TRIGD: Trigonometrical Functions with Degree Arguments

Alan Barnes
School of Engineering & Applied Science
Aston University, Aston Triangle,
Birmingham B4 7ET
GREAT BRITAIN (now retired)
Email: Alan.Barnes45678@gmail.com

1 Introduction

This module provides facilities for the numerical evaluation and algebraic simplification of expressions involving trigonometrical functions with arguments given in degrees rather than in radians. The degree-valued inverse functions are also provided.

Any user at all familiar with the normal trig functions in REDUCE should have no trouble in using the facilities of this module. The names of the degree-based functions are those of the normal trig functions with the letter D appended, for example `SIND`, `COSD` and `TAND` denote the sine, cosine and tangent repectively and their corresponding inverse functions are `ASIND`, `ACOSD` and `ATAND`. The secant, cosecant and cotangent functions and their inverses are also supported and, indeed, are treated more as *first class* objects than their corresponding radian-based functions which are often converted to expressions involving sine and cosine by some of the standard REDUCE simplifications rules.

Below I give a brief description of the facilities available together with a few examples of their use. More examples and the output that they should produce may be found in the test files `trigd-num.tst` and `trigd-simp.tst` and their corresponding log files with extension `.rlg` which may be found in the directory `packages/misc` of the REDUCE distribution along with the source code of the module.

These degree-based functions are probably best regarded as functions defined for *real* values only, but complex arguments are supported for completeness. The numerical evaluation routines are fairly comprehensive for both real and complex arguments. However, few simplifications occur for trigd functions with complex arguments.

The range of the principal values returned by the inverse functions is consistent with those of the corresponding radian-valued functions. More precisely, for `ASIND`, `ATAND` and `ACSCD` the (closure of the) range is [-90, 90] whilst for `ACOSD`, `ACOTD` and `ASECD` the (closure of the) range is [0, 180]. In addition the operator `ATAN2D` is the degree valued version of the two argument inverse tangent function which returns an angle in the interval (-180, 180] in the correct quadrant depending on the signs of its two arguments. For $X > 0$, `ATAN2D(Y, X)` returns the same numerical value as `ATAND(Y/X)`. If $X = 0$ then ± 90 is returned depending on the sign of Y.

It might be thought that the facilities provided in this module couldbe easily provided by defining suitable rule lists to convert between the radian and degree-based versions of the trig functions. For example:

```
1: operator sind, asind$
2:  d2r_list := {sind(~x) =>
        sin(x*pi/180), asind(~x) => 180*asin(x)/pi}$
3:  r2d_list := {sin(~x) =>
      sind(180x/pi), asin(~x) => pi*sind(x)/180}$
4:  sind(x+360) where d2r_list$
5:  ws where r2d_list;
         sind(x)
6:  sind(360) where d2r_list;
      0
```

However, this approach *seldom works* — try it! The result produced by step 4 defeats the current rule[1] used to simplify expressions of the form $\sin(x + 2\pi)$ although it does manage step 6. The rule list approach is more reliable if differentiation, integration or numerical evaluation of expressions involving `SIND` etc. is required. However it is not particularly convenient even if the rules and operator declarations are stored in a file so that they may be loaded at will.

[1]These rules may be improved in the next version of REDUCE.

This module aims to overcome these deficiences by providing the degree-based trig functions as *first class* objects of the system just like their radian-based cousins. The aim is to provide facilities for numerical evaluation, symbolic simplification and differentiation totally analgous to those for the the basic trig functions and their inverses. It is hoped that the module will be of value to students and teachers at secondary school level as well as being sufficiently powerful and flexible to be of genuine utility in fields where angles measured in degrees (and arc minutes and seconds) are in common usage. For more advanced situations (involving integration, complex arguments and values etc.), users are urged to use the standard trig functions already provided by the system.

2 Simplification

As in other parts of REDUCE, basic simplification of expressions involving the `trigd` functions takes place automatically (bracketted terms are multiplied out, like terms are gathered together, zero terms removed from sums and so on). The system *knows* and automatically applies the basic properties of the functions to simplify the input. For example `SIND(0)` is replaced by 0 and `SIND(-X)` by `SIND(X)`. If the switch `ROUNDED` is `OFF` all arithmetic is exact and transcendental functions such as `SIND` are not evaluated numerically even if their arguments are purely numerical.

The built-in simplification rules are totally analogous to those of the standard trig functions namely:

- Replacement of a function application by its value if a simple analytical value is known. For example `cosd(60) => 1/2` and `acscd(1) => 90`. Currently the only argument values where simplification takes place correspond to angles that are integral multiples of 15^o.
- Use of the odd and even properties of the trig. functions so that for example `sind(-x) => -sind(x)`, `cosd(-x) => cosd(x)` and `acosd(-x) => 180 - acosd(x)`.
- Argument shifts by integral multiples of 180^o so that any residual numerical argument lies in the range $-90^o \ldots 90^o$.
  Thus `sind(x+540) => -sind(x)`, `cosd(x+350) => cosd(x-10)`.
- Removal of argument shifts of $\pm 90^o$ so that for example
  `sind(x-90) => -cosd(x)` and `cotd(x+90) => -tand(x)`.

- Replacement of `tand(x)` by `sind(x)/cosd(x)`, `secd(x)` by `1/cosd(x)` and the like, but *only when the final result is simpler than the original.*
- Basic properties relating a function and it inverse so that for example `sind(asind(x)) => x`.
- A few basic rules for `ATAN2D` when the signs of its arguments can be determined. For example `atan2d(Y, 0)` is replaced by ± 90 depending on the sign of Y.

Extra rules can be added by the user for example addition formulae, double angle rules and tangent half-angle formulae as and when required as described in section 11 of the main REDUCE manual.

Rules are provided for the symbolic differentiation of all the trig functions and their inverses. These rules are sufficient fot the power series of the trig functions and their inverses to be found using either the `TPS` or `TAYLOR` packages in the standard way.

3 Numerical Evaluation

When the switch `ROUNDED` is `ON` and the arguments of the operators evaluate to numbers, then the floating point value of the expression is calculated to the currently specified `PRECISION` in the normal way. The *bigfloat* capabilities are the same as for the standard trig functions.

If these functions are supplied with complex numerical arguments, numerical evaluation will *NOT* be performed when the switch `ROUNDED` is `ON`, but the switch `COMPLEX` is `OFF` — the input expression will be returned basically unaltered. Similarly inputs such as `ASIND(2)` or `ASECD(0.5)` are not evaluated numerically. The values of these expressions are, of course, complex.

If the switch `COMPLEX` is also `ON` , numerical evaluation is performed. For example:

```
1: load_package trigd$
2:  on rounded;
3:  asecd(2);
         60.0
4:  asecd(0.5);
       asecd(0.5)
5: on complex;
```

```
    *** Domain mode rounded changed to complex-rounded
6:  asecd(0.5);
     75.4561292902*i
```

The function `ATAN2D` (like `ATAN2`) is only defined if *BOTH* its arguments are real. If they are also numerical, it will be evaluated whenever `ROUNDED` is `ON`. Attempting to evaluate it with complex numerical arguments will cause either the unaltered expression to be returned or an error to be raised when the switch `COMPLEX` is `OFF` or `ON` respectively.

3.1 Conversion between Degrees and Radians

There are a number of utility routines for converting an angle in radians to degrees and vice-versa. `RAD2DEG` converts the radian value to an angle in degrees expressed as a single floating point value (according to the currently specified `PRECISION`). The value to be converted may be an integer, a rational or a floating value or indeed any expression that simplies to a rounded value. In particular numerical constants such as π may be used in the input expression.

`RAD2DMS` converts the radian value to an angle expressed in degrees, minutes and seconds returned as a three element list. The degree and minute values are integers the latter in the range $0 \ldots 59$ inclusive and the seconds value is a floating point value in the interval [0, 60.0). There are also operators `DEG2RAD` and `DEG2DMS` whose purpose should be obvious.

The purpose of the operators `DMS2RAD` and `DMS2DEG` should also be obvious. The degree, minute and second value to be converted is passed to the conversion function as a three element list. There is considerable flexibilty allowed in format of the list supplied as parameter – all three values may be integers, rational numbers or rounded values or any combination of these; the minute and second values need not lie between zero and sixty. The list supplied is simplified with the appropriate *carrys and borrows* performed (in effect at least) between the three values. For example

```
{60.5, 9.2, 11.234}      =>   {60, 39, 23.234}
{45, 0, -1}              =>   {44, 59, 59}
```

These operators are not actually part of the `TRIGD` module but of the REDUCE core system. However, they are not currently documented in the main manual. Currently they are *purely numeric* operators; when `ROUNDED` is `OFF` they basically return the input expression (perhaps with their parameter simplified somewhat).

Note the sine of an angle specified in degrees, minutes and seconds *cannot* be calculated by calling `SIND` directly with a dms list (i.e. as a list of length 3). Instead one must first convert the dms values to degrees using a call to `DMS2DEG` and then call `SIND` on the result. Applied directly to a list (of any length) any `TRIGD` function wil be applied to each member of the list separately just like most other REDUCE operators. Here is an example illustrating tese points:

```
1: load_package trigd$
2: on rounded;
3: sind dms2deg {60, 45, 30};
0.872567064923
4: sind {60,45, 30};
{0.866025403784,0.707106781187,0.5}
5: off rounded;
6: sind{60, 45, 30};
  sqrt(3)   sqrt(2)   1
{---------,---------,---}
     2         2      2
```

Of course the results will be formatted much more attractively on a terminal supporting nice graphics.

3.2 The operators `ARGD` and `ARG`

Although not directly related to the trig functions, the module `TRIGD` also provides an operator `ARGD`; when the switches `ROUNDED` and `COMPLEX` are both `ON`, it will return the argument in degrees of the complex number supplied as its parameter — supplying zero as the parameter causes an error to be raised. If only `ROUNDED` is `ON`, `ARGD` will return the argument of the real numerical value supplied as its parameter — this will be 0 or 180 when the value is positive or negative respectively.

The operator `ARG` is similar to `ARGD`, but returns the argument expressed in radians. There is also an operator `NORM` which returns the modulus (or absolute value or norm) of a complex number. `ARG` and `NORM` are actually part of the REDUCE core system, but are not currently documented in the main manual. Currently they are *purely numeric* operators; when `ROUNDED` is `OFF` they basically return the input expression (perhaps with their parameter simplified).

Example

```
1: load_package trigd$
2: on rounded;
3: {argd(-5), argd(1+i)};
{180.0,  argd(i + 1)}
4: on complex;
*** Domain mode rounded changed to complex-rounded
5: {argd(1+i), argd(-1-i)};
{45.0,  -135.0}
6: {arg(3+4i), norm(3+4i)};
{0.927295218002,  5.0}
```

4 Bugs, Restrictions and Planned Extensions

The behaviour of the numerical evaluation routines for inverse trig functions with complex arguments at branch points could be improved; these values are *undefined* and attempting to evaluate such a function at one of its branch points *ought* to raise an error, however sometimes the input expression will be returned unaltered. It is hoped to improve this behaviour in due course.

Currently there are no facilities analogous to those provided in the module `TRIGSIMP` for the standard trig. functions. There users have a wide range of standard simplification formulae available for use and can control which are to be used depending on the requirements of their particular application: whether to eliminate `sin` in favour of `cos` or vice-versa or to get rid of both in favour of `tan` of half-angles; or whether to use the trigonometrical addition formulae in order to transform trig functions whose arguments are sums into a form where the arguments are single terms or whether to perform the inverse transformations. It is hoped to make the `TRIGSIMP` faciliites available for use with the `TRIGD` functions in the near future.

Integration is not directly supported although the approach using rule-lists to convert the `TRIGD` functions to standard trig ones should work well. Introducing direct support for integration will not therefore be a priority.

For the standard sine function there is a rule for imaginary arguments namely: `sin(I*X) => I*sinh(X)`. The corresponding rule for the degree version is `sind(I*X) => I*sinh(X*PI/180)`. However, currently such rules are *NOT* implemented by the system. They may be implemented in future, but it is not a high priority as it is felt that the radian-based trig functions are best suited for such symbolic calculations.

There are *NO D* versions of the hyperbolic functions — that would be a *step too far*! And should the new functions be called `sinhd` and so on? Or perhaps `sindh`[2] etc?

[2]One is perhaps reminded here of the (in)famous bilingual pun: *peccavi* attributed to Charles James Napier — apparently no relation to his logarithmic namesake – see Wikipedia for details!