Maple for Math Majors

Roger Kraft
Department of Mathematics, Computer Science, and Statistics
Purdue University Calumet
roger@calumet.purdue.edu

8. Manipulating and Simplifying Expressions

8.1. Introduction

In this worksheet we go over some of the details of using the **factor**, **combine**, **expand**, **simplify** and **convert** commands. In addition we try to give some guidelines on which command should be used when. For convenience, this worksheet is organized in two ways. First, it is organized by command and for each command we give a brief overview of how it can be used. Second, this worksheet is organized by the kinds of expressions that can be manipulated and for each kind of expression we show how it can be manipulated using appropriate Maple commands.

```
> 
```

8.2. factor

Maple's **factor** command works in a way that may seem strange to students in calculus courses. Recall that factoring a polynomial like $x^2 - 5\,x + 6$ is equivalent to solving the equation $x^2 - 5\,x + 6 = 0$, that is, if the number a solves the equation, then $x - a$ is a factor of the polynomial. Maple will readily solve the following equation for us and also factor the polynomial.

```
> solve( x^2-5*x+6=0, x );
> factor( x^2-5*x+6 );
```

Now consider the equation $x^2 - 2\,x - 1 = 0$. Maple's **solve** command will solve this equation, but Maple's **factor** command does not factor the polynomial $x^2 - 2\,x - 1$.

```
> solve( x^2-2*x-1=0, x );
> factor( x^2-2*x-1 );
```

The **factor** command seems to be ignoring a basic fact from high school algebra.

```
> 
```

Let us look at another example, one that gives us a hint of what **factor** is trying to do. Consider the polynomial $6\,x^2 - 13\,x + 6$. Let us see how Maple factors this polynomial and how it solves the equation $6\,x^2 - 13\,x + 6 = 0$.

```
> solve( 6*x^2-13*x+6=0, x );
> factor( 6*x^2-13*x+6 );
```

Notice that **factor** did not factor $6\,x^2 - 13\,x + 6$ as $6\left(x - \frac{2}{3}\right)\left(x - \frac{3}{2}\right)$. Since the coefficients in

the polynomial given to **factor** are integers, **factor** returns a factored form that only uses integer coefficients.

```
> 
```

In general, **factor** returns a factored form that uses the same kind of coefficients as the polynomial being factored. If we go back to the example $x^2 - 2x - 1$, we see that **factor** would need to use radicals in the coefficients of the factored form, but the polynomial has only integer coefficients. So **factor** does not factor this polynomial.

Another example of where **factor** will not factor a polynomial is if the factors need to use complex numbers. Consider the polynomial $x^2 - 4x + 13$.

```
> solve( x^2-4*x+13=0, x );
> factor( x^2-4*x+13 );
```

Since the polynomial has integer coefficients but the factored form would need complex coefficients, **factor** does not factor the polynomial.

```
> 
```

There are several ways to get Maple to factor polynomials like $x^2 - 2x - 1$ and $x^2 - 4x + 13$. In the first polynomial, we can tell **factor** that it is OK to use a coefficient containing a certain radical in the factored form. We do this as follows.

```
> factor( x^2-2*x-1, sqrt(2) );
```

Of course, this requires that we know exactly what radical the **factor** command will need. If the factored polynomial needs some other radical, then we need to tell **factor**.

```
> solve( x^2-x-1=0, x );
> factor( x^2-x-1 );
> factor( x^2-x-1, sqrt(5) );
```

Here is an example where more than one radical is needed.

```
> x^4-8*x^2+15=0;
> solve( %, x );
```

Without telling **factor** about any radicals, we get only partial factorization.

```
> factor( x^4-8*x^2+15 );
```

If we tell **factor** about only one of the radicals, then we get more factorization but not full factorization.

```
> factor( x^4-8*x^2+15, sqrt(3) );
> factor( x^4-8*x^2+15, sqrt(5) );
```

If we tell factor about both of the radicals, then we get full factorization.

```
> factor( x^4-8*x^2+15, {sqrt(3), sqrt(5)} );
> 
```

Let us look at a couple of cases where the factored form uses complex numbers. The roots of the polynomial $x^2 - 4x + 13$ are complex numbers whose real and imaginary parts are integers.

```
```

```
> solve( x^2-4*x+13=0, x );
```
Since the polynomial has integer coefficients, the **factor** command has no problem with the integers in the factors but we need to tell **factor** that it is OK to use the imaginary number **I** in the factored form.
```
> factor( x^2-4*x+13, I );
>
```

It is possible for a polynomial to have roots that are complex numbers and the complex roots have radicals in their real or imaginary parts. Here is an example.
```
> solve( x^2+x+1=0, x );
```
To get **factor** to factor such a polynomial, we need to tell it to use both the imaginary number **I** and also the correct radical. Here is an example.
```
> factor( x^2+x+1 );
> factor( x^2+x+1, {I, sqrt(3)} );
>
```

Exercise: Get **factor** to factor this polynomial, $x^2 - 2\sqrt{3}\,x + 1$.
```
>
```

A very simple way to force **factor** to completely factor a polynomial is to make one of the coefficients of the polynomial a decimal number. In that case, **factor** will produce factors, using decimal numbers, that only approximate the true factors. Notice that in the next two commands, there is a decimal point in the last coefficient of each polynomial.
```
> factor( 6*x^2-13*x+6. );
> factor( x^2-2*x-1. );
```
Another way to tell **factor** to use approximate, decimal factors is to put the keyword **real** after the polynomial.
```
> factor( 6*x^2-13*x+6, real );
> factor( x^2-2*x-1, real );
```
Putting a decimal point somewhere in the polynomial does not help for complex factors, as shown by the next example.
```
> factor( x^2-4*x+13. );
```
In the case of complex factors, we need to use the keyword **complex** to tell **factor** to produce approximate, decimal factors.
```
> factor( x^2-4*x+13, complex );
```
Here are two more examples.
```
> factor( x^4-8*x^2+15. );
> factor( x^2+x+1, complex );
>
```

Another way to completely factor each of the above polynomials is to use the following combination of (three) Maple commands. Notice that this method does not need to know beforehand what radical

is needed or whether **I** is needed..

```
> evala(AFactor( x^2-2*x-1 ));
> convert( %, radical );

> evala(AFactor( x^2-x-1 ));
> convert( %, radical );

> evala(AFactor( x^2-4*x+13 ));
> convert( %, radical );

> evala(AFactor( x^2+x+1 ));
> convert( %, radical );
```

Notice that in this method, the form of the factorization can be a little bit different than in the first method.

```
>
```

A third way to factor the above polynomials is to use a special Maple command from the **PolynomialTools** package. The **Split** command always splits a polynomial into linear factors.

```
> PolynomialTools[Split]( x^2-2*x-1, x );
> convert( %, radical );

> PolynomialTools[Split]( x^4-8*x^2+15, x );
> convert( %, radical );

> PolynomialTools[Split]( x^2+2*x+3, x );
> convert( %, radical );
>
```

Notice the difference between **factor** and **Split** with the polynomial $6\,x^2 - 13\,x + 6$.

```
> 6*x^2-13*x+6;
> factor( % );
> PolynomialTools[Split]( %%, x );
>


>
```

8.3. combine

The **combine** command can perform a number of different transformations. Here is a partial list of some of the transformations that it does. (Notice that, in some sense, **combine** is a kind of "factorization" for non-polynomials.)

```
    x^y*x^z          ==>  x^(y+z)
```

```
(x^y)^z          ==>  x^(y*z)
x^n*y^n          ==>  (x*y)^n
sqrt(-x)         ==>  I*sqrt(x)
exp(x)*exp(y)    ==>  exp(x+y)
exp(x)^y         ==>  exp(x*y)
exp(x+n*ln(y)) ==>  y^n*exp(x)  where n is an integer
sin(x)*sin(y)    ==>  1/2*cos(x-y) - 1/2*cos(x+y)
sin(x)*cos(y)    ==>  1/2*sin(x-y) + 1/2*sin(x+y)
cos(x)*cos(y)    ==>  1/2*cos(x-y) + 1/2*cos(x+y)
y*ln(x)          ==>  ln(x^y)
ln(x)+ln(y)      ==>  ln(x*y)
```

Each of these transformation rules is considered to be of a certain type. For example, the first four are of type **power**, the next three of type **exp** (and also type **power**), the next three of type **trig**, and the last two of type **ln**. The **combine** command has some special features and restrictions that are related to specific types of transformations. The following help pages give the details for each of the most useful types of transformations.

```
> ?combine,power
> ?combine,radical
> ?combine,exp
> ?combine,ln
> ?combine,trig
> ?combine,arctan
```

One important feature of the **combine** command is that you can tell **combine** to restrict the type of transformation it can use on an expression. For example, let **f** denote the following expression.

```
> f := exp(x)*exp(y) + sin(x)*sin(y) + sqrt(2)*sqrt(x+1);
```

In the next command, only the trig expressions in **f** are combined.

```
> combine( f, trig );
```

In the next command, only the radical expressions are combined.

```
> combine( f, radical );
```

In the next command, only the exponential and radical expressions are combined (notice that multiple options are enclosed in a pair of brackets).

```
> combine( f, [exp,radical] );
```

In the next command, which does not have any options, all of the possible combinations are performed on **f**.

```
> combine( f );
>
```

It is important to note that not all of the transformations that **combine** can do are correct for all values of the variables in the expression. Consider the third transformation listed above.

```
x^n*y^n  ==>  (x*y)^n
```

If we let **x** and **y** have the value –1 and we let **n** be **1/2**, then this transformation is not correct (why?). In the following **combine** command, since Maple does not know anything about what values **x** and **y** might have, the **combine** command refuses to perform the transformation on the

grounds that Maple does not wish to state something that may not be true.

```
> x^(1/2)*y^(1/2);
> combine( % );
```

There is a way however to force **combine** to perform the transformation.

```
> combine( x^(1/2)*y^(1/2), symbolic );
```

The use of the keyword **symbolic** tells **combine** to do a transformation even if it may produce an incorrect result for some values of the variables. The keyword **symbolic** only works with **combine** for some expression types and sometimes the expression type must also be specified. Here are a few more examples of using the **symbolic** keyword.

```
> combine( ln(x)+ln(y) );
> combine( ln(x)+ln(y), ln, symbolic );
> combine( ln(x)+ln(y), symbolic );
> combine( exp(x)^y );
> combine( exp(x)^y, symbolic );
> combine( (x^y)^z );
> combine( (x^y)^z, symbolic );
>
```

Exercise: Consider the second transformation listed above.

```
(x^y)^z  ==>  x^(y*z)
```

Find specific values for **x**, **y**, and **z** so that **(x^y)^z** is not equal to **x^(y*z)**.

```
>
```

The keyword **symbolic** tells **combine** to do a transformation without any regard for the values of the variables in the expression. In the previous worksheet we saw how to tell Maple something about the kinds of values that we want a variable to represent. For example, we may want to tell Maple that such and such a variable represents a positive integer (without ever assigning the variable a value). Once we have informed Maple about the possible values of the variables in an expression, then the **combine** command can make use of this information to decide if a possible transformation is valid on the expression. For example, the following two commands tell Maple that **a** and **b** are positive.

```
> assume( a>0 );
> assume( b>0 );
```

Now **combine** will do the following transformation without the need for the keyword **symbolic**, since the transformation is correct for any two positive numbers **a** and **b**.

```
> combine( a^(1/2)*b^(1/2) );
```

Now return **a** and **b** back into unassigned variables without any assumptions on them.

```
> a,b:='a','b':
>
```

Exercise: Create an example of **combine** implementing each of the transformations in the list at the beginning of this section.

```
```

```
> 

> 
```

8.4. expand

For polynomials, **expand** is the opposite of **factor**. For many other kinds of functions, **expand** is (almost) the opposite of **combine**. Here is a partial list of some of the transformations that **expand** can do (though some of these transformations need assumptions on the variables before **expand** will do them).

```
x^(y+z)          ==>  x^y*x^z
x^(y*n)          ==>  (x^y)^n
(x*y)^n          ==>  x^n*y^n
exp(x+y)         ==>  exp(x)*exp(y)
exp(x*n)         ==>  exp(x)^n
exp(x+n*ln(y))   ==>  y^n*exp(x)
cos(2*x)         ==>  2*cos(x)^2-1
sin(2*x)         ==>  2*sin(x)*cos(x)
cos(x+y)         ==>  cos(x)*cos(y)-sin(x)*sin(y)
sin(x+y)         ==>  sin(x)*cos(y)+cos(x)*sin(y)
ln(x^y)          ==>  y*ln(x)
ln(x*y)          ==>  ln(x)+ln(y)
```

Here are examples of using **expand** on different kinds of expressions, along with the commands that undo what **expand** does.

For polynomials, **expand** distributes products over sums.

```
> (x+y)*(a+b);
> expand( % );
> factor( % );
> (x+2)^3;
> expand( % );
> factor( % );
> 
```

For the exponential function, **expand** converts exponents into products.

```
> exp(x+y);
> expand( % );
> combine( % );
> simplify( %% ); # this also undoes the expand
> 
```

For trig functions, **expand** produces mostly sum identities. Here are a few examples.

```
> sin(x+y);
> expand( % );
> combine( % );
```

```
> tan(x+y);
> expand( % );
> combine( % );   # this doesn't work
> cos(3*x);
> expand( % );
> combine( % );
> sin(2*x);
> expand( % );
> combine( % );
>
```

One useful feature of the **expand** command is that you can give it an option that prevents the expansion of certain expressions. Here is an example.

```
> expand( (x+y)*(a+b), a+b );
```

The expression **a+b** after the comma kept **expand** from expanding that expression. Here is another example.

```
> (x+y)*(a+b)*(x+2)^3;
> expand( %, x+y, (x+2)^3 );
```

In this example, two expressions were not expanded, **x+y** and **(x+2)^3**.

```
>
```

A word of warning. The option to the **expand** command works differently than the options to some of the other expression manipulation commands. Consider the following example. Let **f** and **g** represent the following expressions.

```
> f := sin(x+y) + exp(x+y);
> g := expand( f );
```

Now look carefully at the results of the following two commands.

```
> combine( g, exp );
> expand( f, exp );
```

In the **combine** command, the **exp** option caused the command to only work on the exponential term. In the **expand** command, the **exp** option *prevented* the expansion from working on the exponential term. So in one command the **exp** option specifies which terms the command does work on, and in the other command the **exp** option specifies which terms the command does not work on. In addition, notice that the **exp** option is playing very different roles in the two commands. In the **combine** command, **exp** is one of only about 15 keywords that can be used as an option to **combine**. But in the **expand** command, **exp** is an expression (as opposed to a keyword) and any valid expression can be used as an option to **expand**. Whatever expression is used as an option will not be expanded by **expand**. Here is a simple example.

```
> f := sin(x+y) + exp(x+y) + exp(a+b);
> expand( f );                 # expand everything
> expand( f, exp );            # don't expand either exponential
> expand( f, exp(a+b) );       # don't expand one of the
```

```
  exponentials
> expand( f, exp(a+b), sin ); # don't expand the sin function
  either
```

Notice another difference between the **expand** and **combine** commands. When we give the **combine** command multiple options to specify which types of expressions to combine, we put the multiple options inside a pair of brackets. But when we give the **expand** command multiple options to specify which expressions not to expand, we just separate the options with commas, no brackets or braces are used.

```
>
```

Exercise: Create an example of **expand** implementing each of the transformations in the list at the beginning of this section.

```
>
```

```
>
```

8.5. simplify

The first thing that should be said about the **simplify** command is that you should not take its name too literally. For one thing, there is no well defined notion in mathematics of what makes one form of an expression "simpler" than an equivalent form of the expression. For example, which of the following two equivalent expressions, $(1-2x)^2$ or $4x^2-4x+1$ is "simpler"? If you need to differentiate (or integrate) the expression, then the second form is simpler to work with since the first form needs the chain rule (or a substitution for integration). If you need to find the roots of the expression, then the first form (the factored form) is much simpler to work with. The only real rule of simplification that one can state is that the simplest form of an expression is the one that makes the next step of your problem easier to do.

Another reason not to take the name **simplify** to seriously is that the **simplify** command does not always manipulate an expression into what most people would think is a simpler form. Consider this example.

```
> simplify( sin(x)^3 );
>
```

Exercise: What exactly did **simplify** do to the expression $\sin(x)^3$?

```
>
```

Consider the following odd behavior of **simplify**.

```
> simplify( (1-x)^9 );
> simplify( (1-x)^9 + 1 );
```

Just a slight change in the first expression caused **simplify** to do a completely different transformation, and neither transformation did much to "simplify" the original expressions.

```
>
```

Even though the name **simplify** may be a bit misleading, this is one of Maple's most useful and important commands. If you have a complicated expression and you want to see if Maple can make some improvements on it, it is always worth trying the **simplify** command.

```
> 
```

Like the **combine** and **expand** commands, **simplify** can perform a number of different transformations on many different kinds of expressions. Here is a partial list of some of the transformations that it does. (Notice that in some cases these transformations are the same as for **combine** and in some other cases they are the same as for **expand**.)

```
x^y*x^z          ==>  x^(y+z)
(x^y)^z          ==>  x^(y*z)
(x*y)^n          ==>  x^n*y^n
sqrt(x^2)        ==>  csgn(x)*x
exp(x)*exp(y)    ==>  exp(x+y)
exp(x)^y         ==>  exp(x*y)
exp(x+n*ln(y))   ==>  y^n*exp(x)
sin(x)^2         ==>  1-cos(x)^2
arcsin(sin(x))   ==>  x
arccos(cos(x))   ==>  x
arctan(tan(x))   ==>  x
ln(x^y)          ==>  y*ln(x)
ln(x*y)          ==>  ln(x)+ln(y)
ln(exp(x))       ==>  x
```

Just as for **combine**, each of these transformations is of a certain type. The following help pages give the details for each of the most useful types of transformations.

```
> ?simplify,power
> ?simplify,radical
> ?simplify,sqrt
> ?simplify,trig
> ?simplify,ln
```

Using these types as options for the **simplify** command we can specify which kinds of subexpressions we want **simplify** to work on. Here are some examples.

```
> f := cos(x)^2+sin(x)^2 + (2^x)^(-3);
> simplify( f );
> simplify( f, trig );
> simplify( f, power );  # look carefully at the result
> simplify( f, trig, power );
> simplify( f, radical );
```

Notice that, unlike the **combine** command (and somewhat more like the **expand** command) when we want to specify several different options in one **simplify** command, we *do not* put the options inside a pair of brackets.

```
> 
```

There is some overlap between the **simplify** command and some of the other expression

manipulating commands, like **combine** and **expand**. Here are a few examples.

```
> f := exp(x)*exp(y);
> simplify( f );
> combine( f );
> g := ln(3*x);
> simplify( g );
> expand( g );
> h := exp(x+3*ln(y));
> simplify( h );
> combine( h );
> expand( h );
```

When there is an overlap between **simplify** and some other command, the exact details of the overlap can be a bit mysterious. Notice how in the next example, **simplify** with an option is equivalent to **combine** without any option, but **simplify** without an option does more simplification. This example brings up two obvious questions. Why doesn't **combine** do the further combining with the exponent? And what is it that **simplify** is using, besides the rules for powers, that allow it to make the further simplification?

```
> simplify( (2^x)^3, power );
> combine( (2^x)^3 );
> simplify( (2^x)^3 );
>
```

Like the **combine** command, **simplify** can perform some transformations that are not always correct. For example, consider the transformation

(a^b)^c ==> **a^(b*c)**

which can be done by both **simplify** and **combine**. If we let **a** be –1, **b** be 2 and **c** be 1/2, then this transformation is not correct. So if **simplify** does not know anything about the values of **a**, **b**, and **c**, then **simplify** will not perform this transformation.

```
> simplify( (a^b)^c );
```

But we can use the keyword **symbolic** to force **simplify** to do the transformation.

```
> simplify( (a^b)^c, symbolic );
```

Using the **symbolic** keyword is often easier than trying to figure out what assumptions are needed for a transformation to be true. But the **symbolic** keyword can be dangerous to use since it can lead to incorrect results.

```
>
```

Exercise: Create an example of **simplify** implementing each of the transformations in the list at the beginning of this section.

```
>
```

```
>
```

8.6. convert

```
> 
```

8.7. Polynomial expressions

The two most important Maple commands for working with polynomials are **factor** and **expand**, but there are also several other more specialized commands, like **sort**, **collect**, **coeffs**, **coeff**, **lcoeff**, **tcoeff**, **degree**, and **ldegree**. In this section we give examples of how each of these commands can be used with polynomial expressions.

Maple can work with both univariate and multivariate polynomials. A polynomial is **univariate** if it has only one unknown and a polynomial is **multivariate** if it has two or more unknowns. Here are some examples of univariate polynomials.

```
> 2*x^3 - 3*x^2 - 17*x + 2;
> -t^5 + sqrt(12)*t^3 + (1/3)*t;
> (1-a)*(a^101 + a^33-10);
```

Here are a few examples of multivariate polynomials.

```
> 1 + x + y + x^2 + x*y + y^2;
> (1+a)*u + (2-3*b)*v;
> (theta-phi)^3 + (3*s-9*t)^2;
> 
```

Exercise: Which of the following expressions are polynomials?

$$3\,w^3 - 5\,w^{(-2)}$$

$$a\,t + b^2\,t^3 - 4\,t^3 - 2\,t$$

$$1 + 2\,\sqrt{x} + 3\,x + 4\,x^2$$

$$\sqrt{1 - y - y^2 - y^3}$$

$$a\,x^n + b\,c^d$$

Hint: If you are not sure, you can ask Maple by using the **type** command. Here is an example.

```
> -t^5 + sqrt(12)*t^3 + (1/3)*t;
> type( %, polynom );
> 
```

Exercise: Is the following polynomial univariate or multivariate?

```
> a[0]+a[1]*x+a[2]*x^2+a[3]*x^3;
> 
```

The **factor** command can work with both univariate and multivariate polynomials.

```
> y^3 - 2*y^2 + y + sqrt(2)*y^2 - 2*y*sqrt(2) + sqrt(2);
> factor( % );
```

```
> 9*u*v^2 + 6*u*v*t + u*t^2 - 18*s*v^2 - 12*s*v*t - 2*s*t^2;
> factor( % );
```

Similarly for the **expand** command.

```
> 3*(phi-1)^2 + 3*phi^4-2*phi^2;
> expand( % );
> (u-2*s)*(3*v+t)^2;
> expand( % );
>
```

Sometimes the **factor** command needs some help with factoring a polynomial.

```
> factor( x^2-2*x-1 );
```

By using the **solve** command, we can see that **factor** will need to use $\sqrt{2}$ in order to factor this polynomial.

```
> solve( x^2-2*x-1, {x} );
```

So we need to tell **factor** that it should use $\sqrt{2}$ when it factors the polynomial.

```
> factor( x^2-2*x-1, sqrt(2) );
```

To factor the next polynomial we need to tell **factor** that it should use complex numbers.

```
> factor( x^2+1 );
> factor( x^2+1, I );
```

We say more about this in the section above about **factor**.

```
>
```

After Maple has done some operation on a polynomial, the terms of the polynomial are often left in some strange order. This can make it difficult to read and analyze the polynomial.

```
> (x-2)^2 + (1-x)^4 + (1-2*x)^3;
> expand( % );
```

When the terms of a polynomial are mixed up like this, the **sort** command can be used to put the terms in order of descending powers.

```
> sort( % );
```

The **sort** command is a bit unusual among the commands that manipulate expressions. To see how, consider the following polynomial.

```
> p := (2-x)*(39*x-45+x^3-11*x^2);
```

Notice that if we apply the **factor** command to **p**, this does not actually change **p** itself.

```
> factor( p );
> p;   # p hasn't changed
```

Similarly, if we apply the **expand** command to **p**, this does not change **p** itself.

```
> expand( p );
> p;   # p hasn't changed
```

But if we apply the **sort** command to **p**, then **p** itself is changed.

```
> sort( p );
> p;   # p has changed
```

The reason that **sort** acts this way is that Maple stores only one copy of any polynomial. If a

polynomial is reentered into Maple with a slightly different order of the terms, Maple will continue to use the order it remembers. Here is an example.

```
> -4*x+x^2+4;
```

If we now expand `(x-2)^2`, we will get the output printed just like the last result, rather than the more expected $x^2 - 4x + 4$.

```
> expand( (x-2)^2 );
```

Here is another example.

```
> 1 - x^3 + x - x^2;
```

Maple printed the terms in the order that we entered them. Now let us reenter this polynomial.

```
> -x^2 - x^3 + x + 1;
```

Maple printed the terms in the order that we first entered them. Maple even keeps track of polynomials when they appear as subexpressions in some larger expression.

```
> cos(exp(x+1-x^2-x^3))/(1-x^2+x-x^3);
```

Now apply `sort` to the polynomial.

```
> sort( %% );
```

Since the `sort` command will force the terms to be reordered in the stored copy of the polynomial, the sorted polynomial becomes the one that Maple remembers from now on.

```
> %%;
```

It is useful to keep this fact about `sort` in mind as you work through the following examples. If you execute a few of the following `sort` commands and then re-execute them, the second time you execute them the results may be different because the way that Maple stores the polynomial has been changed.

```
>
```

For multivariate polynomials, the `sort` command orders the terms by descending total power of each term. The total power of a term is the sum of all the powers of all the unknowns in the term.

```
> p1 := a^3*x^2 + x + a*x - x*a^2 + 2*a + x^4*a;
> sort( p1 );
```

For multivariate polynomials, the `sort` command can also be given a second parameter that tells `sort` specific variables to sort by. For example, the next command tells `sort` to order the terms by descending total power and to put the `a` variable before the `x` variable in each term, and if two terms have the same total degree (like 5), then put the term with the higher power of `a` first.

```
> sort( p1, [a,x] );
```

The next command tells `sort` to put the `x` variable before the `a` variable in each term, and if two terms have the same total degree (like 5), then put the term with the higher power of `x` first.

```
> sort( p1, [x,a] );
```

We can also give `sort` a single variable to sort by.

```
> sort( p1, x );
> sort( p1, a );
```

Notice carefully that in the last result, the terms are no longer sorted by their total power. In that result there is a term of degree 3 before a term of degree 5. When `sort` is sorting with respect to just one variable, it uses the other variables as "coefficients". In other words, the single variable that

you give to **sort** will come last in each term and the terms will be sorted in decreasing powers of that variable.

```
> x*10+a*x^3+x*a^2;
> sort( %, x );
> sort( %, a );
```

Notice once again that the last two results are not ordered by total degree.

```
>
```

There is another way to tell **sort** to sort the terms of a multivariate polynomial. Instead of by the total degree of each term, the terms can be sorted in "alphabetical order", where you specify to **sort** which unknowns are "alphabetically before" which others. This is called "**p**ure **lex**icographical ordering" and is denoted with the keyword **plex**.

```
> a + b^2 + x^3 + y^4;
> sort( % );
> sort( %, [a,b,x,y], plex );
> sort( %, [b,a,y,x], plex );
```

The next few commands compare pure lexicographical ordering with total degree ordering.

```
> p2 := x*y^2*z^3 + y*z^2*x^3 + z*x^2*y^3 + x*y^2 + y*z^2 +
  z*x^2;
> sort( p2, [x,y,z], plex );
> sort( p2, [x,y,z] );
```

Notice in the next few commands how, if a variable is not listed in the **sort** command, then it is used as a "coefficient". That is, the variables listed in **sort** come last in any term and the variables not listed in **sort** come first in the terms.

```
> sort( p2, [x,y], plex );
> sort( p2, [x,y] );
> sort( p2, x, plex );
> sort( p2, x );
```

Notice that the last command seems to be using pure lexicographical ordering instead of total degree ordering.

```
>
```

Exercise: If the **sort** command had been consistent and used total degree ordering in the last command (as the online documentation implies that it should), what would the result have been?

```
>
```

If a polynomial is in a partially factored form, then **sort** will sort the factored parts of the polynomial, without expanding them.

```
> (x+x^2-x^3)*x^2 + x + x^3;
> sort( % );
> a*x^3 + (a*10+x*a^2+a*x^2)*a^3 + x*a^3;
```

In the following examples, look carefully at both the "outer" and the "inner" polynomial.

```
```

```
> sort( % );
> sort( %, x );
> sort( %, a );
>
```

The **collect** command is another command for organizing the terms of a polynomial. The command is used to collect all the terms in a polynomial that have the same degree in some particular unknown.

```
> sqrt(2)*x-x^2+sqrt(5)*x+cos(Pi/5)*x^2;
> collect( %, x );
>
```

The **collect** command is especially useful for organizing multivariate polynomials as univariate polynomials. For example, we can use **collect** to view the following polynomial as a polynomial in **x** (with coefficients that are expressions in **a**).

```
> 3*x^2+a*x+a*x^2+a^2*x;
> collect( %, x );
```

Or we can view the original multivariate polynomial as a polynomial in **a** (with coefficients that are expressions in **x**).

```
> collect( %, a );
>
```

It is also possible to collect terms with respect to more than one variable. Let us start with a complicated multivariate polynomial in four unknowns.

```
> p3 := 3*s*u*v - v*u*t^2 + v*s^2*u^2 + v^2*s*u - v^2*t*u
        + v^2*s*t*u^2 - 2*v^2*s + u*v*s*t;
```

First collect **p3** in terms of **u**.

```
> collect( p3, u );
```

Now collect **p3** in terms of **u** and **v**. Notice that in the following command the order of **u** and **v** is important. The result of the next command is the same as collecting in terms of **v** the "coefficients" from the result of the last command. In other words, the collecting of terms is done, in a sense, sequentially, first in terms of **u** and then in terms of **v**.

```
> collect( p3, [u,v] );
```

Now collect **p3** in terms of **u**, **v** and **s**, in that order. Notice how the next result differs from the last result.

```
> collect( p3, [u,v,s] );
```

To see that the order of the collecting matters, here is **p3** collected in terms of **v** and **u**, in that order.

```
> collect( p3, [v,u] );
```

Try collecting **p3** in terms of several other combinations of the unknowns.

```
>
>
```

Exercise: How do you think the commands **collect(collect(p3,u),v)** and

`collect(collect(p3,v),u)` would compare with the commands `collect(p3,[u,v])` and `collect(p3,[v,u])`?

```
[ >
```

Exercise: Create a multivariate polynomial `p4` in the unknowns `w`, `x`, `y`, `z` so that the following five commands will all have different outputs.

```
[ > p4 := ???;
[ > collect( p4, w );
[ > collect( p4, [w,x] );
[ > collect( p4, [w,x,y] );
[ > collect( p4, [w,x,y,z] );
[ >
```

There is another way to collect terms of a polynomial with respect to more than one variable. Instead of collecting in a "sequential" manner, do the collecting in the variables "simultaneously".

```
[ > collect( p3, [u,v], distributed );
```

In a sense, what the last command gives us is a polynomial in the unknowns `u` and `v` with coefficients that are polynomials in `s` and `t`. The next two commands both make it a bit easier to read the last result. (Notice how the order of the unknowns is used by these `sort` commands.)

```
[ > sort( %, [u,v] );
[ > sort( %, [v,u] );
```

The order of the unknowns does not matter to `collect` when collecting terms in this manner.

```
[ > collect( p3, [v,u], distributed );
[ >
```

The effect of `collect` on a multivariate polynomial is not always easy to predict since the effect depends not just on the polynomial, but also on the form that the polynomial is presently in. Here is an example. Start with the same polynomial `p3` from above and collect it in terms of `u`.

```
[ > collect( p3, u );
```

Now collect the result of the last command in terms of `v`.

```
[ > collect( %, v );
```

Now collect the original polynomial in terms of `v`.

```
[ > collect( p3, v );
```

Notice that the last two commands produce slightly different results, even though they are both collecting the same polynomial in terms of `v`. But the two commands are working on different forms of the same polynomial.

```
[ >
```

It is worth mentioning here that the commands `factor`, `expand`, `sort`, and `collect` are not just for polynomials. They can be applied to almost any expression. With some practice, they can become useful tools for manipulating expressions. But these commands have their most intuitive use with polynomials, and the easiest way to become familiar with them is by using them on

polynomials.

Now we consider some commands that extract information from a polynomial rather than manipulate the form of the polynomial.

The **coeffs** command gives us a list of the coefficients in a polynomial.

```
> expand( (u-2*s)*(3*v+t)^2 );
> coeffs( % );
```

Notice that the list of coefficients is not in the same order that they appeared in the polynomial. Also, notice that it is an error to give the **coeffs** command a polynomial that has not been expanded.

```
> coeffs( (u-2*s)*(3*v+t)^2 );
```

If we **collect** the terms of a polynomial first, then we can give **coeffs** a second parameter and get the list of coefficients from the collected polynomial.

```
> p5 := 3*x^2 + a*x + a*x^2 + a^2*x;
> coeffs( p5 );
> collect( p5, a );
> coeffs( %, a );
> collect( p5, x );
> coeffs( %, x );
>
```

The **coeff** command allows us to find the coefficient of a specific term of a polynomial.

```
> coeff( p5, x^2 );
> coeff( p5, a );
```

Notice that the **coeff** command "collected" the appropriate terms together when it found the coefficient.

```
>
```

Two commands related to **coeff** are **lcoeff** and **tcoeff** for "**l**eading **coeff**icient" and "**t**railing **coeff**icient".

```
> -x^3+17*x^5+21-x;
> lcoeff( % );
> tcoeff( %% );
```

The terms leading and trailing coefficients may make more sense once the polynomial has been sorted.

```
> sort( %%% );
>
```

The command **degree** finds the degree of a univariate polynomial, that is, the highest power of the unknown in the polynomial. For multivariate polynomials, the **degree** command is a bit more complicated; see the online documentation.

```
```

```
> 5*t^2-7*t^3-t+2;
> degree( % );
```
The command **ldegree**, for "**l**ow **degree**", finds the lowest power of the unknown in a univariate polynomial.
```
> 10*x^5+11*x^4+12*x^3+13*x^2;
> ldegree( % );
```
Notice the following.
```
> p6 := 5*x^3 + 6*x^2;
> ldegree( p6 );  # low degree
> lcoeff( p6 );   # leading coefficient
```
This choice of terminology is almost amusing (and it sure can be confusing).
```
>
```

Maple has a lot of other commands for working with polynomials. Here we will briefly mention a few of these commands.

The **completesquare** command can be used to complete the square on a quadratic polynomial. Here is how this command is called (it is in the **student** package).
```
> student[completesquare]( 3*x^2-5*x+2 );
>
```

The **quo** and **rem** commands compute the quotient and remainder for polynomial division.
```
> q := quo( x^3+x+1, x^2+x+1, x );
> r := rem( x^3+x+1, x^2+x+1, x );
```
Verify the results.
```
> q + r/(x^2+x+1);
> simplify( % );
```
Verify them another way.
```
> q*(x^2+x+1) + r;
> simplify( % );
> q, r := 'q', 'r':
>
```

The **taylor** command along with the **convert/polynom** command can be used to find Taylor polynomials.
```
> taylor( exp(x), x=0, 6 );
> convert( %, polynom );
> taylor( ln(x), x=1, 8 );
> convert( %, polynom );
>
```

The command **randpoly** can be used to create randomly generated polynomials.
```
```

```
> randpoly( x );
> randpoly( t );
> randpoly( {u,v}, terms=9 );
```
Go back and re-execute the last three commands.
```
>
```

Exercise: Create a random polynomial with ten terms in the unknowns **w**, **x**, **y**, **z**, and then **collect** and **sort** so that you have a polynomial in the unknowns **x** and **y** with coefficients that are polynomials in **w** and **z**. Create a list of the coefficient polynomials.
```
>
```

The **convert/horner** command can be used to put a polynomial in **Horner form**. Horner form for polynomials is often used to make evaluating a polynomial more efficient.
```
> 5*x^3+4*x^2+3*x+2;
> convert( %, horner );
>
```

Exercise: Suppose you need to evaluate the polynomial $5\,x^3 + 4\,x^2 + 3\,x + 2$ at $x = 6$ and your calculator has only an addition and multiplication button on it. How many multiplications and additions are needed to evaluate the polynomial? Now suppose you rewrite the polynomial in its Horner form, $2 + (3 + (4 + 5\,x)\,x)\,x$. Now how many multiplications and additions are needed to evaluate the polynomial?
(Note: On most computers, multiplying two numbers is much slower than adding two numbers. So rewriting an expression in a form that replaces multiplication operations with additions can help speed up calculations considerably.)
```
>
```

If you are really interested, some other commands for working with polynomials are compoly, content, discrim, galois, gcd, gcdex, spline, Primitive, cyclotomic, orthopoly. If you use the online help to read about any of these commands, be sure to look at both the help browser at the top of the help page and the "See also" section at the bottom of the help page to find many other closely related commands.
```
>
```

```
>
```

8.8. Rational expressions

All of the commands from the last section on polynomial expressions can also be used with rational expressions. In addition, there are a few commands specific to rational expressions. In this section we first go over how the commands from the last section work on rational expressions, and then we go into the commands that are specific to rational expressions, **numer**, **denom**, **normal**, **convert/parfrac**, **convert/contfrac**, and **laurent**

A rational expression is an expression that can be written as a quotient (or ratio) of two polynomials. Notice carefully how this definition was stated. An expression is rational if it *can be* written as a quotient of two polynomials. A rational expression need not actually be written as a quotient. Here are a couple of examples of rational expressions, both univariate and multivariate ones.

```
> (1+3*x-100*x^2)/(sqrt(11)-x^3);
> (s-t)^3/(s^2-s*t+t^2);
> randpoly( [x,y], terms=4 )/randpoly( [y,x], terms=4 );
```

The following are also rational expressions, though it may not be obvious at first.

```
> x^2+3*x^(-2);
> type( %, ratpoly );
> 1/(x-1) + 2/(x^2-1);
> type( %, ratpoly );
```

Notice that Maple uses the abbreviation **ratpoly**, short for "**rat**ional **poly**nomial", as its official name for rational expressions. Another synonym for rational expression that is used by Maple is "rational function".

```
>
```

Exercise: Show that the last two expressions are really rational by writing each one as a quotient of two polynomials.

```
>
```

Exercise: Explain why every polynomial is also a rational expression. (The next command gives an example.)

```
> x^2-x-1;
> type( %, ratpoly );
>
```

Exercise: Why does the following command not produce what one would expect it to produce? Find several ways to fix it.

```
> randpoly(x)/randpoly(x);
>
```

All of the command used to manipulate polynomials can also be applied to rational expressions. However, what they will do to rational expressions is not always obvious. Let us examine how the **factor**, **expand**, **sort**, and **collect** commands act on rational expressions.

The **factor** command will factor the numerator and denominator of a rational expression and cancel any common terms.

```
> (1+2*x+x^2)/(1-x^6);
> factor( % );
> x/(x+x^2);
> factor( % );
```

Notice that in each example, a common term was canceled after the factorization.

```
> 
```

The **expand** command will expand the numerator of a rational expression (but not the denominator) and then distribute the division over the sum in the numerator.

```
> ((x-1)*(x-2))/((x-3)*(x-4));
> expand( % );
> (1+x+x^2+x^3)/x^4;
> expand( % );
> 
```

The **sort** command will sort each of the numerator and denominator of a rational expression.

```
> r1 := (b*x^2+x^4+b^2*x+b*x^5)/(3*b-4*x^2-b^2*x+x^4);
> sort( r1 );
> sort( r1, b );
```

The **sort** command can also be applied to a rational expression that is written in its expanded form.

```
> expand( r1 );
> sort( %, x );
```

Here is another example.

```
> r2 := b*x^(-2)+x^3+2*x+b^2/x^2+b*x^2+3/b^2-b^(-1);
> sort( r2 );
```

Exercise: Explain the last result in terms of sorting terms by their total degree. Is it correct?

```
> 
```

Another example.

```
> sort( r2, b );
> simplify( r2 );
> sort( %, b );
> 
```

Exercise: Show that **sort(simplify(r2),b)** and **simplify(sort(r2,b))** do not produce the same result. (Hint: You can either use the results of the last three commands, or you can try to execute the two command combinations, but if you do the latter, be sure to remember the important difference between **sort** and the other expression manipulating commands.)

```
> 
```

The **collect** command has two ways of working with rational expressions. One way is to collect terms in the numerator and denominator of a rational expression.

```
> (b*x^2+x^2+b^2*x+b*x^3)/(3*b-4*x^2-b*x+x);
> collect( %, b );
> collect( %, x );
```

To see that this is not always the way that **collect** works with rational expressions, consider this next example.

```
> (2*x^2+x^2*b^2+b)/(b*x);
> collect( %, x );
```

Notice that **collect** did not collect the numerator and denominator separately. If it had, the result would have been the following.

```
> ((2+b^2)*x^2+b)/(b*x);
```

What **collect** did was to view the rational expression as a "general polynomial" in **x** and then collect the terms of the general polynomial. To help us see how **collect** viewed this rational expression, let us expand it.

```
> expand( % );
```

To **collect**, this is a general polynomial in **x**, with two terms of degree 1 and one term of degree –1. And so **collect** will collect the two degree 1 terms together. (Compare the next result with the previous **collect** result).

```
> collect( %, x );
```

Here is another example. The following rational expression can be considered a general polynomial in **x** but not in **b**.

```
> (x+b*x^2+b)/(b*x+a*x+b*a*x);
```

So if we collect it in **b**, **collect** will collect the numerator and denominator separately.

```
> collect( %, b );
```

If we collect the rational expression in **x**, **collect** treats it as a general polynomial in **x**.

```
> collect( %%, x );
```

One more example. The following rational expression can be considered a general polynomial in either **x** or **b**.

```
> f := b*x^(-2)+2*x/b+b*x^2+b*x;
> collect( f, x );
> collect( f, b );
```

If we rewrite the rational expression as a numerator over a denominator and then collect, notice that the results are slightly different than the two previous **collect** commands. In the following two **collect** commands, the "coefficients" after collecting remain in a numerator over denominator form.

```
> f := simplify( f );
> collect( f, x );
> collect( f, b );
```

These commands show once again that the order of doing manipulations can have quite an effect on the form of an expression. Sometimes, trying to find the right commands in the just the right order to put an expression in a desired form can be a challenging process of trial and error.

```
>
```

Now let us look at some commands that are more specific to rational expressions. The commands **numer** and **denom** return the numerator and denominator of a rational expression.

```
> (x-a)^2/(2*x^2-x-1);
```

```
> numer( % );
> denom( %% );
>
```

The **normal** command takes a rational expression, puts it over a common denominator and cancels terms common to both the numerator and denominator. The **normal** command tends to leave the numerator in expanded form.

```
> r5 := 1-2*1/(x-3)+6*1/(x-4);
> normal( r5 );
```

The denominator may or may not be left in factored form. Let us add a common factor to the numerator and denominator and then call **normal** again.

```
> expand( numer(%)*(x-5) )/expand( denom(%)*(x-5) );
> normal( % );
```

Notice that **normal** removed the common factor, but this time it left the denominator in expanded form. Sometimes **normal** will leave the numerator in a partially factored form.

```
> r6 := (x-1)*(x+2)/((x+1)*x)+(x-1)/(1+x)^2;
> normal( r6 );
```

The exact form of the result of the **normal** command depends on the form of its input. Equivalent rational expressions can produce slightly different results from **normal**.

```
> expand( r6 );
> normal( % );
```

This time **normal** left the numerator in expanded form. If you want both the numerator and denominator of the result from **normal** fully expanded, then use the option **expanded**.

```
> normal( r6, expanded );
>
```

The **normal** command has some aspects in common with the **factor** and **simplify** commands. Both **factor** and **simplify** will put a rational expression over a common denominator and cancel common factors. The **factor** command will, in addition, leave the numerator and denominator in factored form. But for very complicated rational expressions, **normal** will work faster than **simplify** or **factor**.

There are situations however where the **factor** command is better at simplifying a rational expression than **normal**. Consider the following example.

```
> (x^7+5*x^6+9*x^5+5*x^4-5*x^3-9*x^2-5*x-1)/(x^2-3*x+2);
> normal( % );
> factor( %% );
```

On the other hand, there are examples like the following one, where **normal** is clearly better than **factor**.

```
> (x^8-x-x^7+1)/(x^2-1);
> normal( % );
> factor( %% );
```

Usually, the difference is not so great between the results from these two commands. In general, there is no way to know ahead of time which command will do a better job of simplifying a particular rational expression. Trying to find the best form for an expression is often a process of trial and error, and you need to experiment with several different commands.

```
> 
```

A command that is almost the opposite of **normal** is **convert/parfrac** which converts a rational expression into its partial fraction expansion. Here is how we use this command.

```
> (x^2+1)/(x^2-1);
> convert( %, parfrac, x );
```

The **normal** command will convert this partial fraction expansion back to its original form.

```
> normal( %, expanded );
> 
```

Exercise: Recall from our discussion of Maple names in a previous worksheet that **convert/parfrac** is a helper function for **convert**. Give an example of a direct call to this helper function.

```
> 
```

Here are some examples that are a bit more complicated.

```
> (10+26*x^2+6*x^4+20*x+13*x^3+x^5)/((1+x)^2*(1+x^2)^2);
> convert( %, parfrac, x );

> (3*x^3-3*x^2-62*x+sqrt(3)*x^2-25*sqrt(3)-8)/(x^2-x-20);
> convert( %, parfrac, x );
```

Notice the decimal point in the following rational expression.

```
> (16*x^3-128*x^2+249*x-39)/(24*x^2-192*x+360.0);
> convert( %, parfrac, x );

> (1-2*y+x^2)/(x*(x+y)*y^2);
> convert( %, parfrac, x );
> convert( %%, parfrac, y );
> 
```

The **convert/parfrac** command is very closely related to the **factor** command, since the first step in computing a partial fraction expansion is to factor the denominator of the rational expression. Earlier in this worksheet we mentioned that the **factor** command sometimes needs help in factoring a polynomial.

```
> factor( x^2-2*x-1 );
```

We need to tell Maple that it should use $\sqrt{2}$ when it factors this polynomial.

```
> factor( x^2-2*x-1, sqrt(2) );
```

We must do the same thing when we ask for the partial fraction decomposition of

`1/(x^2-2*x-1)`.

```
> 1/(x^2-2*x-1);
> convert( %, parfrac, x );
> convert( %, parfrac, x, sqrt(2) );
```

We can also get this rational expression's partial fraction expansion the following way.

```
> convert( 1/(x^2-2*x-1), parfrac, x, real );
```

This last command has the advantage that we did not need to know the specific hint to give to `convert/parfrac` (i.e., `sqrt(2)`), but the command has the disadvantage that the result is given only approximately, using decimal numbers. Here is another way to get the partial fraction decomposition that does not need to be told any hints and yet produces a symbolic result.

```
> convert( 1/(x^2-2*x-1), fullparfrac, x );
> simplify( % );
> convert( %, radical );
>
```

Exercise: Look at the second to last result very carefully. Try to make sense out of it. Do not be put off by the funny looking variables _α and _*Z*. The result is a sum, just like the last result. Why does the sum have two terms? Evaluate the sum by hand; it is not too difficult. (We discussed RootOf expressions in a previous worksheet.)

```
>
```

In the next example, `convert/parfrac` is able to compute part of the partial fraction decomposition, but cannot complete it without a hint.

```
> r := (4*x^3-6*x^2-2)/(x^4-2*x^3-2*x+4);
> convert( r, parfrac, x );
```

We need to give `convert/parfrac` a hint so that it can factor the `x^3-2` term in the denominator.

```
> convert( r, parfrac, x, 2^(1/3) );
```

If you are wondering where these hints come from, we can get them from the `solve` command applied to the denominator of the rational expression.

```
> solve( denom(r), {x} );
```

If we give `convert/parfrac` two further hints, then it can factor the remaining quadratic from one of the above denominators and we can get the complete, complex partial fraction expansion.

```
> convert( r, parfrac, x, {I, sqrt(3), 2^(1/3)} );
```

Here is another way to get the complex partial fraction expansion using the `convert/fullparfrac` and `convert/radical` commands. Notice that this method does not need the hints.

```
> convert( r, fullparfrac, x );
> convert( %, radical );
```

Here is still another way to get the complex partial fraction expansion.

```
> convert( r, parfrac, x, complex );
>
```

Exercise: Look carefully at the result of the last command. How can you reasonably "simplify" this numerical result? (Hint: Compare the numerators in the last result with the numerators in the second to last result.)

```
> 
```

Recall from calculus that converting a rational expression into its partial fraction expansion is a common technique for finding the antiderivative of the expression.

```
> r := (-b+a)/(x^2-x*b-a*x+a*b);
> convert( r, parfrac, x );
```

Integrate the partial fraction expansion.

```
> int( %, x );
> combine( %, ln, symbolic );
```

Integrate the original rational expression.

```
> int( r, x );
> simplify( % );
> combine( %, ln, symbolic );
> 
```

Here is another example.

```
> r := 2*sqrt(2)/(x^2-2*x-1);
> convert( r, parfrac, x, sqrt(2) );
```

Now integrate the partial fraction expansion.

```
> int( %, x );
> combine( %, ln, symbolic );
```

This time, integrate the original rational expression.

```
> int( r, x );
> simplify( % );
> convert( %, ln );
> simplify( % );
> combine( %, ln, symbolic );
```

This last example provides an important lesson. Even when Maple has a powerful command like **int** that will do a job for you, it is still sometimes better to use your own mathematical knowledge to help Maple arrive at an answer. In this last example, finding the partial fraction expansion of the rational expression before integrating it lead to a much more direct and simple answer than integrating the rational expression itself.

```
> 
```

Finally, let us look at two interesting commands that are related to rational expressions, **convert/confrac** and **laurent**.

The **convert/confrac** command can be used to put a rational expression into its **continued fraction form**. Like Horner form for polynomials, continued fraction form for rational expressions

is often used to make evaluating a rational expression more efficient.

```
> (x^3+x^2+x+1)/x^4;
> convert( %, confrac, x );
>
```

Exercise: Part (a): Suppose you need to evaluate the rational expression $\frac{x^3+x^2+x+1}{x^4}$ at $x = 6$ and your calculator only has an addition, a multiplication, and a division button. What is the total number of multiplications and divisions needed to evaluate the rational expression? Suppose you rewrite the rational expression in its continued fraction form. What is the total number of multiplications and divisions needed to evaluate the continued fraction?

```
>
```

Part (b): Suppose you rewrite the polynomial that appears in the continued fraction from part (a) in its Horner form. What is the total number of multiplications and divisions needed to evaluate the resulting expression?

```
>
```

Part (c): Suppose you rewrite the polynomial that appears in the numerator of the rational expression from part (a) in its Horner form. What is the total number of multiplications and divisions needed to evaluate the resulting rational expression?

```
>
```

Part (d): Suppose you rewrite the rational expression from part (a) in its expanded form. What is the total number of multiplications and divisions needed to evaluate the expanded expression?

```
>
```

Here is an example of an important application of rational functions. A function like $\cot(x)$ does not have a Taylor series at $x = 0$ because the function has a vertical asymptote there.

```
> taylor( cot(x), x=0 );
```

But $\cot(x)$ does have what is called a Laurent series at $x = 0$. A **Laurent series** approximates a function around an asymptote, just as a Taylor series approximates a function around a point. We can calculate a Laurent series using the **laurent** command from the **numapprox** package.

```
> numapprox[laurent]( cot(x), x=0, 7 );
```

We can convert this Laurent series into a rational function.

```
> convert( %, polynom );
> l := normal( % );
```

The next graph shows how this rational function (the red graph) approximates $\cot(x)$ (the green graph) for x near zero.

```
> plot( [cot(x), l], x=-3*Pi/2..3*Pi/2, -10..10,
        discont=true, color=[green,red]);
```

Notice that the red graph is nearly a perfect match for the green graph near the vertical asymptote at $x = 0$.

```
>
```

Exercise: The third parameter to the **laurent** command determines the degree of the approximating rational function. Try increasing the degree of the rational function (by a fair amount) and observe how that affects the graph of the approximation.

```
[ >
```

```
[ >
```

8.9. Power expressions

In this section we consider the following three power identities. The Maple commands needed to demonstrate these identities are **simplify/power**, **combine/power**, and **expand**.

$$x^y\, x^z = x^{(y+z)}$$

$$(x^y)^z = x^{(y\,z)}$$

$$x^z\, y^z = (x\,y)^z$$

For the first identity, Maple will automatically simplify the left hand side into the right hand side if *y* and *z* are rational numbers.

```
[ > x^(2/3)*x^(-1/2);
```

For the general case, we can use either the **simplify/power** command or the **combine/power** command.

```
[ > x^y*x^z;
[ > simplify( %, power );
[ > combine( %%, power );
```

Notice that the **simplify** and **combine** commands, without the **power** option, also work.

```
[ > x^y*x^z;
[ > simplify( % );
[ > combine( %% );
```

For the other direction, the **expand** command works.

```
[ > x^(y+z);
[ > expand( % );
[ >
```

For the second identity, Maple will automatically simplify the left hand side into the right hand side if *y* and *z* are integers.

```
[ > (x^2)^(3);
```

The **simplify** and **combine** commands will simplify the left hand side into the right hand side if *z* is an integer.

```
[ > (x^y)^3;
[ > simplify( % );
[ > combine( %% );
```

Instead of using specific integers, we can tell Maple to make assumptions about the variables. If we tell Maple to assume that *z* is an integer, then **simplify** and **combine** will transform the left hand side into the right hand side.

```
[
```

```
> assume( z, integer );
> (x^y)^z;
> simplify( % );
> combine( %% );
> z := 'z';
>
```

For anything other than integers, the second identity has problems. Here is one way to see that there is a problem. Consider the following chain of equalities.

$$(x^y)^z = x^{(y\,z)} = x^{(z\,y)} = (x^z)^y.$$

The middle equality is obviously true, and the two outer equalities follow from our second power identity. But it is easy to show that the extreme left and right hand sides of this equation need not be equal. For example, let x be -1, y be 2 and z be 1/2. If you work through the equation with those numbers, you will see that the first equality is false, but the next two equalities are correct. This leads to the following situation. For our second power identity, Maple will automatically simplify the left hand side into the right hand side for some choices of rational numbers y and z, but not for other choices. As the following examples show, it is not at all obvious which ones will get simplified.

```
> (x^(1/2))^2;
> (x^2)^(1/2);
> (x^(2/3))^(1/2);
> (x^(3/2))^(1/3);
> (x^(3/2))^(2/3);
> (x^(2/3))^(3/2);
>
```

Exercise: In Maple, is $(-1)^{\left(\frac{2}{3}\right)}$ the same as $((-1)^2)^{\left(\frac{1}{3}\right)}$ or is it the same as $\left(-1^{\left(\frac{1}{3}\right)}\right)^2$? In a calculus textbook, what would be the (decimal) value of $(-1)^{\left(\frac{2}{3}\right)}$? In Maple, what is the (decimal) value of $(-1)^{\left(\frac{2}{3}\right)}$?

```
>
```

Exercise: (Hard) Find values for x, y, and z so that the three expressions $(x^y)^z$, $x^{(y\,z)}$, and $(x^z)^y$ all have different values.

```
>
```

For the general case of our second power identity, without any assumptions on the variables, the **simplify** and **combine** commands will work if they are given the **symbolic** option (but remember, this will not be a correct transformation for all values of the variables).

```
> (x^y)^z;
```

```
> simplify( %, symbolic );
> combine( %%, symbolic );
```

For the other direction of our second identity, the **expand** command will work but only if one of *y* or *z* is an integer.

```
> x^(y*3);
> expand( % );
```

Notice that if we tell Maple to assume that one of *y* or *z* is an integer, **expand** will not do the transformation.

```
> assume( z, integer );
> x^(y*z);
> expand( % );
> z := 'z';
>
```

Now let us turn to our third power identity. Like our second identity, the third one is not true for all possible values of the variables. For example, $\sqrt{-1}\,\sqrt{-1}$ is not equal to $\sqrt{(-1)\,(-1)}$. So we should not expect Maple to implement this identity without some assumptions. Unfortunately, Maple seems to have a lot of trouble with this identity, at least in the direction of transforming the left hand side into the right hand side. For example, if *z* is an integer, then the identity is clearly true. So let us tell Maple to assume that **z** is an integer.

```
> assume( z, integer );
```

Now **simplify** can do the transformation.

```
> x^z*y^z;
> simplify( % );
```

But **combine/power** cannot do the transformation.

```
> combine( %%, power );
> z := 'z';
```

Now replace **z** with an actual integer.

```
> x^3*y^3;
```

And now, strangely enough, neither **simplify** nor **combine/power** can do the transformation!

```
> simplify( % );
> combine( %%, power );
```

This identity is also true if one of *x* or *y* is positive. Let us tell Maple to assume that **y** is positive.

```
> assume( y, positive );
```

But neither **simplify** nor **combine/power** will do the transformation.

```
> x^z*y^z;
> simplify( % );
> combine( %%, power );
```

But **simplify** will work if we use an undocumented option, **commonpow**. (At least this option is undocumented in Maple's online documentation. I found out about it in *A Guide to Maple*, by E. Kamerich, page 195. And it is not really an option, it is a helper function, **simplify/commonpow**

)
```
> simplify( %%%, commonpow );
> y := 'y';
```
If we tell only **simplify** about the assumptions, then the integer assumption still works and the positive assumption only works with the **commonpow** option.
```
> x^z*y^z;
> simplify( % ) assuming z::integer;
> simplify( %% ) assuming y::positive;
> simplify( %%%, commonpow ) assuming y::positive;
```
For the general case, without any assumptions, we need the (undocumented) **simplify/commonpow** function with the **symbolic** option (but remember, this will not be a correct transformation for all values of the variables).
```
> x^z*y^z;
> simplify( %, commonpow, symbolic );
```
Notice that **combine/power** does not work, even with the **symbolic** option.
```
> combine( %%, power, symbolic );
>
```

For the other direction of our third power identity, things are a bit better. We get automatic simplification if *z* is an integer.
```
> (x*y)^3;
```
But if we tell Maple to assume that z is an integer, then neither **expand**, **simplify** nor **simplify/commonpow** will do the transformation.
```
> assume( z, integer );
> (x*y)^z;
> expand( % );
> simplify( %% );
> simplify( %%%, commonpow );
> z := 'z';
```
If one of *x* or *y* is a rational number, then **expand** and **simplify** will both work.
```
> (5/2*y)^z;
> expand( % );
> simplify( %% );
```
But **simplify/commonpow** will not do the transformation.
```
> simplify( %%%, commonpow );
```
If we tell Maple to make the assumption that one of *x* or *y* is positive, then **expand** and **simplify** will both do the transformation.
```
> assume( y, positive );
> (x*y)^z;
> expand( % );
> simplify( %% );
> y := 'y';
```

```
> 
```

In the general case, with no assumptions at all, we need the **simplify** command with the **symbolic** option (but remember, this will not be a correct transformation for all values of the variables).

```
> (x*y)^z;
> simplify( %, symbolic );
> 
```

In the next section, we give more examples of working with powers in the special case where the exponents are rational numbers, i.e., radical expressions. And in a later section we give some specific information about the power identities from this section in the special case of the base being **e**, i.e., exponential expressions.

```
> 
```

```
> 
```

8.10. Radical expressions

Maple has a few commands that are specifically for working with expressions containing radicals, i.e., exponents that are rational numbers. The commands are **radsimp**, **radnormal**, **rationalize**, **surd** and **root**. In addition, the **simplify** command has the helper functions **simplify/radical** and **simplify/sqrt**, and the **combine** command has the helper function **combine/radical**.

Trying to simplify expressions containing radicals can be tricky. Here is an example.

```
> r := (sqrt(2)-sqrt(3))/(sqrt(2)+sqrt(3));
```

The **radsimp** command does not do anything with this expression,

```
> radsimp( r );
```

unless we also give it an extra option.

```
> radsimp( r, ratdenom );
```

Now expand this result,

```
> expand( % );
```

and combine the last result.

```
> combine( % );
```

The **radnormal** command does not do much with our original expression,

```
> radnormal( r );
```

unless we provide it also with an extra option, in which case it does a bit more than **radsimp** did.

```
> radnormal( r, rationalized );
```

We can also use the **rationalize** command on our expression and get the same result that **radsimp** provided.

```
> rationalize( r );
```

The **simplify/radical** command does not do much.

```
```

```
> simplify( r, radical );
```
Neither does **simplify/sqrt**.
```
> simplify( r, sqrt );
```
The **combine/radical** command does nothing to this expression.
```
> combine( r, radical );
```
Surprisingly enough, even though the expression that we are working with does not have any variables in it, the **factor** command works!
```
> factor( r );
```
These commands show that it is not at all obvious which commands will work to simplify a radical expression. In the rest of this section, we give a lot of examples that try to demonstrate some of what can and cannot be done with radical expressions.
```
>
```

Here is another expression that we can try several different commands on.
```
> g := (x^2-1)/(1+sqrt(x));
```
The **rationalize** command provides a simplified expression (in a factored form).
```
> rationalize( g );
```
The **radsimp** command does nothing,
```
> radsimp( g );
```
unless we give it the **ratdenom** option.
```
> radsimp( g, ratdenom );
```
The **radnormal** command does nothing,
```
> radnormal( g );
```
unless we give it the **rationalized** option.
```
> radnormal( g, rationalized );
```
The **factor** command only does the obvious factorization of the numerator,
```
> factor( g );
```
unless we give it a (somewhat unusual) hint, in which case it produces the same result as **rationalize**.
```
> factor( g, sqrt(x) );
```
The following three commands do not do anything.
```
> simplify( g, radical );
> simplify( g, sqrt );
> combine( g, radical );
>
```

Exercise: Try all of the above commands on the expression $\frac{\sqrt{x}-y}{x-y^2}$.
```
>
```

Exercise: Part (a): Derive the identity $\frac{x^2-1}{1+\sqrt{x}} = (\sqrt{x}-1)(x+1)$ yourself using paper and pencil.

```
[ >
```

Part (b): Given that the command `factor(g,sqrt(x))` worked on the expression `g`, what do you think the command `factor(x-1,sqrt(x))` should produce? Does it?

```
[ >
```

Working with radicals in Maple can be different from what you are used to from algebra and calculus classes. In particular, square roots and cube roots in Maple can act differently from what you might expect. For example, in the last section we saw that Maple interprets the expression `(-1)^(2/3)` differently from what it is in calculus. As another example, since $(-2)^3 = -8$, it is reasonable to expect `(-8)^(1/3)` to evaluate to –2. But this is not what Maple does.

```
[ > (-8)^(1/3);
[ > simplify( % );
```

To get the answer –2 (instead of a complex number) we need to use a special command with a very strange name, `surd`.

```
[ > surd(-8,3);
```

The expression `surd(x,n)` (where `n` is supposed to represent an integer) is a special way of denoting `x^(1/n)`.

```
[ > surd(x,n);
[ > convert( %, power );
```

The most common use for `surd` is, as above, to get real (instead of complex) answers for odd roots of negative numbers. Here is another example of this. In calculus, the function `x^(1/3)` is considered the inverse of the function `x^3`, so their graphs should be reflections of each other with respect to the line $y = x$. But this is not how Maple graphs them.

```
[ > plot( [x^(1/3), x^3], x=-1..1, scaling=constrained );
```

In the above graph, Maple computed complex numbers for the cube roots of the negative `x`'s and then it could not graph these complex numbers, so there is no graph of `x^(1/3)` for negative `x`. If we instead plot the function `surd(x,3)` along with `x^3`, then we get the graph that we were expecting.

```
[ > plot( [surd(x,3), x^3], x=-1..1, scaling=constrained );
```

Here is a way to use a trick, instead of the `surd` function, to get the above graph.

```
[ > x/abs(x)^(2/3);
[ > plot( [%, x^3], x=-1..1, scaling=constrained );
[ >
```

Now let us turn to an example with square roots. In calculus, the expression $\sqrt{x^2}$ simplifies to $|x|$, and if we know that x is a positive number, then $\sqrt{x^2}$ simplifies to x. But that is not how Maple handles this expression. Let us try various commands to simplify `sqrt(x^2)`.

```
[ > simplify( sqrt(x^2) );
```

This result is correct for all complex numbers, which is what Maple, by default, assumes that a

variable represents. But it is not what we were expecting. Let us try **radsimp**.

```
> radsimp( sqrt(x^2) );
```

So it seems that **radsimp** did not assume, as did **simplify**, that **x** is a complex number. We can get this same result from **simplify** by using the option **symbolic**.

```
> simplify( sqrt(x^2), symbolic );
```

But what about the case where **x** is assumed real and the simplification should produce the absolute value of **x**? We can tell **simplify** to assume that **x** is real, instead of complex, this way.

```
> simplify( sqrt(x^2) ) assuming real;
```

These last few examples show that even with radicals as simple as square roots and cube roots, working with them need not be all that simple and it helps to be aware of the specifics of how Maple deals with these and other radicals.

```
>
```

Many of the Maple commands that work with polynomials actually work with "general polynomials" where the terms can have variables raised to any rational exponent. Here is an example of a general polynomial.

```
> f := b*x^(-1/2)+2*sqrt(x)/b+b*x^2+b*sqrt(x)-1/(2*sqrt(x));
```

We can sort this general polynomial.

```
> sort( f, x );
```

We can use **collect** to collect terms of the general polynomial.

```
> collect( f, x );
```

And **coeff** can find the coefficient of a radical term.

```
> coeff( f, x^(-1/2) );
>
```

Finally, let us look at some conventions and notations that Maple has for working with radicals.

Maple will automatically rationalize some numbers.

```
> sqrt(1/2);
> 1/sqrt(3);
```

But not other numbers.

```
> 1/(1+sqrt(2));
```

For this number we need to specifically ask that it be rationalized.

```
> rationalize( % );
```

Even though Maple will automatically rationalize a numeric expression like **1/sqrt(2)**, it will not do so with a variable expression.

```
> 1/sqrt(x);
```

In fact, Maple cannot rationalize the previous expression.

```
> rationalize( % );
```

The reason is because of the following automatic simplification.

```
> sqrt(x)/x;
```

In other words, Maple will automatically rationalize numeric expressions like $\frac{1}{\sqrt{2}}$ into $\frac{\sqrt{2}}{2}$, but it will automatically simplify symbolic expressions like $\frac{\sqrt{x}}{x}$ into $\frac{1}{\sqrt{x}}$.

Maple will automatically convert the square root of a negative number into its imaginary number form.

```
> sqrt(-5);
> sqrt(-3/2);
>
```

Exercise: Why do you think that Maple does not automatically convert an expression like `sqrt(-x)` into $I\sqrt{x}$?

```
>
```

The `root` command can be used to express fractional exponents. The expression `root(x,n)` can be used in place of `x^(1/n)`, as long as `n` represents an integer (either positive or negative).

```
> root(x, 3);
> x^(1/3);
> root(3^4, 5);
> 3^(4/5);
> root(x, -2);
> x^(-1/2);
```

Notice that the following kind of expression is not allowed by `root`.

```
> root(3, 4/5);
```

There is another notation for the `root` command.

```
> root[5](10);
```

The `root` command is not completely equivalent to using radical exponents. Here is one example.

```
> root[5](x^4) = x^(4/5);
```

The `root` command does some different simplifications than when just using exponents.

```
> root[5](9) = 9^(1/5);
> root[3](4/5) = (4/5)^(1/3);
> root[-2](x^3) = x^(-3/2);
```

Here is a real interesting example. Is the following equation true? (It is.)

```
> root[3](24) = 24^(1/3);
```

The `root` command also allows the use of the keyword `symbolic`.

```
> root[2](x^2);
> root[2](x^2, symbolic);
>
```

```
>
```

8.11. Exponential expressions

The two most basic identities for the exponential function are

$$\mathbf{e}^{x}\,\mathbf{e}^{y} = \mathbf{e}^{(x+y)}$$

$$(\mathbf{e}^{x})^{y} = \mathbf{e}^{(xy)}.$$

The Maple commands associated with these two identities are **simplify/power**, **combine/exp**, and **expand**.

The first identity is true for all complex values of the variables. So we should expect Maple to do the associated transformations without any need for assumptions on the variables. We can transform $\mathbf{e}^{x}\,\mathbf{e}^{y}$ with either the **simplify** or the **combine** commands.

```
> exp(x)*exp(y);
> simplify( % );
> combine( %% );
```

And we can transform $\mathbf{e}^{(x+y)}$ using the **expand** command.

```
> exp(x+y);
> expand( % );
>
```

Now let us turn to the second identity. This identity is not true for all values of the variables. For example, let x be $-\pi I$ and let y be 1/2.

```
> identity := exp(x)^y = exp(x*y);
> subs( x=-Pi*I, y=1/2, identity );
> simplify( % );
```

In fact, we can even find values for x and y so that each of the expressions $(\mathbf{e}^{x})^{y}$, $\mathbf{e}^{(xy)}$, and $(\mathbf{e}^{y})^{x}$ has a different value.

```
> terms := [ exp(x)^y, exp(x*y), exp(y)^x ];
> subs( x=-Pi*I, y=2*Pi*I, terms );
> simplify( % );
```

The following command will simplify things some more.

```
> evalc( % );
```

Notice that the first number is very small, the second is quite large, and the third is exactly 1. The following command will make this clear.

```
> evalf( % );
```

So we should not expect Maple to do the transformations associated with this identity without some assumptions about the variables.

```
>
```

The second identity is true if y is an integer, or if both x and y are real numbers. The following commands show that with an appropriate assumption, both **combine** and **simplify** can be used to make the transformation in one direction, and **expand** can be used in the other direction. First,

transform $(\mathbf{e}^x)^y$ where y is an integer

```
> exp(x)^5;
> simplify( % );
> combine( %% );
```

Here is the general case where we assume that y represents an integer.

```
> exp(x)^y;
> simplify( % ) assuming y::integer;
> combine( %% ) assuming y::integer;
```

Now use **simplify** and **combine** to do the same transformation with the assumption that both x and y are real numbers.

```
> exp(x)^y;
> simplify( % ) assuming real;
> combine( %% ) assuming real;
```

We can use **simplify** and **combine** with the **symbolic** keyword to transform $(\mathbf{e}^x)^y$ in the case where we make no assumptions about the variables.

```
> exp(x)^y;
> simplify( %, symbolic );
> combine( %%, symbolic );
>
```

Now transform $\mathbf{e}^{(x\,y)}$ where one of either x or y is an integer

```
> exp(5*x);
> expand( % );
```

We can also tell Maple to assume that one of the variables, say y, is an integer, but unfortunately, **expand** will not make the transformation.

```
> exp(x*y);
> expand( % ) assuming y::integer;
```

And if we tell Maple to assume that both x and y are real numbers, then **expand** still cannot do the transformation.

```
> exp(x*y);
> expand( % ) assuming real;
>
```

For identities that involve exponentials together with logarithms, see the section below on logarithmic expressions.

```
>
```

```
>
```

8.12. Trigonometric expressions

Maple knows a lot of trigonometric identities. In this section we show which Maple commands are

needed for the most important and common trig identities. The commands that we use the most are **simplify/trig**, **combine/trig**, **expand**, and **convert**.

Before going into the details of trigonometric manipulations, a bit of a warning. Using Maple commands to manipulate trig expressions can be very non intuitive. For example, the **expand** command does nothing to the following trig expression.

```
> expand( sin(x)^3 );
```

However, the **simplify** command does "expand" the expression (and it hardly seems to make it simpler).

```
> simplify( sin(x)^3 );
```

The **combine** command will also "expand" this expression, and it is hard to figure out what is being "combined".

```
> combine( sin(x)^3 );
```

As we have mentioned before, learning to use Maple's abilities at algebraic manipulation involves a lot of practice and quite a bit of trial and error. When faced with a particular trigonometric expression that you wish to simplify, do not be surprised if you need to experiment with a variety of commands in a variety of orders before you get what you want (or something close to it).

```
>
```

Maple can do a lot of automatic simplifications of simple trigonometric expressions. Here are some examples.

```
> sin(Pi/3); cos(2*Pi/3); tan(Pi/4);
> sin(I); cos(I); tan(I);
> sin(-x); cos(-x);
> sin(x+2*Pi);
> cos(x-Pi/2);
> cos(x+Pi/2);
> arcsin(cos(x));
> arccos(sin(x));
> sin( arcsin(x) );
> cos( arcsin(x) );
> sin( arctan(x) );
> sec( arccsc(x) );
> radsimp( % );
>
```

The **combine/trig** command does the following trigonometric transformations.

$$\sin(x)\sin(y) \quad => \quad \frac{\cos(x-y)}{2} - \frac{\cos(x+y)}{2}$$

$$\cos(x)\cos(y) => \quad \frac{\cos(x-y)}{2} + \frac{\cos(x+y)}{2}$$

$$\sin(x)\cos(y) \;=\!=\!> \frac{\sin(x-y)}{2} + \frac{\sin(x+y)}{2}$$

For example.

```
> combine( sin(x)*sin(y) );
```

Notice the following two special cases of the above transformations.

$$\sin(x)^2 =\!=\!> \frac{1-\cos(2x)}{2}$$

$$\cos(x)^2 =\!=\!> \frac{1+\cos(2x)}{2}$$

```
> combine( sin(x)^2 );
```

The **combine/trig** command will use the transformations above to convert sums and products of powers of $\sin(x)$ and $\cos(x)$ into sums of terms of the form $\sin(n\,x)$ and $\cos(m\,x)$ for integers n and m.

```
> sin(x)^2*cos(x)^3 + cos(x)^2;
> combine( % );
> cos(x)^4;
> combine( % );
```

Let us look carefully at how **combine/trig** did the transformations that produced the last result. First, $\cos(x)^4$ can be considered $\cos(x)^2\,\cos(x)^2$. Here is how **combine/trig** transforms $\cos(x)^2$

```
> combine( cos(x)^2 );
```

Now take that result and multiply it with itself.

```
> % * %;
```

Have **expand** multiply this out *without* expanding the **cos(2*x)** term.

```
> expand( %, cos(2*x) );
```

Now we have another power of cos that needs to transformed by **combine**. The following command will transform the $\cos(2\,x)^2$ term.

```
> combine( % );
```

One important thing to notice about these steps is that while **combine** was in the process of transforming $\cos(x)^4$, **combine** reached a step where it needed to call itself with another term to transform, the $\cos(2\,x)^2$ term. This idea, of a transforming procedure needing to call itself, will come up repeatedly in later worksheets and it has a name, **recursion**.

```
>
```

Exercise: Show the details of the steps that **combine** goes through as it transforms the expression $\sin(x)^3$.

```
> sin(x)^3;
> combine( % );
>
```

Exercise: One of the most often used trig identities is of course $\cos(x)^2 + \sin(x)^2 = 1$. The

combine command can do this transformation, even though it does not really "know" this identity. Show the details of how the **combine** command uses its trigonometric transformations to transform $\cos(x)^2 + \sin(x)^2$ into 1.

```
> cos(x)^2+sin(x)^2;
> combine( % );
>
```

Exercise: Explain how **combine** does the following two transformations.

```
> cos(3*x)^4;
> combine( % );
> sin(exp(x))^3*cos(cos(x));
> combine( % );
>
```

The **expand** command takes expressions of the form $\cos(m\,x + n\,y)$ or $\sin(m\,x + n\,y)$, with m and n integers, and "expands" them into "multivariate polynomials" in $\cos(x)$, $\sin(x)$, $\cos(y)$, and $\sin(y)$ with integer coefficients. Here is an example.

```
> cos(2*x-3*y);
> expand( % );
```

In addition, **expand** takes expressions of the form $\tan(m\,x + n\,y)$, with m and n integers, and converts them into "multivariate rational functions" in $\tan(x)$ and $\tan(y)$ with integer coefficients. Here is an example.

```
> tan(2*x-3*y);
> expand( % );
> normal( % );
```

The **expand** command can transform sums of the above kinds of expressions.

```
> cos(-3*x) + sin(2*x) - tan(2*x);
> expand( % );
```

And **expand** can also transform products of the above kinds of expressions.

```
> cos(-3*x)*sin(2*x)^2;
> expand( % );
>
```

The **expand** command does these kinds of transformations by repeatedly using the following transformations.

$$\cos(2\,x) = => 2\,\cos(x)^2 - 1$$

$$\sin(2\,x) = => 2\,\sin(x)\,\cos(x)$$

$$\tan(2\,x) = => \frac{2\,\tan(x)}{1 - \tan(x)^2}$$

$$\cos(x + y) = => \cos(x)\,\cos(y) - \sin(x)\,\sin(y)$$

$$\cos(x - y) = => \cos(x)\,\cos(y) + \sin(x)\,\sin(y)$$

$$\sin(x+y) = => \sin(x)\cos(y) + \cos(x)\sin(y)$$
$$\sin(x-y) = => \sin(x)\cos(y) - \cos(x)\sin(y)$$
$$\tan(x+y) = => \frac{\tan(x)+\tan(y)}{1-\tan(x)\tan(y)}$$
$$\tan(x-y) = => \frac{\tan(x)-\tan(y)}{1+\tan(x)\tan(y)}$$

For example, let us see what steps are needed for **expand** to transform $\cos(3x)$.

```
> expand( cos(3*x) );
```

To arrive at this result, first consider $\cos(3x)$ as $\cos(2x+x)$ and apply the fourth transformation above to this expression. We will do this with Maple, but we need to use a trick, in order to avoid an automatic simplification.

```
> expand( cos(2*x+u) );
> subs( u=x, % );
```

Now replace the expression $\sin(x)^2$ with $1-\cos(x)^2$.

```
> subs( sin(x)^2=1-cos(x)^2, % );
```

Now multiply this out, and we get our final result.

```
> expand( % );
```

Using a procedure similar to this, we can see how **expand** can transform any expression of the form $\cos(mx)$ for any integer m (and $\sin(mx)$ also).

Exercise: Explain the steps that **expand** uses to transform $\sin(x+2y)$.

```
> sin(x+2*y);
> expand( % );
>
```

Notice that the **combine** command produces expressions exactly of the form that **expand** can transform. And the **expand** command produces expression exactly of the form that **combine** can transform. It might seem that each of these commands will undo the other's transformation. For many trig expressions that is the case, but not for all trig expressions. Here is an example where **expand** does not undo what **combine** did.

```
> cos(x)^2*sin(x)^2;
> combine( % );
> expand( % );
```

Here are two examples where **combine** does not undo what **expand** did.

```
> cos(4*x)*sin(3*x)+sin(x);
> expand( % );
> combine( % );
> cos(4*x)*sin(2*x)-sin(2*x);
> expand( % );
> combine( % );
```

```
> 
```

According to the online documentation, the **simplify/trig** function does only one, simple, transformation.

$$\sin(x)^2 \;=\!>\; 1-\cos(x)^2$$

But the following example shows that what **simplify/trig** does is more complicated than just this one transformation. Consider this simplification.

```
> cos(2*x)+sin(x)^2;
> simplify( % );
```

The online documentation implies that the result should have been $\cos(2x)+1-\cos(x)^2$. It seems that **simplify** must have done something with the $\cos(2x)$ term. Let us try **simplify** on this term by itself.

```
> simplify( cos(2*x) );
```

The command did not do anything. Let us try **expand** on the $\cos(2x)$ term.

```
> expand( cos(2*x) );
```

If we add this result to $1-\cos(x)^2$ (i.e., the result of **simplify** applied to $\sin(x)^2$), then we get the desired result, $\cos(x)^2$. So **simplify** can expand $\cos(2x)$, but it will only do so within a certain context. We can conclude that the online documentation is not really describing all of what **simplify** will do to trigonometric expressions. And I do not know what exactly **simplify** will and will not do with trig expressions.

```
> 
```

Exercise: Explain how each of the following three commands does its transformation.

```
>  combine( sin(3*x)^2 );
>   expand( sin(3*x)^2 );
> simplify( sin(3*x)^2 );
> 
```

Exercise: We saw earlier that **combine** knows the identity $\cos(x)^2+\sin(x)^2=1$. The **simplify** command knows it also. Describe exactly how **simplify** does the following transformation.

```
> sin(x)^2+cos(x)^2;
> simplify( % );
```

Notice that this gives us an example of a transformation that can be done by two different commands but each command does the transformation in a completely different way.

```
> 
```

There are several variations on the basic identity $\cos(x)^2+\sin(x)^2=1$, such as

$$1+\tan(x)^2=\sec(x)^2$$

$$\sec(x)^2-\tan(x)^2=1$$

$$\csc(x)^2 - \cot(x)^2 = 1.$$

Interestingly, the **combine** command cannot do any of these three transformations, but **simplify** can do the last two (and I have no idea how).

```
> sec(x)^2-tan(x)^2;
> simplify( % );
> csc(x)^2-cot(x)^2;
> simplify( % );
```

Notice that **simplify** cannot transform $1 + \tan(x)^2$ into $\sec(x)^2$.

```
> 1+tan(x)^2;
> simplify( % );
```

But with a little help from the **convert/sincos** command, **simplify** can almost complete the transformation.

```
> convert( %, sincos );
> simplify( % );
> convert( %, sec );
>
```

Besides the two transformations mentioned just above, there are some other trigonometric transformations that **simplify** will do that are not mentioned in the online documentation. For example, **simplify** will simplify the composition of a trig function with its inverse (given the appropriate assumption on the variable).

```
> arcsin( sin(x) );
> simplify( %, assume=RealRange(-Pi/2,Pi/2) );
```

The next example shows an automatic simplification followed by the **simplify** command.

```
> arcsin( cos(x) );
> simplify( %, assume=RealRange(0,Pi) );
>
```

The next example shows that **simplify** can cancel sin's and cos's from within trig functions like tan and csc.

```
> tan(x)*cos(x) + sin(x)*csc(x);
> simplify( % );
>
```

Exercise: Find a sequence of Maple commands that will derive the transformation

$$\cot(x) + \cot(y) \;\; => \;\; \frac{\sin(x+y)}{\sin(x)\,\sin(y)}.$$

```
> cot(x)+cot(y);
>
```

Maple can do a number of conversions between different kinds of trigonometric forms for expressions. The following help pages do a good job of describing these conversions.

```
> help(`convert/trig`);      # from exponentials to trig
> help(`convert/sincos`);    # from trig to sin,cos,sinh,cosh
> help(`convert/tan`);       # from trig to tan
> help(`convert/expsincos`); # from trig to exp,sin,cos
> help(`convert/exp`);       # from trig to exponentials
> help(`convert/ln`);        # from arctrig to logarithms
> help(`convert/expln`);     # from elementary functions to
  exp,ln
>
```

Here are some examples of the **convert** command.

```
> sin(x)/cos(x);
> convert( %, tan );
> 1/cos(x);
> convert( %, tan );
> sec(x)^2-tan(x)^2;
> convert( %, sincos );
> convert( exp(x), trig );
> convert( exp(I*x), trig );
> convert( cos(x), exp );
> combine( % );
>


>
```

8.13. Logarithmic expressions

The two most basic identities for the logarithm function are

$$\ln(x\,y) = \ln(x) + \ln(y)$$

$$\ln(x^y) = y\,\ln(x).$$

The Maple commands associated with these two identities are **simplify/ln**, **combine/ln**, and **expand**.

Here are the Maple commands associated with the first identity (without any assumptions about the variables).

```
> simplify( ln(x*y), symbolic );
> combine( ln(x)+ln(y), ln, symbolic );
```

Here are the Maple commands associated with the second identity (again, without any assumptions about the variables).

```
> simplify( ln(x^y), symbolic );
> combine( y*ln(x), ln, anything, symbolic );
>
```

Maple can also do the transformation $\ln\left(\frac{x}{y}\right) = \ln(x) - \ln(y)$.

```
> simplify( ln(x/y), symbolic );
> combine( ln(x)-ln(y), ln, symbolic );
>
```

A funny quirk of the **simplify** command is that **simplify/power** does much of what **simplify/ln** does. This is only important to know in the case where you want to simplify a subexpression of a larger expression without simplifying some other parts of the expression. Consider the following commands.

```
> f := sin(theta)^2 + cos(theta)^2 + (x^a)^b + ln(x/y);
> simplify( f, symbolic );
```

The **simplify** command simplified both the trig, power, and log parts of the expression. If we only want to simplify the logarithmic part, then we need the **ln** option.

```
> simplify( f, ln, symbolic );
```

But if we want to simplify only the power part of the expression, and we use the **power** keyword, then we (somewhat unexpectedly) get both the power and log parts simplified.

```
> simplify( f, power, symbolic );
>
```

If we wish to combine some logarithmic terms at the same time that we combine some other kinds of terms, then we need to explicitly list the types of terms that we want **combine** to work on, as in the next two examples.

```
> 2*cos(a)*sin(a)+ln(x)+ln(y);
> combine( %, {ln, trig}, symbolic );
> exp(x)*exp(y)+y*ln(x);
> combine( %, {ln, exp}, anything, symbolic );
```

Try deleting the **trig**, **exp**, or **ln** options from the last two **combine** commands to see what happens.

```
>
```

The two logarithmic identities above are not true for all values of the variables. For example, in the first identity, let both x and y be -1.

```
> ln(x*y) = ln(x)+ln(y);
> subs( x=-1, y=-1, % );
> simplify( % );
```

In the second identity, let x be -1 and let y be 2.

```
> ln(x^y) = y*ln(x);
> subs( x=-1, y=2, % );
> simplify( % );
>
```

Exercise: Find other values of *x* and *y* that make the two logarithmic identities false.

```
> 
```

Since the identities are not always true, the **simplify** and **combine** commands need the keyword **symbolic** when we do not make any assumptions about the variables. An example of an assumption that will make the identities true is that both *x* and *y* are positive real numbers.

```
> ln(x*y);
> simplify( % ) assuming positive;
> ln(x^y);
> simplify( % ) assuming positive;
```

The **simplify** command does not do the transformations in the other direction, and the **combine** command does not always seem to work well with the **assuming** command.

```
> ln(x)+ln(y);
> combine( % ) assuming positive;
> y*ln(x);
> combine( %, ln, anything ) assuming positive; # doesn't work
```

Let us try the last command using **assume** instead of **assuming**.

```
> assume(x, positive);
> assume(y, positive);
> y*ln(x);
> combine( %, ln, anything );
```

With these assumptions in place, we can also use **expand** for the transformations in the other direction (i.e., from left hand side to right hand side in the identities).

```
> expand( ln(x*y) );
> expand( ln(x^y) );
```

Its possible to weaken our assumptions. Let us remove the assumption from *y*.

```
> y := 'y';
```

Now try **expand**. and **combine** on the first identity.

```
> expand( ln(x*y) );
> combine( ln(x)+ln(y) );
```

Try **expand** and **combine** on the second identity with these assumptions.

```
> expand( ln(x^y) );
> combine( y*ln(x), ln, anything );
```

expand did not work but **combine** did. Put back an assumption on *y*. Let us assume that *y* is real.

```
> assume(y, real);
```

Now try **expand** again on the second identity.

```
> expand( ln(x^y) );
> about(x,y);
> x,y := 'x','y';
> 
```

Now we turn to some identities involving both the exponential and logarithm functions.

To give you a sense of how working with Maple can be a bit confusing for a calculus student, consider the following two formulas from first year calculus,

$$\ln(\mathbf{e}^{x}) = x \text{ for all real numbers } x, \text{ and}$$

$$\mathbf{e}^{\ln(x)} = x \text{ for all positive real numbers } x.$$

Since the first equation is true for all real numbers but the second equation has a restriction to positive real numbers, one might expect Maple to make it easier to simplify **ln(exp(x))** than to simplify **exp(ln(x))**. But just the opposite is true. Maple will not, by default, simplify **ln(exp(x))**.

```
> ln(exp(x));
> simplify( % );
```

But Maple will, by default, automatically simplify **exp(ln(x))**.

```
> exp(ln(x));
```

The reason is that Maple assumes that all variables represent complex numbers. And for (nonzero) complex numbers, $\mathbf{e}^{\ln(x)} = x$ is always true. But $\ln(\mathbf{e}^{x}) = x$ need not be true. Here is an example.

```
> ln(exp(2*Pi*I));
```

We can get Maple to make the transformation $\ln(\mathbf{e}^{x}) = x$ by telling the **simplify** command to work symbolically, meaning that **simplify** does not worry about the validity of the transformation it makes.

```
> simplify( ln(exp(x)), symbolic );
```

Or we can tell Maple to assume that **x** is a real number.

```
> simplify( ln(exp(x)) )assuming real;
```

In fact, with the assumption that **x** is real, we do not even need the **simplify** command, Maple will automatically make the simplification.

```
> ln(exp(x)) assuming real;
>
```

An important variation on the last identity above is $\mathbf{e}^{(y\ln(x))} = x^{y}$. In the case where y is a number, Maple will automatically simplify $\mathbf{e}^{(y\ln(x))}$.

```
> exp(2/3*ln(x));
```

In the case where y is a variable, Maple will not do the simplification automatically.

```
> exp(y*ln(x));
```

But both the **simplify** and the **expand** commands will make the transformation.

```
> simplify( % );
> expand( %% );
```

The **combine** command will also work, but it is very awkward, since it needs three options.

```
> combine( %%%,  ln, anything, symbolic );
>
```

I do not know of a way to convert x^{y} to $\mathbf{e}^{(y\ln(x))}$. The following would seem like a reasonable guess,

but it does not work.

```
> convert( x^y, expln );
>
```

Exercise: Notice that **simplify** will simplify $\mathbf{e}^{(y \ln(\mathbf{e}^x))}$ to $(\mathbf{e}^x)^y$.

```
> exp(y*ln(exp(x)));
> simplify( % );
```

For another example, let x be I and let y be $I\,\pi$.

```
> exp(I*Pi*ln(exp(I)));
> simplify( % );
```

Now reverse the roles of x and y. Why does $\mathbf{e}^{(I \ln(\mathbf{e}^{(I\,\pi)}))}$ not simplify to $(\mathbf{e}^{(I\,\pi)})^I$?

```
> exp(I*ln(exp(I*Pi)));
> exp(I*Pi)^I;
```

Are the last two expressions equal?

```
>
```

Another important logarithmic identity is the definition of logarithm functions to other bases,

$$\log_b(x) = \frac{\ln(x)}{\ln(b)}$$

Along with this definition are the two identities

$$\log_b(b^x) = x$$

$$b^{\log_b(x)} = x.$$

The Maple notation for the logarithm function to base b, $\log_b$, is **log[b]**. Notice that Maple will automatically simplify the expression $\log_b(x)$ into $\frac{\ln(x)}{\ln(b)}$.

```
> log[b](x);
```

The **simplify** command will simplify $b^{\log_b(x)}$ to x.

```
> b^log[b](x);
> simplify( % );
```

The **simplify** command will simplify $\log_b(b^x)$ to x if we use the keyword **symbolic**.

```
> log[b](b^x);
> simplify( %, symbolic );
```

The reason that **simplify** needed the keyword **symbolic** should be clear from our discussion of the identities $\ln(x^y) = y \ln(x)$ and $\ln(\mathbf{e}^x) = x$.

```
>
```

Exercise: Part (a): Show that $\log_b(b^x) = x$ need not be true.

```
>
```

Part (b): Simplify the expression `log[b](b^x)` without using the keyword `symbolic`. Instead, use appropriate assumptions.

```
> 
```

```
> 
```

8.14. Online information on manipulating expressions

Here are the help pages that give Maple's definition of a polynomial expression.

```
> ?polynom
> ?type,polynom
```

Here are help pages for some important commands for working with polynomials

```
> ?sort
> ?collect
> ?coeffs
> ?coeff
> ?degree
> ?student,completesquare
> ?quo
> ?rem
> ?randpoly
> ?taylor
> ?convert,horner
```

Here are the help pages that gives Maple's definition of a rational expression.

```
> ?ratpoly
> ?type,ratpoly
```

Here are help pages for some important commands for working with rational expressions.

```
> ?numer
> ?denom
> ?normal
> ?convert,parfrac
> ?convert,fullparfrac
> ?convert,confrac
> ?numapprox,laurent
```

The next two help pages give kind of a definition of what a "radical expression" is.

```
> ?type,radfun
> ?type,radalgfun
```

Here are help pages for some important commands for working with radical expressions.

```
> ?root
> ?radsimp
> ?radnormal
```

```
```

```
> ?rationalize
> ?surd
```

Here is the help page for the **factor** command.

```
> ?factor
```

Here are the help page for the **Split** and **convert/radical** commands.

```
> ?PolynomialTools,Split
> ?convert,radical
```

Here are the help pages for the **evala** and **AFactor** commands.

```
> ?evala
> ?AFactor
```

Here are the help pages for the **combine** command and its most useful helper functions.

```
> ?combine
> ?combine,power
> ?combine,radical
> ?combine,exp
> ?combine,ln
> ?combine,trig
> ?combine,arctan
```

Here is the help page for the **expand** command.

```
> ?expand
```

Here are the help pages for the **simplify** command and its most useful helper functions.

```
> ?simplify
> ?simplify,power
> ?simplify,radical
> ?simplify,sqrt
> ?simplify,trig
> ?simplify,ln
```

Here is the help page for the **convert** command. This help page is a hyperlinked list of all of the helper functions for **convert**.

```
> ?convert
```

```
>
```