURN(FAIL);
   end try;

   if patmatch(RHS,(p::anything)^(the_power::anything),'la') then
      assign(la);
      if the_power=1 then
         RETURN(FAIL);
      fi;
      if patmatch(p,a::anything+b::anything*x+c::anything*y(x),'la') then
         assign(la);
         if has(a,x) or has(a,y(x)) then RETURN(FAIL); fi;
         if has(b,x) or has(b,y(x)) then RETURN(FAIL); fi;
         if has(c,x) or has(c,y(x)) then RETURN(FAIL); fi;
         RETURN(a,b,c,the_power);
      else
```

```
        if patmatch(p,b::anything*x+c::anything*y(x),'la') then
           assign(la);
           if has(b,x) or has(b,y(x)) then RETURN(FAIL); fi;
           if has(c,x) or has(c,y(x)) then RETURN(FAIL); fi;
           RETURN(0,b,c,the_power);
        else
           RETURN(FAIL);
        fi;
     fi;
  else
     RETURN(FAIL);
  fi;
end proc:
```

Examples usages

```
ode:=diff(y(x),x)=(3+9*x+8*y(x))^(1/2):
homog_c_parse(ode,y(x));
DEtools:-odeadvisor(ode);
                                    1
                          3, 9, 8, -
                                    2
            [[_homogeneous, class C], _dAlembert]

ode:=diff(y(x),x)=(3+9*x+8*y(x))^(2):
homog_c_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                          3, 9, 8, 2
             [[_homogeneous, class C], _Riccati]

ode:=(x+y(x))*diff(y(x),x)=1:
homog_c_parse(ode,y(x));
DEtools:-odeadvisor(ode);
                        0, 1, 1, -1
     [[_homogeneous, class C], [_Abel, 2nd type, class C], _dAlembert]

ode:=diff(y(x),x)=sqrt(x+y(x)+1):
homog_c_parse(ode,y(x));
DEtools:-odeadvisor(ode);
                                    1
```

```
                          1, 1, 1, -
                                   2
            [[_homogeneous, class C], _dAlembert]

ode:=diff(y(x),x)=1/(x+y(x)+1):
homog_c_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                         1, 1, 1, -1
    [[_homogeneous, class C], [_Abel, 2nd type, class C], _dAlembert]

ode:=diff(y(x),x)=1/(x+y(x)+sin(x)):
homog_c_parse(ode,y(x));
DEtools:-odeadvisor(ode);

                             FAIL
                [[_Abel, 2nd type, class C]]

ode:=diff(y(x),x)=(x+y(x)^2)^(2/3):
homog_c_parse(ode,y(x));
DEtools:-odeadvisor(ode);
                             FAIL
                          [y=_G(x,y')]
```

1.4 Detect first order linear ode of form $A(t)y'(t) + B(t)y(t) = f(t)$

And also find the values of A, B, f in the above. This can be used to parse first order ode after checking it is linear ode (linear in y' and y).

The parsing routing returns either FAIL or A, B, f if pattern is matched.

```
restart;
linear_first_order_parse:=proc(ode::`=`,func::function(name),$)
local y:=op(0,func);
local x:=op(1,func);
local LHS;
local la,A,B,F;

   LHS:= lhs(ode)-rhs(ode);
   LHS:= numer(normal(LHS)); #this is needed to normalize the ode
         #for example 1/y' + x = 0 will becomes 1+x y'  = 0, else
```

```
        #the pattern will fail

   #must collect on y and y' to get the pattern to work OK for all cases
   LHS:= collect(LHS,diff(y(x),x));
   LHS:= collect(LHS,y(x));

   if patmatch(LHS,A::anything*diff(y(x),x)+B::anything*y(x)+F::anything,'la') then
      assign(la);
      if has(A,y(x)) then RETURN(FAIL); fi;
      if has(B,y(x)) then RETURN(FAIL); fi;
      if has(F,y(x)) then RETURN(FAIL); fi;
      RETURN(A,B,F);
   else
      if patmatch(LHS,A::anything*diff(y(x),x)+F::anything,'la') then
         assign(la);
         if has(A,y(x)) then RETURN(FAIL); fi;
         B:=0;
         if has(F,y(x)) then RETURN(FAIL); fi;
         RETURN(A,B,F);
      else
         RETURN(FAIL);
      fi;
   fi;
end proc:
```

Notice that we had to check for the general case $Ay' + By + F$ first and if that fails then check for the specific case $Ay' + F$. This is because if y term is missing then Maple will not match the pattern. i.e. it will not give $B = 0$. So have to make separate pattern matching for the case with missing y in the ode.

For the case $Ay' + F$ Maple will match $F = 0$ if this term is missing. So we do not need to make more specialized pattern for just Ay'.

This is typical. We start with the most general pattern and if that fail, we go to more specialized ones.

Examples usages

```
ode:=1/diff(y(t),t)+t=0;
linear_first_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                              t, 0, 1
```

```
                          [_quadrature]


ode:=1/diff(y(t),t)+t=y(t);
linear_first_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                                FAIL
[[_homogeneous, class C], [_Abel, 2nd type, class C], _dAlembert]



ode:=t*diff(y(t),t)+5=0;
linear_first_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                              t, 0, 5

                          [_quadrature]

ode:=t^2*diff(y(t),t)+t*y(t)=sin(t);
linear_first_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);

                           2
                          t , t, -sin(t)

                            [_linear]

ode:=y(t)*diff(y(t),t)+t*y(t)=sin(t);
linear_first_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);

                                FAIL

                    [[_Abel, 2nd type, class A]]

ode:=sin(t)*diff(y(t),t)-99*ln(t)*y(t)=sin(t)+y(t);
linear_first_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);

                   sin(t), -99 ln(t) - 1, -sin(t)
```

```
                              [_linear]

ode:=sin(t)*diff(y(t),t)-99*ln(t)*y(t)=sin(t)+9*diff(y(t),t)/t+ln(t);
linear_first_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);

                          9
                sin(t) - -, -99 ln(t), -ln(t) - sin(t)
                          t

                              [_linear]

ode:=diff(y(t),t)-y(t)^2=sin(t);
linear_first_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);

                                 FAIL

                              [_Riccati]
```

1.5 Detect second order linear ode of form $A(t)y''(t) + B(t)y'(t) + C(t)y(t) = f(t)$

And also find the values of A, B, C, f in the above. This can be used to parse second order ode after checking it is linear ode (linear in y'', y', y).

The parsing routing returns either FAIL or A, B, C, f if pattern is matched.

```
linear_second_order_parse:=proc(ode::`=`,func::function(name),$)
local y:=op(0,func);
local x:=op(1,func);
local LHS;
local la,A,B,C,F;

#   DEBUG();
   LHS:= lhs(ode)-rhs(ode);
   LHS:= numer(normal(LHS)); #this is needed to normalize the ode
         #for example 1/y'' + x = 0 will becomes 1+x y''  = 0, else
         #the pattern will fail
   LHS:= collect(LHS,diff(y(x),x$2));
```

```
   LHS:= collect(LHS,diff(y(x),x));
   LHS:= collect(LHS,y(x));

   if patmatch(LHS,A::anything*diff(y(x),x$2)
                   +B::anything*diff(y(x),x)
                   +C::anything*y(x)+F::anything,'la') then
      assign(la);
      if has(A,y(x)) then RETURN(FAIL); fi;
      if has(B,y(x)) then RETURN(FAIL); fi;
      if has(C,y(x)) then RETURN(FAIL); fi;
      if has(F,y(x)) then RETURN(FAIL); fi;
      RETURN(A,B,C,F);
   elif patmatch(LHS,A::anything*diff(y(x),x$2)
                   +C::anything*y(x)+F::anything,'la') then
         assign(la);
         if has(A,y(x)) then RETURN(FAIL); fi;
         B:=0;
         if has(C,y(x)) then RETURN(FAIL); fi;
         if has(F,y(x)) then RETURN(FAIL); fi;
         RETURN(A,B,C,F);
   elif patmatch(LHS,A::anything*diff(y(x),x$2)
                   +B::anything*diff(y(x),x)+F::anything,'la') then
         assign(la);
         if has(A,y(x)) then RETURN(FAIL); fi;
         if has(B,y(x)) then RETURN(FAIL); fi;
         C:=0;
         if has(F,y(x)) then RETURN(FAIL); fi;
         RETURN(A,B,C,F);
    elif
         patmatch(LHS,A::anything*diff(y(x),x$2)+F::anything,'la') then
         assign(la);
         if has(A,y(x)) then RETURN(FAIL); fi;
         B:=0;
         C:=0;
         if has(F,y(x)) then RETURN(FAIL); fi;
         RETURN(A,B,C,F);
     else
         RETURN(FAIL);
     fi;
end proc:
```

We see we had to check for all possible patterns. $Ay''+By'+Cy+f$ and $Ay''+By'+f$ and $Ay''+Cy+f$ and $y''+f$. This is because the pattern for say By' will not return $B=0$ if the term By' is missing. In this case there are 4 possible pattern to check for for second order linear ode.

It is possible to do this parsing without using pattern ofcourse (see my ode section below, using the command `DEtools:-convertAlg(ode,y(x));` will do it. But the point here is to do this using patterns.

For expression of many terms, many patterns are need to cover all possible inputs. This is because there is no option to tell Maple to match `A::anything*y(x)` and give $A=0$ when the whole term is missing. If there was such an option, it will make life much simpler.

Examples usages

```
ode:=t/diff(y(t),t$2)+t=0:
linear_second_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);

                            t, 0, 0, t
                  [[_2nd_order, _quadrature]]


ode:=t*diff(y(t),t$2)+5=0:
linear_second_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                            t, 0, 0, 5
                  [[_2nd_order, _quadrature]]

ode:=t*diff(y(t),t$2)+diff(y(t),t)+5*t=0:
linear_second_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                           t, 1, 0, 5 t

                   [[_2nd_order, _missing_y]]

ode:=t*diff(y(t),t$2)+99*y(t)=0:
linear_second_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                           t, 0, 99, 0

                      [[_Emden, _Fowler]]
```

```
ode:=t*diff(y(t),t$2)+99*y(t)=sin(t)*y(t)+diff(y(t),t):
linear_second_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                    t, -1, -sin(t) + 99, 0

            [[_2nd_order, _with_linear_symmetries]]

ode:=y(t)*diff(y(t),t$2)+99*y(t)=sin(t)*y(t)+diff(y(t),t):
linear_second_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                              FAIL

             [[_2nd_order, _reducible, _mu_x_y1],

ode:=diff(y(t),t)*diff(y(t),t$2)+99*y(t)=sin(t)*y(t):
linear_second_order_parse(ode,y(t));
DEtools:-odeadvisor(ode);
                              FAIL
                             [NONE]
```

Using Mathematica pattern the above is a little simpler, since there one can use default zero if term is missing, so no need to check for all possible patterns. Here is an example

```
rule = (aa : a_. y''[x] : 0) + (bb : b_. y'[x] : 0) + (cc : c_. y[x] :0) + any_.
          :> {Plus@a, Plus@b, Plus@c, any};

x^2*y''[x] + x*y'[x] + x*y[x] - 3*x /. rule
```

Gives `{x^2, x, x, -3 x}` and if a term is missing, same pattern still works

```
x^2*y''[x] - 3*x /. rule
```

Gives `{x^2, 0, 0, -3 x}` so we did not have to make pattern for possible missing terms. I was not able to do the same using Maple's pattern match as it has no default option. In this example, we see that Mathematica's pattern matching is better. If I find a way to do the same in Maple I will update this.

1.6 pattern example 1

Given list $3, 4, x, x^2, x^3$, find those elements of form x^n where n is anything other than 1.

In Maple

```
L:=[3,4,x,x^2,x^3];
map(X->`if`(patmatch(X,x^n::nonunit(anything)),X,NULL),L)

     [x^2, x^3]
```

The above can also be written in the long form

```
restart;

L:=[3,4,x,x^2,x^3];

map(proc(X)
    if patmatch(X,x^n::nonunit(anything)) then
       X;
    else
      NULL;
    fi;
    end proc,L)


     [x^2, x^3]
```

In Mathematica

```
Cases[{3, 4, x, x^2, x^3}, x^_]

         {x^2, x^3}
```

1.7 pattern example 2

Given list $3, 4, x, x^2, x^3$, find those elements of form x^n where n is now allowed to be 1. In Maple

```
L:=[3,4,x,x^2,x^3];
map(X->`if`(patmatch(X,x^n::anything),X,NULL),L)

     [x, x^2, x^3]
```

In Mathematica

```
Cases[{3, 4, x, x^2, x^3}, x^_.]

         {x,x^2,x^3}
```

1.8 pattern example 3

Given list $3, 4, x, x^2, x^3$, return list of exponents of x, other than 1.

In Maple

```
L:=[3,4,x,x^2,x^3];
f:=proc(X::anything,x::symbol)
   local la,n;
   if patmatch(X,x^n::nonunit(anything),'la') then
       eval(n,la);
    else
       NULL;
    fi;
end proc;

map(X->f(X,x),L)

   [2, 3]
```

Another option is to use inlined if, like this

```
restart;
L:=[3,4,x,x^2,x^3];
map(proc(X) local la,n;
```

```
    if  patmatch(X,x^n::nonunit(anything),'la') then
        eval(n,la);
    else
        NULL;
    fi;
    end proc, L);


    [2, 3]
```

In Mathematica

```
Cases[{3, 4, x, x^2, x^3}, x^n_ -> n]
        {2, 3}
```

1.9 pattern example 4

Gives list `{f(a,a),f(a,b),f(c,d),f(b,b)}` return `{g(a),f(a,b),f(c,d),g(b)}` where only function that takes two arguments which are the same is replaced by $g(x)$ where x here is the argument in the original list.

In Maple

```
restart;
L:={f(a,a),f(a,b),f(c,d),f(b,b)};
map(X->[unassign('x'),`if`(patmatch(X,f(x::anything,x::anything),'la'),[assign(la),g(x)]

                {f(a, b), f(c, d), g(a), g(b)}
```

Or using the long form of map, which is more clear, even though the code is longer

```
restart;
L:={f(a,a),f(a,b),f(c,d),f(b,b)};

map(proc(X) local la;
    if  patmatch(X,f('x'::anything,'x'::anything),'la') then
        g(eval('x',la));
    else
        X;
    fi;
```

```
    end proc, L
);

         {f(a, b), f(c, d), g(a), g(b)}
```

In Mathematica

```
L = {f[a, a], f[a, b], f[c, d], f[b, b]}
L /. f[x_, x_] -> g[x]

       {g[a], f[a, b], f[c, d], g[b]}
```

1.10 pattern example 5

Given `f(a)+f(b)` transform it using pattern `f(x::anything)->x^2` to obtain `a^2+b^2`

In Maple

```
expr:=f(a)+f(b);
map( proc(X) local la,x;
     if patmatch(X,f(x::anything),'la') then
        eval(x,la)^2;
     else
        NULL;
     fi;
     end proc, expr);


           a^2 + b^2
```

In Mathematica

```
f[a] + f[b] /. f[x_] -> x^2

    a^2+b^2
```

1.11 pattern example 6

Given `[1, x, x^2, x^3]` write pattern to change all `x^n::anything` to `r(n)`

In Maple

```
L:=[1, x, x^2, x^3];
map( proc(X) local la,n;
     if patmatch(X,x^n::'nonunit'(anything),'la') then
        r(eval(n,la));
     else
        X;
     fi;
     end proc, L);


         [1, x, r(2), r(3)]
```

In Mathematica

```
{1, x, x^2, x^3} /. x^n_ -> r[n]

   {1, x, r[2], r[3]}
```

1.12 pattern example 7

Given `f(a + b) + f(a + c) + f(b + d)` apply transformation `f(a + x_) + f(c + y_) -> p(x, y)`

Here to do not use map, since we want to apply pattern on the whole expression. In Maple

```
restart;
expr:=f(a+b)+f(a+c)+f(b+d):

if patmatch(expr,f(a+x::anything)+f(c+y::anything)+z::anything,'la') then
   p(eval(x,la),eval(y,la))+eval(z,la);
else
   expr;
fi;

        p(b, a) + f(b + d)
```

In Mathematica

```
f[a+b]+f[a+c]+f[b+d]/. f[a+x_]+f[c+y_]->p[x,y]

     f[b+d]+p[b,a]
```

1.13 pattern example 8

Write pattern to change `f(a^b)` to `p(a^b,b)`

In Maple

```
restart;
expr:=f(a^b);

if patmatch(expr,f( (a::anything)^b::anything ),'la') then
   print(la);
   p( eval(a,la)^eval(b,la) , eval(b,la) );
else
   expr;
fi;


     p(a^b, b)
```

In Mathematica

```
f[a^b] /. f[x : _^n_] -> p[x, n]

     p[a^b,b]
```

1.14 pattern example 9

Write pattern to select from list `[3,-4,5,-2]` only the numbers that are negative.

In Maple, we use the form `patmatch(X, conditional(patttern,condition))`

```
restart;
L:=[3,-4,5,-2];
map( proc(X) local la;
     if patmatch(X, conditional(n::integer,n<0),'la') then
        eval(n,la);
```

```
     else
        NULL;
     fi;
     end proc, L);


    [-4, -2]
```

In Mathematica

```
Cases[{3, -4, 5, -2}, x_ /; x < 0]

    {-4,-2}
```

1.15 pattern example 10

Write pattern to select from list `[z(1, 1), z(-1, 1), z(-2, 2)]` only those with first argument which is negative.

In Maple, we use the form `patmatch(X, conditional(patttern,condition))`

```
restart;
L:=[z(1, 1), z(-1, 1), z(-2, 2)];
map( proc(X) local la,a,b;
     if patmatch(X, conditional(z(a::integer,b::integer),a<0),'la') then
        z(eval(a,la),eval(b,la));
     else
        NULL;
     fi;
     end proc, L
    );


         [z(-1, 1), z(-2, 2)]
```

In Mathematica

```
Cases[{z[1, 1], z[-1, 1], z[-2, 2]}, z[x_ /; x < 0, y_]]

    {z[-1,1],z[-2,2]}
```

1.16 pattern example 11

Write pattern to select from list `1+a,2+a,-3+a]` only those that have negative number added to a and output $p(x)$ where x is that number.

In Maple, we use the form `patmatch(X, conditional(patttern,condition))`

```
L:=[1+a,2+a,-3+a];
map( proc(X) local la,x;
     if patmatch(X, conditional(a+x::integer,x<0),'la') then
        p(eval(x,la));
     else
        X;
     fi;
     end proc, L
    );


       [1 + a, 2 + a, p(-3)]
```

In Mathematica

```
{1+a,2+a,-3+a}/. (x_/;x<0)+a->p[x]

   {1 + a, 2 + a, p[-3]}
```

1.17 pattern example 12

Use pattern to square only numbers in a list.

```
L:=[2.3,4,7/8,a,b];
map( proc(X) local la,x;
     if patmatch(X, x::numeric,'la') then
        eval(x,la)^2;
     else
        X;
     fi;
     end proc, L
    );

       [5.29, 16, 49/64, a, b]
```

In Mathematica

```
{2.3, 4, 7/8, a, b} /. (x_ /; NumberQ[x]) -> x^2

   {5.29,16,49/64,a,b}
```

1.18 pattern example 13

Use pattern to convert `p(4.5) + p(3/2) + p(u)` to `22.5 + p(u)` by squaring and adding arguments of those function which has its argument numeric.

```
p:=x->`if`(x::numeric,x^2,x);
p(4.5) + p(3/2) + p(u);



      22.50000000 + u
```

In Mathematica

```
p[x_?NumberQ] := x^2
p[4.5] + p[3/2] + p[u]

   22.5 + p[u]
```

1.19 pattern example 14

Write pattern to pick all elements from list which are not integers.

```
L:=[a, b, 0, 1, 2, x, y]:
map( proc(X) local la,x;
     if patmatch(X, conditional( x::anything, not _type(x,integer) ),'la') then
        eval(x,la);
     else
        NULL;
     fi;
     end proc, L
    );

     [a, b, x, y]
```

This can also be done without pattern matching like this

```
select(not type,L,integer)

      [a, b, x, y]
```

In Mathematica

```
Cases[{a, b, 0, 1, 2, x, y}, Except[_Integer]]

   {a, b, x, y}
```

1.20 pattern example 15

Replace variable x in expression with a value 1.

```
L:=[x, x^2,y,z, sin(x), exp( tan(x) )]:

f:=proc(X::anything,x::symbol)
    local la,y;

    if patmatch(X, conditional( y::anything, _has(y,x) ),'la') then
        eval(y,la);
        eval(%,x=1);
     else
        X;
     fi;
end proc;

map(X->f(X,x),L);


     [1, 1, y, z, sin(1), exp(tan(1))]
```

Actually, we do not need patmatch for this, but the above is just an exercise. This can be done in one line in Maple as follows

```
L:=[x, x^2,y,z, sin(x), exp( tan(x) )]:
map(X->eval(X,x=1),L)

   [1, 1, y, z, sin(1), exp(tan(1))]
```

In Mathematica

```
L={x, x^2,y,z, Sin[x], Exp[Tan[x]]}
L/.x->1

   {1,1,y,z,Sin[1],Exp[Tan[1]]}
```

1.21 pattern example 16

Replace variable x in expression with a list.

```
L:=[x, x^2,y,z]:

f:=proc(X::anything,x::symbol)
    local la,y;

    if patmatch(X, conditional( y::anything, _has(y,x) ),'la') then
        eval(y,la);
        eval(%,x={a,b});
     else
        X;
     fi;
end proc;

map(X->f(X,x),L);


     [{a, b}, {a, b}^2, y, z]
```

Actually, we do not need patmatch for this, but the above is just an exercise. This can be done in one line in Maple as follows

```
L:=[x, x^2,y,z]:
map(X->eval(X,x={a,b}),L)

   [{a, b}, {a, b}^2, y, z]
```

In Mathematica

```
{x, x^2, y, z} /. x -> {a, b}
```

```
   {{a, b}, {a^2, b^2}, y, z}
```

1.22 pattern example 17

Replace *sin* by *cos*

```
f:=proc(X::anything)
    local la,x;

    if patmatch(X, conditional( x::anything, _has(x,sin) ),'la') then
        eval(x,la);
        eval(%,sin=cos);
     else
        X;
     fi;
end proc;

L:=[sin(x), cos(x)+sin(2*x),tan(x)]:
map(X->f(X), L);
```

```
    [cos(x), cos(x) + cos(2*x), tan(x)]
```

Actually, we do not need patmatch for this, but the above is just an exercise. This can be done in one line in Maple as follows

```
L:=[sin(x), cos(x)+sin(2*x),tan(x)]:
map(X->eval(X,sin=cos), L)
```

```
    [cos(x), cos(x) + cos(2*x), tan(x)]
```

In Mathematica

```
{Sin[x], Cos[x] + Sin[2*x], Tan[x]} /. Sin -> Cos
```

```
   {Cos[x], Cos[x] + Cos[2 x], Tan[x]}
```

1.23 pattern example 18

Use pattern to replace x^n by $f(n)$

```
restart;
expr:=1 + x^2 + x^4;

F:=proc(X::anything,x::symbol)
    local la,y,n;
    if patmatch(X, (y::anything)^(n::'nonunit'(anything)) ,'la') then
        if eval(y,la)=1 then #bug in maple?
           X;
        else
           f(eval(n,la));
        fi;
     else
        X;
     fi;
end proc;

map(X->F(X,x), expr);


    f(4) + f(2) + 1
```

In Mathematica

```
1 + x^2 + x^4 /. x^p_ :> f[p]

   1 + f[2] + f[4]
```

1.24 pattern example 19

replaces a whole expression by default. In Maple, this is easier to do it without pattern, like this

```
restart;
expr:=f(1, 2, z);
e:=map(X->`if`(X::integer,"int",X),[op(expr)]);
f(op(e))
```

```
    f("int", "int", z)
```

In Mathematica, a pattern works easier

```
Replace[f[1, 2, z], _Integer :> "int", {1}]

   f["int", "int", z]
```

The main issue is that it is hard to match a sequence in Maple. For example, in Mathematica this works

```
f[1, 2, z] /. f[x___] :> x

       Sequence[1, 2, z]
```

But in Maple

```
expr:=f(1, 2, z);
patmatch(expr, f(x::anything))

      false
```

So pattern matching does work to match sequence of arguments.

2 OOP in Maple

2.1 How to use new object method calling in Maple 2021?

In Maple 2021, it is now possible to use `object:-method(arg)` notation. This makes is easier to use OOP in maple. To do this, use `_self` as follows

```
restart;

module person()
 option object;
 local name::string:="";
 local age::integer:=0;

 export get_name::static:=proc(_self,$)
   return _self:-name;
```

```
 end proc;

 export set_name::static:=proc(_self,name::string,$)
   _self:-name:=name;
 end proc;

 export get_age::static:=proc(_self,$)
   return _self:-age;
 end proc;

 export set_age::static:=proc(_self,age::integer,$)
   _self:-age:=age;
 end proc;

end module;
```

And now make an object and use it as follows

```
o:=Object(person)
                  o := Object<<1846759887808>>

o:-get_name();
              ""

o:-get_age();
              0

o:-set_name("joe doe");
o:-get_name();
           "joe doe"
```

2.2 How to make a constructor for an Object?

Add `ModuleCopy` proc in the class. This will automatically be called to initialize the object.

Here is an example

```
restart;

module ODE()
 option object;
```

```
 local ode:=NULL;
 local y::symbol;
 local x::symbol;
 local sol;

 export ModuleCopy::static := proc( _self::ODE, proto::ODE, ode, func, $ )
    print("Initilizing object with with args: ", [args]);
    _self:-ode:= ode;
    _self:-y:=op(0,func);
    _self:-x:=op(1,func);
 end proc;

 export dsolve::static:=proc(_self,$)
   _self:-sol := :-dsolve(ode,y(x));
 end proc;

 export get_sol::static:=proc(_self,$)
   return sol;
 end proc;

end module;
```

And now make an object and use it as follows

```
o:=Object(ODE, diff(y(x),x)+y(x)=sin(x), y(x));
o:-dsolve():
o:-get_sol();

        #y(x) = -1/2*cos(x) + 1/2*sin(x) + exp(-x)*_C1
```

So a constructor just makes it easier to initialize the object without having to make a number of `set()` calls to initialize each member data.

2.3 How to make different constructors for an Object?

This is done using `overload` with different `ModuleCopy` proc in the class.

Here is an example. Lets make a constructor that takes an ode and initial conditions, and one that only takes an ode with no initial conditions.

```
restart;
```

```
module ODE()
 option object;
 local ode:=NULL;
 local y::symbol;
 local x::symbol;
 local ic:=NULL;
 local sol;

 export ModuleCopy::static:= overload(
    [
       proc( _self::ODE, proto::ODE, ode, func, $ ) option overload;
          _self:-ode:= ode;
          _self:-y:=op(0,func);
          _self:-x:=op(1,func);
       end proc,

       proc( _self::ODE, proto::ODE, ode, func, ic, $ ) option overload;
          _self:-ode:= ode;
          _self:-y:=op(0,func);
          _self:-x:=op(1,func);
          _self:-ic :=ic;
       end proc
    ]
 );

 export dsolve::static:=proc(_self,$)
    if evalb(ic=NULL) then
       sol := :-dsolve(ode,y(x));
    else
       sol := :-dsolve([ode,ic],y(x));
    fi;
 end proc;

 export get_sol::static:=proc(_self,$)
   return sol;
 end proc;

end module;
```

And now use it as follows

```
o:=Object(ODE, diff(y(x),x)+y(x)=sin(x), y(x), y(0)=0);
o:-dsolve():
o:-get_sol();

         #y(x) = -1/2*cos(x) + 1/2*sin(x) + 1/2*exp(-x)

o:=Object(ODE, diff(y(x),x)+y(x)=sin(x), y(x));
o:-dsolve():
o:-get_sol();

    #y(x) = -1/2*cos(x) + 1/2*sin(x) + exp(-x)*_C1
```

2.4 How to do OOP inheritance?

In the child class you want to extend from the parent class, add `option object(ParentName);`

Here is an example

```
restart;
module ODE()
   option object;
   local ode;

   export set_ode::static:=proc(_self,ode,$)
     _self:-ode :=ode;
   end proc;

  export get_ode::static:=proc(_self,$)
     return _self:-ode;
   end proc;
end module;

#create class/module which extends the above
module second_order_ode()
   option object(ODE);
   export get_ode_order::static:=proc(_self,$)
      return 2;
   end proc;
end module;
```

In the above `second_order_ode` inherits all local variables and functions in the `ODE` class and adds new proc. Use as follows

```
o:=Object(second_order_ode); #create an object  instance
o:-set_ode(diff(y(x),x)=sin(x));
o:-get_ode();
o:-get_ode_order();
```

Note that the child class can not have its own variable with the same name as the parent class. This is limitation. in `C++` for example, local variables in extended class overrides the same named variable in the parent class.

Even if the variable have different type, Maple will not allow overriding. For example, this will fail

```
restart;
module ODE()
   option object;
   local ode;
   local id::integer;

   export set_ode::static:=proc(_self,ode,$)
     print("Enter ode::set_ode");
     _self:-ode :=ode;
   end proc;

  export get_ode::static:=proc(_self,$)
     return _self:-ode;
   end proc;
end module;

module second_order_ode()
   option object(ODE);
   local id::string;
   export get_ode_order::static:=proc(_self,$)
      return 2;
   end proc;
end module;

    Error, (in second_order_ode) local `id` is declared more than once
```

There might be a way to handle this, i.e. to somehow explicitly tell Maple to override parent class proc or variable name in the child. I do not know now. The above is using Maple 2021.1

2.5 How to extend a base class and override its method with different one?

This is called polymorphism in OOP. This is a base class animal_class which has `make_sound` method. This method acts just as a place holder (interface), which the extending class must extends (override) with an actual implementation.

The class is extended to make cat class and implementation is made.

```
module animal_class()
 option object;

 export make_sound::static:=proc(_self,$)
     error("Not implemented, must be overriden");
 end proc;
end module;

%---------------
module cat_class()
 option object(animal_class);

 make_sound::static:=proc(_self,$)  #note. DO NOT USE export
     print("mewooo");
 end proc;

end module;
```

And now

```
o:=Object(animal_class);
o:-make_sound();

Error, (in anonymous procedure) Not implemented, must be overriden
```

The above is by design. As the animal_class is meant to be extended to be usable.

```
my_cat:=Object(cat_class);
my_cat:-make_sound();
```

```
"mewooo"
```

So the base class can have number of methods, which are all meant to be be have its implementation provided by an extending class. Each class which extends the base class must provide implementation.

2.6 How to extend a class and call base class function from the extended class??

Once a base class is extended, all methods in the base class become part of the extending class. So to call a base class method just use same way as if calling any other method in the extending class itself.

Here is an example.

```
module person()
    option object;
    local age::integer:=100;

    export ModuleCopy::static:= proc(_self,proto::person, age::integer,$)
           _self:-age := age;
    end proc;

    local base_class_method::static:=proc(_self,$)
        print("In base class method...");
    end proc;


end module;

#---- extend the above class
module young_person()
   option object(person);
   export process::static:=proc(_self,$)
      print("In young_person process");
     _self:-base_class_method();
   end proc;
end module;
```

Here is an example of usage

```
o:=Object(young_person,20);
o:-process()

                    "In young_person process"
                    "In base class method..."
```

The above in Maple 2023.1

2.7 How to use object as user defined record inside a proc?

A Maple Object can be used a record type in other languages, such as Ada or Pascal. This example shows how to define a local type inside a proc and use it as record.

```
restart;
foo:=proc(the_name::string,the_age::integer)::person_type;
     local module person_type() #this acts as a record type
         option object;
         export the_name::string;
         export the_age::integer;
     end module;

     local person::person_type:=Object(person_type);

     person:-the_name:=the_name;
     person:-the_age:=the_age;
     return person;
 end proc;

o:=foo("joe doe",100);

o:-the_name;
                              "joe doe"
o:-the_age;
                                 100
```

In the above `person` is local variable of type `person_type`. In the above example, the local variable was returned back to user. But this is just an example. One can declare such variables and just use them internally inside the proc only. This method helps one organize related variables into one record/struct like type. The type can also be made global if needed.

2.8 How to make copy of list of objects?

Suppose we have list L1 of objects and we want to copy this list to another list, say L2. If we just do `L2 = L1` then this will not make an actual copy as any changes to L1 are still reflected in L2. The above only copies the reference to the same list.

To make a physical copy, need to use the `copy` command as follows

```
restart;

module person_type()
  option object;
  export the_name::string:="doe joe";
  export the_age::integer:=100;
end module;

L1:=[];
for N from 1 to 5 do
    o:=Object(person_type);
    o:-the_name:=convert(N,string);
    L1:= [ op(L1), o];
od:

L2:=map(Z->copy(Z),L1):
```

Now making any changes to L1 will not affect L2. If we just did `L2 = L1` then both will share same content which is not what we wanted.

2.9 How to use OOP to implement ode solver?

This is a basic example of using OOP in Maple to implement an ode solver. There is a base module called ode_base_class (I will be using class instead of module, as this is more common in OOP)

This program will for now support first order and second order ode's only.

The base ode class will contain the basic operations and data which is common to both first order and second order ode's.

Next, we will make a first order ode class, and second order ode class. Both of these will extend the base ode class.

Next, we will have more ode classes. For example, for first order ode, there will be linear

first order ode class, and separable first order ode class, and Bernoulli ode class and so on. Each one these classes will extend the first order ode class which in turn extends the base ode class.

Same for second order ode's. There will be second order constant coefficients ode class, and second order variable coefficient ode class and so on. Each one of these classes will extend the second order ode class which in turn extends the base ode class.

Let `base_ode_class` be the base of an ode class which will contain all the necessary methods that are generic and applicable to an ode of any order and type.

These can be the ode expression itself, its order and any other data and operations which applicable to any ode type.

Now we want to create a first order class. This will have its own operations and private variables that are specific and make sense only to any first order ode.

Then these will be the first order separable ode class, which has methods that implement solving the ode using separable method and has other methods which makes sense only for first order separable ode. The following diagram is partial illustration of the is-A relation among possible classes.

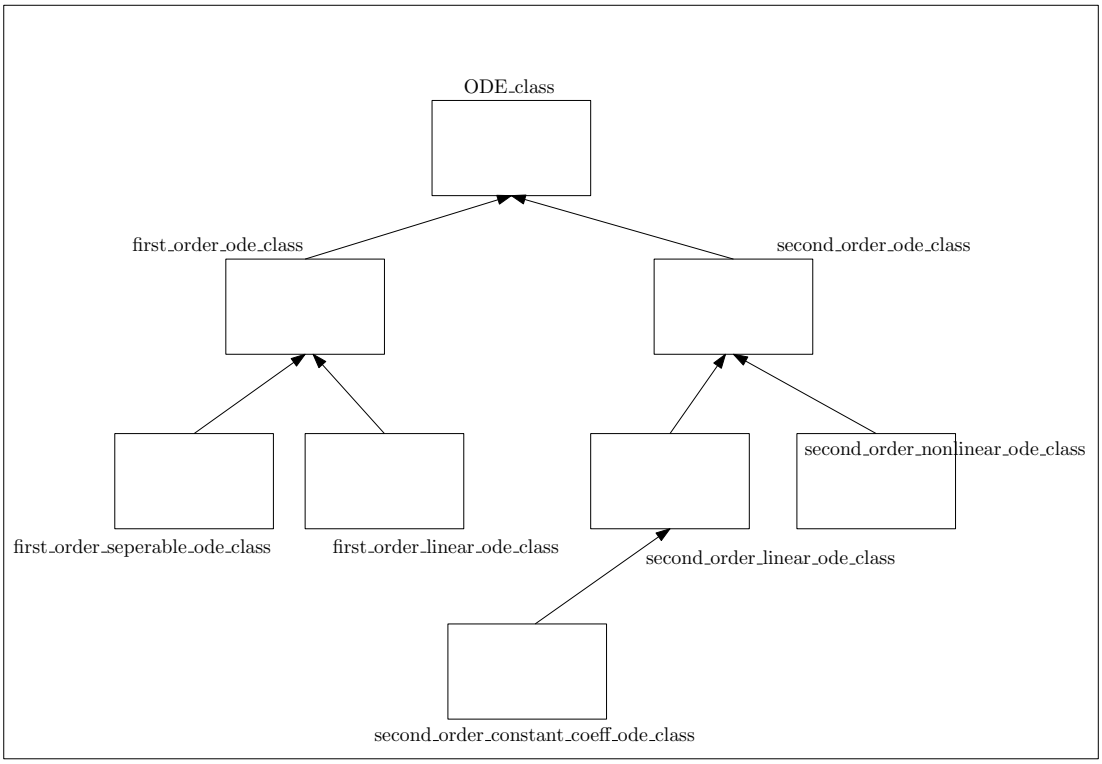


Figure 1: The is-A relation between class for ODE solver

First we define the base ode class and define private variables and method that are common to any ode type.

```
restart;

module base_ode_class()
  option object;
  local the_ode;
  local the_order::integer;

  export get_ode::static:=proc(_self,$)
         RETURN(_self:-the_ode);
  end proc;

  export get_order::static:=proc(_self,$)
         RETURN(_self:-the_order);
  end proc;

end module;
```

Note that the base ode class does not have constructor. Since it is meant to be extended only.

The following is the first order ode class.

```
module first_order_ode_class()
  option object(base_ode_class);

  local  initial_conditions;
  local  solution_found;

  export get_IC::static:=proc(_self,$)
         RETURN(_self:-initial_conditions);
  end proc;

  export get_solution::static:=proc(_self,$)
         RETURN(_self:-solution_found);
  end proc;

  export verify_solution::static:=proc(_self,$)::truefalse;
      #code to verify if the solution found is valid or not
```

```
      #using odetest()
  end proc;

end module;
```

The following is the first order separable ode class which extends the above first order ode class.

```
module first_order_separable_ode_class()
  option object(first_order_ode_class);
  local f,g;  #ode of form  y'=f(x)*g(y)

  export ModuleCopy::static:= proc(_self,proto::first_order_separable_ode_class,
                                   ode, IC,$)
      _self:-the_ode  := ode;
      _self:-initial_conditions := IC;
  end proc;

  export dsolve::static:=proc(_self,$)
         #solve the ode for now we will use Maple but in my code
         #I have my own solver ofcourse.
         _self:-solution_found:=  :-dsolve([_self:-the_ode, _self:-initial_conditions]);
  end proc;
end module;
```

In the above, when we create a instance of `first_order_separable_ode_class` then it now have the whole chain of classes into one. i.e. first order separable class extending the first order class which in turn extends the base ode class. For example

```
o:=Object(first_order_separable_ode_class,diff(y(x),x)=3*sin(x)*y(x),y(0)=1)
o:-dsolve();
o:-get_solution()
       #y(x) = exp(3)*exp(-3*cos(x))
o:-get_ode()
       #diff(y(x), x) = 3*sin(x)*y(x)
```

The above calls will all work, even though the first order separable class has no method in it called `get_ode` but it extends a class which does, hence it works as it.

Now we will do the same for second order ode's.

Advantage of this design, is that methods in base classes that are being extended can be reused by any class which extends them. Only methods that applies and specific to the lower level classes need to be implemented.

As we add more specific solvers, we just have to extend the base classes and the new solvers just need to implement its own specific `dsolve` and any specific methods and data that it needs itself.

Ofcourse in practice the above design is not complete as is. The user should not have to specify which class to instantiate, as user does not care what the ode type or class it is. They just want to to do

```
o:=Object(ode_class,diff(y(x),x)=3*sin(x)*y(x),y(0)=1)
o:-dsolve();
o:-get_solution()
```

To solve this problem we have to make a factory method which is called to make the correct instance of the class and return that to the user. The factory method figures out the type of ode and it creates the correct instance of the correct class and returns that. So the call will become

```
o := make_ode_object( diff(y(x),x)=3*sin(x)*y(x), y(0)=1)
o:-dsolve();
o:-get_solution()
```

The function `make_ode_object` above is the main interface the user will call to make an ode object.

This will be explained next with examples. One possibility is to make the factory function a global function or better, a public method in a utility module. For now, it is given here as standalone function for illustration. The user calls this method to make an object of the correct instance of the ode. Here is complete implementation of all the above including the factory method.

```
#factory method. Makes objects for users
make_ode_object:=proc(ode::`=`,func::function(name))
   local x,y,the_order;
   y:=op(0,func);
   x:=op(1,func);
   the_order := PDEtools:-difforder(ode,x);
   if the_order=1 then
        RETURN(first_order_ode_class:-make_ode_object(ode,func));
```

```
   elif the_order=2 then
        #RETURN(second_order_ode_class:-make_ode_object(ode,func));
        #implement later
        NULL;
   else
        error "Only first and second order ode's are currently supported";
   fi;

end proc:

#########################
module base_ode_class()
  option object;
  local the_ode;
  local the_order::integer;

  #methods bound to the object
  export get_ode::static:=proc(_self,$)
         RETURN(_self:-the_ode);
  end proc;

  export get_order::static:=proc(_self,$)
         RETURN(_self:-the_order);
  end proc;

end module:
##################
module first_order_ode_class()
  option object(base_ode_class);

  local  initial_conditions;
  local  solution_found;

  #public factory method not bound to the object.
  export make_ode_object:=proc(ode::`=`,func::function(name))
     local x,y,ode_type::string;
     y:=op(0,func);
     x:=op(1,func);
     ode_type:="separable"; #code here which determined first order ode type
     if ode_type="separable" then
        RETURN( Object(first_order_separable_ode_class,ode,func));
```

```
    elif ode_type="linear" then
        RETURN( Object(first_order_linear_ode_class,ode,func));
    fi;
    #more ode types added here
  end proc;

  #methods bound to the object
  export get_IC::static:=proc(_self,$)
         RETURN(_self:-initial_conditions);
  end proc;

  export get_solution::static:=proc(_self,$)
         RETURN(_self:-solution_found);
  end proc;

  export verify_solution::static:=proc(_self,$)::truefalse;
      #code to verify if the solution found is valid or not
      #using odetest()
  end proc;

end module:

###################
module first_order_separable_ode_class()
  option object(first_order_ode_class);
  local f,g;  #ode of form  y'=f(x)*g(y)

  export ModuleCopy::static:= proc(_self,proto::first_order_separable_ode_class,ode,func
      _self:-the_ode  := ode;
  end proc;

  export dsolve::static:=proc(_self,$)
         #print("Enter first_order_separable_ode_class:-dsolve");
         #solve the ode for now we will use Maple but in my code
         #I have my own solver ofcourse.
         _self:-solution_found:=  :-dsolve(_self:-the_ode);
         NULL;
  end proc;
end module:
```

It is used as follows

```
o:=make_ode_object(diff(y(x),x)=sin(x)*y(x),y(x));
o:-dsolve();
o:-get_solution();
                    y(x) = c__1 exp(-cos(x))
```

2.10 How to make a complete OOP ode solver in Maple?

Here, I will start making a complete small OOP ode solver in Maple.

At each step more classes are added and enhanced until we get a fully working small ode solver based on OOP design that solves a first and second order ode, this is to show how it all works. Another solvers can be added later by simply extending the base class.

The base class is called `Base_ode_class`. There will be `Second_order_ode_class` and `First_order_ode_class` and these classes extend the base ode class. We can later add `higher_order_ode_class`.

Next, there are different classes which extend these. There is `First_order_linear_ode_class` and `First_order_separable_ode_class` and so on, and these extend the `First_order_ode_class`.

For example, if a user wanted to solve a first order ode which happened to be say separable, then object of class `First_order_separable_ode_class` will be created and used.

Since the user does not know and should not know what object to create, then the factory class will be used. The factory class is what the user initially calls to make the ode object.

It is the factory class which determines which type of class to instantiate based on the ode given.

The factory class is singleton (standard module in Maple, not of type object), which has the `make_ode` method which is called by the user. This method parses the ode and determines its order and then based on the order determine which subclass to use, and then instantiate this and returns the correct object to the user to use.

This object will have the `dsolve` method and other methods the user can use on the object.

The `make_ode` method in the factory module accepts only the ode itself the function such as $y(x)$. A typical use is given below

```
ode :=  ODE_factoy_class:-make_ode( diff(y(x),x)=sin(x), y(x) );
ode:-set_IC(....);
ode:-set_hint("the hint");
.....
```

```
ode:-dsolve();  #solves the ode
ode:-is_solved(); #checks if ode was successfully solved
ode:-verify_sol(); #verifies the solution using maple odetest()
ode:-is_sol_verified(); #asks if solution is verified
ode:-get_number_of_sol(); #returns number of solutions, 1 or 2 etc...
ode:-get_sol(); #returns the solutions found in list
#and more method
...
```

Examples at the end will show how all the above works on actual odes's.

The initial call to make an ode does not have initial conditions, or hint and any other parameters. This is so to keep the call simple. As the factory method only makes the concrete ode object.

Additional methods are then used to add more information if needed by using the returned object itself, such as initial conditions, and hint and so on before calling the dsolve method on the object.

Here is a very basic setup which include the base ode class and extended to first order two sub classes for now.

```
restart;
ODE_factory_class :=module()
   #notice, normal module. No option object.

   export make_ode:=proc(ode::`=`,func::function(name),$)
       local dep_variables_found::list,item;
       local y::symbol;
       local x::symbol;
       local ode_order::integer;

       if nops(func)<>1 then
           error("Parsing error, dependent variable must contain one argument, found ",
       fi;
       y:=op(0,func);
       x:=op(1,func);
       if not has(ode,y) then
          error ("Supplied ode ",ode," has no ",y);
       fi;
```

```
    if not has(ode,x) then
       error ("Supplied ode ",ode," has no ",x);
    fi;

    if not has(ode,func) then
        error ("Supplied ode ",ode," has no ",func);
    fi;

    ode_order := PDEtools:-difforder(ode,x);

    #this will check that the dependent variable will show with
    #SAME argument in the ode. i.e. if y(x) and y(t) show up in same ode, it
    #will throw exception, which is what we want.

    try
        dep_variables_found := PDEtools:-Library:-GetDepVars([y],ode);
    catch:
        error lastexception;
    end try;

    #now go over dep_variables_found and check the independent
    #variable is same as x i.e. ode can be y'(z)+y(z)=0 but function is y(x).
    for item in dep_variables_found do
        if not type(item,function) then
           error("Parsing error. Expected ",func," found ",item," in ode");
        else
           if op(1,item) <> x then
               error("Parsing error. Expected ",func," found ",item," in ode");
           fi;
        fi;
    od;

   #now go over all indents in ode and check that y only shows as y(x) and not as j
   #as the PDEtools:-Library:-GetDepVars([_self:-y],ode) code above does not detect
   #i.e. it does not check  y'(x)+y=0

   if numelems(indets(ode,identical(y))) > 0 then
        error("Parsing error, Can not have ",y," with no argument inside ",ode);
   fi;

   if ode_order=1 then
```

```
         RETURN(make_first_order_ode(ode,y,x));
      elif ode_order=2 then
         RETURN(make_second_order_ode(ode,y,x));
      else
         RETURN(make_higher_order_ode(ode,y,x));
      fi;
  end proc;

  local make_first_order_ode:=proc(ode::`=`,y::symbol,x::symbol)
     #decide on what specific type the ode is, and make instant of it

     if first_order_ode_quadrature_class:-is_quadrature(ode,y,x) then
        RETURN(Object(first_order_ode_quadrature_class,ode,y,x));
     elif first_order_ode_linear_class:-is_linear(ode,y,x) then
        RETURN(Object(first_order_ode_linear_class,ode,y,x));
     fi; #and so on
  end proc;

  local make_second_order_ode:=proc(ode::`=`,y::symbol,x::symbol)
     #decide on what specific type the ode is, and make instant of it
     #same as for first order
  end proc;
end module;
#------------------------
module solution_class()
   option object;
   local the_solution;
   local is_verified_solution::truefalse:=false;
   local is_implicit_solution::truefalse:=false;

   export ModuleCopy::static:= proc(_self,proto::solution_class,the_solution::`=`,is_imp
       _self:-the_solution:=the_solution;
       _self:-is_implicit_solution:=is_implicit_solution;
   end proc;

   export get_solution::static:=proc(_self,$)
        RETURN(_self:-the_solution);
   end proc;

   export is_verified::static:=proc(_self,$)
        RETURN(_self:-is_verified_solution);
```

```
   end proc;

   export is_implicit::static:=proc(_self,$)
        RETURN(_self:-is_implicit_solution);
   end proc;

   export is_explicit::static:=proc(_self,$)
        RETURN(not(_self:-is_implicit_solution));
   end proc;

   export verify_solution::static:= overload(
   [
      proc(_self, ode::`=`,$) option overload;
          local stat;
          stat:= odetest(_self:-the_solution,ode);
          if stat=0 then
             _self:-is_verified_solution:=true;
          else
            if simplify(stat)=0 then
                _self:-is_verified_solution:=true;
            else
                _self:-is_verified_solution:=false;
            fi;
         fi;
      end,

      proc(_self, ode::`=`,IC::list,$) option overload;
          local stat;
          stat:= odetest([_self:-the_solution,IC],ode);
          if stat=[0,0] then
             _self:-is_verified_solution:=true;
          else
            if simplify(stat)=[0,0] then
                _self:-is_verified_solution:=true;
            else
                _self:-is_verified_solution:=false;
            fi;
         fi;
      end
   ]);
end module:
```

```
#------------------------

module ODE_base_class()
   option object;
   local y::symbol;
   local x::symbol;
   local func::function(name); #y(x)
   local ode::`=`;
   local ode_order::posint;
   local IC::list:=[];
   local parsed_IC::list:=[];
   local the_hint::string:="";
   local solutions_found::list(solution_class):=[];

  #exported getters methods
  export get_ode::static:=proc(_self,$)
         RETURN(_self:-ode);
  end proc;

  export get_x::static:=proc(_self,$)
         RETURN(_self:-x);
  end proc;

  export get_y::static:=proc(_self,$)
         RETURN(_self:-y);
  end proc;

  export get_ode_order::static:=proc(_self,$)
         RETURN(_self:-ode_order);
  end proc;

  export get_IC::static:=proc(_self,$)
         RETURN(_self:-IC);
  end proc;

  export get_parsed_IC::static:=proc(_self,$)
         RETURN(_self:-parsed_IC);
  end proc;

  export get_sol::static:=proc(_self,$)
         local L:=Array(1..0):
```

```
        local sol;
        for sol in _self:-solutions_found do
            L ,= sol:-get_solution();
        od;
        RETURN(convert(L,list));
  end proc;

  #exported setters methods
  export set_hint::static:=proc(_self,hint::string,$)
        #add code to check if hint is valid
        _self:-the_hint:=hint;
  end proc;
end module;

#-------------------------
module first_order_ode_quadrature_class()
  option object(ODE_base_class);
  local f,g;  #ode of form  y'=f(x)*g(y)

  #this method is not an object method. It is part of the module but does
  #not have _self. It is called by the factory class to find if the ode
  #is of this type first
  export is_quadrature:=proc(ode::`=`,y::symbol,x::symbol)::truefalse;
        RETURN(true); #for now
  end proc;

  export ModuleCopy::static:= proc(_self,proto::first_order_ode_quadrature_class,ode::`=
      _self:-ode  := ode;
      _self:-y  := y;
      _self:-x  := x;
      _self:-func  := _self:-y(_self:-x);
      _self:-ode_order :=1;
  end proc;

  export dsolve::static:=proc(_self,$)
        local sol,o;
        #print("Enter first_order_ode_quadrature_class:-dsolve");
        #solve the ode for now we will use Maple but in my code
        #I have my own solver ofcourse.
        sol:= :-dsolve(_self:-ode,_self:-func);
        o:=Object(solution_class,sol,false);
```

```
        _self:-solutions_found:= [o];
        NULL;
  end proc;
end module:

#-------------------------
module first_order_ode_linear_class()
  option object(ODE_base_class);
  local f,g;  #ode of form  y'=f(x)*g(y)

  #this method is not an object method. It is part of the module but does
  #not have _self. It is called by the factory class to find if the ode
  #is of this type first
  export is_linear:=proc(ode::`=`,y::symbol,x::symbol)::truefalse;
        RETURN(true); #for now
  end proc;

  export ModuleCopy::static:= proc(_self,proto::first_order_ode_linear_class,ode::`=`,y:
      _self:-ode  := ode;
      _self:-y  := y;
      _self:-x  := x;
      _self:-func  := _self:-y(_self:-x);
      _self:-ode_order :=1;
  end proc;

  export dsolve::static:=proc(_self,$)
        local sol,o;
        sol:= :-dsolve(_self:-ode,_self:-func);
        o:=Object(solution_class,sol,false);
        _self:-solutions_found[1]:= [o]:
  end proc:
end module:
```

Example usage is

```
o:=ODE_factory_class:-make_ode(diff(y(x),x)=x,y(x))
o:-get_ode()
                          d
                         --- y(x) = x
                          dx
```

```
o:-dsolve();
o:-get_sol()
                        [       1  2       ]
                        [y(x) = - x  + c__1]
                        [       2          ]
```

3 Differential equations

3.1 How to check if ode is of certain type?

The command `DEtools:-odeadvisor(ode,y(x));` gives the ode type names.

See `https://maplesoft.com/support/help/maple/view.aspx?path=DEtools/odeadvis or` for known names of ode type in Maple.

But we can also tell it to check if the ode is of specific type using `DEtools:-odeadvisor(ode,y(x),[name]);` For an example

```
ode:=x^2+3*x*diff(y(x),x)=y(x)^4+2*y(x);
DEtools:-odeadvisor(ode,y(x),[Chini])

              [_Chini]
```

If the name we have given is not the type of the ode, Maple will return `[NONE]`.

For an example

```
ode:=x^2+3*x*diff(y(x),x)=y(x)^4+2*y(x);
DEtools:-odeadvisor(ode,y(x),[separable])

              [NONE]
```

3.2 How to force dsolve to use specific method for solving?

To find what methods dsolve uses do

```
`dsolve/methods`[1]

     [quadrature, linear, Bernoulli, separable, inverse_linear,
      homogeneous, Chini, lin_sym, exact, Abel, pot_sym]
```

The above gives methods for first order ODEs. More are given by

```
`dsolve/methods`[1,'semiclass']

     [Riccati, inverse_Riccati, equivalent_to_Abel,
      linearizable, linearizable_by_differentiation]

`dsolve/methods`[1,'high_degree']

     [WeierstrassP, WeierstrassPPrime, JacobiSN, linearizable_by_differentiation,
      missing, dAlembert, homogeneous_B, sym_implicit]

`dsolve/methods`[1,"development"]

     [linearizable_by_differentiation, linearizable, con_sym,
      WeierstrassP, WeierstrassPPrime, equivalent_to_Abel,
      Abel_AIR, special, Riccati_symmetries]

`dsolve/methods`[1,extra]
     [inverse_Riccati, Abel_AIL, `sym_pat/[F(x)*G(y),0]`, `sym_pat/[F(x),G(x)]`,
      `sym_pat/[F(x),G(y)]`, `sym_pat/[F(x)+G(y),0]`,
      `sym_pat/[F(x),G(x)*y+H(x)]`, sym_pat, exp_sym]
```

For example given a first order ode, we can ask dsolve to solve it using one of these methods as follows

```
restart;
ode:=diff(y(x),x)=sqrt( sqrt( y(x) )* sin(x))/sqrt(y(x))
dsolve(ode,y(x),[`exact`])

     4/5*y(x)^(5/4) + Intat(-sqrt(sqrt(y(x))*sin(_a))/y(x)^(1/4), _a = x) + c__1 = 0
```

We can ask it to use symmetry with specific pattern as follows

```
restart;
ode:=diff(y(x),x)=sqrt( sqrt( y(x) )* sin(x))/sqrt(y(x))
dsolve(ode,y(x),[`sym_pat/[F(x),G(x)*y+H(x)]`])

    c__1 + 4/5*y(x)^(3/2)*sqrt(sin(x))/sqrt(sqrt(y(x))*sin(x)) - Int(sqrt(sin(x)), x) =

#or using short cut, same thing can be done as follows
```

```
dsolve(ode,y(x),Lie)

    c__1 + 4/5*y(x)^(3/2)*sqrt(sin(x))/sqrt(sqrt(y(x))*sin(x)) - Int(sqrt(sin(x)), x) =

#another short cut for Lie is
dsolve(ode,[`sym_pat`])

    c__1 + 4/5*y(x)^(3/2)*sqrt(sin(x))/sqrt(sqrt(y(x))*sin(x)) - Int(sqrt(sin(x)), x) =
```

To find all methods, and for higher order ode, we first use indices as follows

```
indices(`dsolve/methods`)

    [high, linear_nonhomogeneous],
    [1],
    [1, high_degree],
    [3, linear_homogeneous],
    [2, "linear_homogeneous all methods"],
    [2, "linear_homogeneous other"],
    [2, linear_homogeneous],
    [2, "special_functions"],
    [3, linear_nonhomogeneous],
    [2, "linear_homogeneous in Normal Form"],
    [2, "development"], [2, "hypergeometric"],
    [1, extra],
    [high, linear_homogeneous],
    [high, nonlinear],
    [1, "special"],
    [1, "development"],
    [2, nonlinear],
    [high, "development"],
    [1, semiclass],
    [3, nonlinear],
    [2, linear_nonhomogeneous],
    [2, "linear_homogeneous as given"],
    [3, "development"]
```

Using the above, to find all methods for second order linear_homogeneous ode, the command is

```
`dsolve/methods`[2, "linear_homogeneous all methods"]
```

```
    [quadrature, const_coeffs, Euler, linear_1,
     `linear/missing_y`, Kovacic, special_functions, to_const_coeffs,
      exact_linear, sym_1, Mathieu, MeijerG, Heun, HeunG, HeunC, HeunB, HeunD,
      HeunT, mu_xy, equivalent_to_Bessel, to_Riccati, Bessel, elliptic,
      Legendre, Whittaker, Kummer, cylindrical, hypergeometric, hypergeom1,
      hypergeom2, Riemann, RNF, hypergeometricsols, rationalize_lode, with_periodic_func

`dsolve/methods`[2, "linear_homogeneous other"]

    [exact_linear, sym_1, to_const_coeffs, mu_xy, equivalent_to_Bessel,
     to_Riccati, with_periodic_functions]
```

For example, given the ode $y'' + 3y' + y = 0$, we can now do

```
dsolve(ode,y(x))
     y(x) = c__1*exp(1/2*(sqrt(5) - 3)*x) + c__2*exp(-1/2*(3 + sqrt(5))*x)

dsolve(ode,y(x),[`const_coeffs`])
    y(x) = c__1*exp(1/2*(sqrt(5) - 3)*x) + c__2*exp(-1/2*(3 + sqrt(5))*x)

dsolve(ode,y(x),[`Kovacic`])
    y(x) = c__1*exp(1/2*(sqrt(5) - 3)*x) + c__2*exp(-1/2*(3 + sqrt(5))*x)
```

Not all methods ofcourse will work, as it depends on the ode type.

This function below lists all methods

```
ind:=indices(`dsolve/methods`);
for item in ind do
    cat("`dsolve/methods`",String(item));
    eval(parse(%))
od;
```

Which gives

```
"`dsolve/methods`[high, linear_nonhomogeneous]"
    [quadrature, fully_exact_linear, linear_nonhomogeneous_[0,1],
    exact_linear_nonhomogeneous, linear, exp_reduce]
```

```
"`dsolve/methods`[1]"
  [quadrature, linear, Bernoulli, separable, inverse_linear,
   homogeneous, Chini, lin_sym, exact, Abel, pot_sym]

"`dsolve/methods`[1, high_degree]"
    [WeierstrassP, WeierstrassPPrime, JacobiSN,
     linearizable_by_differentiation, missing, dAlembert,
     homogeneous_B, sym_implicit]


"`dsolve/methods`[3, linear_homogeneous]"
     [quadrature, const_coeffs, Euler, fully_exact_linear,
      to_const_coeffs, linear, exp_reduce, exact_linear,
      with_periodic_functions]


"`dsolve/methods`[2, "linear_homogeneous all methods"]"
     [quadrature, const_coeffs, Euler, linear_1, linear/missing_y,
      Kovacic, special_functions, to_const_coeffs, exact_linear,
      sym_1, Mathieu, MeijerG, Heun, HeunG, HeunC, HeunB, HeunD,
      HeunT, mu_xy, equivalent_to_Bessel, to_Riccati, Bessel,
      elliptic, Legendre, Whittaker, Kummer, cylindrical,
      hypergeometric, hypergeom1, hypergeom2, Riemann, RNF,
      hypergeometricsols, rationalize_lode, with_periodic_functions]


"`dsolve/methods`[2, "linear_homogeneous other"]"
      [exact_linear, sym_1, to_const_coeffs, mu_xy,
       equivalent_to_Bessel, to_Riccati, with_periodic_functions]

"`dsolve/methods`[2, linear_homogeneous]"
                     [linear_homogeneous]

"`dsolve/methods`[2, "special_functions"]"
    [Bessel, elliptic, Legendre, Kummer, Whittaker, hypergeometric, Mathieu]


"`dsolve/methods`[3, linear_nonhomogeneous]"
     [quadrature, fully_exact_linear, linear_nonhomogeneous_[0,1],
      exact_linear_nonhomogeneous, linear, exp_reduce]
```

```
"`dsolve/methods`[2, "linear_homogeneous in Normal Form"]"
                            [linear_1]

"`dsolve/methods`[2, "development"]"
    [mu_xyp, mu_xyp2, mu_formal, mu_heuristic, exp_reduce, linear,
     Bessel2, Whittaker_old, nonlinear_homogeneous,
     exact_linear_nonhomogeneous, mu_y1, mu_x_y1, mu_y_y1,
     mu_poly_yn, exp_sym, sym_pat, sym_8]


"`dsolve/methods`[2, "hypergeometric"]"
                        [hypergeom1, hyper3]

"`dsolve/methods`[1, extra]"
      [inverse_Riccati, Abel_AIL, sym_pat/[F(x)*G(y),0],
       sym_pat/[F(x),G(x)], sym_pat/[F(x),G(y)],
       sym_pat/[F(x)+G(y),0], sym_pat/[F(x),G(x)*y+H(x)], sym_pat,
       exp_sym]


"`dsolve/methods`[high, linear_homogeneous]"
     [quadrature, const_coeffs, Euler, fully_exact_linear,
       to_const_coeffs, linear, exp_reduce, exact_linear,
       with_periodic_functions]

"`dsolve/methods`[high, nonlinear]"
     [linearizable_by_differentiation, linearizable, reducible,
      exact_nonlinear, missing, mu_formal, lin_sym]

"`dsolve/methods`[1, "special"]"
                             [80, 81]

"`dsolve/methods`[1, "development"]"
     [linearizable_by_differentiation, linearizable, con_sym,
      WeierstrassP, WeierstrassPPrime, equivalent_to_Abel, Abel_AIR,
      special, Riccati_symmetries]


"`dsolve/methods`[2, nonlinear]"
    [Liouville, WeierstrassP, JacobiSN, linearizable,
```

```
    linearizable_by_differentiation, mu_xy_2, missing,
    mu_xyp2_dynamical_symmetries_fully_reducible,
    mu_xyp_singularcases, sym_1, exact_nonlinear, reducible,
    lin_sym, S-function, mu_xyp_generalcase,
    mu_xyp2_dynamical_symmetries_not_fully_reducible]


"`dsolve/methods`[high, "development"]"
    [k25, RNF, mu_heuristic, MeijerG, nonlinear_homogeneous,
     mu_poly_yn, exp_sym]

"`dsolve/methods`[1, semiclass]"
    [Riccati, inverse_Riccati, equivalent_to_Abel, linearizable,
     linearizable_by_differentiation]


"`dsolve/methods`[3, nonlinear]"
   [linearizable_by_differentiation, linearizable, missing,
     exact_nonlinear, reducible, mu_formal, lin_sym]


"`dsolve/methods`[2, linear_nonhomogeneous]"
     [quadrature, fully_exact_linear, linear_nonhomogeneous_[0,1],
      linear_nonhomogeneous_[0,F(x)], linear_nonhomogeneous]

"`dsolve/methods`[2, "linear_homogeneous as given"]"
     [quadrature, const_coeffs, Euler, linear_1, linear/missing_y,
      Kovacic, RNF, special_functions, MeijerG, Heun,
      hypergeometricsols, rationalize_lode]

"`dsolve/methods`[3, "development"]"
     [k25, RNF, mu_heuristic, linear_patterns, MeijerG,
      nonlinear_homogeneous, mu_y2, mu_poly_yn, exp_sym, pFq, 3F2,
      2F2, 1F2, 0F2]
```

3.3 How to find a particular solution to ODE?

```
restart;
ode:=diff(y(x),x)+y(x)^2*sin(x)-2*sin(x)/cos(x)^2 = 0;
yp:=DETools:-particularsol(ode);
```

To step into the code, do

```
restart;
ode:=diff(y(x),x)+y(x)^2*sin(x)-2*sin(x)/cos(x)^2 = 0;
stopat(`DEtools/particularsol`);
DETools:-particularsol(ode);
```

To print it do

```
print(`DEtools/particularsol`);
```

3.4 How to find basis solutions for homogeneous ode?

Use the `output=basis` option

```
ode:=diff(y(x),x$2)-x*diff(y(x),x)-x*y(x)=0;
dsolve(ode,output=basis);
```

3.5 How to solve a differential equation with initial conditions?

To solve

$$y'' - 3y' + 2y = 10e^{5x}$$

with $y(0) = 1, y'(0) = 5$ do

```
eq1:= diff(y(x),x$2)-3*diff(y(x),x)+2*y(x)=10*exp(5*x);
dsolve({eq1,y(0)=1,D(y)(0)=5},y(x));

Methods for second order ODEs:
Trying to isolate the derivative d^2y/dx^2...
Successful isolation of d^2y/dx^2
--- Trying classification methods ---
trying a quadrature
trying high order exact linear fully integrable
trying differential order: 2; linear nonhomogeneous with symmetry [0,1]
```

```
trying a double symmetry of the form [xi=0, eta=F(x)]
<- double symmetry of the form [xi=0, eta=F(x)] successful
....
```

The above can also be written using `D@@` notation, like this

```
eq:= (D@@2)(y)(x) - 3*D(y)(x) +2*y(x) = 10*exp(5*x);
IC := y(0)=1,D(y)(0)=5;
dsolve({eq,IC},y(x));
```

3.6 How to verify that the ODE solution given is correct?

use `odetest` and check if it gives zero.

```
eq1:= diff(diff(y(x),x),x)-3*diff(y(x),x)+2*y(x)=10*exp(5*x);
ans:=dsolve({eq1,IC},y(x));
odetest(ans,eq1);

                0
```

3.7 How to know the type of ODE?

Maple can classify the ODE.

```

eq1:= diff(y(x),x$2)-3*diff(y(x),x)+2*y(x)=10*exp(5*x);
R0 := DEtools['odeadvisor'](eq1,y(x));

        R0 := [[_2nd_order, _with_linear_symmetries]]
```

To get help on this type of ODE, do

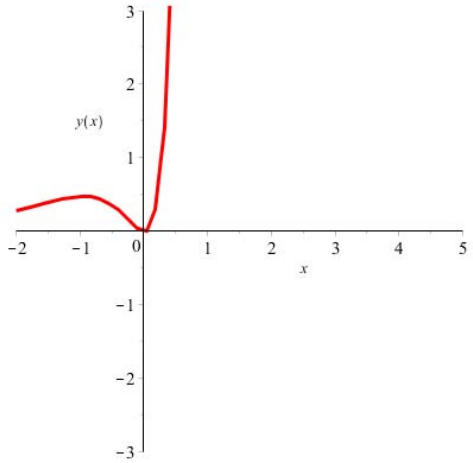
```
DEtools['odeadvisor'](eq1,'help');
```

3.8 What packages to load for differential equations?

Use `with(DEtools);`

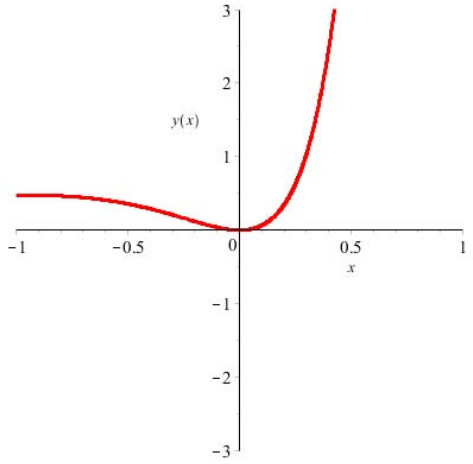
3.9 How to plot solution of differential equations?

```
restart;
eq1:= diff(y(x),x$2)-3*diff(y(x),x)+2*y(x)=10*exp(5*x);
DEtools[DEplot](eq1,y(x),x=-2..5, [ [y(0)=0, D(y)(0)=0]], y=-3..3,linecolor=red);
```



To get a better plot, change the stepsize and independent variable range

```
restart;
eq1:= diff(y(x),x$2)-3*diff(y(x),x)+2*y(x)=10*exp(5*x);
DEtools[DEplot](eq1,y(x),x=-1..1,[[y(0)=0,D(y)(0)=0]],y=-3..3,stepsize=0.001,linecolor=r
```



3.10 On High precision. Using Taylor to solve ODE

```
From: Robert Israel (israel@math.ubc.ca)
Subject: Re: given precision in Maple
Newsgroups: comp.soft-sys.math.maple
Date: 2003-07-16 20:19:06 PST

Set Digits:= n and all calculations from this point will be done with n
digits.  Mathematical functions will be correct to n digits as well (to
the extent this is practical).

If you want high-accuracy numerical ODE solutions, on the other hand,
it's not so simple.  I think the best way is using the taylorseries
method.  For example, consider the problem y' = y^2, y(1) = 1, where
the exact solution y = 1/(2-x) has y(1.9) = 10.

> Digits:= 30:
  sol:= dsolve({D(y)(x)=y(x)^2, y(1) = 1}, y(x), numeric,
            method=taylorseries, abserr=1e-25):
  sol(1.9);

          [x = 1.9, y(x) = 9.999999999999999999999797691]

> 10 - eval(y(x),%);

                                        -23
                              0.202309 10

The other methods (in particular the default rkf45) do not give results
anywhere near this good.
```

3.11 Obtain ODE in canonical coordinates (Lie symmetry)

How to obtain the ODE in canonical coordinates $S(R)$ for an ode using Lie symmetry?

Here is a function I wrote with the help of Maple docs, that does that. For now, I am using first order ode. It takes the ode and the variables (coordinates) to use (R, S) is what normally used, and returns back the ODE in canonical coordinates. The ode should always be a quadrature for first order.

```
get_ODE_in_canonical:=proc(ode::`=`,y::symbol,x::symbol,S::symbol,R::symbol)
```

```
  local infinitesimals,tr,itr,ODE;
  infinitesimals := [DEtools:-symgen(ode)];
  if nops(infinitesimals)=0 then
     RETURN(FAIL);
  fi;
  infinitesimals:=infinitesimals[1];
  tr  := DEtools:-canoni(infinitesimals,y(x),S(R));
  itr := op(1,[solve(tr,{x,y(x)})]);
  ODE:= PDEtools:-dchange(itr,ode,[R,S(R)],simplify);
  ODE:= op(solve(ODE,{diff(S(R),R)}));
  RETURN(ODE);
end proc:
```

Example

```
ODE := diff(y(x),x) = 1/x^2+y(x);
get_ODE_in_canonical(ODE,y,x,S,R)

    #diff(S(R), R) = 1/(exp(R)*R^2)

ODE := diff(y(x),x) = x^2-2*x*y(x)+y(x)^2;
get_ODE_in_canonical(ODE,y,x,S,R)
   #diff(S(R), R) = 1
```

The following function is small change of the above. It accepts your choice of ξ, η to use instead of the first one obtained by Maple. This makes it possible to experiment with different infinitesimals. Note that different infinitesimals will/can result in different canonical ODE, but the final solution for $y(x)$ of course will always be the same after applying the transformation back to x, y coordinates.

```
restart;
get_ODE_in_canonical_v2:=proc(ode::`=`,y::symbol,x::symbol,S::symbol,R::symbol,xi,eta)
  local infinitesimals:=[xi,eta],tr,itr,ODE;
  tr  := DEtools:-canoni(infinitesimals,y(x),S(R));
  itr := op(1,[solve(tr,{x,y(x)})]);
  ODE:= PDEtools:-dchange(itr,ode,[R,S(R)],simplify);
  ODE:= op(solve(ODE,{diff(S(R),R)}));
  RETURN(simplify(ODE,symbolic));
end proc:
```

And now

```
ODE:=diff(y(x),x) = x^2-2*x*y(x)+y(x)^2;
get_ODE_in_canonical_v2(ODE,y,x,S,R,1,1)

    #diff(S(R), R) = 1/(R^2 - 1)
```

3.12 How to use hint with symgen?

Here is an example.

```
ode:=diff(y(x),x)=-y(x)^2/(exp(x)-y(x))
DEtools:-symgen(ode,y(x),HINT=[c1*x+c2*y+c3,c4*x+c5*y+c6]);

      [_xi = 1, _eta = y]
```

Or

```
ode:=diff(y(x),x)=-y(x)^2/(exp(x)-y(x))
DEtools:-symgen(ode,y(x),HINT=[g(x),f(x)*y]);

      [_xi = 1, _eta = y]
```

So not use `y(x)` but only `y` in the hint.

3.13 How to parse a single ode?

The following function takes in a single ode and parses it to verify it is valid. It returns back 3 values. The dependent variable, the independent variable and the ode order.

```
#added 12/8/2022.
interface(warnlevel=4);
kernelopts('assertlevel'=2):

parse_single_ode:=proc(ode::`=`)::symbol,symbol,integer;
#parses single ode. returns back  dep_var,indep_var,ode_order
#throws exception when it detects parsing errors

local func;
local y,x;
local dep_variables_found::list,item;
local the_order;
```

```
func:=PDEtools:-Library:-GetDepVars("not given",ode,onlydifferentiatedfunctions=true);
if nops(func)=0 then
   error ("not differential equation ",ode);
fi;

func := func[1];

if nops(func)<>1 then
   error("Parsing error, dependent variable must contain one argument, found ", func);
fi;

y:=op(0,func);
x:=op(1,func);

#basic verification
if not has(ode,y) then
   error ("Supplied ode ",ode," has no ",y);
fi;

if not has(ode,x) then
   error ("Supplied ode ",ode," has no ",x);
fi;

if not has(ode,func) then
   error ("Supplied ode ",ode," has no ",func);
fi;

the_order := PDEtools:-difforder(ode,x);
#if the_order=0 then
#   error ("No derivative found in ",ode,". Input is not differential equation");
#fi;

#note that the following call will also return y(x) if the input is not an ode
#this will check that the dependent variable will show with SAME argument in the ode
#i.e. if y(x) and y(t) show up in same ode, it will throw exception, which is what
#we want.
try
   dep_variables_found := PDEtools:-Library:-GetDepVars([y],ode);
catch:
   error lastexception;
```

```
end try;

#now go over dep_variables_found and check the independent variable is same as x
#i.e. ode can be y'(z)+y(z)=0 but function is y(x).
for item in dep_variables_found do
    if not type(item,function) then
       error("Parsing error. Expected ",func," found ",item," in ode");
    else
       if op(1,item) <> x then
          error("Parsing error. Expected ",func," found ",item," in ode");
       fi;
    fi;
od;

#now go over all indents in ode and check that y only shows as y(x) and not as just y
#as the PDEtools:-Library:-GetDepVars([_self:-y],ode) code above does not detect this.
#i.e. it does not check  y'(x)+y=0

if numelems(indets(ode,identical(y))) > 0 then
   error("Parsing error, Can not have ",y," with no argument inside ",ode);
fi;

return y,x,the_order;
end proc:
```

To use do

```
y,x,the_order := parse_single_ode(diff(y(x),x)+y(x)=0);
```

An alternative to the above is to pass the dependent function itself as well as the ode. This is what I do myself in my ode solver. Like this

```
parse_single_ode:=proc(ode::`=`,func::function(name))::symbol,symbol,integer;
#parses single ode. returns back  dep_var,indep_var,ode_order
#throws exception when it detects parsing errors

local y,x;
local dep_variables_found::list,item;
local the_order;

    if nops(func)<>1 then
```

```
      error("Parsing error, dependent variable must contain one argument, found ", func
   fi;

   y:=op(0,func);
   x:=op(1,func);

   #basic verification
   if not has(ode,y) then
      error ("Supplied ode ",ode," has no ",y);
   fi;

   if not has(ode,x) then
      error ("Supplied ode ",ode," has no ",x);
   fi;

   if not has(ode,func) then
      error ("Supplied ode ",ode," has no ",func);
   fi;

   the_order := PDEtools:-difforder(ode,x);

   #note that the following call will also return y(x) if the input is not an ode
   #this will check that the dependent variable will show with SAME argument in the o
   #i.e. if y(x) and y(t) show up in same ode, it will throw exception, which is what
   #we want.
   try
      dep_variables_found := PDEtools:-Library:-GetDepVars([y],ode);
      #print("dep_variables_found=",dep_variables_found);
   catch:
      error lastexception;
   end try;

   #now go over dep_variables_found and check the independent variable is same as x
   #i.e. ode can be y'(z)+y(z)=0 but function is y(x).
   for item in dep_variables_found do
       if not type(item,function) then
          error("Parsing error. Expected ",func," found ",item," in ode");
       else
          if op(1,item) <> x then
             error("Parsing error. Expected ",func," found ",item," in ode");
          fi;
```

```
        fi;
    od;

    #now go over all indents in ode and check that y only shows as y(x) and not as jus
    #as the PDEtools:-Library:-GetDepVars([_self:-y],ode) code above does not detect t
    #i.e. it does not check  y'(x)+y=0

    if numelems(indets(ode,identical(y))) > 0 then
       error("Parsing error, Can not have ",y," with no argument inside ",ode);
    fi;

    return y,x,the_order;
end proc:
```

To use do the same as before, but need to add $y(x)$ as second argument. Like this

```
y,x,the_order := parse_single_ode(diff(y(x),x)+y(x)=0, y(x));
```

3.14 How to check if single ODE is valid?

This is very much the same as above, but this function returns True if the ode is syntactically valid, else false. Can be used just to check if the ode is valid before using it.

It takes in the ode and the function $y(x)$ and returns either true or false.

```
#March 15, 2024

export is_valid_single_ode :=proc(ode_in::`=`,func::function(name))::truefalse;
    local x:=op(1,func),y:=op(0,func);
    local ode:=ode_in;
    local dep_variables_found::list;
    local item;

    if nops(func)<>1 then RETURN(false); fi;

    if not has(ode,diff) then ode:=convert(ode,diff); fi;

    if `or`(not has(ode,diff), not has(ode,x), not has(ode,func))  then
       RETURN(false);
    fi;
```

```
    try
        dep_variables_found := PDEtools:-Library:-GetDepVars([y],ode);
    catch:
       RETURN(false);
    end try;

    map( X-> `if`( `or`(not type(X,function), op(1,X) <> x) ,RETURN(false),NULL),
       dep_variables_found
    );

    #check there is no y on its own. Should always be y(x)
    if nops(indets(ode,identical(y))) <> 0 then
       RETURN(false);
    fi;

    RETURN(true);
end proc:
```

Example usages

```
is_valid_single_ode(diff(y(x),x)+sin(x)=0, y(x) )

       true
```

To test, do

```
L:=[[3*(D@@2)(y)(x)+diff(y(x),x)+1=sin(x),y(x),true],
                  [diff(y(x),x)^2=z,y(x),true],
                  [diff(y(x),x)^2=z,y(x),true],
                  [diff(y(x),x)^2=y(x)^2,y(x),true],
                  [diff(y(x),x)=y(z)^2,y(x),false],
                  [1/diff(y(x),x)=y(x)^2,y(x),true],
                  [y(x)=1,y(x),false],
                  [diff(y(x),z)=1,y(x),false],
                  [D(y)(x)=1,y(x),true],
                  [(D@@2)(y)(x)+diff(y(x),x)=sin(x),y(x),true],
                  [diff(y(x),x)+diff(y(z),z)=1,y(x),false],
                  [diff(y(x),x)+diff(g(z),z)=1,y(x),true]
                  ]:
```

```
map(X->evalb(is_valid_single_ode(X[1],X[2])=X[3]),L);
if andmap(x->evalb(x=true),%) then
   print("all tests passed");
else
   print("WARNING, not all tests passed");
fi;

#gives

 [true, true, true, true, true, true, true, true, true, true, true, true]
                        "all tests passed"
```

3.15 How to check if an ode has y' in it?

Suppose to want to check if an ode has $y'(x)$ in it. We can not write `has(ode, diff(y(x),x))` and see if this gives true or not, because this will also match y'' and y''' and any higher order.

One way is to first convert the ode to `D` form and then use `has` on `D(y)(x)`. This will not match y'' anymore which is what we wanted, because y'' becomes `(D@@2)(y)(x)` and so the check for first order diff will work as expected.

```
ode:=diff(y(x),x$2)+3*y(x)=0;
ode:=convert(ode,D);
has(ode,(D)(y)(x))

            #false
```

Another example

```
ode:=diff(y(x),x$2)+1/diff(y(x),x)=0;
ode:=convert(ode,D);
has(ode,(D)(y)(x))

            #true
```

The same thing if we wanted to check if the ode has y'' or not.

```
ode:=diff(y(x),x$3)+diff(y(x),x)=0;
ode:=convert(ode,D);
has(ode,(D@@2)(y)(x))
```

```
      #false
```

3.16 How to check if an ode is linear ode?

```
ode:=diff(y(x),x$3)+ x*diff(y(x),x)+sin(x)=0;
if has(DEtools:-odeadvisor(ode,y(x),['linear']),_linear) then
   print("linear");
else
   print("not linear ode");
fi
```

The above prints `"linear"`

```
ode:=y(x)*diff(y(x),x$3)+ x*diff(y(x),x)+sin(x)=0;
if has(DEtools:-odeadvisor(ode,y(x),['linear']),_linear) then
   print("linear");
else
   print("not linear ode");
fi
```

The above prints `"not linear ode"`. This works for any ode order.

3.17 How to find the order of an ode?

```
ode:=diff(y(x),x$3)+ x*diff(y(x),x)+sin(x)=0;
PDEtools:-difforder(ode)
```

Gives 3.

3.18 How to find the coefficients of a linear ode?

Given a linear ode of the form $Ay'' + By' + Cy = f(x)$ how to find $A, B, C, f(x)$?

```
ode:=diff(y(x),x$3)+ x*diff(y(x),x)+99*y(x)=sin(x);
L:=DEtools:-convertAlg(ode,y(x)); #this only works on linear ode's
```

Gives `L := [[99, x, 0, 1], sin(x)]`.

L is a list. The second entry is $f(x)$ and the first entry of L is a list which gives the coefficients

of the ode. Notice they are ordered from lowest order to highest order of the ode. Since this is third order ode, then there are 4 entries. The first is the coefficient of $y(x)$, the second is the coefficient of y' and the third is the coefficient of y'' and the fourth entry is the coefficient of y'''. Notice that coefficient of y'' is zero since y'' is missing from the ode.

This function `DEtools:-convertAlg` only works on linear ode's. Therefore we need to check if the ode is linear first before using. How to check for linear ode is given above.

3.19 How to find the coefficients of any ode, linear or not and any order?

This function below finds the ode coefficients for any ode, linear or not. It takes as input the ode and returns back two items. The first is list of coefficients, and the second is the RHS (i.e. $f(x)$).

So give an ode such as $3y'' + 5y' = \sin x$ it will return `[0,5,3],sin(x)` where the order of coefficients is $y, y', y'', \ldots$. If terms is missing, zero is placed there.

```
#version Dec 3, 2024, 3PM. by Nasser M. Abbasi
get_ode_coefficients_any_order:=proc(ode_in::`=`,func::function(name),$)::list,anything;

#find coefficients. For example, for 5*y''''+y'''+y'+3*y=sin(x); it will return
#   [3,1,0,1,5],sin(x)
#Where the list is in order y,y',y'',y''',y''''

local y:=op(0,func);
local x:=op(1,func);
local ode:=ode_in,i,j,tmp;
local a::list;
local c:=0;
local f:=0;
local ode_order::integer;
local found_diff::truefalse;

    ode:= expand(lhs(ode))-expand(rhs(ode));
    ode_order:= PDEtools:-difforder(ode,x);
    if ode_order<1 then
        error "ODE expected in get_ode_coefficients_any_order, found ", ode;
    fi;

    a:=[0$ode_order+1]; #for [y,y',y'',...,y(n)], that is why added 1
```

```
    if ode::`+` then
        for i from 1 to nops(ode) do
            tmp := op(i,ode);
            found_diff:=false;
            for j from ode_order to 1 by -1 do
                if has(tmp, diff(y(x),x$j) ) then
                    a[j+1] := a[j+1] + tmp/ diff(y(x),x$j);
                    found_diff:=true;
                    break;
                fi;
            od;
            if not found_diff then
               if has(tmp,y(x)) then
                    a[1] := a[1] + tmp/y(x);
                else
                    f := f+tmp;
                fi;
            fi;
        od;
    else
        found_diff:=false;
        for j from ode_order to 1 by -1 do
            if has(ode, diff(y(x),x$j) ) then
               a[j+1] := ode/ diff(y(x),x$j);
               found_diff:=true;
               break;
            fi;
        od;
        if not found_diff then
            if has(ode,y(x)) then
                a[1] := ode/y(x);
            else
                f :=ode;
            fi;
        fi;
    fi;

    return a,(-f);

end proc:
```

Examples

```
ode:=sin(x)+y(x)*diff(y(x),x$5)+diff(y(x),x)-1/x+99*diff(y(x),x$3)=0;
get_ode_coefficients_any_order(ode,y(x))
                                                    1
              [0, 1, 0, 99, 0, y(x)], -sin(x) + -
                                                    x

ode:=diff(y(x),x$3)=0:
get_ode_coefficients_any_order(ode,y(x))
                         [0, 0, 0, 1], 0

ode:=diff(y(x),x$3)=sin(y(x)):
get_ode_coefficients_any_order(ode,y(x))

                   [  sin(y(x))          ]
                   [- ---------, 0, 0, 1], 0
                   [    y(x)             ]


ode:=sin(x)+1/x*diff(y(x),x$3)+y(x)=exp(x)+9;
get_ode_coefficients_any_order(ode,y(x))

                [         1]
                [1, 0, 0, -], -sin(x) + exp(x) + 9
                [         x]
```

3.20 find order and degree of highest derivative

I need to find the degree of the highest derivative in an ode.

For example, if the input is

$$x\left(\frac{d^2}{dx^2}y(x)\right)^5 + x\left(\frac{d^2}{dx^2}y(x)\right)^2 + \left(\frac{d}{dx}y(x)\right)y(x) + \left(\frac{d^3}{dx^3}y(x)\right)^4 \sin(x) + 5y(x) + \sin(x) + \frac{d^6}{dx^6}r(x) + \left(\frac{d^3}{dx^3}y(x)\right)^4 \cos(x) = 0$$

Then the degree for highest derivative of y w.r.t. x is 4. For

$$y''(x)^2 + y'(x) + y(x) = 0$$

It will be 2.

This function returns the order and degree of such term.

```
find_all_derivatives_of_specific_order:=proc(expr,y::symbol,x::symbol,N::posint)::set;
local t1,t2;

if not has(expr,diff(y(x),x$N)) then
   return {};
fi;

t1  := identical(diff(y(x),x$N))^anything;
t2  := identical(diff(y(x),x$N));
return indets[flat](expr,{t1,t2});  #MUST use flat

end proc:

get_order_and_degree_of_largest_derivative:=proc(expr,y::symbol,x::symbol)::integer,anyt
local the_order,the_degree;
local cand::set;
local the_exponent;
local item;

    if not has(expr,diff(y(x),x)) then return 0,0; fi;
    the_order := PDEtools:-difforder(expr,x);
    cand       := find_all_derivatives_of_specific_order(expr,y,x,the_order);

    if nops(cand)=0 then
        the_degree := 0;
    else
        the_degree:=1;
        for item in cand do
            if type(item,`^`) then
                the_exponent := op(2,item);
                if type( the_exponent,symbol) or not type( the_exponent,numeric) then #d
                    the_degree :=the_exponent;
                else
                    if type(the_degree,symbol) or not  type( the_exponent,numeric) then
                        next;
                    else
                        if op(2,item)>the_degree then
```

```
                        the_degree := op(2,item);
                    fi;
                fi;
            else
                if type(the_degree,symbol) then
                    next;
                else
                    if the_degree=0 then
                        the_degree := 1;
                    fi;
                fi;
            fi;
        od;
    fi;

    return the_order,the_degree;
end proc:
```

And now it can be called as follows

```
ode:=diff(y(x),x$2)+diff(y(x),x)+y(x)=0;
get_order_and_degree_of_largest_derivative(lhs(ode),y,x)

    #2,1

ode:=diff(y(x),x$2)*diff(y(x),x$2)^4+diff(y(x),x)+y(x)=0;
get_order_and_degree_of_largest_derivative(lhs(ode),y,x)

   #2,5

ode:=y(x)=0;
get_order_and_degree_of_largest_derivative(lhs(ode),y,x)

   #0,0

ode:=diff(y(x),x)^n=0;
get_order_and_degree_of_largest_derivative(lhs(ode),y,x)

   #1,n
```

3.21 How to move all derivatives to one side in an equation?

Given

$$a\left(\frac{d}{dx}y(x)\right)+b\left(\frac{d^2}{dx^2}y(x)\right)+x+\cos(x)+y(x)+c\left(\frac{d}{dx}y(x)\right)^2=\sin(x)$$

How to move all terms with derivative to LHS side and everything else to RHS?

```
ode:= a*diff(y(x),x)+b*diff(y(x),x$2)+x+cos(x)+y(x)+c*diff(y(x),x)^2=sin(x);
ode:=lhs(ode)-rhs(ode);
LHS,RHS:=selectremove(has,ode,'diff');
new_ode:=LHS=-RHS;
```

$$a\left(\frac{d}{dx}y(x)\right)+b\left(\frac{d^2}{dx^2}y(x)\right)+c\left(\frac{d}{dx}y(x)\right)^2=-x-\cos(x)-y(x)+\sin(x)$$

3.22 How to obtains list of all derivatives in expression?

I had a need to find all derivatives of form `diff(y( anything), anything)` in an ode so to check that y argument is not different among them.

For an example, given

$$\begin{aligned}a\left(\frac{d^2}{dx^2}y(x)\right)\left(\frac{d^3}{dx^3}y(x)\right)-\sqrt{1+\left(\frac{d^2}{dx^2}y(x)\right)^2}+\frac{1}{\sin\left(\frac{d^5}{dx^5}y(x)\right)}\\=\frac{d}{dz}y(z)+\mathrm{e}^{y(x)+\frac{d}{dx}y(x)}+\frac{d}{dx}r(x)\end{aligned}$$

the result should be

$$\begin{bmatrix}\frac{d^5}{dx^5}y(x)\\\frac{d^4}{dx^4}y(x)\\\frac{d^3}{dx^3}y(x)\\\frac{d^2}{dx^2}y(x)\\\frac{d}{dx}y(x)\\\frac{d}{dz}y(z)\end{bmatrix}$$

One issue is how to check for diff and also check that the dependent variable is y so as not to pick other dependent variables such as z in this example. This was done by converting `diff` to `D` otherwise it will not work.

```
restart;
expr:=a*diff(y(x),x$2)*diff(y(x),x$3)-sqrt(1+ diff(y(x),x$2)^2)+1/sin(diff(y(x),x$5))=di

#this finds all derivatives
list_of_diffs:=indets(expr,'satisfies'(s->op(0,s)='diff' and op([0,1],convert(s,D))=y));

#This finds all dependent variables
list_of_diffs:=convert(list_of_diffs,list);
list_of_indep_variables   := map(x->PDEtools:-Library:-GetIndepVars(x)[-1],list_of_diffs

#this converts it to set. If the ODE is valid, then the list_of_indep_variables should
#have one entry $x$ in it and nothing else.
list_of_indep_variables   := convert(ListTools:-Flatten(list_of_indep_variables),set);

if nops(list_of_indep_variables)>1 then
  error( cat("Only one independent variable expected in differential form, found ",
                  convert(list_of_indep_variables,string)) );
fi;

if list_of_indep_variables[1]<>x then
   error( cat("Independent variable expected in differential form not same as independen
      convert(list_of_indep_variables,string)) );
fi;
```

Another option instead of doing all the above is to do this

```
expr:=a*diff(y(x),x$2)*diff(y(x),x$3)-sqrt(1+ diff(y(x),x$2)^2)+1/sin(diff(y(x),x$5))=di
try
   PDEtools:-Library:-GetDepVars([y(x)],expr);
catch
  error "functions with name [y] having different dependency: [[y(x), y(z)]]"
end try;
```

The function `PDEtools:-Library:-GetDepVars([y(x)],expr)` checks that only $y(x)$ dependency shows up. It throws an error otherwise. So if an error is thrown, then this means y shows up with different independent variables.

3.23 How to invert roles of dependent variable and independent variable in an ode?

Sometimes it is useful to invert an ode. i.e. make the independent variable the dependent variable, and the dependent variable the independent. For example, given

$$1+\left(\frac{x}{y\left(x\right)}-\sin\left(y(x)\right)\right)\left(\frac{d}{dx}y(x)\right)=0$$

We want the ode to become

$$-\sin\left(y\right)y+y\left(\frac{d}{dy}x(y)\right)+x(y)=0$$

This can be done as follows

```
restart;

ode:=1+ (x/y(x)-sin(y(x) ))*diff(y(x),x)=0;
tr:={x=u(t),y(x)=t};
ode:=PDEtools:-dchange(tr,ode);
ode:=eval(ode,[t=y,u=x]);
ode:=simplify(ode);
```

$$\frac{-\sin\left(y\right)y+y\left(\frac{d}{dy}x(y)\right)+x(y)}{y\left(\frac{d}{dy}x\left(y\right)\right)}=0$$

In this case, we can get rid of the denominator, but this is a manual step for now.

```
ode:=numer(lhs(ode))=0;
```

$$-\sin\left(y\right)y+y\left(\frac{d}{dy}x(y)\right)+x(y)=0$$

The above can now be solved more easily for $x(y)$ than solving the original ode for $y(x)$.

3.24 How to find the indicial equation for an ODE?

For say Bessel ode of order zero:

```
eq:= x^2*diff(y(x),x$2)+x*diff(y(x),x)+x^2*y(x)=0;
DEtools[indicialeq](eq,x,0,y(x));
          #x^2 = 0
```

The third argument above is the singularity point of interest. So we have two roots, both zero. These are now used for finding the power series solution $y(x)$ if needed.

Another example, is Bessel of order 1

```
eq:= x^2*diff(y(x),x$2)+x*diff(y(x),x)+(x^2-1)*y(x)=0;
DEtools[indicialeq](eq,x,0,y(x));
          #x^2-1 = 0
```

3.25 How to write derivative

To write $y'(x) = x$, one way is `diff(y(x),x)=x` and another is `D(y)(x)=x`. To write $y''(x) = x$, one way is `diff(y(x),x$2)=x` and another is `(D@@2)(y)(x)=x`.

To convert from one form to another use `convert(eq,diff)` or `convert(eq,D)`

3.26 How to solve heat PDE in 1D in Maple 2017?

to solve $\frac{\partial u(x,t)}{\partial t} = k\frac{\partial^2 u(x,t)}{\partial x^2}$ with homogeneous dirichlet boundary conditions $u(0,t) = 0, u(L,t) = 0$ the commands are

```
restart;
pde:=diff(u(x,t),t)=k*diff(u(x,t),x$2);
bc:=u(0,t)=0,u(L,t)=0;
sol:=pdsolve([pde,bc]) assuming 0<L:
```

Which gives

$$u(x,t) = \sum_{\_Z1=1}^{\infty} \_C1(\_Z1) \sin\left(\frac{\pi \_Z1\, x}{L}\right) \mathrm{e}^{-\frac{k\pi^2 \_Z1^2 t}{L^2}}$$

Which can be made more readable as follows

```
sol:=algsubs(_Z1=n,sol):
```

```
sol:=algsubs(Pi*n/L=lambda(n),sol);
```

$$u(x,t) = \sum_{n=1}^{\infty} \_C1(n) \sin(x\lambda(n)) \, \mathrm{e}^{-kt(\lambda(n))^2}$$

For homogeneous Neumann B.C., at $x = 0$, let $\frac{\partial u}{\partial x} = 0$ and at $x = L$ let $u(L,t) = 0$, the solution it gives looks different than my hand solution

```
restart;
pde:=diff(u(x,t),t)=k*diff(u(x,t),x$2);
bc:=D[1](u)(0,t)=0,u(L,t)=0;
pdsolve([pde,bc]) assuming 0<L;
```

It gives

$$u(x,t) = \_C3 \, \_C2 \left(\mathrm{e}^{1/4 \, \frac{2\,i\pi\,xL - k\pi^2 t}{L^2}} + \mathrm{e}^{-1/4 \, \frac{\pi\,(2\,ixL + k\pi\,t)}{L^2}} \right)$$

I need to look more into the above and see if this comes out to be the same as my hand solution.

Another example, with initial conditions now given

```
restart;
pde:=diff(u(x,t),t)=k*diff(u(x,t),x$2);
bc:=D[1](u)(0,t)=0,u(L,t)=0;
ic:=u(x,0)=f(x);
sol:=pdsolve([pde,bc,ic],u(x,t)) assuming 0<L;
sol1:=algsubs(_Z2=n,sol);
```

The result is

$$u(x,t) = \sum_{n=1}^{\infty} \left(2 \, \frac{1}{L} \mathrm{e}^{-1/4 \, \frac{k\pi^2 t(1+2\,n)^2}{L^2}} \cos\left(1/2 \, \frac{\pi\,x(1+2\,n)}{L} \right) \int_0^L f(x) \cos\left(1/2 \, \frac{\pi\,x(1+2\,n)}{L} \right) \, \mathrm{d}x \right)$$

Another example

```
restart;
pde:=diff(u(x,t),t)=k*diff(u(x,t),x$2);
```

```
bc:=D[1](u)(0,t)=0,u(L,t)=0;
ic:=u(x,0)=3*sin(Pi*x/L)-sin(3*Pi*x/L);
sol:=pdsolve([pde,bc,ic],u(x,t)) assuming 0<L;
sol1:=algsubs(_Z2=n,sol);
```

$$u(x,t)=\sum_{n=1}^{\infty}768\,\frac{1}{\pi\,(16\,n^4+32\,n^3-136\,n^2-152\,n+105)}\mathrm{e}^{-1/4\,\frac{k\pi^2t(1+2\,n)^2}{L^2}}\cos\left(1/2\,\frac{\pi\,x(1+2\,n)}{L}\right)$$

Another example

```
restart;
pde:=diff(u(x,t),t)=k*diff(u(x,t),x$2);
bc:=u(0,t)=0,u(L,t)=0;
ic:=u(x,0)=3*sin(Pi*x/L)-sin(3*Pi*x/L);
sol:=pdsolve([pde,bc,ic],u(x,t)) assuming 0<L;
```

$$u(x,t)=\sin\left(\frac{\pi\,x}{L}\right)\mathrm{e}^{-9\,\frac{\pi^2kt}{L^2}}\left(-2\,\cos\left(2\,\frac{\pi\,x}{L}\right)+3\,\mathrm{e}^{8\,\frac{\pi^2kt}{L^2}}-1\right)$$

The above answer seems wrong. There is not even a summation in it. It is different from my hand solution. Look more into it.

3.27 How to make Maple display diff(y(x),x) as y'(x) or as y' ?

Add this

```
expr:=diff(y(x),x);
Typesetting:-Settings(typesetprime=true, prime=x):
```

The above will display the expression as $y'(x)$. To make it now show the x do

```
expr:=diff(y(x),x);
Typesetting:-Settings(typesetprime=true, prime=x):
Typesetting:-Suppress(y(x));
```

Now it will show the expression as just y'. For all the above to work, make sure you have `Typesetting level` set to `Extended` in the GUI.

This is done inside `Tools->Options->Display` menu.

To clear all the above `Typesetting`, do `restart` or do `Typesetting:-Unsuppress(y(x))`

3.28 How to set boundary conditions for dsolve or pdsolve?

The Maple syntax for setting initial and boundary conditions is very confusing, as compared to Mathematica, which seems to me to be simpler. So I wrote this to remind me of the syntax each time.

For PDE, assuming dependent variable is $u(x,t)$ then

Conditions	Maple code
$u(0,t)=0$	`u(0,t)=0`
$\frac{\partial u}{\partial x}=0$ at $x=0$	`D[1](u)(0,t)=0`
$\frac{\partial^2 u}{\partial x^2}=0$ at $x=0$	`D[1,1](u)(0,t)=0`
$\frac{\partial^3 u}{\partial x^3}=0$ at $x=0$	`D[1,1,1](u)(0,t)=0`
$\frac{\partial u}{\partial t}=0$ at $t=0$	`D[2](u)(x,0)=0`
$\frac{\partial^2 u}{\partial t^2}=0$ at $t=0$	`D[2,2](u)(x,0)=0`
$\frac{\partial^3 u}{\partial t^3}=0$ at $t=0$	`D[2,2,2](u)(x,0)=0`

Notice the syntax for the last one above. It is `(D[1]@@2)(u)(0,t)=0` and not `(D@@2)[1](u)(0,t)=0`

For an ODE, assuming dependent variable is $y(x)$ then the syntax is

Conditions	Maple code
$y(0)=0$	`y(0)=0`
$\frac{dy}{dx}=0$ at $x=0$	`D(y)(0)=0`
$\frac{d^2y}{dx^2}=0$ at $x=0$	`(D@@2)(y)(0)=0`

3.29 How force dsolve to use Lie?

Use `dsolve(ode,Lie)`

To find symmetries, do

```
DEtools:-symgen(ode,y(x),HINT=[c__1+c__2*x+c__3*y,c__4+c__5*x+c__6*y])
```

or just

```
DEtools:-symgen(ode,y(x))
```

To debug it do

`stopat(‘ODEtools/symgen‘);` before calling `dsolve` or `DEtools:-symgen`

3.30 How to do change of variables on the dependent variable for an ODE?

given an ode

$$y(x) = \left(\frac{d}{dx}y(x)\right)^3 y(x)^2 + 2x\left(\frac{d}{dx}y(x)\right)$$

do change of variable $u(x) = y(x)^2$

```
restart;
ode:=y(x)=diff(y(x),x)^3*y(x)^2+2*x*diff(y(x),x);
new_ode:=PDEtools:-dchange({y(x)=sqrt(u(x))},ode,{u});
```

$$\sqrt{u(x)} = \frac{\left(\frac{d}{dx}u(x)\right)^3}{8\sqrt{u(x)}} + \frac{x\left(\frac{d}{dx}u(x)\right)}{\sqrt{u(x)}}$$

3.31 How to do change of variable on the independent variable for an ODE?

given an ode

$$\frac{d^2}{dt^2}y(t) + y(t) = 2t$$

do change of variable $t = \tau + \pi$

```
restart;
ode:=diff(y(t),t$2)+y(t)=2*t;
PDEtools:-dchange({t=tau+Pi},ode,known={t},unknown={tau},params=Pi)
```

$$\frac{d^2}{d\tau^2}y(\tau) + y(\tau) = 2\tau + 2\pi$$

it is important to use `params=Pi`. Watch what happens if we do not do that

```
restart;
ode:=diff(y(t),t$2)+y(t)=2*t;
PDEtools:-dchange({t=tau+Pi},ode,known={t},unknown={tau});
```

$$\frac{d^2}{d\tau^2}y(\tau,\pi)+y(\tau,\pi)=2\tau+2\pi$$

Which is not what we want.

3.32 ODE change of variable on both dependent and independent variable?

This verifies solution given in `https://math.stackexchange.com/questions/3477732/can-t-see-that-an-ode-is-equivalent-to-a-bessel-equation` Where a change of variables on

$$\xi^2\frac{d^2\eta}{d\xi^2}+\xi\frac{d\eta}{d\xi}-(\xi^2+n^2)\eta=0$$

Was made using

$$\eta=\frac{y}{x^\alpha},\quad \xi=\beta x^\gamma,$$

To produce the ode

$$\frac{d^2y}{dx^2}-\frac{(2\alpha-1)}{x}\frac{dy}{dx}-(\beta^2\gamma^2x^{2\gamma-2}+\frac{n^2\gamma^2-\alpha^2}{x^2})y=0.$$

In Maple the above is done using

```
restart;
ode := zeta^2*diff(eta(zeta),zeta$2) + zeta*diff(eta(zeta),zeta) - (zeta^2 + n^2)*eta(ze
the_tr:={zeta=beta*x^gamma,eta(zeta)=y(x)/x^alpha};
PDEtools:-dchange(the_tr,ode,{y(x),x},'known'={eta(zeta)},'uknown'={y(x)},'params'={alph
simplify(%);
numer(lhs(%))=0;
simplify(numer(lhs(%))/(x^(1-alpha)))=0;
numer(lhs(%))=0;
collect(%,[y(x),diff(y(x),x),diff(y(x),x$2)]);
```

Which gives

$$\left(-\gamma^2x^{-1+2\gamma}\beta^2x-\gamma^2n^2+\alpha^2\right)y(x)+(-2\alpha x+x)\left(\frac{d}{dx}y(x)\right)+x^2\left(\frac{d^2}{dx^2}y(x)\right)=0$$

Here is another example. Here to make change of variables to polar coordinates by making $x=r\cos\theta$ and $y=r\sin\theta$ The ode is

$$\frac{y-xy'}{\sqrt{1+(y')^2}}=x^2+y^2$$

In Maple

```
restart;
ode := (y(x)-x*diff(y(x),x))/sqrt(1+ diff(y(x),x)^2) = x^2+y(x)^2;
the_tr:={x=r(t)*cos(t),y(x)=r(t)*sin(t)};
PDEtools:-dchange(the_tr,ode,{r(t),t},'known'={y(x)},'uknown'={r(t)});
```

Which gives

$$\frac{r(t)\sin(t)-\frac{r(t)\cos(t)\left(\left(\frac{d}{dt}r(t)\right)\sin(t)+r(t)\cos(t)\right)}{\left(\frac{d}{dt}r(t)\right)\cos(t)-r(t)\sin(t)}}{\sqrt{1+\frac{\left(\left(\frac{d}{dt}r(t)\right)\sin(t)+r(t)\cos(t)\right)^2}{\left(\left(\frac{d}{dt}r(t)\right)\cos(t)-r(t)\sin(t)\right)^2}}}=r(t)^2\left(\cos^2(t)\right)+r(t)^2\left(\sin^2(t)\right)$$

Here is another example. Where we want to change $R(r)$ to $y(x)$ everywhere

$$\frac{d^2}{dr^2}R(r)+\frac{d}{dr}R(r)+R(r)=0$$

```
restart;
ode:=diff(R(r),r$2)+diff(R(r),r)+R(r)=0;
the_tr:={r=x,R(r)=y(x)};
PDEtools:-dchange(the_tr,ode,{y(x),x},'known'={R(r),r},'uknown'={y(x),x});
```

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

The format of the transformation is `old_independent_variable=new_independent_variable` and `old_dependent_variable=new_dependent_variable`

3.33 How to make phase plot of first order ODE?

Make a phase plot of

$$y'(x)=\sqrt{y(x)^2-1}$$

The phase plot has x on the x axis and has y on the y axis. It shows the family of solutions for different initial conditions.

```
restart;
ode:=diff(y(x), x) = sqrt(y(x)^2 - 1);
DEtools:-DEplot(ode,y(x),x=-2..2,y=1..3)
```

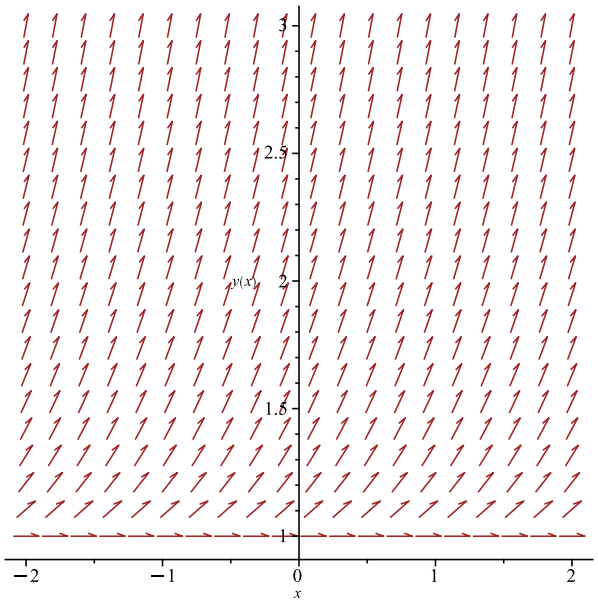


Figure 2: Phase Plot

To show specific solution curve that passes via some initial conditions such as $y(0) = 2$ then do

```
restart;
ode:=diff(y(x), x) = sqrt(y(x)^2 - 1);
DEtools:-DEplot(ode,y(x),x=-2..2,y=1..3,[y(0)=2])
```

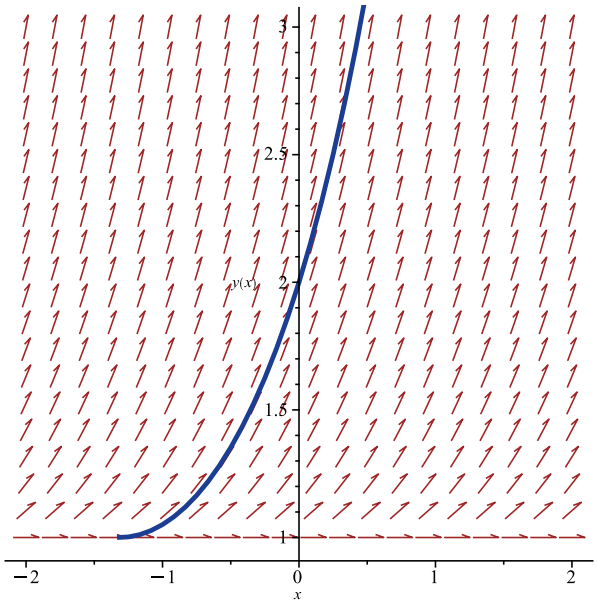


Figure 3: Phase Plot

3.34 How to make phase plot of second order ODE?

Make a phase plot of

$$\frac{d^2}{dt^2}x(t) + \frac{\frac{d}{dt}x(t)}{2} + x(t) = u(t)$$

By plotting $x(t)$ vs $x'(t)$ without solving the ODE.

```
restart;
alias(DS=DynamicSystems):
ode := diff(x(t),t$2) +1/2*diff(x(t),t)+ x(t) = u(t);
sys:=DS:-DiffEquation(ode,'outputvariable'=[x(t)],'inputvariable'=[u(t)]);
sys0:=DS:-StateSpace(sys);
eq1:=diff(x1(t),t)=sys0:-a[1,..].Vector([x1(t),x2(t)]);
eq2:=diff(x2(t),t)=sys0:-a[2,..].Vector([x1(t),x2(t)]);
DEtools:-DEplot([eq1,eq2],[x1(t),x2(t)],t=0..35,[[x1(0)=1,x2(0)=1]],x1=-2..2,x2=-2..2,
        numpoints=200, linecolor=black, axes=boxed);
```

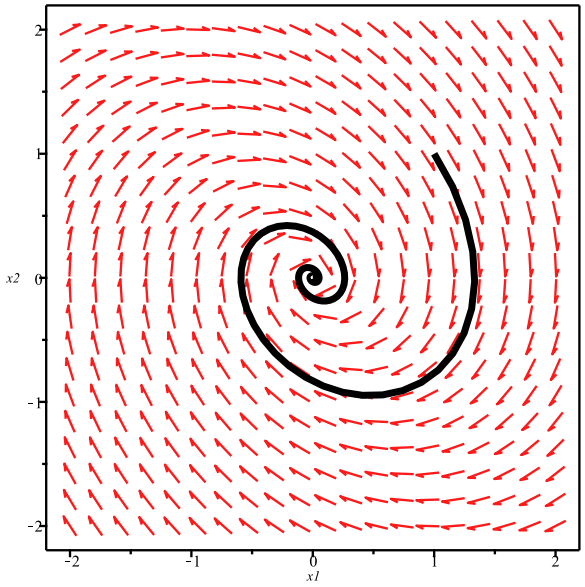


Figure 4: Phase Plot

3.35 How to move all terms with y to one side?

Given ode where y is dependent variable and x is independent variable, how to normalize the ode so that all terms with $y(x)$ are on left side and everything is on the RHS? This makes it easier to see what the forcing function is. i.e. we want to write the ode as

$$ay'' + by' + cy = f(x)$$

Here is function to do this. This works for any ode or any expression.

```
export move_y_x_to_each_side:=proc(expr_in::`=`,y::symbol,x::symbol)::`=`;
    local expr:=expr_in;
    local stuff_with_y,stuff_not_with_y;
    expr  := lhs(expr)-rhs(expr);
    if not expr::`+` then
      RETURN(expr_in);
    fi;
    expr  := numer(normal(expr));
    stuff_with_y,stuff_not_with_y := selectremove(has,expr,y);
    stuff_not_with_y := convert(stuff_not_with_y,diff);

    RETURN(stuff_with_y = -stuff_not_with_y);
end proc:
```

Examples

```
ode:=diff(y(x),x$2)+x*y(x)+sin(x)-3=tan(x);
expr:=move_y_x_to_each_side(ode,y,x);

        # diff(y(x), x, x) + x*y(x) = 3 - sin(x) + tan(x)
```

And

```
ode:=diff(y(x),x$2)+x*y(x)+sin(x)-3=tan(x)+y(x)+Pi;
expr:=move_y_x_to_each_side(ode,y,x);

        # diff(y(x), x, x) + x*y(x) - y(x) = 3 - sin(x) + tan(x) + Pi
```

And

```
ode:=move_y_x_to_each_side(diff(x(t),t$2)/x(t) + diff(x(t),t)/x(t) - F(t)/x(t) =1,x,t)
```

```
expr:=move_y_x_to_each_side(ode,x,t);

        # diff(x(t), t, t) + diff(x(t), t) - x(t) = F(t)
```

3.36 How to find all constants of integration in an expression?

Sometimes I make solution for an ode with constants of integration. These must be of form `_Cn` where n is positive integer, such as `_C1` or `_C5`. Now to integrate this solution once more, need to come up with new constant that does not already show in the solution.

To find current constants in an expression use this

```
sol:=x-_C1*x+_C3*x^2;
indets(sol,And(symbol, suffixed(_C, nonnegint)));

         {_C1, _C3}
```

Therefore, one way to make a new constant of integration, is to find the largest numbered one and increase by one. Like this

```
restart;
sol:=x-_C1*x+_C3*x^2;
myconstants:=indets(sol,And(symbol, suffixed(_C, nonnegint)));
map(X->String(X),myconstants);
map(X->X[3..],%);
map(X->:-parse(X),%);
n:=max(%);
new_constant:=_C||(n+1);
```

3.37 How to find if an ode is missing the dependent variable?

Sometimes it is useful to know if ode is missing $y(x)$ as this allows one to do the substitution $y' = u$ and reduce the order of the ode.

This is a function which takes in an ode and returns true if it is missing $y(x)$ or false otherwise.

```
#added 2/18/2024. checks if an ode is missing y
export is_ode_missing_y:=proc(ode_in::`=`,y::symbol,x::symbol)::truefalse;
local ode:=ode_in;
local ode_order::posint := PDEtools:-difforder(ode_in);
local N::posint;
```

```
    ode:=lhs(ode)-rhs(ode);
    ode:=convert(ode,D);
    for N from 1 to ode_order do
        ode:=eval(ode,[(D@@N)(y)(x)=Y||N]);
    od;

    RETURN(not has(ode,y(x)));

end proc;
```

To use do

```
restart;
ode:=diff(y(x),x$2)+diff(y(x),x$4)+x=0;
is_ode_missing_y(ode,y,x);

       #true

ode:=diff(y(x),x$2)+diff(y(x),x$4)+x=sin(x)+y(x);
is_ode_missing_y(ode,y,x);

      #false
```

3.38 How to convert Riccati general ode to second order ode?

The general Riccati ode is

$$y'(x) = f_0(x) + f_1(x)y(x) + f_2(x)y^2(x)$$

This can be converted to second order linear ode in $u(x)$ using the transformation

$$y = \frac{-u'}{f_2 u}$$

Which results, after some simplifications in the ode

$$u'' - \left(\frac{f_2'}{f_2} + f_1\right) u' + f_0 f_2 u = 0$$

The following Maple code does the above

```
riccati_ode:= diff(y(x),x)= f__0(x)+f__1(x)*y(x)+f__2(x)*y(x)^2;

Typesetting:-Unsuppress('all'); #always do this.
Typesetting:-Settings(prime=x,'typesetprime'=true); #this says to use y'(x) instead of
Typesetting:-Suppress(u(x)); # this says to use y' and not y'(x)

PDEtools:-dchange( {y(x)= -diff(u(x),x)/(f__2(x)*u(x))},riccati_ode,{u});
(lhs-rhs)(%);
numer(normal(%));
collect(%,[diff(u(x),x),diff(u(x),x$2)]);
map(X->X/f__2(x),%);
new_U_ode:=-%=0;
```

Gives

```
   new_U_ode := -(f__2(x)*f__1(x) + diff(f__2(x), x))*diff(u(x), x)/f__2(x) + f__0(x)*f_
```

3.39 How to reverse the role of the dependent and independent variables in ode?

Sometimes an ode can be solved much easier if we change the roles, and make the independent variable as the dependent variable.

For example, the ode $y'(x) = \frac{1}{e^{4y(x)}+2x}$ which is not easy to solve, becomes $x'(y) = 2x(y) + e^{4y}$ which is linear. After solving the simpler ode, we solve for y from the result which gives the solution to the original ode.

To do the conversion, use the command `convert(ode, y_x)`. Below are examples

```
original_ode:= diff(y(x),x)=1/(exp(4*y(x))+2*x);
sol_of_original := dsolve(ode);

       y(x) = ln(-c__1 - sqrt(c__1^2 + 2*x))/2, y(x) = ln(-c__1 + sqrt(c__1^2 + 2*x))/2

new_ode:= convert(original_ode, y_x);

         diff(x(y), y) = exp(4*y) + 2*x(y)

sol_of_new := dsolve(new_ode);
sol_of_new := eval(sol_of_new,x(y)=x);
sol_of_new := y(x) = solve(sol_of_new,y);
```

```
       y(x) = (ln(-c__1 + sqrt(c__1^2 + 2*x))/2, ln(-c__1 - sqrt(c__1^2 + 2*x))/2)
```

We see that we get the same solution ofcourse, but solving ode after changing the roles is much easier now.

3.40 function to solve reduced Riccati ode in Maple?

This is a standalone function that solves $y' = ax^n + by^2$. This is called the specific or reduced Riccati ode. The formulas used are from Eqworld ode0106 and Dr Dobrushkin web page

```
#solves y' = a*x^n + b*y^2, using standard formulas for n=-2 and n<>-2

special_riccati_dsolve := proc(n,a,b,func::function(name),$)

    local y:=op(0,func);
    local x:=op(1,func);
    local sol;
    local C1,C2;

    if not (n::'integer' or n::'fraction') then
       print("argument n  must be integer or fraction");
       RETURN(NULL);
    fi;

    if n=-2 then
        proc()
            local z,sol_z,lambda;
            try
                sol_z:=timelimit(30,[solve(b*z^2+z+a=0,z)]);
                if nops(sol_z)=0 then
                   error "Unable to solve for root";
                fi;
            catch:
                error "Unable to solve for root";
            end try;

            z:=sol_z[1]; #pick one, any will work
            sol:=y(x)= z/x - x^(2*b*z)/( b*x/(2*b*z+1)*x^(2*b*z) + c__1);
        end proc();
    else
```

```
        proc()
            local k,w;
            k:=1+n/2;
            C1:= c__1;
            C2:= c__2;
            try
                if evalb(a*b >0) then
                    w:=  sqrt(x) * ( C1*BesselJ(1/(2*k),1/k*sqrt(a*b)*x^k) + C2* BesselY
                else
                    w:=  sqrt(x) * ( C1*BesselI(1/(2*k),1/k*sqrt(-a*b)*x^k) + C2* Bessel
                fi;
            catch:
                w:=  sqrt(x) * ( C1*BesselJ(1/(2*k),1/k*sqrt(a*b)*x^k) + C2* BesselY(1/(
            end try;
            sol:= simplify(-1/b*diff(w,x)/w);
            sol:= y(x)=eval(sol,C2=1);
        end proc();
    fi;

    RETURN(simplify(sol));

end proc:
```

Example usages

```
a:=1:n:=-2:b:=1:
ode:=diff(y(x),x)=a*x^n+b*y(x)^2;
sol:=special_riccati_dsolve(n,a,b,y(x));

    sol := y(x) = ((-x^(sqrt(3)*I) - 3*c__1)*sqrt(3)*I + 3*x^(sqrt(3)*I) + 3*c__1)/(2*(x

a:=5:n:=-8/3:b:=3:
ode:=diff(y(x),x)=a*x^n+b*y(x)^2;
sol:=special_riccati_dsolve(n,a,b,y(x));

   sol := y(x) = -sqrt(15)*((sqrt(15)*x - 15*c__1*x^(2/3) + c__1*x^(4/3)/3)*cos(3*sqrt(1



a:=5:n:=5:b:=2:
ode:=diff(y(x),x)=a*x^n+b*y(x)^2;
```

```
sol:=special_riccati_dsolve(n,a,b,y(x));

   sol := y(x) = -sqrt(10)*x^(5/2)*(BesselY(-6/7, (2*sqrt(10)*x^(7/2))/7) + BesselJ(-6/7
```

Ofcourse, dsolve can be used to solve these also.

3.41 How to guess a solution to an ode?

Sometimes we need to guess a particular solution to an ode. For example when solving the Riccati ode $y' = f_0 + f_1 y + f_2 y^2$.

Typical guesses for this ode are $y = ax^b$ and $y = ae^{bc}$. How can we find using Maple if such a guess works or not and what are the values of the constants a, b?.

To do this, we first use `odetest` on the guess. This gives some residual. Then we use `solve` command with the `identity`, to solve for the values a, b that would make the residual zero. If solve finds a solution, then we have found the guess solution.

`solve` with identity used below, basically says to find constants a, b which will make the residual zero for any x. This is why we must use `identity(the_residual=0,x)` below.

Here is an example

```
restart;

ode:=diff(y(x),x)=-(1+x+x^2)-(2*x+1)*y(x)-y(x)^2; #riccati ode
guess_1:=y(x)=a*x^b;
the_residual:= odetest(guess_1,ode);

# the_residual := a^2*x^(2*b) + 2*a*x^(1 + b) + a*b*x^(b - 1) + a*x^b + x^2 + x + 1

solve(identity(the_residual=0,x),[a,b])

        [[a = -1, b = 1]]
```

It worked. The above says that $y = -x^{-1}$ is a particular solution. Now we know a particular solution, we can solve the Riccati ode and find the general solution.

Let use try the other guess and see what happens

```
guess_2:=y(x)=a*exp(b*x);
ode:=diff(y(x),x)=-(1+x+x^2)-(2*x+1)*y(x)-y(x)^2;
the_residual:=odetest(guess_2,ode)
```

```
    #the_residual := a^2*exp(2*b*x) + a*b*exp(b*x) + 2*a*exp(b*x)*x + a*exp(b*x) + x^2

solve(identity(the_residual=0,x),[a,b])

      []
```

No solution. This means this guess did not work. This way we can make a number of guesses and try them hoping one will work. As we know, there is no general method to solve every Riccati ode. but if we know one particular solution, then it can be solved. So this method of guessing a solution can lead to a way to solve a Riccati ode which otherwise can be hard to solve using other methods.

This is another example, given ode

$$y' = \frac{4x^2+5x}{x-1} - \frac{4(2x+1)}{x-1}y + \frac{4}{x-1}y^2$$

Try to guess a solution. Since coefficients are polynomials of most degree 2, we start by a guess of form $y = ax + b$ and if this does not work, we try $y = ax^2 + bx + c$. But it is better just to use same order polynomial as the largest one in the coefficients.

```
restart;
ode:=diff(y(x),x)= (4*x^2+5*x)/(x-1)-4*(2*x+1)/(x-1)*y(x)+4/(x-1)*y(x)^2;
guess_1:=y(x)=a*x^2+b*x+c;

the_residual:=odetest(guess_1,ode)

#the_residual := -4*a^2*x^4/(x - 1) - 8*a*b*x^3/(x - 1) - 8*a*c*x^2/(x - 1)
                + 8*a*x^3/(x - 1) - 4*x^2*b^2/(x - 1) + 6*a*x^2/(x - 1)
                - 8*b*c*x/(x - 1) + 8*x^2*b/(x - 1) - 2*a*x/(x - 1) + 5*b*x/(x - 1)
                - 4*c^2/(x - 1) + 8*c*x/(x - 1) - 4*x^2/(x - 1) - b/(x - 1) + 4*c/(x -

solve(identity(the_residual=0,x),[a,b,c])


#[[a = 0, b = 1, c = 1/2], [a = 0, b = 1, c = 1/2], [a = 0, b = 1, c = 1/2], [a = 0, b
```

There are 4 different particular solutions. Using any one of these, now we can solve the Riccati ode easily using the transformation $y = y_p + u(x)$ or using $y = y_p + \frac{1}{u}$. Where y_p is any of the above 4 particular solutions.

4 Symbolic manipulation, expression filtering, structured types

4.1 on maple data types

See http://www.maplesoft.com/applications/view.aspx?SID=1533&view=html&L=G

4.2 how to extract elements from a list based on some selection?

use select. For example

```
>restart;
>my_list:=[1,3.4,3+I,5];
>select(x->evalb(Im(x)=0),my_list);
        [1, 3.4, 5]
```

4.3 how to test if all elements of a matrix are integers?

```
restart;
m:=Matrix( [[1.3,2,3],[3,4,4] ]);
matrixTestQ := proc(m::Matrix)

    local r,c,i,j;

    (r,c):=LinearAlgebra[Dimensions](m);
    for i from 1 to r do
       for j from 1 to c do
           if( not evalb( whattype(m[i,j]) = integer) )  then
               return(false);
            end if;
        end do;
    end do;

    return true;
end proc;

>matrixTestQ(m);

               false
```

I am sure there is a better way than the above. Need to find out.

4.4 How to select terms with sqrt or radical

Given

$$3+x+\sqrt{-4ac+b^2}+\sin(y)+x^3\sqrt{39}+\sqrt{\cos x}$$

Find terms that are sqrt. Use `indets`

```
restart;
expr_with_radical:= 3+x+sqrt(b^2-4*a*c)+sin(y)+x^3*sqrt(39)+sqrt(cos(x));|
indets(expr_with_radical, algebraic^fraction)
```

$$\{\sqrt{39},\sqrt{-4ac+b^2},\sqrt{\cos x}\}$$

Alternative is to use type `radical`

```
restart;
expr_with_radical:= 3+x+sqrt(b^2-4*a*c)+sin(y)+x^3*sqrt(39)+sqrt(cos(x));|
indets(expr_with_radical, radical)
```

$$\{\sqrt{39},\sqrt{-4ac+b^2},\sqrt{\cos x}\}$$

4.5 How to find all csgn() and replace them by 1

I wanted to simplify an expression which could have `csgn()` in it, and find all the arguments.

$$\frac{1+\operatorname{csgn}(a)\,a}{3\operatorname{csgn}(b)\,b}$$

One way is

```
restart;
expr:=(1+csgn(a)*a)/(3*csgn(b)*b):
expr:=subsindets(expr,'specfunc( anything, csgn )',f->1);
```

$$\frac{1+a}{3b}$$

4.6 How to find symbols inside csgn() in an expression?

Given `sol:=1/2*2^(1/2)*csgn(x)*x*csgn(y);` how to find all symbols inside csgn which will be x, y in this case?

```
restart;
sol:=1/2*2^(1/2)*csgn(x)*x*csgn(y);
indets(sol,'specfunc( anything, csgn )');
vars:=subsindets(%,'specfunc( anything, csgn )',f->op(f))
```

Gives `{x, y}`

Now if we want to simplify the above solution by assuming that all variables inside `vars` are positive, how to do that?

```
restart;
sol:=1/2*2^(1/2)*csgn(x)*x*csgn(y);
indets(sol,'specfunc( anything, csgn )');
vars:=subsindets(%,'specfunc( anything, csgn )',f->op(f));
simplify(sol) assuming op(map2(`<`,0,vars))
```

Gives $\frac{\sqrt{2}x}{2}$. Notice in the above the use of `op(map2('<',0,vars))`, this will generate the sequence `0 < x, 0 < y` automatically. `op` is needed otherwise the result will be `{0 < x, 0 < y}` which will give syntax error when passed to `assuming`

Ofcourse, it would have been also possible to just write

```
simplify(sol) assuming positive;
```

And get the same result. But sometimes we might want to specify which variables are to be assumed positive and not all of them at once in the expression.

4.7 How to replace all abs(expr) by expr

I wanted to replace `|expr|` by `(expr)`

One way is

```
restart;
expr:=u(x) = _C1*exp(-3*x^(1/3)*sqrt(c))*(3*x^(1/3)*sqrt(c) + 1) + _C2*exp(3*x^(1/3)*sqr
```

$$u(x) = \_C1\, \mathrm{e}^{-3x^{\frac{1}{3}}\sqrt{c}}\left(3x^{\frac{1}{3}}\sqrt{c}+1\right) + \_C2\, \mathrm{e}^{3x^{\frac{1}{3}}\sqrt{c}}|-1+3x^{\frac{1}{3}}\sqrt{c}|$$

```
expr:=subsindets(expr,'specfunc( anything, abs )',f->op(f));
```

$$u(x) = \_C1\, \mathrm{e}^{-3x^{\frac{1}{3}}\sqrt{c}}\left(3x^{\frac{1}{3}}\sqrt{c}+1\right) + \_C2\, \mathrm{e}^{3x^{\frac{1}{3}}\sqrt{c}}\left(-1+3x^{\frac{1}{3}}\sqrt{c}\right)$$

4.8 How to obtain list of all occurrences of some function in an expression?

For an example, How to find list of all ln functions in this expression?

$$\ln\left(|x+1|\right) + 2x\ln\left(x\right) + \sin\left(x\right)$$

```
restart;
expr:=ln(abs(x+1))+2*x*ln(x)+sin(x);
tmp := indets(expr,'specfunc(anything,ln)');

          #    tmp := {ln(x), ln(abs(x + 1))}
```

To pick only ln functions which has *abs* inside them anywhere, replace the above with

```
restart;
expr:=ln(abs(x+1))+2*x*ln(x)+sin(x);
lis:=indets(expr,'specfunc(anything,ln)');
select(Z->has(Z,abs),lis)

          #    tmp := {ln(abs(x + 1))}
```

Or, better alternative to the above is

```
restart;
expr:=ln(abs(x+1))+2*x*ln(x)+sin(x);
indets(expr,'specfunc( satisfies(u->has(u,abs)) ,ln  )');

          #    tmp := {ln(abs(x + 1))}
```

4.9 How to replace $\ln(|x|)$ with $\ln(x)$ in an expression?

Given

$$\sin(x)+\ln(|x|)+\ln\left(x+\frac{|y|}{\sqrt{|x+3|}}\right)+\ln\left(x^3\right)+\cos(|x|)$$

How to remove the absolute, the ones only inside each ln in the above expression?

```
restart;
expr:=sin(x)+ln(abs(x))+ln(x+abs(y)/sqrt(abs(x+3)))+ln(x^3)+cos(abs(x));
expr:=evalindets(expr,'specfunc(ln)',f->evalindets(f,'specfunc(abs)',f->op(1,f)))

#   sin(x) + ln(x) + ln(x + y/sqrt(x + 3)) + ln(x^3) + cos(abs(x))
```

$$\sin(x)+\ln(x)+\ln\left(x+\frac{y}{\sqrt{x+3}}\right)+\ln\left(x^3\right)+\cos(|x|)$$

4.10 How to find all signum functions in expression and simplify it?

Given

$$-\frac{\left(\ln\left(\frac{\left(b+\sqrt{b^2+y(x)^2}\,\mathrm{signum}(b)\right)b}{y(x)}\right)+\ln(2)\right)\mathrm{signum}(b)}{b}=\_C1+\frac{-\ln(a)+\ln(x)-\ln\left(a+\sqrt{a^2+x^2}\,\mathrm{signum}(a)\right)-$$

How to find all arguments of `signum` and simplify the above by assuming they are all positive?

```
restart;
expr:=-(ln((b + sqrt(b^2 + y(x)^2)*signum(b))*b/y(x)) + ln(2))*signum(b)/b = _C1 + (-ln(
lis:=indets(expr,'specfunc(anything,signum)');
assum:=convert(map(x->op(1,x)>0,lis),list);
simplify(expr,assume=assum);
```

$$\frac{-\ln(b)-\ln\left(\frac{b+\sqrt{b^2+y(x)^2}}{y(x)}\right)-\ln(2)}{b}=\frac{\_C1a-\ln(a)-\ln\left(a+\sqrt{a^2+x^2}\right)+\ln(x)-\ln(2)}{a}$$

4.11 How to replace all signum functions in expression by 1?

Given

$$-\frac{\left(\ln\left(\frac{\left(b+\sqrt{b^2+y(x)^2}\,\mathrm{signum}(b)\right)b}{y(x)}\right)+\ln(2)\right)\mathrm{signum}(b)}{b}=\_C1+\frac{-\ln(a)+\ln(x)-\ln\left(a+\sqrt{a^2+x^2}\,\mathrm{signum}(a)\right)-$$

How to replace all `signum` by 1?

```
restart;
expr:=-(ln((b + sqrt(b^2 + y(x)^2)*signum(b))*b/y(x)) + ln(2))*signum(b)/b = _C1 + (-ln(
evalindets(expr, 'specfunc(anything,signum)', f -> 1);
```

$$-\frac{\ln\left(\frac{\left(b+\sqrt{b^2+y(x)^2}\right)b}{y(x)}\right)+\ln(2)}{b}=\_C1+\frac{-\ln(a)+\ln(x)-\ln\left(a+\sqrt{a^2+x^2}\right)-\ln(2)}{|a|}$$

4.12 How to find if some function is present in an expression

Given expression $3\sin(x)+t+3f(x,t)\,t+g(x,t)$ find if it contains function $f()$.

Use `indets` with `specfunc(f)`

```
restart;
expr := 3*sin(x)+t+3*f(x,t)*t+g(x,t);
res  := indets(expr, specfunc(f));
if numelems(res)<>0 then
   print("Found f(x,t)");
else
   print("could not find f(x,t)");
fi;
```

"Found f(x,t)"

4.13 How to find all functions in an expression?

Given expression $3\sin(x)+t+3f(x,t)\,t+g(x,t)$ find all functions, if any, in the expression.

Use `indets` with `function`

```
restart;
expr := 3*sin(x)+t+3*f(x,t)*t+g(x,t);
res  := indets(expr,function);
if numelems(res)<>0 then
   print("Found these functions",res);
else
   print("could not find any function)");
fi;
```

"Found these functions", f(x, t), g(x, t), sin(x)

4.14 find all functions except builtin math functions

Given expression $3\sin(x)+t+3f(x,t)\,t+g(x,t)$ find all functions, if any, in the expression but exclude the math functions such as sin in the above.

```
restart;
expr := 3*sin(x)+t+3*f(x,t)*t+g(x,t);
res  := indets(expr, And( function, Not(typefunc(mathfunc))));
if numelems(res)<>0 then
   print("Found these functions",res);
else
   print("could not find any function)");
fi;
```

"Found these functions", f(x, t), g(x, t)

4.15 How to obtain a list of all arguments of a single function?

Given function such as $f(x,t)$, how to find all its arguments x,t? use `op`

```
restart;
op(1..,f(x,t))

    x, t
```

Note that `op(0,f(x,t))` finds the function name.

4.16 How to obtain a list of all arguments of a single function in an expressions?

Given an expression, we want to find list of all the arguments of such function, say $y(.)$ in that expression. This can be used for example to check if the arguments of such a function are valid during parsing.

One way is to use `indets` to select all such functions from the expression, then use `op` as above to find the arguments. Here is an example

```
expr:=y(3)+y(x)+7+y(Pi)+y([a+b])+h(z)+exp(x);
L:=indets(expr,'specfunc'(anything,y));
map(X->op(1,X),L);

    {3, Pi, x, [a + b]}
```

4.17 How to obtain a list of all arguments of the D function in an expressions?

Given an expression used for initial/boundary conditions, such as `D(y)(3)+(D@@2)(y)(0)=1`, the question is, how to obtain all arguments of each D in the above to verify that it does not include the independent variable x in this example?

One way is to use `indets` to select all such functions from the expression, then use `op` as above to find the arguments.

```
expr:=D(y)(3)+(D@@2)(y)(0)+1/(D@@3)(y)(x)=0;
(lhs-rhs)(expr);
L:=indets(%,':-De');
map(X->op(1,X),L);
if has(%,x) then
   error "Can not have x in argument for D used for initial/boundary conditions";
fi;
```

Which gives

`Error, Can not have x in argument for D used for initial/boundary conditions` because we had `(D@@3)(y)(x)` there.

4.18 Find functions whose first argument is z

```
restart;
expr := 3*sin(z)+t+3*f(z,t,y)*t+g(x,t);
res  := indets(expr, patfunc(identical(z), anything));
if numelems(res)<>0 then
   print("Found these functions",res);
else
   print("could not find any function)");
fi;
```

gives

"Found these functions", f(z, t, y), sin(z)

4.19 Find functions whose second argument is t?

```
expr := 3*sin(z)+t+3*f(z,t,y)*t+g(x,t);
res  := indets(expr, patfunc(anything, identical(t), anything));
if numelems(res)<>0 then
   print("Found these functions",res);
else
   print("could not find any function)");
fi;
```

gives

"Found these functions", f(z, t, y), g(x, t)

4.20 How to use select with own type to find sub expressions?

Given expression such as $3 + (1 + x)\sin x$ or $3 + (1 + x)\sin^2 x$ use select to find any `polynomial * sin^n` subexpressions.

```
restart;
mytype_1 := ''`*`'( {polynom(And(algebraic, satisfies(u -> not has(u, I))),x),
                     Or( 'specfunc(sin)'^integer, 'specfunc(sin)')
                    }
                   )';
select(type, 3+(1+x)*sin(x),mytype_1);
select(type, 3+(1+x)*sin(x)^2,mytype_1);
```

Gives

```
                        (1 + x) sin(x)

                                      2
                       (1 + x) sin(x)
```

4.21 How to write structured types to match some expressions?

4.21.1 type for $\sin^m(x)\cos^n(x)$

```
restart;
my_type:=''`*`'( { Or('specfunc'(sin),'specfunc'(sin)^Or(integer,rational)),
                   Or('specfunc'(cos),'specfunc'(cos)^Or(integer,rational))})';

type(sin(x)^2*cos(x)^3,my_type);
type(sin(x)^2*cos(x),my_type);
type(sin(x)*cos(x),my_type);
type(cos(x)*sin(x)^(1/2),my_type);


           true
           true
           true
           true
```

I could not find a way to avoid writing `Or('specfunc'(sin),'specfunc'(sin)^Or(integer,rational)` in order to match both $\sin x$ and $\sin^2 x$. For these things, I find Mathematica patterns more flexible. The above can be done as follows in Mathematica

```
ClearAll[x,n,m,any]
patt=any_.*Sin[_]^n_. * Cos[_]^m_.
MatchQ[Sin[x]^2*Cos[2*x]^3,patt]
MatchQ[Sin[x]^2*Cos[x],patt]
MatchQ[Sin[x]*Cos[x],patt]
MatchQ[Cos[x]*Sin[x],patt]

      True
      True
      True
```

```
      True
```

In Mathematica `n_.` says basically to match $\sin x$ or $\sin^2 x$ since the dot says to match zero or more. So no need to duplicate things as I did above in Maple. There might be a way to do the same in Maple using structured type, but I could not find it. In General, I find patterns in Mathematica more flexible and easier to use for this sort of thing. Maple has `patmatch` command, but not as easy to use as Patterns in Mathematica.

4.22 select only indexed variables from an expression

use `indets` with type `'indexed'`

```
expr:=16*a[3]+6*a[1];
terms:=indets(expr,'indexed');

         terms := {a[1], a[3]}

#to find maximum index, then do

map(x->op(x),terms)

      {1, 3}
```

4.23 Given an expression, how to find all variables and functions in it?

Given say $\frac{d^2}{dx^2}y(x)+n\left(\frac{d}{dx}y(x)\right)+3=\sin(x)$ how to find all variables and functions in it, not including math functions such as $\sin x$?

So the result should be $n, x, y(x)$.

```
ode:=diff(y(x),x$2)+n*diff(y(x),x)+3=sin(x);
vars:=indets(ode, Or(  And(symbol,Not(constant)), And(function,Not(typefunc(mathfunc)) )

#gives
#           vars := {n, x, diff(y(x), x), diff(y(x), x, x), y(x)}
```

I still need to work on excluding derivatives from the search.

4.24 How to check if an expression is integer, when it has symbols in it?

I had case where I needed to check if something is integer or not. The problem is that the result had a symbol n in it. I need a way to tell Maple that to check if the result can be an integer given that n is also an integer.

Using `type` does not work, since can't use assumptions. One way is to use `coulditbe` as follows

```
restart;
expr:=n-1+2*m;
vars:=indets(expr,And(symbol,Not(constant)));
coulditbe(expr,integer) assuming op(map(Z->Z::integer,vars))

#  true
```

In the above `indets(expr,And(symbol,Not(constant)))` picks all variables in the expression, and `assuming op(map(Z->Z::integer,vars))` makes assumption that each is integer.

4.25 How to find parameters such as π in an expression?

Use

```
restart;
expr:=4*Pi+sin(x);
indets(expr,And(name,constant))

        {Pi}
```

4.26 How to find all derivatives $y'(x)$ in an expression?

Given an expression such as

$$\sin(x)+\left(\frac{d}{dx}y(x)\right)^3+\left(\frac{d}{dx}y(x)\right)\ln\left(y(x)\left(\frac{d}{dx}y(x)\right)^2\right)+\sqrt{\frac{d}{dx}y(x)}+\frac{x}{\left(\frac{d}{dx}y(x)\right)^7}$$

Find all $y'(x)$ for any power that show up, so the result should be

$$\left\{\frac{1}{\left(\frac{d}{dx}y\left(x\right)\right)^{7}},\left(\frac{d}{dx}y(x)\right)^{2},\left(\frac{d}{dx}y(x)\right)^{3},\sqrt{\frac{d}{dx}y\left(x\right)},\frac{d}{dx}y(x)\right\}$$

Use `indets` with type `identical(diff(y(x),x))^anything` is used. But must use the `flat` option to work correctly.

```
restart;

expr:= y(x)*diff(y(x),x)^(1/3)+sin(x)*diff(y(x),x)^3 + z*diff(y(x),x)*ln(y(x)*diff(y(x),
t1:=identical(diff(y(x),x))^anything;
t2:=identical(diff(y(x),x));
indets[flat](expr, 'Or'(  t1, t2 ));
```

gives

```
{diff(y(x), x)^(1/3), 1/diff(y(x), x)^7, diff(y(x), x)^2, diff(y(x), x)^3, diff(y(x), x)
```

Without using `flat` it will given wrong result. For example

```
restart;

expr:=diff(y(x),x)^2;
t1:=identical(diff(y(x),x))^anything;
t2:=identical(diff(y(x),x));
indets(expr, 'Or'(  t1, t2 ));
```

Gives

```
{diff(y(x), x)^2, diff(y(x), x)}
```

You see, it has extra `diff(y(x), x)` showing up. Adding `flat` it gives

```
restart;

expr:=diff(y(x),x)^2;
t1:=identical(diff(y(x),x))^anything;
t2:=identical(diff(y(x),x));
indets[flat](expr, 'Or'(  t1, t2 ));
```

Now it gives

```
{diff(y(x), x)^2}
```

Which is the correct result.

4.27 How combine log terms?

To go from $\ln(AB)$ to $\ln A + \ln B$ need to use `simplify` with `ln` option but add assumptions that one of the terms is positive. Else it will not do it

```
restart;
simplify(ln(A*B),ln); # no change
simplify(ln(A*B),ln) assuming A>0;  # ln(A) + ln(B)
simplify(ln(A*B),ln) assuming B>0;  # ln(A) + ln(B)
```

To go from $\ln(\frac{A}{B})$ to $\ln A - \ln B$ need to use `simplify` with `ln` option but add assumptions that $B > 0$.

```
restart;
simplify(ln(A/B),ln);  # no change
simplify(ln(A/B),ln) assuming A>0;  # ln(A) + ln(1/B)
simplify(ln(A/B),ln) assuming B>0;  # do this: -ln(B) + ln(A)
```

To go from $\ln A + \ln B$ to $\ln(AB)$ need to use `combine` with assumptions that either A or B is positive, else it will not do it.

```
restart;
combine( ln(A) + ln(B),ln); # no change
combine( ln(A) + ln(B),ln) assuming A>0; # ln(A*B)
combine( ln(A) + ln(B),ln) assuming B>0  # ln(A*B)
```

To go from $\ln A - \ln B$ to $\ln \frac{A}{B}$ need to use `combine` with assumptions that either B is positive.

```
restart;
combine( ln(A) - ln(B),ln);  #no change
combine( ln(A) - ln(B),ln) assuming A>0;  # -ln(B/A)
combine( ln(A) - ln(B),ln) assuming B>0   # use this  ln(A/B)
```

4.28 Find position in a list of items that are not numeric

Given list such as `[1,2,3,4,5,x,y,8,9,Pi]` find position of elements that are not numeric. In this case the answer should be `[6,7,10]`

```
restart;
lis:=[1,2,3,4,5,x,y,8,9,Pi];
lis2:=select(x->not(type(x,numeric)),lis);
map(x->ListTools:-Search(x,lis),lis2)

    [6,7,10]
```

I could not find a way to do it using one command like with Mathematica. The first command above uses `select` to first find non numeric entries. The second command `ListTools:-Search` then find the index/position.

Maple's `ListTools:-Search` should really have a version that allows one to select the element directly. Something like this

```
lis:=[1,2,3,4,5,x,y,8,9,Pi];
ListTools:-Search(x->not(type(x,'numeric')),lis,all)
```

4.29 How to change $\arctan(y,x)$ to $\arctan\left(\frac{y}{x}\right)$ in an expression

I needed to do this as I was translating Maple code to Sagemath. Where sagemath supports arctan with only one argument.

Given an expression such as

$$x+\arctan(y,x)+\sin(x)$$

Convert it to

$$x+\arctan\left(\frac{y}{x}\right)+\sin(x)$$

```
evalindets(expr,'specfunc(arctan)',f->`if`(nops(f)=2,arctan(op(1,f)/op(2,f)),f))
```

4.30 How to find all symbols that represent variables in an expression?

Given an expression such as $a + \sin(x) + \pi + y + f(r)$ how to find all symbols in it, which will be a, x, y, r?

One way

```
expr:=a+sin(x) + Pi + y+ f(r);
indets(expr,And(symbol,Not(constant)));
```

Another is

```
expr:=a+sin(x) + Pi + y+ f(r);
indets(expr,assignable(name));
```

Both give `{a, r, x, y}`

4.31 How to change first argument of function?

This question came up in `https://mathematica.stackexchange.com/questions/274535/replacing-only-variables-in-specific-locations-with-replace-all` where the user wanted to `I would like to replace R0 with r, but only in the arguments of functions.`

The input they had is `line = R0*f[R0,x] + R0^2*42*D[g[R0,x],x]`

In Maple, this can be done as follows, which I think is a easier thanks to Maple's strong type system.

```
line := R0*f(R0,x) + R0^2*42*diff(g(R0,x),x);
evalindets(line, 'patfunc(identical(R0),anything)', Z-> subsop(1 = r, Z ));
```

Which gives

$$R0 f(r,x) + 42 R0^2 \left(\frac{\partial}{\partial x} g(r,x) \right)$$

4.32 How to change last argument of function?

Given an expression which contains some different functions each with different number of arguments. Suppose we want to change only the last argument of each function if the last argument is x, and change it to say x^2.

how to do that?

Hence given `R0*f(R0,x)+42*R0^2*D[2](g)(R0,x)+h(x,y,z,r,x)` we want to change it to `R0*f(R0,x^2)+42*R0^2*D[2](g)(R0,x^2)+h(x,y,z,r,x^2)`.

```
expr:= R0*f(R0,x) + R0^2*42*diff(g(R0,x),x)+ h(x,y,z,r,x);
evalindets(expr, 'patfunc[reverse](anything,identical(x))', Z-> subsop(-1 = x^2, Z ));
```

Note the use of `subsop(-1 = x^2, Z )` where -1 means the last entry in the argument of the function. This is basically same as last example, but uses `patfunc[reverse]` instead of just `patfunc`

4.33 How to remove last argument of function?

This is the same example as the above, but now we want to remove the last argument instead of changing it.

Hence, given an expression which contains some different functions each with different number of arguments. Suppose we want to remove only the last argument of each function if the last argument is x.

how to do that?

Hence given `R0*f(R0,x)+h(x,y,z,r,x)` we want to change it to `R0*f(R0)+h(x,y,z,r)`.

```
expr:=R0*f(R0,x)+h(x,y,z,r,x)
evalindets(expr, 'patfunc[reverse](anything,identical(x))', Z-> subsop(-1 = NULL, Z ));
```

Be careful using the above on expression that has `diff(f(y,x),x)` as this will give 0, because we basically removed the variable of the differentiation.

4.34 How to find a pattern inside an expression

Given $3\,e^{x}+\sin\left(a\,e^{x}\right)f(e^{5x})$ how to find all terms with pattern `anything*exp` ? If we do this

```
expr:=3*exp(x)+sin(a*exp(x))*f(exp(5*x));
indets(expr,`&*`(anything,'specfunc(exp)'));

#        {a*exp(x), 3*exp(x)}
```

We see it did not find `exp(5*x)` this is because there is nothing multiplying the `exp` function. To find this we add `Or` like this to count for both cases

```
expr:=3*exp(x)+sin(a*exp(x))*f(exp(5*x));
indets[flat](expr,Or( `&*`(anything,'specfunc(exp)'), 'specfunc(exp)' ))


#        {a*exp(x), 3*exp(x), exp(5*x)}
```

The `flat` option is needed, as without it this will be the result

```
expr:=3*exp(x)+sin(a*exp(x))*f(exp(5*x));
indets(expr,Or( `&*`(anything,'specfunc(exp)'), 'specfunc(exp)' ))

#       {a*exp(x), 3*exp(x), exp(x), exp(5*x)}
```

4.35 How to find parts of a Sum?

Given an inert Sum such as

$$r=\sum_{n=2}^{\infty}a_{n-1}x^{n-1}$$

How to obtain the body of the sum, the index variable, the lower starting value and the upper limit?

```
r:=Sum(a[n - 1]*x^(n - 1), n = 2 .. infinity);
op(0,r)  #head
                              Sum
op(1,r) #body
                        a[n - 1] x^(n - 1)

op(2,r) # sum specs
                        n = 2 .. infinity
```

```
lhs(op(2,r))  #name of summation index
                                n

rhs(op(2,r))  # lower..upper limits
                           2 .. infinity

op(1,rhs(op(2,r)))  #lower limit
                                2

op(2,rhs(op(2,r)))  #upper limit

                      infinity
```

4.36 find if sequence or list is inside another and the indices

Given a 1D container, such as vector or list, called A, how to find if this sequence is inside another sequence say B and the indices in B where the sequence A is located?

Use `ArrayTool` in version 2023

```
A:=[1,3,5]; B:=[1,3,4,5];

status,indices_list := ArrayTools:-IsSubsequence( A, B ,'output' = ['check','indices'],'
if status then
   print("Sequence ",A," Was found in ",B," At indices ",indices_list);
else
   print("Sequence ",A," Was not found in ",B);
fi;
```

The above gives

```
 "Sequence ", [1, 3, 5], " Was found in ", [1, 3, 4, 5], " At indices ", [1, 2, 4]
```

If A was not sequence inside B, then `status` will be false otherwise.

4.37 find if some type inside some expression and its location

Use `membertype`

```
restart;
expr:=cos(x)+x+9+sin(x)*(x^2+4);
membertype( polynom(anything,x), expr,'loc' );
       true
loc;
       2
```

The above says there is a polynomial in x inside the expression at `op(2,expr)` notice that x^2+4 is not a polynomial since the expression will expand and $\sin x$ will be multiplied by it causing it not to become polynomial. So only $9+x$ is the polynomial. The location is where the member starts at. Notice that Maple sorts polynomial from lower to higher powers.

```
restart;
expr:=cos(x)+x+9+sin(x)*(x^2+4);
membertype( integer, expr,'loc' );
       true
loc;
       3
```

The above says that there is an integer (which is 9 in this example) inside the expression at `op(3,expr)`

4.38 How to check that all entries in a list have the same type?

Given a list, we need to check that all entries are same type, say `'='`. But the type can be anything.

One way is to use `andmap` with the `type` function. Here is an example

```
restart;
L:=[3+x=0, 2+y(x)-3=0, 5=0, 7*x=2];
#check all entries are of type `=`
andmap(X->type(X,`=`),L);

   #true

L:=[3+x=0, 2+y(x)-3=0, 5];
andmap(X->type(X,`=`),L);
```

```
  #false
```

4.39 How to replace generic function inside derivative?

I had case where I wanted to substitute `_F=(y-x^2)/x` in an expression obtained which is `x*diff(F1, x) + 2*y*diff(F1, y)`

Using eval does not work

```
restart;
eval(x*diff(F1, x) + 2*y*diff(F1, y),F1=(y-x^2)/x);
  0
```

Also using delayed does not work

```
restart;
eval('x*diff(F1, x) + 2*y*diff(F1, y)',F1=(y-x^2)/x);
value(%)
  0
```

Also using subs does not work

```
restart;
subs(F1=(y-x^2)/x,x*diff(F1, x) + 2*y*diff(F1, y))
  0
```

However, delayed with subs finally worked

```
restart;
subs(F1=(y-x^2)/x,'x*diff(F1, x) + 2*y*diff(F1, y)');
value(%)

x*(-2 - (-x^2 + y)/x^2) + 2*y/x
```

So the rule is, if you want to replace a function inside diff, use subs and not eval, and make sure to delay evaluation of the expression and then use `value()` to obtain the final result.

4.40 Examples how to match types

These are examples how to match expression types

1. expr is product `a*b` which is matched using

```
type(a*b,`&*`(anything,anything))
                true
```

   The above can also be written using infix notation

```
type(a*b,  anything &* anything)
                true
```

   However, the above only match product of two terms. To match `3*a*b` use

```
type(3*a*b,`*`)
                true
```

2. expr is division `a/b` which is matched using same as `*`

```
type(a/b,`&*`(anything,anything))
                true
```

3. Match type of `a*f(x)`, which is anything times a function that takes one argument x.

```
type(a*f(x),`&*`(anything,patfunc(identical(x),anything)))
                true
```

   Using `patfunc` is better than using `function`

```
type(a*f(x,y),`&*`(anything,function(identical(x))))
                true
```

   Because `patfunc` matches on $f(x,y,z,\dots)$ and not just function which takes only one argument $f(x)$. But if you know your function takes only one argument, then use `function`

4. To match `3*y/x`. This was tricky. Had to use

```
expr:=3*y/x;
type(expr, `&*`(anything,identical(y),`^`(identical(x),-1)))
```

```
        or
type(expr, `&*`(identical(y),`^`(identical(x),-1)))

             true
```

I used `or` to account for possible `expr:=y/x`, i.e. missing constant at front. Note that

```
expr:=3*y/x;
type(3*y/x, `&*`(anything,identical(y/x)))
            false
```

Does not match, since `3*y/x` is actually `3*y` times `1/x` internally.

5. Match on `f(b* y/x)`

```
type(f(3*y/x),function(`&*`(anything,identical(y),`^`(identical(x),-1))))

            true
```

6. Match on `f(b* y/x)` or `f(y/x)`

```
expr:=f(3*y/x);
type(expr, function(`&*`(anything,identical(y),`^`(identical(x),-1))))
   or
type(expr, function(`&*`(identical(y),`^`(identical(x),-1))))

            true
```

Again, had to use or to account for missing constant multiplier.

7. Match on `a*f(b* y/x)`

```
expr:=a*f(3*y/x);
type(expr, `&*`(anything,patfunc(`&*`(anything,identical(y),`^`(identical(x),-1))))
  true
```

8. Match against `9+3 y/x`

```
expr:=9+3*y/x;
selector:=`&+`(anything,`&*`(anything,identical(y),`^`(identical(x),-1))):
type(expr, selector);
```

```
            true

select(type,[expr],selector);
     [9 + 3*y/x]
```

Notice in the above, when using `select` we need to put the expression inside a list, as select looks at each operand, This way the whole expression is taken as one. If we just used `select(type,expr,selector);` it would not have found it.

To use `patmatch` the command becomes

```
patmatch(expr, a::anything+b::anything*y/x,'la');
la

    [a = 9, b = 3]
```

The nice thing about patmatch is that it allowed one to assign variable to parts of the expression automatically.

9. Match against `9+f(3 y/x)` where now f is function. Using `patmatch`. I could not do this in one command, as all my attempts failed:

```
expr:=3+4*x*f(3*y/x);
body_of_function:=C::anything*y/x;
patmatch(expr,A::anything+B::anything*F::function(C::anything*y/x),'la');
patmatch(expr,A::anything+B::anything*F::patfunc(C::anything*y/x),'la');

Error, (in PatternMatching:-AlgStruct:-Match) testing against an invalid type
Error, (in type/patfunc) testing against an invalid type
```

So I had to do it in two steps. First match on the function as whole, then use that to match on `f(3*y/x)` in second stage, like this

```
expr:=3+4*x*f(3*y/x);
patmatch(expr,A::anything+B::anything*F::function(anything),'la');
la;

[A = 3, B = 4*x, F = f(3*y/x)]
```

And now

```
assign(la);
A:='A';
patmatch (op(1,F),A::anything*y/x,'la');
la

[A = 3]
```

Overall, I find Mathematica's pattern matching constructs much simpler and more intuitive to use and easier to learn as there are many examples and tutorials. For example, the last example above in Mathematica could be done as follows

```
expr=9+(4*x)*f[3*y/x];
Cases[{expr},any0_.+ any1_.*any2_[any3_.*y/x]:>{any0,any1,any2,any3}]

{{9, 4 x, f, 3}}
```

Maple's help pages are not good at all and provide little or no examples to learn from compared to Mathematica's excellent help pages. For any serious pattern matching tasks, I would use Mathematica. Maple has a better debugger and hence easier to debug the code because of this. So it is a trade between these two systems.

4.41 On the order of terms when using indents

I was trying to match on anyfunction that has x inside its arguments. It turned out that matching $f(ax)$ vs. $f(a+x)$ needed to have the `identical(x)` being placed first when it is a sum and last when it is a product. Very strange. Just be aware of this

```
indets(f(a+x),function(`&+`(anything,identical(x))));
#failed
    {}
```

But

```
indets(f(a+x),function(`&+`(identical(x),anything)));

   {f(x + a)}
```

While with product, it is the other way around

```
indets(f(a*x),function(`&*`(anything,identical(x))));
```

```
    {f(a*x)}
```

But now this fail

```
indets(f(a*x),function(`&*`(identical(x),anything)));
#fail
    {}
```

I have not yet figure how to tell it that the order does not matter. Maple 2023.1

A better way than the above, if I want to find any function that takes in x or y as arguments is to use `patfunc` like this

```
indets(f(a*x+b+y),patfunc(anything,`Or`({identical(x),identical(y)})));
     {f(a*b + y)}

indets(f(b+y),patfunc(anything,`Or`({identical(x),identical(y)})));
     {f(b + y)}

indets(exp(a*x),patfunc(anything,`Or`({identical(x),identical(y)})));
      {exp(a*x)}

indets(exp(a+x),patfunc(anything,`Or`({identical(x),identical(y)})));
{exp(x + a)}
```

5 General FAQ

5.1 How to convert Mathematica expression to Maple?

```
restart;
with(MmaTranslator); #load the package
FromMma(`Integrate[Cos[x],x]`);
```

Or

```
restart;
with(MmaTranslator); #load the package
convert(`Integrate[Cos[x],x]`, FromMma);
```

5.2 How to debug and stopat internal procedures, such as dsolve?

```
f:=proc()
eq:=x*diff(y(x),x)+y(x)=exp(2*x);
dsolve(eq,y(x));
end proc;
```

Then used the command `stopat(f);` then called the procedure `f();` and now the debugger comes up. Did `step` command and now it steps inside `dsolve`

Some examples

```
stopat(`ODEtools/symtest`);
stopat(`ODEtools/test`);
stopat(`ODEtools/normal/expanded`);
stopat(`ODEtools/odepde`);
stopat(`ODEtools/odeadv`);  #for DEtools:-odeadvisor
stopat(`odsolve/dAlembert`);
stopat(`odsolve/dAlembert/integrate`);
stopat(`odsolve/answer`);
stopat(`odsolve/homogeneous`); #for all A,C,D,G types
stopat(`odsolve/homogeneous_C/integrate`);
stopat(`odsolve/exact`); #for solving exact ODE
stopat(`odsolve/exact/integrate`);
stopat(`odsolve/exact/integrate`(f,y,x,M,N)); #where f here is RHS of y'=RHS;


DEtools:-symtest([-3,y],ode,y(x));

`ODEtools/normal/expanded`
```

For exact ode, can also do

```
ode:=....# write your ode here
Student:-ODEs:-Solve:-Exact(ode,y(x),output=steps);
```

5.3 How to display source code of a function?

For integration use

```
infolevel[`evalf/int`]:=5;infolevel[int]:=5;
```

Another option

```
restart;
interface(verboseproc=3) #(try 2 also)
```

then `print(procedure);` or `eval(procedure_name);` for example

```
restart:
interface(verboseproc=3):
print(LinearAlgebra:-GramSchmidt);
print(lcm);
```

Also can use `showstat`, in this case `interface(verboseproc=3)` is not needed. Also `showstat` gives line numbers and I think it is easier to read. Some examples

```
showstat(`odsolve/2nd_order`)
showstat(`evalf/hypergeom`);
showstat(`evalf/exp/general`);
showstat(`evalf/Psi`);
showstat(`evalf/int`);
showstat(`dsolve/SERIES`);

#these 3 shows the main 3 functions by each solver
showstat(`odeadv/dAlembert`);  #used by advisor
showstat(`odsolve/dAlembert`);  # main API.
showstat(`odsolve/dAlembert/integrate`); #used to integrate the ode

showstat(`ODEtools/odeadv`);
showstat(DEtools:-odeadvisor);
showstat(`dsolve/series/froben/inhom`)
showstat(`dsolve/series/froben`)
```

To stop at anyone of these functions in debugger do

```
stopat(`dsolve/series/froben/inhom`)
```

```
#code here, say dsolve command.
```

The above will stop in the debugger in the above function.

There is also a function by Joe Riel here here is the post by Joe Riel:

"A disadvantage of showstat, particularly if you want to cut and paste the output, is that it includes line numbers. Here is a simple procedure I threw together to remove the line numbers."

```
PrintProc := proc(p::name,lines::{posint,posint..posint})
local width;
option `Copyright (C) 2004 by Joseph S. Riel. All rights reserved.`;
description "Print like showstat, but without line numbers";
width := interface('screenwidth'=200);
try
printf("%s",
StringTools:-RegSubs(
"\n ...." = "\n"
,debugopts('procdump'=
`if`(nargs=1,p,[args]))))
catch "procedure name expected":
error "%1 is not a procedure name",p
finally interface('screenwidth'=width)
end try;
NULL
end:
```

To print source code to file using the above, do the following

```
currentdir("C:\\data");
interface('prettyprint'=1):
interface('verboseproc'=3):
writeto("listing.txt")
PrintProc('singular');
writeto('terminal'):
```

Now the output will show up in the file "listing.txt" and also no line wrapping. The above I found is the best solution so far to do this.

5.4 How to display trace of a function as it runs in maple?

```
trace(foo);
untrace(foo);
```

also see `debug(foo);`

Also

```
infolevel[all]:=5:
printlevel:=10:
```

See `http://www.mapleprimes.com/questions/35951-How-To-Debugtrace-Things-In-Maple`

Also look at `kernelopts(opaquemodules=true)`

Here is a useful post by Carl Love from Maple prime forum that summarizes all of these

Here are four things that you can do to get more information. I have listed them in order by how structured the information is, with the most structured first.

1. Set

   ```
   infolevel[all]:= 5;
   ```

   That will cause programs to print out additional information of the programmers' choosing. You can use higher or lower numbers for more or less information. Most programs don't use levels higher than 5.

2. Print the code of procedures with showstat:

   ```
   showstat(int);
   showstat(sin);
   showstat(cos);
   ```

3. Trace the execution of particular procedures with trace:

   ```
   trace(int);
   trace(sin);
   ```

4. Trace the execution of everything with printlevel:

   ```
   printlevel:= 10000:
   ```

   You can use higher or lower numbers for more or less information.

5.5 How to display a build in function code?

Some examples

```
interface(verboseproc=3);
print(DEtools)
print(`ODEtools/symgen`);
print(`symgen/methods`);
print(`symgen/do`);
```

To stop the debugger at symgen do

```
stopat(`ODEtools/symgen`);
```

To get infolevel on symgen do

```
infolevel[`symgen`]:=5;
```

Or to see line numbers

```
interface(verboseproc=3);
showstat(dsolve)
```

Or can use the `Browse();` command

```
with(LibraryTools);
Browse();
```

Another option I found is

```
s:=debugopts(procdump=`showstat`);
```

Then the above produces listing that can be copied as string with line wrapping ok.

5.6 How to build a LIST or a SET on the fly?

One way

```
L:=[]:
for i from 1 to 3 do :
    L:=[op(L),i];
end do;
```

But a better way is to use `seq` if one knows the length

```
L:=[seq(i,i=1..3)];

       L := [1, 2, 3]
```

Since list is immutable, a more efficient method, for long lists, is to use Array, and then convert the result back to list at the end since Array can grow dynamically without preallocation each time something is inserted as follows

```
L:=Array():
for i from 1 to 3 do :
    L(i):=i;
```

```
end do;

for i from 1 to numelems(L) do :
    print(L[i]);
end do;

L := convert(L,list)
```

Which will print

```
                              L := [1]

                            L := [1, 2]

                          L := [1, 2, 3]

                                1

                                2

                                3

                          L := [1, 2, 3]
```

Notice that to add to an Array, `()` is used. But to access an entry in an array `[]` is used.

And finally, using Array also, it can be done without using any indexing as follows

```
L:=Array(1..0):
for i from 1 to 3 do :
    L ,= i;
end do;

L := convert(L,list)
```

For the above to work, the array must be declared using `Array(1..0)`. The new syntax `A ,= i` will append to the array, and there is no need to write `A(i) := i`

5.7 make function display more information

By Carol Devore on the net:

```
Use infolevel.
For example, to show what logic dsolve uses, do this:

First try
> infolevel[all]:= 5;

That will probably give more information than you want, but if not,
then try
> printlevel:= 1000;

If you want information about a specific procedure, you can use debug.
 For example,
restart;
debug(`int/int`);
int(p, x= 0..1);

To find out what procedures are being called without getting too much
extra information, use excallgraph.
```

Trying on dsolve

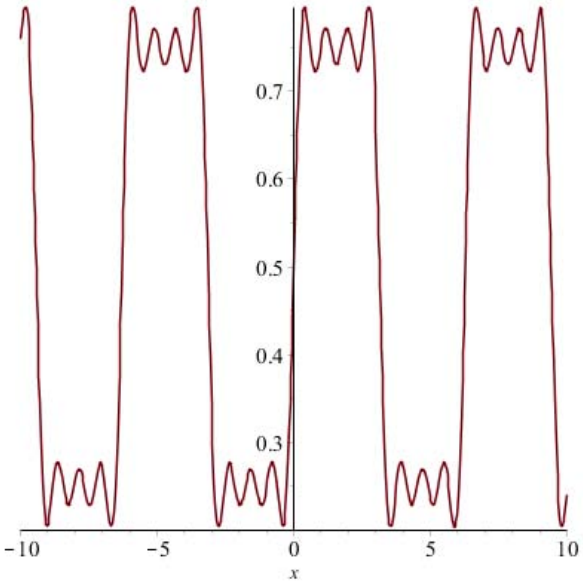
```
infolevel[dsolve]:= 3;
dsolve({eq1},y(x));

Methods for second order ODEs:
Trying to isolate the derivative d^2y/dx^2...
Successful isolation of d^2y/dx^2
--- Trying classification methods ---
trying a quadrature
trying high order exact linear fully integrable
trying differential order: 2; linear nonhomogeneous with symmetry [0,1]
trying a double symmetry of the form [xi=0, eta=F(x)]
<- double symmetry of the form [xi=0, eta=F(x)] successful
```

5.8 How to plot a function?

Here, I am looking at Fourier series expansion of $f(x) = 0$ between π and 0, and $f(x) = 1$ between 0 and π.

The Fores series expansion is worked out to be as below. This shows that the series approximate the above $f(x)$ as more terms are added

```
restart;
f:=(x)-> 1/2 + (1/Pi)*(sin(x)+sin(3*x)/3+sin(5*x)/5+sin(7*x)/7);
plot(f(x),x=-10..10);
```



5.9 How to run maple from command line?

From DOS, point to where your cmaple is

```
>"C:\Program Files\Maple 7\BIN.WNT\"cmaple
```

To make it execute maple commands use the `< foo.txt` to pipe maple commands in the file to it.

5.10 How to use matrices in maple?

```
A:= Matrix( [ [1, 2, 3] ,
              [3, 6, 7] ,
              [5, 6, 9] ,
              [7, 7, 7]
            ]);
whattype(A);
       Matrix
size:=LinearAlgebra:-Dimension(A);
     size := 4, 3
row:=size[1];
      row := 4
col:=size[2];
      col := 3
```

You can extract any part of the matrix like this:

```
B:=A[1..3,2..2];
```

$$\begin{bmatrix} 2 \\ 6 \\ 6 \end{bmatrix}$$

By Carl Devore http://mathforum.org/kb/message.jspa?messageID=1570678

```
Maple list and sequence structures are more flexible than Matrices, which are
highly structured.  A Maple list of lists (called a listlist in Maple) is akin
to a matrix in some other languages.  Many matrix
operations can be performed directly on the listlist form, but to do
serious linear algebra, you should convert to a Matrix.  Of course, it is
trivial to convert a listlist to Matrix:

LL:= [[1,2], [3,4]];
M:= Matrix(LL);

So here is another solution in line with your original wishes.  This is
"index free", but the table-based solution I gave earlier should be
faster.  (It is usually considered bad form to repeatedly append to a list or sequence.)

L:= [][]; # Create a NULL sequence
```

```
do
   line:= readline(file);
   if line::string then
      if line contains valid data then
         Z:= a list of that data;
         L:= L, Z
      fi
   else
      break
   fi
od

A:= Matrix([L]); # Note []: seq -> list.
```

To move move a column into a matrix: Here, I want to copy 2nd column to the 3rd column:

A;

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 6 & 7 \\ 5 & 6 & 9 \\ 7 & 7 & 7 \end{bmatrix}$$

```
B:=A[1..row,2];
```

$$\begin{bmatrix} 2 \\ 6 \\ 6 \\ 7 \end{bmatrix}$$

```
 A[1..row,3]:=B: A;
```

$$\begin{bmatrix} 1 & 2 & 2 \\ 3 & 6 & 6 \\ 5 & 6 & 6 \\ 7 & 7 & 7 \end{bmatrix}$$

5.11 return more than value from a procedure

Maple can return multiple values. Make sure to use the comma "," in the body of the procedure to separate each return value. Example:

```
size_matrix:=proc(x) 3*x, 4*x; end proc;
row,col :=size_matrix(5);
```

5.12 How does maple handle procedure arguments?

When passing a variable to maple procedure, the variable VALUE is passed to the procedure (This is different from say Fortran where the default is pass by reference). But this is the same as with Mathematica.

For example, if a variable X had value 10, then you call a procedure FOO passing it X, then inside FOO, X will be the number 10, not the argument variable X. So, this means one can not have X on the left hand side inside FOO. Like this `x:=1`

The only way to assign new value to the input and return new value, is to use a local variable, like this:

```
one:= proc(x)
        local y;
        print(x);
        y:=x+ 1;
        print(x);
        y;
        end proc;

z:='z';
z:=5;
f:=one(z);

      f := 6
```

5.13 How to define your own data types?

Use ‘type/name‘ to define new type name.

```
`type/char`:= x-> x::string and length(x)=1;

P:= proc(c::char) print(c) end proc:
P("x");
                              "x"
P("xy");
Error, invalid input: P expects its 1st argument, c, to be of type char, but received xy

> `type/byte`:= x-> x::integer and (x>= 0 and x<256);

#will define a byte (unsigned integer)
```

5.14 find max element and position in matrix

Code from net by Carl Devore:

```
MMax:= proc(M::{Matrix,matrix})
   local C,r,c,mx,L,p;
   C:= op(`if`(M::Matrix, [1,2], [2,2,2]), eval(M));
   L:= map(op, convert(M, listlist));
   mx:= max(L[]);
   member(mx,L,'p');
   r:= iquo(p, C, 'c');
   mx, `if`(c=0, [r,C], [r+1,c])
end;
```

Code below from C W

```
A:=matrix(12,12,rand(100));
Ao:=array((proc(E)
            local i; [seq(i=(rhs=lhs)(E[i]),i=1..nops(E))]end)
            (sort(op(3,eval(A)),proc(E1,E2) if rhs(E1)>rhs(E2)
             then
                  true
             else
                  false
             fi
```

```
            end)));
            Ao[1];
```

5.15 How to create a package?

First create the module:

```
restart;

nma:= module()
      option package;
      export getMaxMatrix;
      getMaxMatrix := proc (M::{matrix, Matrix})
          local C, r, c, mx, L, p;
          C := op(`if`(M::Matrix,[1, 2],[2,2,2]),eval(M));
          L := map(op,convert(M,listlist));
          mx := max(L[]); member(mx,L,'p');
          r := iquo(p,C,'c');
          mx, `if`(c = 0,[r, C],[r+1, c])
       end proc;
end module;

A:= Matrix( [ [1, 2, 3] ,
              [3, 6, 7] ,
              [5, 6, 9] ,
              [7, 7, 7]
            ]);

nma[getMaxMatrix](A);|
```

Gives `9, [3, 3]`. Now save the module.

```
savelibname := "C:/MAPLE_PACKAES";
march('create', savelibname, 20);
```

now save the library to disk. `savelib(nma);`

Now we can test everything by reinitialize everything and reload the library.

```
>restart
#Add my library to LIBNAME
```

```
>libname:="C:/MAPLE_PACKAGES",libname;
> A:=matrix( [ [1,2,3],[4,6,9] ]);
>with(nma);
>nma[getMaxMatrix](A);
```

Now to print a proc() in the package, do

```
>interface(verboseproc=3);
> print(nma[getMaxMatrix]);
```

Now you can list what packages exist in the archive:

```
march('list',savelibname);
march('extract',savelibname,":-1.m","C:MAPLE_PACKAGES/t.m")
```

Some notes. need to clean later

```
> module1lib:=`module1\\lib`;
> system("md "||module1lib);
> march('create',module1lib,100);
> makehelp(module1,`module1/module1.mws`,module1lib):
> makehelp(`module1/export1`,`module1/export1.mws`,module1lib):
> savelibname:=module1lib: ### doesn't affect current libname
> savelib(module1); ### no error message
> restart;
> module1lib:="module1\\lib":
> libname:=module1lib,libname; ### now Maple will find module1
> with(module1);
> ?module1
```

Also there is a long thread here on Maple prime on making personal packages in Maple `How-To-Create-A-Personal-Package`

5.16 How to convert from floating point to Hex?

```
From: Robert Israel (israel@math.ubc.ca)
Subject: Re: Getting non-integral results in hex
Newsgroups: comp.soft-sys.math.maple
Date: 2003-06-13 00:07:37 PST

I assume you mean floating-point numbers.  Note that
Maple floats (as opposed to "hardware floats") are
in fact stored in base 10.  To convert a float to hex
with n digits after the ".", you can use this:

> `convert/hexfloat`:= proc(x::numeric, n::nonnegint)
  local A,B,ax,R;
  if nargs = 1 then return procname(x,round(Digits*log[16](10))) fi;
  if x = 0 then return cat(`0.`,`0`$n) fi;
  ax:= abs(x);
  A:= floor(ax);
  B:= round(frac(ax)*16^n);
  if B = 16^n then A:= A+1; B:= 0 fi;
  R:= cat(convert(A,hex),`.`);
  if x < 0 then R:= cat(`-`,R) fi;
  cat(R,substring(convert(16^n+B,hex),2..-1));
  end;

And then, e.g.:

> convert(1234.5678, hexfloat, 4);

                                4D2.915B
```

5.17 How to find Taylor series expansion of functions?

```
mtaylor(sin(x),[x],10);
```

$$x - 1/6\,x^3 + \frac{x^5}{120} - \frac{x^7}{5040} + \frac{x^9}{362880}$$

5.18 How to print elements of a matrix?

```
restart;
a:=Matrix([  [2,3,4],[4,5,6]   ]);
nRow,nCol :=LinearAlgebra[Dimension](a);
for i from 1 to nRow do
    for j from 1 to nCol do
        printf("a(%d,%d)=%d\n",i,j,a[i,j]);
    end do;
end do;

a(1,1)=2
a(1,2)=3
a(1,3)=4
a(2,1)=4
a(2,2)=5
a(2,3)=6
```

5.19 How to find determinant of matrix?

```
restart;
a:=Matrix([ [2,4],[5,7] ]);
LinearAlgebra:-Determinant(a);
         -6
```

5.20 How to generate Hilbert matrix?

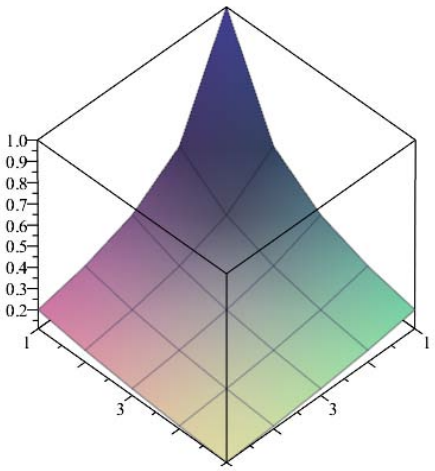
```
H := LinearAlgebra:-HilbertMatrix(5);
```

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

5.21 How to plot matrix data?

Matlab is much easier here. In maple, need to covert the matrix to a list of list of points first.

```
restart;
H := LinearAlgebra:-HilbertMatrix(5):
nRow,nCol :=LinearAlgebra[Dimension](H):
L:=[seq([seq( [i,j,H[i,j]], i=1..nRow) ], j=1..nCol)]:
plots:-surfdata(L);
```



5.22 How to catch an error from a proc()?

An error in maple raises an exception. So, use try catch to trap it as follows:

```
try
   v,pos:=MMax(4);
catch:
   printf("an error is cought\n");
end try;
```

5.23 How to convert 3456 to 3,456 ?

From the net, by Carl Devor:

```
`print/commas`:= proc(N::integer)
    local n,s,i,b;
    n:= ListTools:-Reverse(convert(abs(N), base, 1000));
    if N<0 then n:= subsop(1= -n[1], n) fi;
    nprintf("%s", sprintf(cat("%d", ",%03d" $ nops(n)-1), n[]))
```

```
 end proc:

commas(456554);

         456,554
```

To convert a string to array of chars use `array(StringTools:-Explode(S))`

```
s:="Nasser M. Abbasi":
r:=array(StringTools:-Explode(s));
              r:=["N" "a" "s" .......]
```

Now can use the string as normal array

```
r[4];
      "s"
```

5.24 How to use units ?

```
Units[GetDimensions](base);
 amount_of_information, amount_of_substance, currency, electric_current, length,
 logarithmic_gain, luminous_intensity, mass, thermodynamic_temperature, time
```

5.25 How to evaluate Catalan number and other sums?

Use the Sum command.

```
restart;
expr:= (-1)^i/(2*i+1)^2;
Sum(expr,i=0..infinity);
evalf(%,50);
          0.91596559417721901505460351493238411077414937428167
```

Notice, if I used the `sum` command instead of the `Sum` command I get this result:

```
sum(expr,i=0..infinity);
          Catalan
```

5.26 write a text file that contains a package, and load it

This shows how to do a simple package and use it without building a library. Just using a plain text file.

Create this `nma_pkg1.txt` file:

```
nma_pkg1 := module()
  export f1;
  option package;

  f1:= proc()
       print("in pakcage nma_pkg1");
  end proc;

end module;
```

now save it, and from maple do

```
>read("c:\\nma_pkg1.txt");
```

now execute f1() as this:

```
>nma_pkg1[f1]();
          "in pakcage nma_pkg1"
```

now put it in a library (so that we can use with, instead of read)

```
> savelibname:=("c:/maple");
> march('create', savelibname, 20);
> savelib(nma_pkg1);
>restart;
> libname := "c:/maple",libname;
> with(nma_pkg1);
> f1();
    "in pakcage nma_pkg1"
```

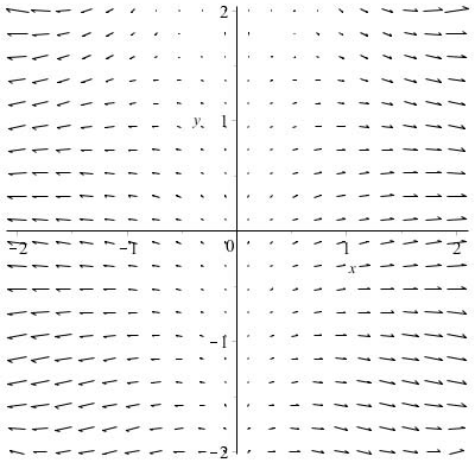
now make changes to the `nma_pkg1.txt` file and updated again as above.

5.27 How to find what packages are included in maple

```
?index,package
```

5.28 How to plot the gradient vector field?

```
restart;
f:=3*x^2 + y* cos(x*y);
the_grad :=linalg[grad](f,[x,y]);
plots[fieldplot](the_grad,x=-2..2,y=-2..2);
```



or

or can do it in just one command: `plots[gradplot](f,x=-2..2,y=-2..2);`

5.29 How to put the digits of Pi into a list?

Suppose you want the 100 digits of Pi put in a list. This is one way to do it:

```
restart;
L:=evalf(Pi,100);
S:=convert(L,string);
the_list:=[seq(parse(S[i]),i=3..length(S))];

    the_list := [1, 4, 1, 5, 9, 2, 6, 5, 3, ..
```

This below now tells how many times each digits occurs.

```
>stats[transform,tally](the_list);
```

```
[Weight(0, 8), Weight(1, 8), Weight(2, 12), Weight(3, 11),
Weight(4, 10), Weight(5, 8), Weight(6, 9), Weight(7, 7),
Weight(8, 13), Weight(9, 13)]
```

5.30 Digits of PI in maple and mma

Written sometime in 2005? I should really record the time when I write something.

I just run these now, August 2014, and now Maple 18 as very fast. So this all below is no longer valid. I will leave it here for now for reference until I update it all later

I have written a few lines of code, which counts how many times each digit occurs after the decimal points of π

Written this in maple first. Then did similar thin in mma 5.0. Both are run on the same PC. No other applications are running at the time when I run the code.

The basic idea of the algorithm is to use `evalf(Pi,digits)` in maple to find π for any number of decimal digits, and to use `N[Pi,digits]` in mma for doing the same. (Where the variable digits above is the number of digits)

Then in maple convert the above π to a string, and generate a sequence of the characters to right of decimal point, then use `stats[transform,tally]` to do the actual counting.

In mma, I use `RealDigits[]` to get a list of the digits, and then use `Count[]` to do the counting.

This is result of some of the runs to find Pi to some digits, and the total time (to find Pi and do the counting)

All times are in cpu seconds, machine is P4, 2.8 Ghz, 500 MB of RAM, single CPU, hyperthreading enabled, running XP home edition. Maple 9.03 student version, and mma 5.0 student version.

Below is the result, and below that I show the maple code and the mma code.

Because of this, before each run in mma, I exited the application and started it fresh. In maple, it does not matter for the above reason.

```
100,000 digits:
             Find_Pi     Total
Maple 9.0      55          84
Mma 5.0        0.9         1.54
```

Mma is 60 times faster in finding pi and about 56 times faster overall

```
300,000 digits:
             Find_Pi     Total
Maple 9.0     309         781
Mma 5.0       3.7         6
```

Mma is 300 times faster in finding Pi, and 130 times faster overall.

```
3,000,000 digits
             Find_Pi     Total
Maple 9.0
Mma 5.0        85        118

Maple time in hours ! Still running.
```

Maple code

```
> restart;
startingTime :=time();
L:=evalf(Pi,100000):
timeToFindPiInSecs:=time()-startingTime;
S:=convert(L,string):
the_list:=[seq(parse(S[i]),i=3..length(S))]:
stats[transform,tally](the_list);
endingTime :=time():
cpuTimeInSecs := endingTime - startingTime;
```

mma code

```
Clear[]
startingTime=TimeUsed[]
t1=N[Pi,100000];
timeToFindPiInSecs=TimeUsed[]-startingTime
{c,d}=RealDigits[t1];
theList=c[[Range[2,Length[c]]]];
f[digit_]:=Count[theList,digit];
r=Range[0,9];
Map[f,r]
```

```
cpuTimeInSecs=TimeUsed[]-startingTime
```

update 12/25/03 Changed maple code on how to do the counting : To use

```
StringTools[CharacterFrequencies](S)
```

Now the counting in maple is much faster. It is always hard to know which is the best function to use.

```
restart;
 startingTime :=time();
 L:=evalf(Pi,300000):
 timeToFindPiInSecs:=time()-startingTime;
 S:=convert(L,string):
StringTools[CharacterFrequencies](S);
endingTime :=time():
cpuTimeInSecs := endingTime - startingTime;
```

5.31 How to find where functions are?

```
From: Ken Lin (maplemath@tp.edu.tw)
Subject: Re: how to find which package a function belongs to?
Newsgroups: comp.soft-sys.math.maple
Date: 2003-12-04 03:49:26 PST

When Maple first loaded, There are only two kinds of "internal"
commands which can be called directly. One is the "kernal" commands
coded in C, and the other includes many "internal" procedures
programmed by the kernal commands which lies in the "Main Library",
There are also many other "external" procedures which were categorized
into so called "packages", plots[display](...) for example, plots[] is
a package(Library), and display() is the procedures inside plots[]. All
the packages can be loaded by with() command, like
> with(plots);


Because Different Packages include user library might have the same
procedure name, Maple doesn't realize the "procedure_name" you type
in, it took it for a "symbol". If you really want to know which
packages provided by Maple the external procedure lies in, just mark
```

```
the procedure_name and press F1 key, the Maple Help Browser will show
you the packages you might be interested.

By the way, plot3d() is a "internal" procedure lies in the Main
Library. You can confirm that by:
> op(0, eval(plot3d));
                          procedure
or in Maple 9
> type( plot3d, 'std' );      #Is it internal?                        true
> type( plot3d, 'stdlib' );   #Does is lie in "Standard(Main) Library"?
                          true
If you are interested the codes inside plot3d()...
> interface(verboseproc=2):   #Turn on verboseproc
> print(plot3d);              #eval() also works
> interface(verboseproc=1):   #Turn off verboseproc

I hope this will give you some help. Have fun with Maple.

Ken Lin
```

5.32 how to use Laplace transform?

```
restart;
f:= t->sin(omega*t) ;
L:=convert(inttrans[laplace](f(t),t,s),int);
```

$$\frac{\omega}{\omega^2 + s^2}$$

To find the inverse, do:

```
inttrans[invlaplace](L,s,t);
```

$$\sin(\omega t)$$

For unit step, use

```
_EnvUseHeavisideAsUnitStep:=true;
f:=Heaviside(t-a);
```

```
INV:=inttrans:-laplace(f,t,s) assuming a>0;  #make sure to use a>0
```

$$\frac{\mathrm{e}^{-sa}}{s}$$

Another example

```
_EnvUseHeavisideAsUnitStep:=true;
f:=Heaviside(t)-Heaviside(t-a);
INV:=inttrans:-laplace(f,t,s) assuming a>0;  #make sure to use a>0
```

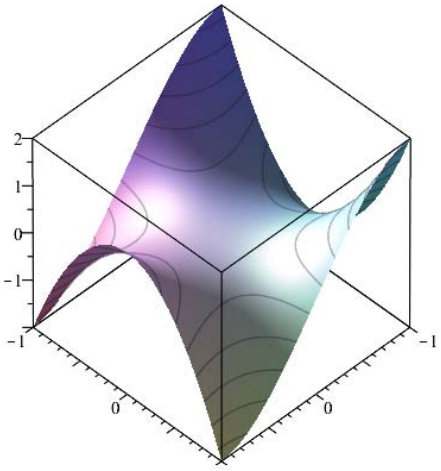
$$\frac{1-\mathrm{e}^{-sa}}{s}$$

5.33 questions I have

```
Any difference between using

`diffalg/Rosenfeld_Groebner`(args)
or
diffalg[Rosenfeld_Groebner](args)
```

5.34 3D plotting

```
restart;
f:= (x,y)->x^3-3*x*y^2;
plot3d(f,-1..1,-1..1,numpoints=2500,style=patchcontour);
```



5.35 How to raise each element in a list to a power?

Use map

```
map(``^``,{1,2,3},3);
      {1, 8, 27}
```

5.36 How to generate a sequence with any increment?

```
incr:=.25; start:=0; last:=3;
seq(start+i*incr,i=1..(last/incr));
```

5.37 What shortcuts are there for matrix manipulation?

read `?MVshortcut, ?MVassignment, and ?Mvextract` and Transpose(R) can be shortened to `R^%T`

5.38 How to solve a set of equations for the derivative?

Written Feb 20, 2004

This is problem 7.4 chapter 4, in the Mary Boas book. Given

$$xs^2 + yt^2 = 1$$
$$x^2 s + y^2 t = xy - 4$$

Find $\frac{dx}{dt}, \frac{dx}{ds}, \frac{dy}{dt}, \frac{dy}{ds}$ at $x = 1, y = -3, s = 2, t = -1$

This is how I did it in maple:

```
restart;
alias(x=x(s,t));
alias(y=y(s,t));
alias(Xt= diff(x(s,t), t));
alias(Xs= diff(x(s,t), s));
alias(Yt= diff(y(s,t), t));
alias(Ys= diff(y(s,t), s));

eq1:= x*s^2+y*t^2=1;
eq2:= x^2*s+y^2*t=x*y-4;

r1:=diff(eq1,t);
r2:=diff(eq1,s);
r3:=diff(eq2,t);
r4:=diff(eq2,s);

sol:=solve({r1,r2,r3,r4},{Xt,Xs,Yt,Ys});
```

$$\frac{\partial}{\partial s}x(s,t) = -\frac{x(s,t)\left(x(s,t)\,t^2 - 4\,y(s,t)\,st + 2\,x(s,t)\,s\right)}{2\,x\left(s,t\right)st^2 - 2\,y\left(s,t\right)ts^2 + x\left(s,t\right)s^2 - y\left(s,t\right)t^2}$$
$$\frac{\partial}{\partial t}x(s,t) = -\frac{y(s,t)\,t(-3\,y(s,t)\,t + 2\,x(s,t))}{2\,x\left(s,t\right)st^2 - 2\,y\left(s,t\right)ts^2 + x\left(s,t\right)s^2 - y\left(s,t\right)t^2}$$
$$\frac{\partial}{\partial s}y(s,t) = -\frac{x(s,t)\left(3\,x(s,t)\,s - 2\,y(s,t)\right)s}{2\,x\left(s,t\right)st^2 - 2\,y\left(s,t\right)ts^2 + x\left(s,t\right)s^2 - y\left(s,t\right)t^2}$$
$$\frac{\partial}{\partial t}y(s,t) = -\frac{y(s,t)\left(4\,x(s,t)\,st - y(s,t)\,s^2 - 2\,y(s,t)\,t\right)}{2\,x\left(s,t\right)st^2 - 2\,y\left(s,t\right)ts^2 + x\left(s,t\right)s^2 - y\left(s,t\right)t^2}$$

```
points:= {x=1,y=-3,s=2,t=-1};
subs(points,sol);
```

5.39 How to solve a set of equations for differentials?

This is problem 7.15 chapter 4 in Boas:

Given $x^2u - y^2v = 1$ and $x + y = uv$ Find $\frac{dx}{du}, v$ and $\frac{dx}{du}, y$

This is the maple code to solve this:

```
restart;
```

```
eq1:=x^2*u-y^2*v=1;
eq2:=x+y=u*v;
r1:=D(eq1);
r2:=D(eq2);
r1_:=subs(D(v)=0,r1);
r2_:=subs(D(v)=0,r2);
sol:=solve({r1_,r2_},{D(x),D(u)});
print("dx/du,v=");
rhs(sol[1])/rhs(sol[2]);

r1_:=subs(D(y)=0,r1);
r2_:=subs(D(y)=0,r2);
sol:=solve({r1_,r2_},{D(x),D(u)});
print("dx/du,y=");

rhs(sol[1])/rhs(sol[2]);
```

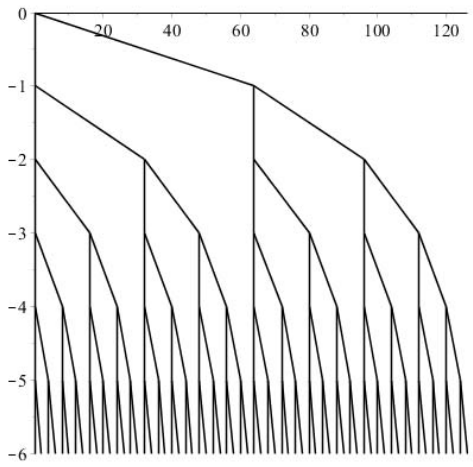
$$eq1 := u\,x^2 - v\,y^2 = 1$$
$$eq2 := x + y = u\,v$$
$$r1 := 2\,\mathrm{D}(x)\,x\,u + x^2\,\mathrm{D}(u) - 2\,\mathrm{D}(y)\,y\,v - y^2\,\mathrm{D}(v) = 0$$
$$r2 := \mathrm{D}(x) + \mathrm{D}(y) = \mathrm{D}(u)\,v + u\,\mathrm{D}(v)$$
$$r1\_ := 2\,\mathrm{D}(x)\,x\,u + x^2\,\mathrm{D}(u) - 2\,\mathrm{D}(y)\,y\,v = 0$$
$$r2\_ := \mathrm{D}(x) + \mathrm{D}(y) = \mathrm{D}(u)\,v$$
$$sol := \left\{\mathrm{D}(u) = \frac{2\,\mathrm{D}(y)\,(u\,x + v\,y)}{x\,(2\,u\,v + x)},\ \mathrm{D}(x) = \frac{\mathrm{D}(y)\,\left(2\,v^2\,y - x^2\right)}{x\,(2\,u\,v + x)}\right\}$$
"dx/du,v="
$$\frac{2\,(u\,x + v\,y)}{2\,v^2\,y - x^2}$$
$$r1\_ := 2\,\mathrm{D}(x)\,x\,u + x^2\,\mathrm{D}(u) - y^2\,\mathrm{D}(v) = 0$$
$$r2\_ := \mathrm{D}(x) = \mathrm{D}(u)\,v + u\,\mathrm{D}(v)$$
$$sol := \left\{\mathrm{D}(u) = -\frac{\mathrm{D}(v)\,\left(2\,u^2\,x - y^2\right)}{x\,(2\,u\,v + x)},\ \mathrm{D}(x) = \frac{\mathrm{D}(v)\,\left(u\,x^2 + v\,y^2\right)}{x\,(2\,u\,v + x)}\right\}$$
"dx/du,y="
$$-\frac{2\,u^2\,x - y^2}{u\,x^2 + v\,y^2}$$

5.40 How to plot binary tree

by `http://www.math.fsu.edu/~bellenot`

```
restart;
t2 := proc(i, x, y)
    if i < 2 then [[x, y], [x, y - 1]], [[x, y], [x + 2^i, y - 1]]
    else [[x, y], [x, y - 1]], [[x, y], [x + 2^i, y - 1]],
```

```
        t2(i - 1, x, y - 1), t2(i - 1, x + 2^i, y - 1)
    end if
end proc;

PLOT(CURVES(t2(6,0,0)));
```



5.41 Problem 12.4 chapter 4, Boas book

```
restart;
z:=   Int( sin(t)/t, t=sin(x)..cos(x));
diff(z,x);
```

$$-\frac{\sin(x)\sin(\cos(x))}{\cos(x)}-\frac{\cos(x)\sin(\sin(x))}{\sin(x)}$$

5.42 example of doing convergence test in maple

```
restart;
c:='c': C:='C': n:='n': P:='P':
C := n -> ((n+2)/(3*n+1))^n:
### WARNING: calls to `C` for generating C code should be replaced by codegen[C]
`The general term is `, c[n]= C(n); ` `;
`The n-th root is:`;
### WARNING: calls to `C` for generating C code should be replaced by codegen[C]
P := C(n)^(1/n):
abs(c[n])^(1/n) = P;
P := simplify(P, assume=positive):
```

```
abs(c[n])^(1/n) = P;
```

$$\text{The general term is , } c_n = \left(\frac{n+2}{3\,n+1} \right)^n$$

$$\text{The n-th root is:}$$

$$|c_n|^{\frac{1}{n}} = \left(\left(\frac{n+2}{3\,n+1} \right)^n \right)^{\frac{1}{n}}$$

$$|c_n|^{\frac{1}{n}} = \frac{n+2}{3\,n+1}$$

5.43 Problem ch 14, 3.18, Boas book. contour integration

```
restart;
f:= 1/(  (1-2*z)*(5*z-4) );
residue(f,z=4/5);
```

$$\frac{-1}{3}$$

5.44 How to find multiple roots to an equation such as $\sin(x) = 0$

```
_EnvAllSolutions:=true;
solve(sin(x)=0);
```

```
Pi _Z1~
```

5.45 Dr Basti Associated Legendre

```
Subject:      Associated Legendre
Author:       Mehran Basti <Basti@worldnet.att.net>
Organization: AT&T Worldnet
Date:         Mon, 25 Nov 2002 02:48:15 GMT


Dear newsgroup:

I had mentioned that my methods will solve classical equations without
```

```
the use of  infinite series.

The following is a Maple code of my old files. Those days I had Maple2 but the
general idea is the same in the process  and you see that we can also
solve the integrals involved.

It does not make sense how are the theory behind it but eventually it will come into lig

Just read the procedures and you can see the solution of associated Legendre AL at the e

> s1:=-diff(p(t),t)+p(t)^2;
>
> s2:=exp(2*int(p(t),t))*T(t);
> s3:=s1+s2;
> s4:=diff(T(t),t)/T(t);
> s5:=-(1/2)*(diff(s4,t))+(1/4)*s4^2;
> s6:=s5+s2;
> p(t):=-1/t+(1)/(2-t);
> s1:=simplify(s1);
> s1:=collect(%,t);
> s2:=simplify(s2);
> s1+s2=(2*t^2-4*t+m^2-1)/(t*(-2+t))^2;
> solve(%,T(t));
> T(t):=simplify(%);
> s2:=simplify(s2);
> s2+s1;
> s3:=simplify(%);
>
> s6:=simplify(s6);
> t*(-2+t);
> simplify(%);
> z:=(r3*t^3+r2*t^2+r1*t+r0)/(%);
>
> simplify(diff(z,t)+z^2-s6);
> s7:=collect(numer(%),t);
>
> coeff(%,t,0);
> solve(%,r0);
> r0:=op(1,{%});
> coeff(s7,t,1);
> solve(%,r1);
```

```
> r1:=simplify(%);
> coeff(s7,t,2);
> solve(%,r2);
> r2:=simplify(%);
> coeff(s7,t,3);
> solve(%,r3);
> r3:=simplify(%);
> simplify(s7);
> s3:=simplify(s3);
> s4:=simplify(s4);
> s6:=simplify(s6);
> T(t):=simplify(T(t));
> z:=simplify(z);
> 1/2*s4+2*p(t)+z;
> s8:=simplify(%);
> exp(int(%,t));
> expand(%);
> g:=(%);
> simplify(g,power);
> g:=%;
> Int(%,t);
> Integralg:=(%);
> int(g1(t),t);
> x1:=-p(t)+g1(t)/(%);
> diff(x1,t)+x1^2-s3;
> simplify(%);
> s10:=numer(%);
> solve(%,int(g1(t),t));
> Ing:=(%);
> simplify(subs(g1(t)=g,%));
>
>  Ing:=(%);
> expand(%);
> Ing:=simplify(%);
> simplify(diff(%,t)-g);
> expand(%);
> simplify(%);
> x:=-p(t)+g/Ing;
> simplify(diff(x,t)+x^2-s3);
>  int(x,t);
> exp(%);
```

```
> expand(%);
> s11:=simplify(%);
> ALT:=t*(2-t)*diff(u(t),t$2)+2*(1-t)*diff(u(t),t)+(2-m^2/(1-(1-t)^2))*u(t);
> -2*(1-t)/(2*t*(2-t));
> int(%,t);
> exp(%);
> s12:=simplify(%,power);
>
> u1:=s12*s11;
> u1:=simplify(%,power);
>  simplify(subs(u(t)=u1,ALT));
> AL:=(1-nu^2)*diff(u(nu),nu$2)-2*nu*diff(u(nu),nu)+(2-m^2/(1-nu^2))*u(nu);
>
> u2:=subs(t=1-nu,u1);
> simplify(subs(u(nu)=u2,AL));
>

The advantage of these methods are that there are ample rooms for advances.

Today my skills for solving classical equations such as Riccati is much advanced.

Highly complicated and more general Riccati equations in its billions now possible.

Sincerely

Dr.M.Basti
```

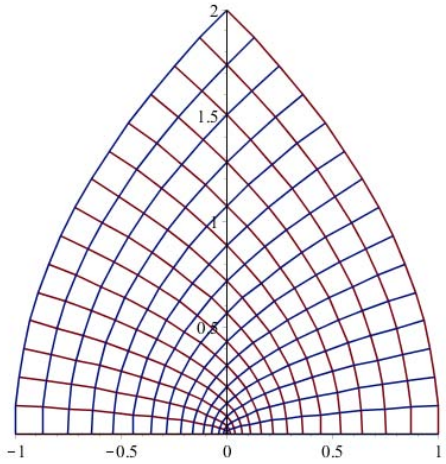
5.46 Understanding conformal mapping in maple

To plot mapping of complex function in maple, use `[plots]conformal` The trick is to how to specify the quadrant in the x-y plane. This example shows how.

Suppose we want to map the first quadrant. Then we specify the `DIAGONAL` points in the range, from the lower left corner to the upper right corner, which then should be `0..1+I` Because 0 is the lower left corner, and $(1, i)$ is the upper right corner. Example:

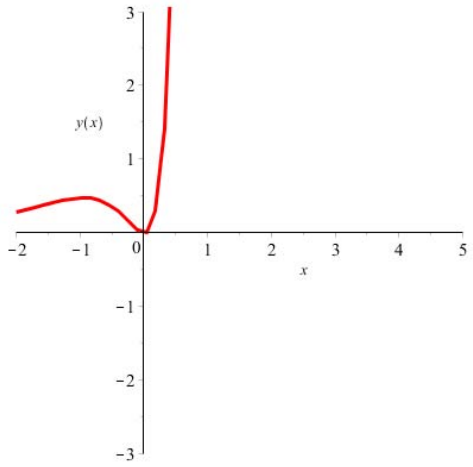
```
restart;
assume(y,real);
assume(x,real);
#f:= z->I+z*exp(I*Pi/4);
f:= z->z^2;
```

```
w:=f(x+I*y);
u:=Re(w);
v:=Im(w);
plots:-conformal(f(z),z=0..1+I,grid=[16,16],numxy=[16,16],scaling=constrained);
```



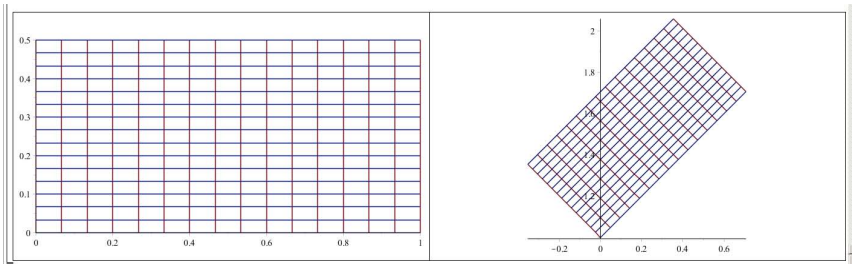
This below uses the first TWO quadrant's, i.e. the upper half of the x-y plane

```
restart;
assume(y,real);
assume(x,real);
#f:= z->I+z*exp(I*Pi/4);
f:= z->z^2;
w:=f(x+I*y);
u:=Re(w);
v:=Im(w);
plots:-conformal(f(z),z=-1-I..1+I,grid=[16,16],numxy=[16,16],scaling=constrained);
```



This below puts the plots next to each others so to see them

```
restart;
assume(y,real);
assume(x,real);
f:= z->I+z*exp(I*Pi/4);
#f:= z->z^2;
w:=f(x+I*y);
u:=Re(w);
v:=Im(w);
A := array(1..2):
A[1]:=plots:-conformal(z,z=0..1+I/2,grid=[16,16],numxy=[16,16],scaling=constrained):
A[2]:=plots:-conformal(f(z),z=0..1+I/2,grid=[16,16],numxy=[16,16],scaling=constrained):
plots:-display(A);
```



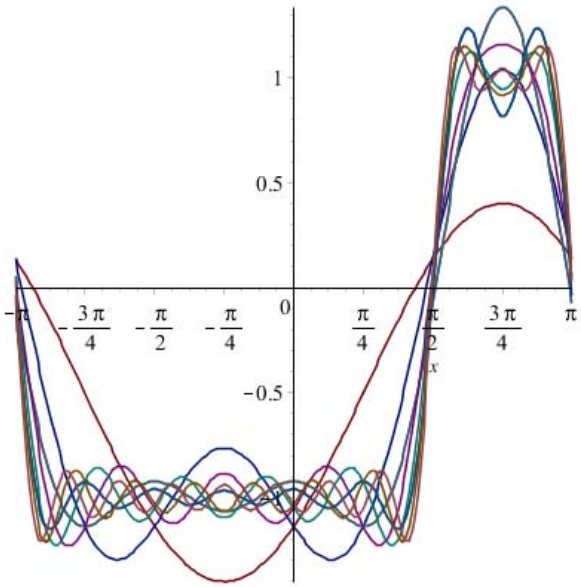
5.47 Hide tilde character when using assumption

`interface(showassumed=0)` removes all tildas and `interface(showassumed=1)` adds the tildas.

5.48 Fourier series in maple

I wrote this to generate FS in Maple for some HW I was doing. I think this was for Math 121A at UC Berkeley in 2003

```
restart;
f:=x->piecewise(-Pi<x and x<Pi/2,-1,
                Pi/2<x and x<1,0,1);

assume(n,integer);

nmaFourier2:=proc(f,freq,from_,to_,maxN)
       local n::integer,denomC,denomS,a,b;
       denomC:=( to_ - from_ ) / 2;
       denomS:=( to_ - from_ ) / 2;

       a:=proc(n)
         int(f(x)*cos(n*freq*x),x=from_..to_) /denomC;
       end proc;

       b:=proc(n)
         int(f(x)*sin(n*freq*x),x=from_..to_) / denomS;
       end proc;

       evalf(denomC);

       1/2*a(0) + sum( a(n) * cos(n*freq*x) ,n=1..maxN)
                + sum( b(n) * sin(n*freq*x) ,n=1..maxN)
end proc;

r:=[seq(nmaFourier2(f,1,-Pi,Pi,nIter),nIter=1..10)];
plot(r,x=-Pi..Pi);
```



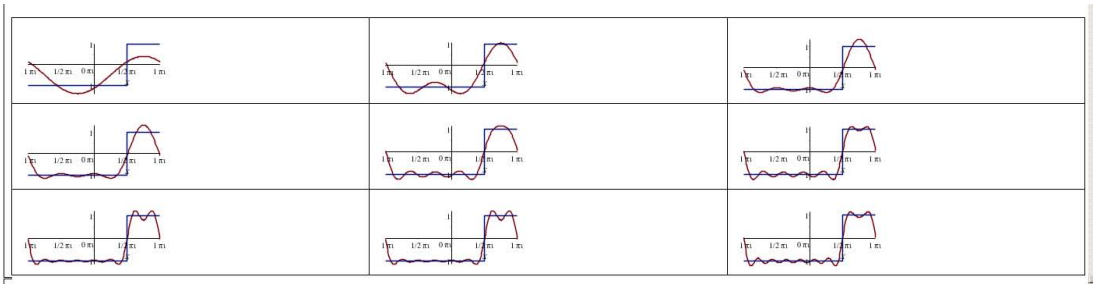
To animate do

```
g:=n->plot(nmaFourier2(f,1,-Pi,Pi,n),x=-2*Pi..2*Pi);
plots:-animate(g,[n],n=1..40);
```

Another version

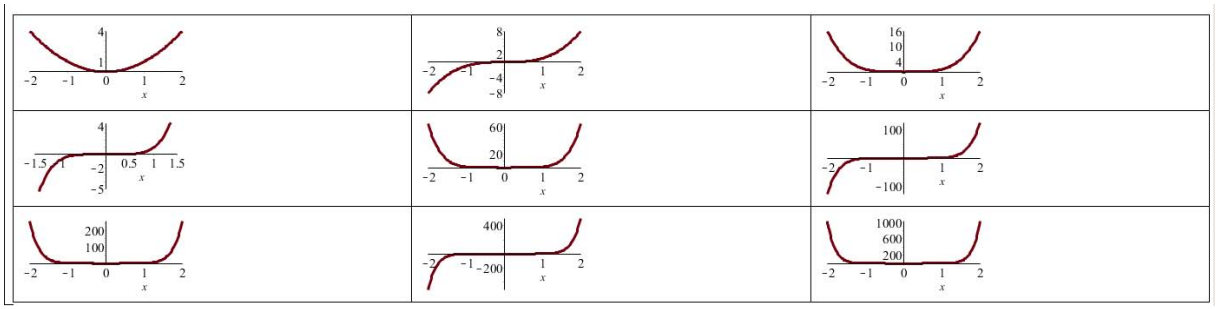
```
restart;
f:=x->piecewise(-Pi<x and x<Pi/2,-1,
                Pi/2<x and x<1,0,1);

assume(n,integer);
nmaFourier2:=proc(f,freq,from_,to_,maxN::integer)
       local n::integer,denomC,denomS,a,b;

       denomC:=( to_ - from_ ) / 2;
       denomS:=( to_ - from_ ) / 2;

       a:=proc(n)
         int(f(x)*cos(n*freq*x),x=from_..to_) /denomC;
       end proc;

       b:=proc(n)
         int(f(x)*sin(n*freq*x),x=from_..to_) / denomS;
       end proc;
```

```
       1/2*a(0) + sum( a(n) * cos(n*freq*x) ,n=1..maxN)
                + sum( b(n) * sin(n*freq*x) ,n=1..maxN)
end proc;

plots[setoptions](title=` `, axesfont=[SYMBOL,8] ,font=[COURIER,1],
   xtickmarks=[seq(evalf(k*Pi/2)=sprintf("%a %s", k/2 ,"pi" ),k= -3..3)],
   ytickmarks=[-1.0="-1",-0.5="",0.0="0",0.5="",1.0="1"]);

B:=array(1..3,1..3);
k:=0;
for i from 1 to 3 do
    for j from 1 to 3 do
       k:=k+1;
       B[i,j]:=plot({f(x),nmaFourier2(f,1,-Pi,Pi,k)},x=-Pi..Pi,size=[200,100]);
    end do;
end do;

plots:-display( B);
```



5.49 How to plot graphs next to each others in a grid like fashion

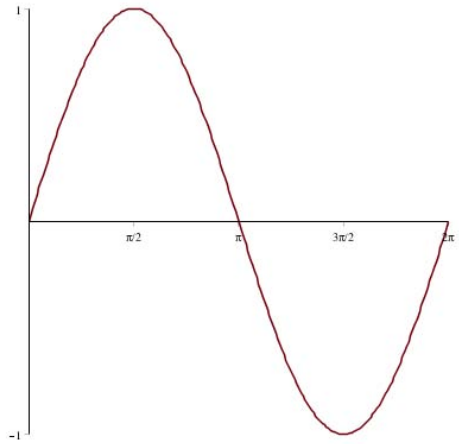
```
restart;
v:=1;
B:=Matrix(3,3);
for i from 1 to 3 do
   for j from 1 to 3 do
       v:=v+1;
       B[i,j]:= plot(x^v,x=-2..2,thickness=3,size=[200,100] );
   end do;
end do;
plots:-display(B);
```



5.50 How to generate Pi on X-axis

From book Maple animation by John Putz

```
plot( sin(x), x=0..2*Pi, xtickmarks=evalf([Pi/2="p/2", Pi="p",
3*Pi/2="3p/2", 2*Pi="2p"]), ytickmarks=[-1,1], axesfont=[SYMBOL,16], labels=["",""] );
```



5.51 How to make output from FunctionAdvisor look better?

From Preben Alsholm

```
res:=FunctionAdvisor(sin):
res2:=op(2,eval(res)):
map(print,res2);
```

or answer by Thomas Richard

```
> FunctionAdvisor( display, sin );
```

5.52 How to do partial fractions?

Use `convert(expr,parfrac)` or `convert(f,fullparfrac)`

5.53 How to generate sequence sum symbolically

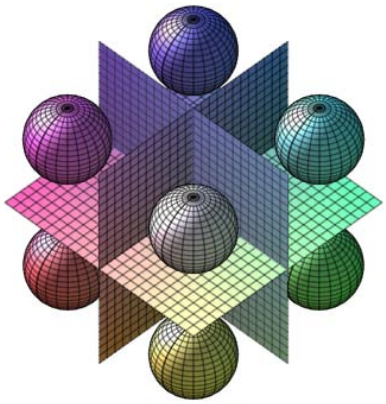
```
n := 7;
f:=sum('a[k]*b[k]','k'=1..n);
```

$$a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4 + a_5b_5 + a_6b_6 + a_7b_7$$

5.54 Nice plot from Maple

from Serge from the net:

```
restart;
with(geom3d):
plane(OYZ,x=0,[x,y,z]):
plane(OXZ,y=0,[x,y,z]):
plane(OXY,z=0,[x,y,z]):
c:=1/2:r:=1/4:
L:=combinat[permute]([-c$3,c$3],3):
S:=seq(sphere(s||i,[point(A||i,op(op(i,L))),r]),i=1..8):
draw([OYZ,OXZ,OXY,S]);
```



5.55 How to check if 2 expressions are the same?

Use `evalb()`. For example `evalb(I*sinh(x)=sin(I*x));` gives `true`

The above does not always work. Only sure way is to do this

```
> m1 := exp(I*n*x);
m2 := (cos(n*x)+I*sin(n*x));
simplify(m1-m2);
simplify(m1-convert(m2,exp));
```

5.56 converting series to factorials

Function by Robert Israel from the net:

```
restart;

thefacts:= [seq(i!,i=2..20)]:
  getfacts:= proc(x::{algebraic,series})
    local i;
    if type(x, {`+`,`*`,series}) then
      map(getfacts,x)
    elif type(x, fraction) then
      getfacts(numer(x))/getfacts(denom(x))
    elif type(x,``^``) then
      getfacts(op(1,x))^op(2,x)
    elif type(x,negint) then
      -getfacts(-x)
    elif type(x,posint) then
      for i from 1 to 19 while irem(x, thefacts[i]) = 0 do od:
      if i = 1 then x
      elif thefacts[i-1] = x then ``(i)!
      else ``(i-1)!*getfacts(x/thefacts[i])
      fi
    else x
    fi
  end;

getfacts(series(sin(x),x));
```

$$\operatorname{series}\left(x-\frac{x^3}{((3))!}+\frac{x^5}{((5))!}+O\left(x^7\right),x,7\right)$$

5.57 How to find what new additions made to Maple?

```
?updates,maple10
```

5.58 Maple can't solve laplace equation and numerically

Maple 2020.

```
restart;
PDE := diff(u(x,y), y$2 ) + diff(u(x,y), x$2) = 0;
BC:= u(x,0)=0, u(x,100)=100, u(0,y)=0, u(10,y)=0;
sol:=pdsolve(PDE,[BC] ,numeric);

Error, (in pdsolve/numeric) unable to handle elliptic PDEs
```

Compare to

```
restart;
PDE := diff(u(x,y), y$2 ) + diff(u(x,y), x$2) = 0;
BC:= u(x,0)=0, u(x,100)=100, u(0,y)=0, u(10,y)=0;
sol:=pdsolve([PDE,BC]);
```

$$u(x,y)=\sum_{n=1}^{\infty}-200\,\frac{\left((-1)^n-1\right)\mathrm{e}^{10\,\pi\,n}\sin\left(1/10\,n\pi\,x\right)\left(\mathrm{e}^{1/10\,n\pi\,y}-\mathrm{e}^{-1/10\,n\pi\,y}\right)}{\pi\,n\left(\mathrm{e}^{20\,\pi\,n}-1\right)}$$

5.59 Some Maple Matrix operations

Create a new matrix, by appending some rows of one matrix to rows from another matrix:

```
restart; with(LinearAlgebra):
A:=< <1|2|3> , <4|5|6> >;
```

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$$

```
B:=< <7|8|10> , <11|12|13> , <14|15|16>  >;
```

$$\begin{bmatrix} 7 & 8 & 10 \\ 11 & 12 & 13 \\ 14 & 15 & 16 \end{bmatrix}$$

Now append first row of A to last 2 rows of B

```
C:=<  A[1,1..-1] ,  B[2..-1,1..-1] >;
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 11 & 12 & 13 \\ 14 & 15 & 16 \end{bmatrix}$$

```
# Now append first column of A to first 2 rows  of B
A[1..-1,1];
B[1..2,1..-1];
C:=< A[1..-1,1] | B[1..2,1..-1]  >;
```

$$\begin{bmatrix} 1 & 7 & 8 & 10 \\ 4 & 11 & 12 & 13 \end{bmatrix}$$

```
#Now remove the middle row of B
B;
B:=<B[1,1..-1] , B[-1,1..-1] >;
```

$$\begin{bmatrix} 7 & 8 & 10 \\ 14 & 15 & 16 \end{bmatrix}$$

```
#now set the diagonal elements of B to be 0
B:=RandomMatrix(3);
for i from 1 to 3 do
    B[i,i]:=0;
end do:
B;
```

$$B := \begin{bmatrix} 0 & 99 & 92 \\ 8 & 0 & -31 \\ 69 & 44 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 99 & 92 \\ 8 & 0 & -31 \\ 69 & 44 & 0 \end{bmatrix}$$

To find inverse.

```
restart;
with(LinearAlgebra):
A:=Matrix( [ [2,0],[4,2] ]);
MatrixInverse(A);
```

$$\begin{bmatrix} 1/2 & 0 \\ -1 & 1/2 \end{bmatrix}$$

To check that for any matrix A, then **A*transpose(A)** is always a matrix which is symmetrical

```
A:=RandomMatrix(2,3);
A.Transpose(A);
```

$$A := \begin{bmatrix} 99 & 44 & -31 \\ 29 & 92 & 67 \end{bmatrix}$$

$$\begin{bmatrix} 99 & 44 & -31 \\ 29 & 92 & 67 \end{bmatrix}$$

how to create a random lower triangular matrix?

```
restart;
with(LinearAlgebra);
A:=RandomMatrix(4,4,outputoptions=[shape=triangular[lower]]);
```

$$\begin{bmatrix} 67 & 0 & 0 & 0 \\ -31 & 92 & 0 & 0 \\ 44 & 29 & 99 & 0 \\ 69 & 8 & 27 & -4 \end{bmatrix}$$

5.60 How set diagonal elements to some value, say 1?

```
restart;
with(LinearAlgebra);
A:=RandomMatrix(5);
LinearAlgebra:-Map[(i,j)->evalb(i=j)](x->1,A);
```

$$A := \begin{bmatrix} 1 & -98 & -76 & -4 & 29 \\ -38 & 1 & -72 & 27 & 44 \\ -18 & 57 & 1 & 8 & 92 \\ 87 & 27 & -32 & 1 & -31 \\ 33 & -93 & -74 & 99 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -98 & -76 & -4 & 29 \\ -38 & 1 & -72 & 27 & 44 \\ -18 & 57 & 1 & 8 & 92 \\ 87 & 27 & -32 & 1 & -31 \\ 33 & -93 & -74 & 99 & 1 \end{bmatrix}$$

5.61 How to multiply roots of a polynomial?

```
eq:=3*x^3+2*x^2+x+5=0;
s:=[evalf(solve(eq,x))];
mul(s[i],i=1..nops(s));
```

Gives

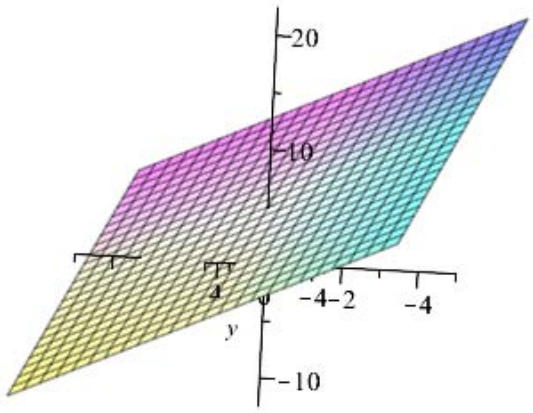
$$eq := 3x^3 + 2x^2 + x + 5 = 0$$

$$s := [-1.342780428,\ 0.3380568807 - 1.061566392\,I,\ 0.3380568807 + 1.061566392\,I]$$

$$-1.666666666 - 1.157693742\,10^{-10}\,I$$

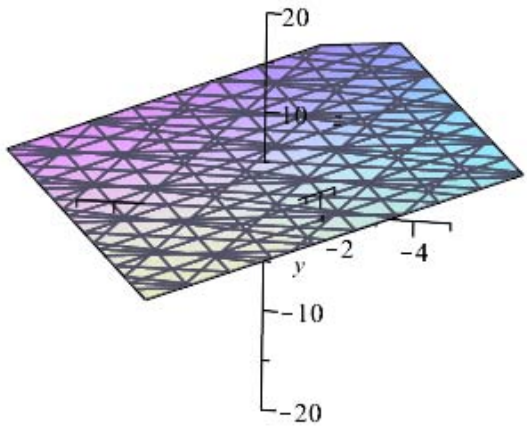
5.62 How to plot a surface in 3D?

```
restart;
eq:=3*x+4*y+2*z=10;
plot3d(solve(eq,z),x=-5..5,y=-5..5,axes=normal);
```



One can also use impliticplot3d

```
restart;
with(plots):
implicitplot3d(3*x+4*y+2*z=10, x=-5..5,y=-5..5, z=-20..20,axes=normal);
```



5.63 How to convert trig to sinc function in an expression

From `http://www.mapleprimes.com/questions/40470-Trigonometric-Function-To-Sinc-Function`

Maple doesn't have a sinc function. If you mean the function sinc(x) = sin(x)/x, you could say something like

```
> eval(expr, {sin = (x -> x*sinc(x)),
              cos = (x -> (x+Pi/2)*sinc(x+Pi/2)),
              tan = (x -> x*sinc(x)/(x+Pi/2)/sinc(x+Pi/2))});
```

5.64 How to find NullSpace and ColumnSpace of a matrix?

```
restart;
with(LinearAlgebra):
A:=Matrix([[1,0,1,0,1],[0,1,0,1,0]]);
NullSpace(A);
ColumnSpace(A);
```

$$A := \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right\}$$

$$\left[\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right]$$

5.65 How to fix the interface to using Maple notation for input?

Go to `tools->optiopn`, and Display, and select Maple notation for input display.

5.66 How to find all solutions using allvalues ?

```
solve(x^2-sin(x),x);
RootOf(-sin(_Z)+_Z^2)

allvalues(%);
RootOf(-sin(_Z)+_Z^2, 0.), RootOf(-sin(_Z)+_Z^2, .8767262154)

evalf(%);
0., .8767262154
```

5.67 Adding only to diagonal of a matrix

Use Map with filter

```
A:=< 1,2,3;4,5,6;7,8,9>;
LinearAlgebra:-Map[(i,j)->evalb(i=j)](x->x+1,A);
```

5.68 How to search help for updates on some package

Go to `http://www.maplesoft.com/support/help/search.aspx`

and type say `updates,Maple17,DE` in the small box there.

5.69 How to work with groups in worksheet

From `http://www.mapleprimes.com/questions/201092-How-To-Insert-New-Paragraph-On-Its-Own` by Carl Love:

```
I use these special keystrokes constantly in my Maple worksheet typing:

   Ctrl-J: Insert execution group below cursor.
   Ctrl-K: Insert execution group above cursor.
   Ctrl-T: Switch from executable code mode to text mode (for entering extended
  formatted comments).
   Ctrl-M: Switch from text mode to executable code mode.
   Shift-Enter (or Shift-Return): Begin a new line in the same execution group.
   Func-3: Split execution group into two (at cursor).
   Func-4: Join cursor execution group with execution group below.
```

5.70 How to read code into worksheet?

Use the read command, as in `read "mycode.mpl"` where `mycode.mpl` is plain text file that contains maple code

5.71 Code editors for Maple

1. `http://www.mapleprimes.com/forum/codeeditormaple`
2. `http://www.mapleprimes.com/blog/joe-riel/emacs-mode-for-maple`
3. `http://www.mapleprimes.com/blog/jacquesc/vim-mode-for-maple`
4. `http://www.maplesoft.com/products/toolboxes/IDE/index.aspx`

5.72 How to find if package is module or table?

New packages are module, which allows using `packageName:-function()` since it is easier. Old packages use tables which needs `packageName[function]()` which is not common.

To find if package is based on module or not, use the command

```
type(combstruct,'`module`');
```

This will return true or false. To know if name is package use the command

```
type(combstruct,'package');
```

5.73 How to replace a string?

```
file_name :=StringTools:-SubstituteAll(file_name,":","-");
```

5.74 How to use geometry and plottools ?

```
restart;
c:= i->([i/(1+i),0],1/(1+i)):
d:= i->([1,1/i],1/i):
geometry:-circle(c1,[geometry:-point(o,2/3,0),1/3],[x,y]):
geometry:-circle(c2,[geometry:-point(o,1,1),1],[x,y]):
geometry:-intersection(o,c1,c2,[u,v]):
plots:-display(plottools:-circle(c(2)),plottools:-circle(d(1)),geometry:-draw(o));
```

To know more about the intersection, use this:

```
geometry:-detail(o);
```

5.75 How to simplify log expressions ?

Use symbolic option

```
restart;
simplify(ln(3^x/2^y) =ln(n),symbolic);
```

5.76 How to simplify hyperbolic expression ?

How to convert

$$\frac{3+2\sinh(x)^2}{\sinh(x)^2\tanh(x)}$$

to

$$3\coth^3(x)-\coth(x)$$

```
restart;
e := (3+2*sinh(x)^2)/(sinh(x)^2*tanh(x));
expand(student[changevar](sinh(x)^2=tanh(x)^2/(1-tanh(x)^2),e));
```

5.77 How to create text file and append string to it?

```
restart;
try
   fd :=-1;
   fd := fopen("C:\\output3.txt",APPEND,TEXT);
catch:
   print(`Unable to open file, error is`);
   print(StringTools:-FormatMessage(lastexception[2]));
end try:

if not(evalb(fd=-1)) then #file open ok
   str:="hello world";
   try
      fprintf(fd,"%s\n",str);
   catch:
      print(`failed to append to file, error is`);
      print(StringTools:-FormatMessage(lastexception[2]));
   finally:
     close(fd);
   end try;
fi:
```

5.78 How to search packages and libraries?

To find in which library a command is do

```
with(LibraryTools);
FindLibrary('int',all); #find which library command int is in

"C:\Program Files\Maple 18\lib\update.mla",
"C:\Program Files\Maple 18\lib\DEsAndMathematicalFunctions18.mla",
"C:\Program Files\Maple 18\lib\maple.mla"
```

To get content of library do

```
restart;
with(LibraryTools):
LibLocation:=cat(kernelopts(mapledir),"/lib/maple.mla");
c:=ShowContents(LibLocation);
```

Then can use this to print the name of each symbol/command, and then use whattype command to find its type

```
seq(c[i,1],i=1..20);
```

To get list of Maple kernel builtin commands and symbols, use this. Written by Acer from Maple prime site:

```
restart:
interface(warnlevel=0):
started := false:
T := 'T':
for i from 1 to 1000 do
  f := eval(parse(cat("proc() option builtin=",i,"; end proc")));
  p := (s->StringTools:-Take(s,StringTools:-Search(";",s)-1))(convert(eval(f),string)[26
  if not type(parse(p),posint) then
    T[i] := p;
    started := true;
  else
    if started then i:=1000; next; end if;
  end if;
end do:
i;
```

```
[ entries(T,nolist) ];
nops(%);
```

The above gives on Maple 18.02 the following

```
["crinterp", "equation", "`{}`", "even", "debugopts",
  "embedded_imaginary", "define_external", "embedded_real",
  "coeff", "cx_zero", "coeffs", "embedded_axis", "conjugate",
  "constant", "convert", "cx_infinity", "dlclose", "identical",
  "divide", "hfloat", "`done`", "function", "`$`", "fraction",
  "denom", "float", "degree", "finite", "disassemble",
  "extended_rational", "diff", "extended_numeric", "frem",
  "`union`", "frontend", "upperbound", "exports", "writeto",
  "factorial", "`xor`", "evalgf1", "type", "expand", "typematch",
  "entries", "unames", "evalb", "unbind",
  "`evalf/hypergeom/kernel`", "atomic", "hfarray", "anything",
  "hastype", "complex", "has", "boolean", "goto", "`:-`",
  "gmp_isprime", "`!`", "genpoly", "anyfunc", "gc", "algebraic",
  "SFloatMantissa", "ssystem", "Scale10", "`stop`", "Scale2",
  "sort", "SearchText", "`[]`", "`~`", "`subset`", "~Array",
  "subsindets", "~Matrix", "streamcall", "~Vector", "subs",
  "Unordered", "table", "ToInert", "system",
  "_hackwareToPointer", "substring", "UpdateSource", "subsop",
  "_maplet", "trunc", "_jvm", "`kernel/transpose`", "_treeMatch",
  "tcoeff", "_savelib", "taylor", "abs", "rtable_num_dims",
  "addressof", "rtable_num_elems", "_unify", "rtable_options",
  "_xml", "rtable_redim", "`and`", "rtable_scale", "andmap",
  "rtable_scanblock", "alias", "rtable_size", "anames",
  "rtable_sort_indices", "assign", "savelib", "assemble",
  "rtable_zip", "array", "select", "appendto", "searchtext",
  "cat", "series", "callback", "selectremove", "bind", "sign",
  "attributes", "setattribute", "ormap", "ArrayOptions", "order",
  "Array", "parse", "`**`", "overload", "`*`", "`::`", "numer",
  "CopySign", "numelems", "`^`", "`or`", "`||`", "op", "nops",
  "seq", "normal", "time", "`not`", "piecewise", "numboccur",
  "`?[]`", "userinfo", "modp2", "inner", "mods", "timelimit",
  "mvMultiply", "traperror", "negate", "rtable_normalize_index",
  "call_external", "rtable_is_zero", "assigned", "rtable_indfns",
  "evalf", "rtable_histogram", "eval", "evaln", "rtable_eval",
  "truefalse", "evalhf", "rtable_convolution", "tabular", "mul",
  "rtableInfo", "zppoly", "`if`", "rtable", "uneval", "remove",
  "sfloat", "rhs", "specfunc", "readlib", "string", "reduce_opr",
  "symbol", "ASSERT", "`?()`", "realcons", "TRACE", "`quit`",
  "relation", "_local", "pointto", "sequential", "add", "print",
```

```
 "set", "SFloatExponent", "iolib", "radical", "SDMPolynom",
 "`int/series`", "protected", "Record", "irem", "procedure",
 "Re", "iquo", "poszero", "isqrt", "real_infinity", "RETURN",
 "is_gmp", "ratpoly", "`+`", "lcoeff", "rational", "OrderedNE",
 "kernelopts", "range", "Object", "NumericEventHandler",
 "icontent", "numeric", "NumericStatus", "igcd", "odd",
 "NumericClass", "ilog10", "nonpositive", "NumericEvent",
 "ilog2", "nonreal", "`implies`", "posint", "NameSpace",
 "indets", "positive", "NextAfter", "indices", "polynom",
 "MPFloat", "`intersect`", "pos_infinity", "MorrBrilCull",
 "`<`", "member", "neg_infinity", "Im", "maxnorm", "name",
 "`<>`", "max", "negint", "`<=`", "map2", "negative", "modp1",
 "nonnegative", "FromInert", "modp", "negzero",
 "EqualStructure", "`minus`", "nonposint", "`>=`", "min",
 "nonnegint", "`>`", "DefaultUnderflow", "lexorder",
 "imaginary", "`=`", "lhs", "indexable", "ERROR", "ldegree",
 "indexed", "EqualEntries", "length", "integer", "macro",
 "list", "DEBUG", "map", "literal", "`..`", "lowerbound",
 "`module`", "Default0", "lprint", "moduledefinition",
 "DefaultOverflow"]
                              296
```

5.79 How to numerically solve a BVP ode and plot the solution?

This one has one solution

```
eq:=diff(u(z),z$2)+(k-1)*diff(u(z),z)/z+lambda*exp(u(z))=0;
sol:=dsolve({subs({k=1,lambda=2},eq),u(0)=1,u(1)=0},numeric,u(z),
            method=bvp[midrich],'abserr'=0.001);
plots[odeplot](sol);
```

This solved coupled ODE's, so there are 2 solutions. Say $x_1(t)$ and $x_2(r)$, It is a little tricky to plot all solutions generated, but here is an example

```
restart;
R := 0.4; px := 32000; Mm := 0.1; Ds := 9; DO2 := 7.2; YXS := 0.3; KS := 10;
Sp := 30; Cb := 8; KO2 := 0.2; R0 := 0.000001; YXO := 0.42857;
Vs := px*1/YXS*(Mm*x2(r))/(KS + x2(r))*x1(r)/(KO2 + x1(r));
Vo := px*1/YXO*(Mm*x2(r))/(KS + x2(r))*x1(r)/(KO2 + x1(r));

eqs := diff(x1(r),r$2) + 2/r*diff(x1(r),r)= Vo/DO2,
diff(x2(r),r$2) + 2/r* diff(x2(r),r)= Vs/Ds;
```

```
ic:=D(x1)(R0)=0,x1(R) = Cb,D(x2)(R0)= 0, x2(R) = Sp;
sol:=dsolve({eqs,ic},numeric,{x1(r),x2(r)},'abserr'=.52,'maxmesh'=1000,output=listproced
```

And now to plot do

```
x1Sol:=rhs(sol[2]);
plot(x1Sol(r),r=0..0.4);

x2Sol:=rhs(sol[4]);
plot(x2Sol(r),r=0..0.4);
```

5.80 How to display on screen for specific width?

This below by Axel Vogt posted on `sci.math.symbolic` which does a nice job of formatting output to specific width.

```
split_for_print:=proc(expr, len)
  # expr = some Maple expression
  # len  = length to split with line breaks
  local L,s,tmp,j;
  s:=convert(expr, string);
  L:=[StringTools:-LengthSplit(s, len)];
  for j from 1 to nops(L) do
  #  if j = nops(L) then printf("%s ;", L[-1])
    if j = nops(L) then printf("%s", L[-1])
    else printf("%s\\\n", L[j]);
    end if;
  end do:
end proc;

evalf[100](Pi);
split_for_print(%, 40);

3.1415926535897932384626433832795028841\
9\
7169399375105820974944592307816406286208\
9986280348253421170 68
```

5.81 Maple IDE links

for VIM

1. `https://code.google.com/p/maplevim/source/browse/trunk/syntax/maple.vim`

in vim, type `set syntax=maple` after putting the file maple.vim in `~/.vim/syntax/maple.vim`. I found maple.vim in above link.

For Maple IDE

MapleIDE18

5.82 loading, remove and finding what packages loaded

use `packages();` to find what packages loaded. use `unwith` to remove package

```
packages();
                                []

with(DynamicSystems):
packages();
                        [DynamicSystems]

unwith(DynamicSystems);
packages();
                                []
```

5.83 some rules of thumbs when using Maple

1. put `restart` in separate execution group
2. do not use `with` inside proc(). Use `uses` instead.

5.84 How to make multiple assumptions on a symbol?

```
assume( A::AndProp(NonZero,constant) );
```

Now can use `is(A,constant);`

5.85 How to check if expression is an equation?

check for ‘=‘ as follows

```
eq:= x=1;
whattype(eq);   #  `=`

if whattype(eq) = `=` then
   print("yes");
else
   print("no");
fi;

      "yes"
```

5.86 How to check if expression is a set?

check for ‘set‘ as follows

```
eq:= {diff(y(x),x)=1,x(0)=1};

if whattype(eq) = `set` then
   print("yes");
else
   print("no");
fi;
                              "yes"
```

5.87 How to export a plot to PDF?

I could only find a way to export to eps

```
plotsetup(default):
plotsetup(postscript, plotoutput=`t.eps`, plotoptions=`color,portrait,height=300`);
plot(sin(x),x=-Pi..Pi,'gridlines');
plotsetup(default):
```

Make sure not to put : at the end of the plot command! else it will not be exported. It has to end with ;

This will same it to `t.eps` in the `currentdir()` location. Then used `ps2pdf t.eps t.pdf`

to convert it to PDF. Or just `ps2pdf t.eps` it will automatically create `t.pdf`

Or `ps2pdf -dCompatibilityLevel=1.4 t.eps` but may it is best to do

`ps2pdf -dCompatibilityLevel=1.4 -dEmbedAllFonts=true t.eps`

Also try adding

`-dPDFSETTINGS=/printer`

to the above. This tells it to optimize it for printing.

Another example of a direction field for an ODE

```
plotsetup(postscript, plotoutput=`t0.eps`, plotoptions=`color,portrait, height=300` );
ode:= diff(y(x),x) = 3*x^2 - 1;
DEtools:-DEplot( ode, y(x), x=-2..2, [y(0) = 0], y=-2..2,
              linecolour=red, color = blue, stepsize=.05,arrows=MEDIUM );
plotsetup(default);
```

5.88 How to find all roots of complex number

To find roots of $(3+4i)^{1/3}$, do

```
fsolve(z^3=(3+4*I),z);

#gives

-1.26495290635775+1.15061369838445*I,
-.363984239564424-1.67078820068900*I,
1.62893714592218+.520174502304545*I
```

5.89 How to convert matrix of matrices to a matrix?

```
A:= Matrix(2, 2, {(1, 1) = 0, (1, 2) = 0, (2, 1) = 0, (2, 2) = 2});
f:=x->`if`(x<>0,x*LinearAlgebra:-IdentityMatrix(2),0*Matrix(2));
B:=map(f,A);
```

Which gives

$$\begin{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \end{bmatrix}$$

now

```
r:=Matrix(convert(B,listlist))
```

Gives

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

5.90 How to do pattern matching in Maple?

Maple has a simple but easy to use pattern matching, which works well. Here are some example. For each case, will show what pattern to detect and how to do it. I am still not very good at pattern matching in Maple and will need to make improvement in this with time.

5.91 Example 1

Detect $\sqrt{xy}$ in expression.

```
restart;
expr:= sin(x)*sqrt(x*y);
if patmatch(expr,a::anything*(b::anything*x*y)^(c::anything),'la') then
    assign(la);
    if c =1/2 or c=-1/2 then
       print("found sqrt(x*y)");
    else
       print("did not find sqrt(x*y)");
    fi;
 fi;
```

But if the expression was $\sin(x)\sqrt{xy}+3$ then the above would fail, because there are a term after $\sqrt{xy}$, so the pattern has to change to

```
restart;
expr:= sin(x)*sqrt(x*y)+3;
if patmatch(expr,a::anything*(b::anything*x*y)^(c::anything)+d::anything,'la') then
    assign(la);
    if c =1/2 or c=-1/2 then
       print("found sqrt(x*y)");
    else
       print("did not find sqrt(x*y)");
    fi;
 fi;
```

5.92 Example 2

There was a case where I wanted to detect form $f(x)g(\frac{y}{x})$, i.e. $f(x)$ can be any expression which is function of x only (it can be constant also) multiplied by a function whose argument must be $\frac{y}{x}$ or a constant multiplied by $\frac{y}{x}$.

This means something like $xg(\frac{y}{x})$ or $x^2e^{3\frac{y}{x}}$ or $f(x)\sin\frac{y}{x}$ or $\cos\frac{y}{x}$ where in this last case $f(x)=1$ which is allowed.

TO FINISH.

5.93 How to find trig identities?

use `trigsubs`, very useful command. For example

```
trigsubs(cos(theta)^3)
```

Gives

$$[1/2\cos(\theta)+1/2\cos(2\theta)\cos(\theta),1/4\cos(3\theta)+3/4\cos(\theta)]$$

5.94 How to find directional derivative of scalar function?

Given $f(x,y,z) = x^2z + y^3z^2 - xyz$ we want to find its directional derivative along the vector n.

One way

```
n:=<-1,0,3>;
g:=VectorCalculus[Gradient](x^2*z+y^3*z^2-x*y*z, [x,y,z]);
Student[VectorCalculus][DotProduct](g,n/LinearAlgebra[Norm](n,2))
```

Gives

$$-\frac{(2xz - yz)\sqrt{10}}{10} + \frac{(6y^3z + 3x^2 - 3xy)\sqrt{10}}{10}$$

Another is

```
Student[MultivariateCalculus][DirectionalDerivative](x^2*z+y^3*z^2-x*y*z, [x,y,z], [-1,0
```

Gives the same result.

5.95 How to check if name is assigned a value?

For simple variable, use `assigned`

```
restart;
x:=10:
assigned(x)
                                true
assigned(y)
                               false
```

For a field in table do

```
restart;
A:=table(["x"=10,"y"=20]):

assigned(A["x"])
                                true
assigned(A["z"])
                               false
```

For field in Record, I do not know how yet, other than using try catch, as assigned does not seem to work for Record fields.

```
restart;
A:=Record('x'=10,'y'=20);
try
   assigned(A:-x)
catch:
   print("no such field in record")
end try;
```

```
                                true
```

```
try
   assigned(A:-z)
catch:
   print("no such field in record")
end try;
```

```
                  "no such field in record"
```

5.96 How to simplify $e^{\ln(x)+\ln(y)}$

given

$$\mathrm{e}^{\frac{2\ln\left(\sqrt{p^2+1}+p\right)+2\ln(a)+\ln\left(p^2+1\right)a}{2a}}+\mathrm{e}^{3x}$$

`simplify(expr)` does not work. So tried `subsindets`

```
restart;
expr := exp((2*ln(sqrt(p^2 + 1) + p) + 2*ln(a) + ln(p^2 + 1)*a)/(2*a))+ exp(3*x);
subsindets(expr,'specfunc( anything, exp )',f->(`if`(has(op(1,f),'ln'),expand(f),f)))
```

$$\left(\sqrt{p^2+1}+p\right)^{\frac{1}{a}}a^{\frac{1}{a}}\sqrt{p^2+1}+\mathrm{e}^{3x}$$

It is possible to also try `simplify(expr,exp)` in some cases, but for the above example, this did not work, i.e. it did not simplify it.

Update December 2023. Trying Maple 2023.2.1, it simplifies the above using `simplify(expr,exp)`

```
restart;
```

```
expr := exp((2*ln(sqrt(p^2 + 1) + p) + 2*ln(a) + ln(p^2 + 1)*a)/(2*a))+ exp(3*x);
simplify(expr,exp)
```

$$sqrtp^2 + 1 \left(\sqrt{p^2+1} + p\right)^{\frac{1}{a}} a^{\frac{1}{a}} + \mathrm{e}^{3x}$$

And

```
restart;
expr:=exp(ln(x)+ln(y));
simplify(expr)
```

$$xy$$

5.97 Basis for Null space, Row space and column space of matrix

Given

$$\begin{bmatrix} 1 & -1 & 0 & 2 \\ 1 & 2 & 2 & -2 \\ 0 & 2 & 3 & -1 \end{bmatrix}$$

Find its Null, Row and Column space basis vectors.

```
restart;
A:=Matrix([[1,-1,0,2],[1,2,2,-2],[0,2,3,-1]]);
LinearAlgebra:-NullSpace(A)
```

$$\left\{ \begin{bmatrix} 0 \\ 2 \\ -1 \\ 1 \end{bmatrix} \right\}$$

```
restart;
A:=Matrix([[1,-1,0,2],[1,2,2,-2],[0,2,3,-1]]);
LinearAlgebra:-RowSpace(A)
```

$$\left[\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & -2 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}\right]$$

```
restart;
A:=Matrix([[1,-1,0,2],[1,2,2,-2],[0,2,3,-1]]);
LinearAlgebra:-ColumnSpace(A)
```

$$\left[\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right]$$

5.98 How to do Gaussian elimination on a Matrix?

Given

$$\begin{bmatrix} 1 & -4 & -3 & -7 \\ 2 & -1 & 1 & 7 \\ 1 & 2 & 3 & 11 \end{bmatrix}$$

Find the new form after Gaussian elimination

```
restart;
A:=Matrix([[1,-4,-3,-7],[2,-1,1,7],[1,2,3,11]]);
LinearAlgebra:-GaussianElimination(A);
```

$$\begin{bmatrix} 1 & -4 & -3 & -7 \\ 2 & -1 & 1 & 7 \\ 1 & 2 & 3 & 11 \end{bmatrix}$$

5.99 How to find Reduced Echelon form of a Matrix?

Given matrix

$$\begin{bmatrix} 5 & 2 & 18 \\ 0 & 1 & 4 \\ 4 & 1 & 12 \end{bmatrix}$$

Find its Reduced Echelon form.

```
restart;
A:=Matrix([[5,2,18],[0,1,4],[4,1,12]]);
Student:-LinearAlgebra:-ReducedRowEchelonForm(A)
```

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

Another option is

```
restart;
A:=Matrix([[5,2,18],[0,1,4],[4,1,12]]);
MTM:-rref(A)
```

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

5.100 How add a new row to bottom of matrix?

Given matrix

$$\begin{bmatrix} 1 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}$$

How to add row

$$[a, b]$$

to end of the matrix?

```
restart;
A:=Matrix([[1,1],[2,3],[4,5]]);
the_row:=convert([a,b],Vector['row']);
ArrayTools:-Concatenate(1,A,the_row);
```

$$\begin{bmatrix} 1 & 1 \\ 2 & 3 \\ 4 & 5 \\ a & b \end{bmatrix}$$

5.101 How to find the cofactor matrix of a matrix?

Use `LinearAlgebra:-Adjoint` and then transpose the result. Since the Adjoint is the transpose of the cofactor.

Given

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$$

then

```
restart;
A:=Matrix([[1,2,3],[4,5,6],[7,8,10]]);
LinearAlgebra:-Transpose(LinearAlgebra:-Adjoint(A))
```

$$\begin{bmatrix} 2 & 2 & -3 \\ 4 & -11 & 6 \\ -3 & 6 & -3 \end{bmatrix}$$

5.102 How to normalize eigenvectors?

When finding eigenvectors of matrix, using `LinearAlgebra`, the vectors are not normalized. How to normalized them so the length is one?

One way is

```
restart;
LA:=LinearAlgebra;
Sx:=Matrix([[0,1,0],[1,0,1],[0,1,0]]);

#this finds eigenvectors in v
lam,v:=LA:-Eigenvectors(Sx);

#this normalize it
B:=map(n -> v[.., n]/norm(v[.., n], 2), [$1..LA:-RowDimension(v)]):
B:=`<|>`(op(B)); #this converts the list back to matrix.
```

$$v = \begin{bmatrix} -1 & 1 & 1 \\ 0 & \sqrt{2} & -\sqrt{2} \\ 1 & 1 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

5.103 How to typeset $\hbar$?

```
expr:=`&hbar;`*x
```

gives

$$\hbar x$$

Notice, the ; is needed. This ‘&hbar‘*x will not work. It must be ‘ℏ‘*x

5.104 How to find the Curl of a vector?

First example

```
restart;
VectorCalculus:-SetCoordinates( 'cartesian'[x,y,z] );
F:=VectorCalculus:-VectorField(<y,-x,0>);
```

$$F = y\bar{\mathbf{e}}_{\mathbf{x}} - x\bar{\mathbf{e}}_{\mathbf{y}}$$

And now

```
VectorCalculus:-Curl(F);
```

$$-2\bar{\mathbf{e}}_{\mathbf{z}}$$

Second example

```
restart;
VectorCalculus:-SetCoordinates( 'cartesian'[x,y,z] );
F:=VectorCalculus:-VectorField(<y*z^2,x*z^2+2,2*x*y*z-1>);
```

$$F = y\,z^2\bar{\mathbf{e}}_{\mathbf{x}} + (x\,z^2 + 2)\bar{\mathbf{e}}_{\mathbf{y}} + (2xyz - 1)\bar{\mathbf{e}}_{\mathbf{z}}$$

And now

```
VectorCalculus:-Curl(F);
```

$$0$$

Since Curl is zero, field is conservative.

Third example, in cylinderical coordinates

```
restart;
VectorCalculus:-SetCoordinates( 'cylindrical'[rho,phi,z] );
F:=VectorCalculus:-VectorField(<0,-rho,2>);
```

$$F = -\rho\bar{\mathbf{e}}_{\mathbf{\Phi}} + 2\bar{\mathbf{e}}_{\mathbf{z}}$$

And now

```
VectorCalculus:-Curl(F);
```

$$2\bar{\mathbf{e}}_{\mathbf{z}}$$

5.105 See all steps in RREF of an augmented matrix

Use `Student:-LinearAlgebra:-GaussJordanEliminationTutor( A, output=steps )` Where A is your augmented matrix.

5.106 How to find column space of matrix?

Do not use the Maple command `LinearAlgebra:-ColumnSpace` for this. it gives the columns in the RREF. The correct way is to obtain the corresponding columns of the pivot columns in the original matrix A. Hence use the command `Basis` like this

```
A:=Matrix([[1,0,0],[1,1,1]]);
LinearAlgebra:-Basis([seq(A[..,i],i=1..LinearAlgebra:-ColumnDimension(A) )]);
```

Which gives

$$\left[\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right]$$

If you use `ColumnSpace` command you'll get this

```
A:=Matrix([[1,0,0],[1,1,1]]);
LinearAlgebra:-ColumnSpace(A);
```

$$\left[\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right]$$

These are different. `Basis` is the correct command to use, which matches the standard definition in textbooks.

5.107 Show step by step. Calculus problem and differential equations.

For integration do

```
Student:-Calculus1:-ShowSolution(Int(x*sin(x),x));
```

The steps are displayed. This does not work all the time. For example

```
integrand:=x*y(x)*diff(y(x),x$2)+x*(diff(y(x),x))^2-y(x)*diff(y(x),x);
Student:-Calculus1:-ShowSolution(Int(integrand,x));
```

gives

Error, (in Student:-Calculus1:-ShowSolution) unable to determine which calculus operation is being applied in this problem; you can provide this information as the 2nd argument on

your call to Rule or Hint

For differential equations, support is limited but these are the steps

```
restart;
ode:=diff(y(x),x)=sin(x);
Student:-ODEs:-ODESteps(ode)
```

Prints the steps. If IC is there, then

```
restart;
ode:=diff(y(x),x)=sin(x);
ic:=y(0)=1;
Student:-ODEs:-ODESteps([ode,ic])
```

5.108 How to obtain list of files with some extension in folder?

Use `FileTools:-ListDirectory`

```
dir_name:="C:/tmp";
currentdir(dir_name); #cd to directory
files_to_process := FileTools:-ListDirectory(dir_name,'all','returnonly'="*.tex"):
numelems(files_to_process)
    100
```

In the above, `files_to_process` is a list of the files in the current folder with extension .tex

5.109 How to delete lines from text file that contains some string?

There was a case when I needed to delete lines from text file that contains a say "foo" as an example.

This is what I did. use readline to read the lines, check, and if the line contains "foo" skip, else write the line to a temporary file. At the line, use Rename to rename the temporary file to the file being read.

```
dir_name:="C:/tmp";
currentdir(dir_name);

tmp_file_name      := "TMP.txt";
source_file_name   := "source.txt";
```

```
file_id             := fopen(tmp_file_name,WRITE):
line                := readline(source_file_name):

while line<>0 do

   if not StringTools:-Has(line,"foo") then
      fprintf(file_id,"%s\n",line);
   fi;

   line := readline(source_file_name):
od:

fclose(file_id);
FileTools:-Rename(tmp_file_name,source_file_name,force=true);
```

5.110 How to truncate a polynomial?

Given $9x^5+4x^4+3x^3+x^2+x+1$ how to truncate it, so that all terms of x^3 and higher are removed?

This can be done as follows

```
restart;
p:=1+x+x^2+3*x^3+4*x^4+9*x^5;
simplify(p,{x^3=0})
```

$$x^2+x+1$$

5.111 How to make a local declare like block inside a proc?

Sometimes it is useful to make a small local piece of code inside a proc, with its own local variables that do not interfere with the variables of the proc. In Ada, this is done using `declare` clause for example. In Maple on can do the same as follows

```
restart;

foo:=proc()
   local n;
   n:=10;
   proc()
```

```
      local n;
      n:=99;
      print("inside inner proc, n=",n);
   end proc();
   print("n=",n);
end proc;

foo();
```

Which prints

```
                "inside inner proc, n=", 99
                          "n=", 10
```

Notice the end of the inner anonymous proc above. It has `end proc();` and not `end proc;` as normal proc. This defines the inner proc and calls it at same time. All the local variables inside the anonymous proc only exist inside that proc and go away after the call, and they do not interfere with the outer proc variables. This way we can declare temporary variables and use them where they are needed.

5.112 Using short name for a proc inside nested modules instead of long name

There was a case where I was making lots of calls from many places to one specific proc inside a module. I did not want to keep using the long name each time.

the command alias did not work. After some trial and error, found that using `use` works. Here is the solution. First this is the original layout

```
restart;

A:=module()
   export B:=module()
        export foo:=proc()
            print("in A:-B:-foo()");
        end proc;
   end module;

   export C:=module()
      export boo:=proc()
             print("in A:-C:-boo()");
```

```
            A:-B:-foo();
      end proc;
   end module;
end module;
```

In the above, the goal is replace `A:-B:-foo();` with just `foo()` and have it bind to `A:-B:-foo();` automatically.

This is done by modifying the above to

```
restart;
A:=module()
   export B:=module()
        export foo:=proc()
            print("in A:-B:-foo()");
        end proc;
   end module;

   use foo=A:-B:-foo in #add this line here
   export C:=module()
      export boo:=proc()
              print("in A:-C:-boo()");
              foo(); #now can just use the short name
      end proc;
   end module;
   end use; #add this line here.
end module;
```

Wrapping the whole module where the short name is used worked.

Any module that needs to use the short name, can do the same. This solved the problem.

5.113 Remove duplicates objects in a list based on condition on a field

I had case where there is list of Objects, and wanted to removed duplicate entries in the list based on if some field is the same among the objects.

This can be done using the command `ListTools:-MakeUnique` and using a proc which checks for the condition. In this example, we want to remove objects where the field `age` in each object is the same.

```
restart;

#create data type
module person_type()
    option object;
    export name::string:="me";
    export age:=25;
end module:

#make few objects
o1:=Object(person_type);
o2:=Object(person_type);
o3:=Object(person_type);
o3:-age:=46;
o4:=Object(person_type);

#make list of them
list_of_people:=[o1,o2,o3,o4];

nops(list_of_people);  #this will print 4

#now delete object if age is same
list_of_people:=ListTools:-MakeUnique(
            list_of_people,
            1,
            proc(a,b)
                evalb(a:-age=b:-age);
            end proc
        );

nops(list_of_people); #this will print 2
```

5.114 How to remove duplicates Vectors from a list?

Converting a list of Vectors to set will not remove duplicates, as each Vector occupies different memory address, even if the structure is the same. To remove duplicate vector, use `ListTools:-MakeUnique` as follows

```
restart;

my_list:=[Vector([1,0]),Vector([1,0]),Vector([2,0])];
```

```
convert(my_list,set); #this will still show the 3 vectors.

ListTools:-MakeUnique(my_list,1,proc(a,b) LinearAlgebra:-Equal(a,b) end proc)

#now only 2 vectors will remain. Duplicate one was removed
```

5.115 How to find all poles and their order of a rational function?

Gives a rational function in x, such as

$$\frac{1}{10\,(x-4)\,(x-5)^3}$$

How to find all its poles which are $x=4$ and $x=5$ and the order of each pole which will be 1 and 3 in this example?

Using `sqrfree` as follows

```
restart;
get_poles_and_order:=proc(r_in,x::symbol)::list;
  local r:=r_in,N::posint;
  local the_poles::list;
  local item;

  r:=normal(r);
  if not type(r,'ratpoly'(anything,x)) then
     error("Not be a polynomial or a rational function in ",x)
  fi;

  the_poles := sqrfree(denom(r),x);
  the_poles := the_poles[2,..]; #we do not need the overall factor
  for N,item in the_poles do
      the_poles[N]:=[solve(item[1]=0,x),item[2]];
  od;
  return the_poles;

end proc:
```

The above proc `get_poles_and_order` returns back a list of lists. Each sublist has its first entry the pole and the second entry the order.

Here are some examples

```
r:=1/(10*(x-4)*(x-5)^3);
get_poles_and_order(r,x)

     #[[4, 1], [5, 3]]
```

The above says there is a pole at $x = 4$ of order 1 and pole at $x = 5$ of order 3.

5.116 find series of function with specific number of terms

Doing `series` in Maple with specific order value, the number of terms generated ofcourse depends on the function. I had need to have the series generated always with same number of terms. I could not find an option in Maple to do that. This function does this. It keeps finding the series for the function with increasing order until the number terms that comes out is what requested. There is an upper limit that can be changed if needed to protect against pathological cases.

```
restart;

get_series_by_terms:=proc(expr,x::symbol,at::numeric,number_terms_needed::posint)
local keep_running::boolean:=true;
local current_order::integer:=0;
local MAX_ORDER_TO_TRY::posint:=100; #change as needed
local result;

    do
        current_order := current_order+1;
        result        := convert(series(expr,x=at,current_order),polynom);
        if nops(result) >=  number_terms_needed or current_order>MAX_ORDER_TO_TRY then
           keep_running:=false;
        fi;
    until keep_running=false;

    return result;
end proc:
```

And now

```
get_series_by_terms(sin(x),x,0,10)
```

returns

$$x-\frac{1}{6}x^3+\frac{1}{120}x^5-\frac{1}{5040}x^7+\frac{1}{362880}x^9-\frac{1}{39916800}x^{11}+\frac{1}{6227020800}x^{13}-\frac{1}{1307674368000}x^{15}+\frac{1}{355687428096000}x^{17}-\frac{1}{121645100408832000}x^{19} \tag{1}$$

5.117 How to call sibling's proc without making the sibling module exported?

Given a parent module A and inside it there are two child modules (local modules) with names say B and C. To call a proc `foo` inside B from another proc inside C, the proc `foo` has to be exported. But the module B does not have to be exported, if we make sure to use `B:-foo()` call instead of full name `A:-B:-foo()` call.

So make sure to use `child:-proc()` from other siblings to avoid having to make each child exported. Making children exported means they can be seen and called directly from outside the parent which his not what we want.

Here is an example

```
restart;

A:=module()  #parent

  export main:=proc()
     C:-foo();
  end proc;

  local B:=module()  #child
     export foo:=proc()
        print("in A:-B:-foo() proc");
     end proc;
  end module;

  local C:=module() #child
     export foo:=proc()
        print("in A:-C:-foo(). About to call A:-B:-foo()");
        B:-foo();  #do this and NOT A:-B:-foo()
     end proc;
  end module;
end module;
```

and now

```
A:-main()
     "in A:-C:-foo(). About to call A:-B:-foo()"
     "in A:-B:-foo() proc"
```

If instead we have written `A:-B:-foo()` in the above call, then Maple will complain with the error `Error, (in foo) module does not export 'B'`

5.118 Convert time to use seconds instead of milliseconds

Maple's command `Value(Time())` returns 13 digits number, which is number of milliseconds from epoch. I wanted this value to be in seconds, to match the file changed time from `FileTools[Status]("A.txt" )` which uses seconds and not milliseconds. I could not find an option to tell `Date` or `Time` to do this. Here is one way to do this.

```
r:=Value(Time());  #r := 16526777498870
length(r);   #13
r:=convert(r, base, 10);
r:=ListTools:-Reverse(r);
r:=r[1..-4]; #remove last 3 digits
nops(r);
r:=parse(cat(op(r)))  #r := 1652677498
length(r); #10
```

This can be made into a function

```
get_time_in_seconds:=proc()::integer;
local r;
r:=Value(Time());
r:=convert(r, base, 10);
r:=ListTools:-Reverse(r);
r:=r[1..-4];
r:=parse(cat(op(r)));
return r;
end proc;

get_time_in_seconds()  #1652679222
```

5.119 Change the summation index letter

I noticed that Maple returns the summation index variable using leading underscore as in `_n` or `_m` which makes the latex looks not as good. Here is an example

```
restart;
dsolve(diff(y(x),x$2)+diff(y(x),x)+y(x)=0,y(x),'formal_series');

y(x) = _C1*Sum((-1/2 - sqrt(3)*I/2)^_n*x^_n/_n!, _n = 0 .. infinity) +
       _C2*Sum((-1/2 + sqrt(3)*I/2)^_n*x^_n/_n!, _n = 0 .. infinity)
```

The latex of the above is

$$y(x) = \_C1\left(\sum_{\_n=0}^{\infty} \frac{\left(-\frac{1}{2}-\frac{i\sqrt{3}}{2}\right)^{-n} x^{-n}}{\_n!}\right) + \_C2\left(\sum_{\_n=0}^{\infty} \frac{\left(-\frac{1}{2}+\frac{i\sqrt{3}}{2}\right)^{-n} x^{-n}}{\_n!}\right)$$

Not seeing an option to change `_n` to `n`, I wrote the following function which takes in the solution, use **subsindets** and remove the leading underscore.

This is the above example showing how to use the function

```
restart;

fix_summation_index:=proc(expr)
local fix_it:=proc(the_sum)
  local the_letter::symbol,the_new_letter::symbol,the_letter_as_string::string;
  the_letter:= op([2,1],the_sum);
  the_letter_as_string:=String(the_letter);
  if the_letter_as_string[1]="_" then
     the_new_letter:=parse(the_letter_as_string[2..]);
     RETURN(subs(the_letter=the_new_letter,the_sum));
  else
     RETURN(the_sum);
  fi;
end proc;

if not(has(expr,Sum)) then
   RETURN(expr);
else
   RETURN(subsindets( expr, 'specfunc( anything, Sum )', f->fix_it(f)));
fi;
```

```
end proc;

sol:=dsolve(diff(y(x),x$2)+diff(y(x),x)+y(x)=0,y(x),'formal_series'):
sol:=fix_summation_index(sol);

y(x) = _C1*Sum((-1/2 - sqrt(3)*I/2)^n*x^n/n!, n = 0 .. infinity) +
       _C2*Sum((-1/2 + sqrt(3)*I/2)^n*x^n/n!, n = 0 .. infinity)
```

The latex now is

$$y(x) = \_C1\left(\sum_{n=0}^{\infty} \frac{\left(-\frac{1}{2}+\frac{i\sqrt{3}}{2}\right)^n x^n}{n!}\right) + \_C2\left(\sum_{n=0}^{\infty} \frac{\left(-\frac{1}{2}-\frac{i\sqrt{3}}{2}\right)^n x^n}{n!}\right)$$

5.120 How to use RootOf to solve an equation?

If we have equation $x = f(t)$ and want to solve for t as function of x and we do not want to use Maple's solve, one option it to explicitly do it use RootOf as follows

```
restart;
eq:= x=f(t);
sol:= t=RootOf(  subs(t=_Z,rhs(eq))-x , _Z);
```

Which gives

```
sol := t = RootOf(f(_Z) - x)
```

The same can be obtained using solve command, but there can be cases where solve fails to do this, so this is an option to use to do it explicitly like this. Just remember to use `_Z` variable.

6 General settings issues and interface

6.1 How to change zoom factor for worksheet to non standard value?

Maple's `interface->default zoom` only lists some values to zoom. For example it does not list 110% or 120% and so on. To make default zoom to something of your choice, on Linux, edit the file

`/home/me/.maple/2025/maplerc` and change the line `Default Zoom=120` to the value of your choice.

Better to do the above when Maple is closed (just in case).

Now Start Maple to see the effect.

This is using Maple 2025 on Linux