Forum

http://wwwopt.mathematik.tu-darmstadt.de/int2ms/

1. register, head-part

2. creating an account

3. login

4. find correct forum (Unix / Maple / General)
   select topic, e.g. new topic, or click on intersting topic

Symbolic Computations

Simplifying an Expression

Maple knows many functions for symbolic expression computations. Here, the most commonly used ones.

The simplify command tries to find a simpler equivalent for a given expression. The rules for the simplification steps follow some heuristics (but of course, the chosen simplification steps themselves are correct).

> $x := 25;$

$$x := 25 \tag{1}$$

> $simplify\left(\sin(x)^2 \cdot x^4 + \cos(x)^2 \cdot x^4\right);$

$$390625 \tag{2}$$

> $restart;$

> $simplify\left(\sin(x)^2 \cdot x^4 + \cos(x)^2 \cdot x^4\right);$

$$x^4 \tag{3}$$

Expanding an Expression

The *expand* command produces a sum of products for polynomials.
A polynomial is a mathematical expression consisting of a sum of terms each of which is a product of a constant and one or more variables with non-negative integral powers. If there is only a single variable, x,
the general form is given by $a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1} x + a_n$, where the a_i are constants (called coefficients).

Examples:

> $p := \left(x^2+3\right) \cdot \left(x^4 + 3 \cdot x^2 - 7\right);$

$$p := \left(x^2+3\right) \left(x^4 + 3\, x^2 - 7\right) \tag{4}$$

> $expand(p);$

$$p \tag{5}$$

> $q := (x+3)\cdot(x-7)\cdot(x+7);\ r := (x+25)\cdot(x-7)\cdot(x+9);\ expand\left(\frac{q}{r}\right);$

$$q := (x+3)\ (x-7)\ (x+7)$$

$$r := (x+25)\ (x-7)\ (x+9)$$

$$\frac{x^2}{(x+25)\ (x+9)} + \frac{10\,x}{(x+25)\ (x+9)} + \frac{21}{(x+25)\ (x+9)} \tag{6}$$

Factorize an Expression

The command *factor* is the opposite of the *expand* command. It factorizes polynomial exprressions.

> $factor(x^2-1);$

$$(x-1)\ (x+1) \tag{7}$$

> $factor(\%\%);$

$$\frac{(x+3)\ (x+7)}{(x+25)\ (x+9)} \tag{8}$$

Normalize fractions

Restructures rational expressions. If possible, an expression is converted to factored normal form. This is the form numerator/denominator, where the numerator and denominator are relatively prime polynomials with integer coefficients.
I.e., common factors are canceled.

> $normal\left(\frac{x^5}{x+1} + \frac{x^4}{x+1}\right);$

$$x^4 \tag{9}$$

> $normal\left(\frac{1}{x} + \frac{x}{x+1}\right);$

$$\frac{x^2+x+1}{x\ (x+1)} \tag{10}$$

> $normal\left(\frac{1}{x} + \frac{x}{x+1}, expanded\right);$

$$\frac{x^2+x+1}{x^2+x} \tag{11}$$

> $simplify\left(\frac{x^5}{x+1} + \frac{x^4}{x+1}\right);$

$$x^4 \tag{12}$$

> $normal\left(\frac{q}{r}\right);$ *#in the output are nominator and denominator relatively prime.*

$$\frac{(x+3)\ (x+7)}{(x+25)\ (x+9)} \tag{13}$$

> $expand\left((1-x)^{10}\right);$

$$1-10\,x+45\,x^2-120\,x^3+210\,x^4-252\,x^5+210\,x^6-120\,x^7+45\,x^8-10\,x^9+x^{10} \tag{14}$$

> $\mathit{normal}\left(\frac{q}{r}, \mathit{expanded}\right);$

>

Programming with Maple

Simple commands

e.g. all direct commands we saw so far.

Comparison Operators (<, >, >, <=, >=)

> $a := 0;\ b := 1;$

$$a := 0$$

$$b := 1 \qquad \mathbf{(15)}$$

> $\mathit{evalb}(a = 0);$ *#evalb prints boolean results to screen*

$$\mathit{true} \qquad \mathbf{(16)}$$

> $\mathit{evalb}(b > 2);$

$$\mathit{false} \qquad \mathbf{(17)}$$

> $\mathit{evalb}(b + a \leq 0);$

$$\mathit{false} \qquad \mathbf{(18)}$$

> $a = 0;$

$$0 = 0 \qquad \mathbf{(19)}$$

Flow Control (if, for, while, ...)

```
if <conditional expression> then <statement sequence>
        | elif <conditional expression> then <statement sequence> |
        | else <statement sequence> |
end if
```

(Note: Phrases located between | | are optional.)

> **if** $(a > 0)$ **then** $f := x^2$ **fi**;

> **if** $(a = 0)$ **then** $f := x^2$ **fi**;

$$f := x^2 \qquad \mathbf{(20)}$$

>

> **if** $(a < 9)$ **then**
   $f := x^2 + 1;$ # *";" is necessary, because: several statements without structure*
   $g := x^2$ # *";" not necessary*
  else
   $g := x^2 + 1;$
   $f := x^2;$
  end if;

$$f := x^2 + 1$$

(21)

$$g := x^2 \tag{21}$$

The **for ...while ... do** loop

```
>
>
```

1) Print even numbers from 6 to 10.

```
> for i from 6 by 2 to 10 do print(i)  end do;
                                   6
                                   8
                                  10                                    (22)
```

2) Find the sum of all two-digit odd numbers from 11 to 99.

```
> mysum := 0;
   for i from 11 by 2 while i < 100 do
     mysum := mysum + i
   end do:
   mysum;
                              mysum := 0
                                 2475                                   (23)
```

3) Multiply the entries of an expression sequence.

```
> restart;
   total := 1 :
  for z in 1, x, y, q^2, 3  do
     total := total·z
   end do:
   total;
   x := 2 :
   q := 3 :
  total;
                                3 x y q^2
                                  54 y                                  (24)
```

3) Add together the contents of a list.

```
> ?cat
> restart;
  y := 3;
   myconstruction := "";
  for z in [1, "+", y, "·", "q^2", "·", 3] do
     myconstruction := cat(myconstruction, z)
  end do;
  myconstruction;
                                 y := 3
                          myconstruction := ""
                          myconstruction := "1"
                         myconstruction := "1+"
                         myconstruction := "1+3"
                        myconstruction := "1+3*"
```

```
                  myconstruction := "1+3*q^2"
                  myconstruction := "1+3*q^2*"
                  myconstruction := "1+3*q^2*3"
                         "1+3*q^2*3"                          (25)
```

```
> ?parse
```

```
> q := 4;
                            q := 4                            (26)
```

```
> qq := parse(myconstruction);
                         qq := 1 + 9 q^2                      (27)
```

```
> qq;
                             145                              (28)
```

Procedures

Flow control constructions, simple commands and comparison operators can be bound together; in a so called
procedure. The simplest possible procedure looks as follow.

```
 proc(parameter sequence)
    statements;
 end proc:
```

```
> restart;
  myfactorial := proc(n)
      local r, i;
      r := 1;
      for i from 1 by 1 to n do
        r := r · i;
        print(r);
      od;
      return r;
    end proc;
myfactorial := proc(n)                                        (29)
    local r, i; r := 1; for i to n do r := r* i; print(r) end do; return r
end proc
```

```
> myfactorial(4);
                              1
                              2
                              6
                             24
                             24                               (30)
```

Maple allows recursive procedure calls:

```
> restart;
   myfactorial2 := proc(n)
```

```
        if (n < 2) then return 1
        else return n·myfactorial2(n − 1);
        fi;
      end proc;
> myfactorial2(4);
                                   myfactorial2(4)                                   (31)
```

Functional-Operators

Maple allows the definition of so called functional operators.

- A functional operator in Maple is a special form of a procedure. Functional operators are written using arrow notation.

```
vars -> result
```

Here, vars is a sequence of variable names (or a single variable) and result is the result of the procedure acting on vars.

- For example, the following

```
x -> x^2
```

represents the function that squares its argument.

- Multivariate and vector functions are also allowed. You must put parentheses around vars or result whenever they are expression sequences. For example, the following functions have the correct syntax.

```
(x,y) -> x^2 + y^2
x -> (2*x, 3*x^4)
(x,y,z) -> (x*y, y*z)
```

```
> restart;
> f := x→ x^4 − 3 ·x + 21;
                                f := x→x^4 − 3 x + 21                                 (32)
> f(3);
                                         93                                           (33)
> g := x^4 − 3 ·x + 21;
                                 g := x^4 − 3 x + 21                                  (34)
> g(4);
                                x(4)^4 − 3 x(4) + 21                                  (35)
> eval(g, x = 3);
                                         93                                           (36)
> h1 := 2·f;
                                      h1 := 2 f                                       (37)
> h1(2);
                                         62                                           (38)
>
> h2 := 2·g;
                                                                                      (39)
```

$$h2 := 2\,x^4 - 6\,x + 42 \tag{39}$$

```
> h2(2);
```

$$2\,x(2)^4 - 6\,x(2) + 42 \tag{40}$$

```
> eval(h2, x = 2);
```

$$62 \tag{41}$$

```
>
> x := 5; simplify(h2);
```

$$x := 5$$

$$1262 \tag{42}$$

```
> h2;
```

$$1262 \tag{43}$$

```
>
>
>
```

The Maple Library

The Maple library consists of for parts:

- the standard library
- the update library
- packages
- share library (user-contributed)

Until now, we only used commands and operations from the standard andthe update library.

Howver: There are so called packeges for more specialized purposes in Maple,
e.g. the LinearAlgebra package for matrx-vector computations or the numtheory-package.
Functions from those packages can be used with the following syntax:

```
 PackageName[FunctionName](FunctionParameters)
```

Here two examples:

```
> restart;
```

$$> A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix};$$

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \tag{44}$$

```
> Transpose(A);
```

$$\mathit{Transpose}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right) \tag{45}$$

```
> LinearAlgebra[Transpose](A); # transposes the matrix A
```

(46)

```
                              [1  3]
                              [    ]                                  (46)
                              [2  4]

> numtheory[divisors](68); # prints the divisors of 68 to the screen
                        {1, 2, 4, 17, 34, 68}                         (47)
```

Often, you want to use a package more intensively. Then you can abbriviate the package-commands with the with()-command:

```
> with(LinearAlgebra);
[&x, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm,     (48)
    BilinearForm, CharacteristicMatrix, CharacteristicPolynomial, Column,
    ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix,
    ConditionNumber, ConstantMatrix, ConstantVector, Copy, CreatePermutation,
    CrossProduct, DeleteColumn, DeleteRow, Determinant, Diagonal, DiagonalMatrix,
    Dimension, Dimensions, DotProduct, EigenConditionNumbers, Eigenvalues, Eigenvectors,
    Equal, ForwardSubstitute, FrobeniusForm, GaussianElimination, GenerateEquations,
    GenerateMatrix, Generic, GetResultDataType, GetResultShape, GivensRotationMatrix,
    GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose, HessenbergForm,
    HilbertMatrix, HouseholderMatrix, IdentityMatrix, IntersectionBasis, IsDefinite,
    IsOrthogonal, IsSimilar, IsUnitary, JordanBlockMatrix, JordanForm, LA_Main,
    LUDecomposition, LeastSquares, LinearSolve, Map, Map2, MatrixAdd, MatrixExponential,
    MatrixFunction, MatrixInverse, MatrixMatrixMultiply, MatrixNorm, MatrixPower,
    MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial, Minor, Modular,
    Multiply, NoUserValue, Norm, Normalize, NullSpace, OuterProductMatrix, Permanent,
    Pivot, PopovForm, QRDecomposition, RandomMatrix, RandomVector, Rank,
    RationalCanonicalForm, ReducedRowEchelonForm, Row, RowDimension, RowOperation,
    RowSpace, ScalarMatrix, ScalarMultiply, ScalarVector, SchurForm, SingularValues,
    SmithForm, StronglyConnectedBlocks, SubMatrix, SubVector, SumBasis, SylvesterMatrix,
    ToeplitzMatrix, Trace, Transpose, TridiagonalForm, UnitVector, VandermondeMatrix,
    VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm, VectorScalarMultiply,
    ZeroMatrix, ZeroVector, Zip]

> with(numtheory);
[GIgcd, bigomega, cfrac, cfracpol, cyclotomic, divisors, factorEQ, factorset, fermat, imagunit,     (49)
    index, integral_basis, invcfrac, invphi, issqrfree, jacobi, kronecker, λ, legendre, mcombine,
    mersenne, migcdex, minkowski, mipolys, mlog, mobius, mroot, msqrt, nearestp, nthconver,
    nthdenom, nthnumer, nthpow, order, pdexpand, ϕ, π, pprimroot, primroot, quadres,
    rootsunity, safeprime, σ, sq2factor, sum2sqr, τ, thue]

> Transpose(A);
                              [1  3]
                              [    ]                                  (50)
                              [2  4]

> factorset(96);
```

```
                              {2, 3}                                   (51)
> divisors(96);
          {1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96}                   (52)
```