Package pw.mac

About pw.mac

Introduction:
This package is for working with piecewise functions symbolically and is written and maintained by Richard Hennessy. This package requires Maxima 5.20.1 or later (at least that was true the last time I checked). The current version of Maxima is 5.25.1 and this package works on this version. Older versions may not work as it is my goal to make pw.mac work with the current release of Maxima only. I do not make an effort to support older versions of Maxima. To use pw.mac type load(pw)$ after placing the file in the share/contrib directory. For your information here is a list of user functions defined in this package in no particular order.

pw(), piecewise(), pwint(), iif(), pwsimp(), pwdefint(), deltaint(), defdeltaint(), maxmin2abs(), max2abs(), if2sum(), iif2sum(), sum2iif(), min2abs(), between2if(), between2iif(), if2signum(), iif2signum(), signum2if(), signum2iif(), abs2iif(), abs2if(), if2ifthen(), iif2ifthen(), abs2signum(), signum2abs(), abs2unitstep(), unitstep2signum(), signum2unitstep(), unitpulse2signum(), unitspike2signum(), signum2hstep(), hstep2signum(), charfun2signum(), simpunitstep(), unit_pulse(), unit_spike(), delta(), diff_delta(), between(), simpsignum(), negunitstep(), periodic(), intperiodic(), simpsignumargs(), pushout(), pwlimit(), pullinif(), pulliniif(), bffind_root(), simp_given(), simp_iif() and gen_deltas() (formerly use_delta()).

The new functions are unitpulse2between(), between2unitpulse(), unitpulse2unitstep(), ifthen2iif(), ifthen2if(), max2iif(), min2iif(), maxmin2iif(). Using ifthen2iif() allows programming constructs to be handled algebraically, sometimes.

convertall2unitstep() has been removed. Functions gen_pwdeltas(), pwdelta() and diff_pwdelta() have been changed to gen_deltas(), delta() and diff_delta().

Pwint() can now express integrals of unit_pulse() in terms of unit_pulse().

There are some global flags that affect output as well. use_between (default is 'false, can be true, applies to the pw() function), coeff_action (default is 'none, can be ‘factor, ‘ratsimp, ‘expand and others, applies to the pushout() function), factor_signum (default is 'true, can be ‘all, ‘false, applies to pwint() and pwdefint()), global (default is 'true, can be false, applies to pwsimp()).

Function pw(), piecewise()
Syntax: Function pw(L, x, [options]) or piecewise(L, x, [options]), assumes L is a list and x is the independent variable (x can be an expression like x+y which can be useful).
The list L should have one of the following forms:
[number1, expr1, number2, expr2, number3, expr3, numbern, exprn, numbern+1]
Or,
[[number1a, number1b, expr1], [number2a, number2b, expr2], ..., [numberna, numbernb, exprn]] with some exceptions detailed below.

The numbers define the domains of the corresponding expressions, outside these domains the function returns 0. The function pw(L, x,[options]) expresses a piecewise function in terms of the signum(), delta(), diff_delta() and unit_spike(), iif(), if then else, %if() or the between() function or an easy to understand matrix like form. If 'array is used in options then pw() displays the function in matrix format and just returns “done”. If 'connect is in options then the pieces of the function are connected at their end points by adding the necessary constant(s) to each piece. Options can also contain, '%if, 'iif, 'ifthen, 'signum, unit_pulse, ‘pulse or 'between. If other options are specified then they can be 'halfopen (default), 'open, 'lclosed, 'rclosed or ’closed which will cause the intervals defined to be either, open intervals (end points not included), left closed, right closed which are open on either the left or right and closed on the other side or closed where all the intervals are closed. The 'halfopen option causes the function to behave differently but only at the endpoints. If 'halfopen is used then at the endpoints the piecewise functions will take on the average of the left and right handed limits of the function at that point. If the intervals overlap then the sum of the values of the expressions in the overlapping region are used (with the possible exception noted before about halfopen). If there are gaps in the defined regions then the function returns zero in these regions, unless option 'connect is present in which case the result is not likely to be zero. If in the list for pw([number1, expr1, number2, expr2, number3],x) number2 is less than number1 or number3 is less than number2 then the output is zero. **IMPORTANT:** This is a change from versions of pw.mac prior to version 5.05. For earlier versions the results in this case can be unpredictable. If ‘array is not in the list of options then the output will be expressed in terms of signum(), delta(), diff_delta() and unit_spike(), iif(), %if(), or the between() functions which represent the piecewise function in a form suitable for manipulation by Maxima. Note: The 'ifthen option which produces the "if then else" form is not very suitable for further manipulation.

If in the list L you see the following form [. . .numbern, exprn, numbern+1. . .] where numbern and numbern+1 are equal then exprn*unit_spike() is inserted in the output. If in the list you see the following form [. . .numbern, [m,exprn], numbern+1. . .] where numbern and numbern+1 are equal and m>=0 then exprn*diff_delta(m, x-numbern) is inserted in the output. If in the list you see the following form [. . .numbern, [exprn], numbern+1. . .] where numbern and numbern+1 are equal then exprn*delta(x-numbern) is inserted in the output. The following forms also may be used […[numbern, numbern, exprn]…] or […[numbern, numbern, [exprn]]…] or […[numbern, numbern, [m,expr2]]…]. This syntax is another way to put exprn*unit_spike(x-numbern), exprn*delta(x-numbern) or exprn*diff_delta(m,x-numbern) in the output. Examples follow.

```
(%i1) piecewise([minf,2,2,-5,inf],x,'array);

            [ If  x  in  (  minf  ,   2   ]  then   2  ]
            [                                          ]
            [ If  x  in  [   2    ,  inf  )  then  - 5 ]

(%o1)
                         Done
(%i2) piecewise([minf,2,2,-5,inf],x,'array,'open);

            [ If  x  in  (  minf  ,   2   )  then   2  ]
            [                                          ]
            [ If  x  in  (   2    ,  inf  )  then  - 5 ]
(%o2)                    Done

(%i3) L:[-10,-(x^2-25)/15,-5,-15/3*sin(3*x),0,x^2/6,5,-(8*x-65)/6, 10]$

(%i4) f(x):= ''(pw(L,x));
                      2                                                            2
               (25 - x )  (signum(x + 10) - signum(x + 5))   5 sin(3 x)  (signum(x + 5) - signum(x))   x  (signum(x) - signum(x -
5))
(%o4)      f(x)      :=      -----------------------------------------      -      -------------------------------------      +
------------------------------
                                   30                                        2                                    12
                                                                                                         (65 - 8 x)  (signum(x - 5) -
signum(x - 10))
                                                                                                                                    +
-------------------------------------------
                                                                                                                              12

(%i5) f(2);
                                   2
(%o5)                              -
                                   3
(%i6) f(-5);
                                5 sin(15)
(%o6)                           ---------
                                   2
(%i7) f(12);

(%o7)                              0
(%i8) f(8);
```

```
                                1
(%o8)                           -
                                6
(%i9) piecewise([minf,2,2,[x],2,-5,inf],x);
                                      5 (signum(x - 2) + 1)
(%o9)                               - --------------------- - signum(x - 2) + x delta(x - 2) + 1
                                                2
(%i10) piecewise([minf,2,2,x,2,-5,inf],x);
                                    5 (signum(x - 2) + 1)
(%o10)                            - --------------------- - signum(x - 2) + x unit_spike(x - 2) + 1
                                              2
(%i11) piecewise([minf,2,2,[0,x],2,-5,inf],x);
                                  5 (signum(x - 2) + 1)
(%o11)                          - --------------------- - signum(x - 2) + diff_delta(0, x - 2) x + 1
                                            2
(%i12) piecewise([minf,2,2,[2,x],2,-5,inf],x);
                                  5 (signum(x - 2) + 1)
(%o12)                          - --------------------- - signum(x - 2) + diff_delta(2, x - 2) x + 1
                                            2
(%i13)
```

Function pwsimp()
Syntax: Function pwsimp(e, x, [options]).

Pwsimp(e, x, options) assumes e is a piecewise expression and x is the independent variable. If options is 'list then the return value is the list L that would create the function if passed to pw(). If 'array is used in the options for pwsimp() then the return value is "done" and an easy to understand matrix like format of the function is displayed for the user. If neither is present then the result is expressed in terms of the iif(), %if(), signum(), between(), if then else, unit_spike(), delta() and diff_delta() functions. . If 'list is in options then the list returned will have the following syntax
[number1, expr1, number2, expr2, number3, expr3 . . . numbern, exprn, numbern+1],
If options contains 'halfopen, 'closed, 'lclosed, 'rclosed or 'open AND no 'array or 'list is specified then the result will as if the option was ommitted. These commands are no longer sent to pw(). IMPORTANT NOTE: If your piecewise expression is hand entered and not the result of a call to pw() then I highly recommend calling pwsimp() right away to put the expression into a form that this package can handle. Pwsimp() does not work well with symbolic expressions. Pwsimp() just produces the expression unchanged if it is unable to sort the list of numbers that define the domains of the corresponding expressions. You can try assume(a>b, b>c, c>d) then pwsimp(f(signum(a), signum(b), signum(c), x), x) may work but if there is an alternative sort pwsimp() will have no effect. There are many examples of problematic expressions (forms that cannot be integrated) with signum calls in exponents, or in denominators just to name a few that can be handled by pwsimp() if they use numerical values and not symbolic constants or if the symbolic constants can be sorted. It should be mentioned that the purpose of pwsimp() was to make piecewise functions easier to integrate. That has changed over time and now pwsimp() can be used for actual simplification or standardization of piecewise forms.

```
(%i1) pwsimp(prod((1+x*unit_step(x-i)),i,0,6),x,'array);


                              [ If  x  in  (  minf  ,   0   )  then                          1                          ]
                              [                                                                                         ]
                              [ If  x  in  [   0    ,   0   ]  then                          1                          ]
                              [                                                                                         ]
                              [ If  x  in  (   0    ,   1   )  then                        x + 1                        ]
                              [                                                                                         ]
                              [ If  x  in  [   1    ,   1   ]  then                          2                          ]
                              [                                                                                         ]
                              [                                                         2                               ]
                              [ If  x  in  (   1    ,   2   )  then                    x  + 2 x + 1                     ]
                              [                                                                                         ]
                              [ If  x  in  [   2    ,   2   ]  then                          9                          ]
                              [                                                                                         ]
                              [                                                      3      2                           ]
                              [ If  x  in  (   2    ,   3   )  then                 x  + 3 x  + 3 x + 1                 ]
                              [                                                                                         ]
                              [ If  x  in  [   3    ,   3   ]  then                         64                          ]
                              [                                                                                         ]
                              [                                                  4      3      2                        ]
                              [ If  x  in  (   3    ,   4   )  then             x  + 4 x  + 6 x  + 4 x + 1              ]
                              [                                                                                         ]
                              [ If  x  in  [   4    ,   4   ]  then                         625                         ]
                              [                                                                                         ]
                              [                                              5      4       3       2                   ]
                              [ If  x  in  (   4    ,   5   )  then         x  + 5 x  + 10 x  + 10 x  + 5 x + 1         ]
                              [                                                                                         ]
                              [ If  x  in  [   5    ,   5   ]  then                        7776                         ]
                              [                                                                                         ]
                              [                                          6      5       4       3       2               ]
                              [ If  x  in  (   5    ,   6   )  then     x  + 6 x  + 15 x  + 20 x  + 15 x  + 6 x + 1     ]
                              [                                                                                         ]
                              [ If  x  in  [   6    ,   6   ]  then                       117649                        ]
                              [                                                                                         ]
                              [                                      7      6       5       4       3       2           ]
                              [ If  x  in  (   6    ,  inf  )  then  x  + 7 x  + 21 x  + 35 x  + 35 x  + 21 x  + 7 x + 1 ]

                                                                      Done

(%i2) signum(x-7)/(a*x+signum(x-9)*a);
                                                                          signum(x - 7)
(%o2)                                                                 ---------------------
                                                                      a signum(x - 9) + a x
(%i3) pwsimp(%,x);
                                                                                2
                                            (9 x + 9) signum(x - 7) - 9 signum(x - 9) + (x  - 9 x - 1) unit_spike(x - 9) - 9
(%o3)                                       --------------------------------------------------------------------------------
                                                                              2
                                                                         9 a x  - 9 a
(%i4) pwsimp(%,x,array);


                                                       [                                        - 9 x - 9   ]
                                                       [ If  x  in  (  minf  ,   7   >  then  ------------ ]
                                                       [                                            2       ]
                                                       [                                       9 a x  - 9 a ]
                                                       [                                                    ]
```

```
                                                          [                                       9 x + 9   ]
                                                          [ If  x  in  <   7    ,   9   )  then  ----------- ]
                                                          [                                          2      ]
                                                          [                                     9 a x  - 9 a ]
                                                          [                                                 ]
                                                          [                                         1       ]
                                                          [ If  x  in  [   9    ,   9   ]  then     ---      ]
                                                          [                                        9 a      ]
                                                          [                                                 ]
                                                          [                                       9 x - 9   ]
                                                          [ If  x  in  (   9    ,  inf  )  then  ----------- ]
                                                          [                                          2      ]
                                                          [                                     9 a x  - 9 a ]

(%o4)                                                                            Done
(%i5) pwsimp(%o3-%o2,x);

                                                                                  0
```

Functions delta(), diff_delta()
Syntax: Function delta(e), diff_delta(n,e).

delta(e) and diff_delta(n, e) returns zero when e # 0 and undefined otherwise.
The delta functions are similar to the Dirac delta function and can be integrated to produce a discontinuity in the result (indefinite). diff_delta(n, x) is the nth derivative of delta(x) and has some usefulness. It behaves in a similar manner to the derivative's of the Dirac delta function. Two examples follow.

```
(%i1) pwint(delta(x-a),x);
                                                          signum(x - a) + 1
(%o1)                                                     -----------------
                                                                  2
(%i2) pwint(x^3*diff_delta(1,x-a),x);
                                                            2
                                                         3 a  (signum(x - a) + 1)
(%o2)                                                  - ------------------------
                                                                   2
```

Function pwint()
Syntax pwint(expr, x) pwint(expr,x,a,b)

This function can do indefinite integrals of many piecewise functions. This function works well with symbolic expressions and can handle delta(), diff_delta(), between(), iif() and unit_pulse(). If the limits of integration are specified then this function computes the anti-derivative of expr at b and at a and subtracts the two. This means that the answer is not always right. Consider pwint(1/(x+a),x,-2*a,2*a), in this case the answer would not yield the definite integral because of the infinity at x = -a.

Function between()
Syntax between(x, a, b, interval_type)

This function returns 1 if x is between a and b and zero otherwise. If a >= b then between() always returns 0. If x = a or x = b then this function by default returns 1/2 depending on interval_type. If interval_type = 'closed then between() returns 1 if x = a or x = b and a < b. If interval_type = 'lclosed then between() returns 1 if x = a and a < b. If interval_type = 'rclosed then between() returns 1 if x = b and a < b. If interval_type = 'open then between() returns 0 if x = a or x = b. If interval_type = 'halfopen or is not specified then between() returns 1/2 if x = a or x = b and a < b.

Function iif()
Syntax iif(<cond>, a, b)

This function returns a or b depending on the truth of <cond>. If <cond> is true iif() returns a else iif() returns b. <cond> can be any inequality or equal() or notequal(). And, or, not and if then are not permitted.

Function deltaint()
Syntax deltaint(expr, x), defdeltaint(expr,x,a,b)

This function can do integrals of piecewise functions. This function expects a piecewise expression given in terms of the signum(), between(), charfun2(), iif(), %if() or unit_step() functions and expresses the result in terms of the unit_step function. This function can handle delta(), diff_delta() and unit_pulse() too.

Function unit_spike()
Syntax unit_spike(e). The unit_spike function returns the value 1 if e = 0 and 0 if e # 0.,

Function unit_pulse()
Syntax unit_pulse(e). The unit_pulse function returns the value 1 if e > 0 and e < 1 and ½ if e = 0 or e = 1, 0 otherwise.

Conversion functions
Syntax maxmin2abs(e), max2abs(e), if2sum(e), iif2sum(e), sum2iif(e), min2abs(e),between2if(e), between2iif(e), if2signum(e), iif2signum(e), signum2if(e), signum2iif(e), abs2iif(e), abs2if(e), if2ifthen(e) and iif2ifthen(e), abs2signum(e), signum2abs(e), abs2unitstep(e), unitstep2signum(e), signum2unitstep(e), unitpulse2signum(e), ***unitpulse2between(), unitpulse2unitstep(), between2unitpulse(), ifthen2iif(), ifthen2if(), max2iif(), min2iif(), maxmin2iif(),*** unitspike2signum(e), convertall2signum(e), charfun2between(e) and charfun2signum(e).

This set of functions converts expressions containing max(), min(), unit_step(), unit_pulse(), unit_spike(), abs(), signum(), iif(), between() or %if() to another form. Max2abs and min2abs are based on Barton Willis' code in abs_integrate.mac and are used with his permission. Abs2signum() also is based partly on Barton Willis' abs_to_signum() function in abs_integrate.mac but is a little more capable. Examples follow.

```
(%i1) abs2signum(maxmin2abs(max(x,x^2)));
           2              2         2
         (x  - x) signum(x  - x) + x  + x
(%o1)    --------------------------------
                        2
(%i2) abs2signum(maxmin2abs(abs(x-a)^3*max(x,x^2)));
              3       3            2              2         2
       (x - a)  signum (x - a) ((x  - x) signum(x  - x) + x  + x)
(%o2) -----------------------------------------------------------
                                 2
(%i3) convertall2signum(unit_pulse((x-b)/2)*abs(x-a)*max(x-a,(x-b)^2));
                                                                          2                  2                2               2
       (x - a) signum(x - a) (signum(x - b) - signum(x - b - 2)) (((x - b)  - x + a) signum(x  - 2 b x - x + b  + a) + (x - b)  +
x - a)
(%o3)
---------------------------------------------------------------------------------------------------------------------------
                                                                        4
(%i4) simpspikes(signum2abs(iif2sum(iif(x>=0,x,-x))));
(%o5)                                                                             abs(x)
```

Functions periodic() and intperiodic()
Syntax periodic(expr, x, a, b), intperiodic(expr, x, a, b),

Periodic takes a function defined on an interval [a,b] and extends it by periodic extension to all x so that f(x) = periodic(f(x), x, a, b) when x>=a and x<=b and outside this range the function repeats its behavior with a period of b-a. The following identity holds.

periodic(f(x), x, a, b) = f((b-a)*((x-a)/(b-a)-floor((x-a)/(b-a)))+a)

intperiodic(expr, x, a, b) is equivalent to integrate(periodic(expr, x, a, b),x).

Function bffind_root()
Syntax bffind_root(expr, funlist x, lowguess, highguess, digits)

bffind_root() can find the roots of functions defined piecewise or non-piecewise and can produce an answer to arbitrary precision depending on the values of digits. The answer will be expressed as a big float. This function uses the Falsi search method and requires two numbers to start the search. One number must cause the expression to be positive and the other must cause the expression to be negative. If the expression is piecewise then this function may not return a root. For example consider the following.

```
(%i1) f(x):=''(pw([minf,2,2,-5,inf],x));

                  5 (signum(x - 2) + 1)
(%o1)   f(x) := - --------------------- - signum(x - 2) + 1
                           2

(%i2) bffind_root(f(x), [], x, 7, 1, 30);

(%o2)                         2.0b0

(%i3) f(%);  /* in this case a boundary point was found where the function changes sign */
                               3
(%o3)                       -  -
                               2
(%i4) g(x):=''(pw([minf,-x^2+4,0,x^2-3,inf],x));
                                                   2                                2
                                                 (x  - 3) (signum(x) + 1)   (4 - x ) (1 - signum(x))
(%o4)                                    g(x) := ------------------------ + ------------------------
                                                            2                          2
(%i5) bffind_root(g(x),[], x, 1,3,25);  /* in this case a root is found */

(%o5)                                                     1.732050807568877293527446b0
(%i6) g(%);
(%o6)                                                        5.551115123125783b-17
```

In the first case the function returned the boundary point where the function changes sign, but the value of the function is not zero at this point. In the second case the function returned a root of the piecewise function to 25 places of accuracy. The all cases the result will always be either a root or a boundary point or a message that will say the 2 guesses cause the expression to have the same sign.

Functions simpsignumargs(expr, x), simpunitstep(expr, x), simpsignum(expr), negunitstep(expr), pullinif(expr) and pulliniif(expr).

These functions try to change the form of the expression that has unit_step(), iif(), %if() or signum() functions in it. They are used internally by pw.mac but can be called by users. Simpsignumargs(expr,x) tries to change an expression that has calls to signum() with quadratic arguments in it to the product of signum() expression that have linear arguments. Simpunitstep() can simplify many expressions involving unit_step() functions. Simpsignum() tries to factor a polynomial expression in the arguments of a signum() expression to a product of signum() expressions that have linear arguments. Negunitstep() turns a negative expression involving unit_step(negative_expression) into 1-unit_step(positive_expression). Pulliniif() transforms iif() expressions so that the last operation performed is the iif().

```
(%i1) signum(x^2-2*a*x+a^2);
                                                                    2            2
(%o1)                                                       signum(x  - 2 a x + a )
(%i2) simpsignumargs(%,x);
(%o2)                                                        1 - unit_spike(x - a)
(%i3) signum(x^2-2*a*x-a^2);
                                                                    2            2
(%o3)                                                       signum(x  - 2 a x - a )
(%i4) simpsignumargs(%,x);
                                                                    2            2
(%o4)                                                       signum(x  - 2 a x - a )
(%i5) simpsignumargs(%,x),a=1;
                                                                   3/2              3/2
                                                              2 - 2                2    + 2
(%o5)                                             signum(x - --------) signum(x - --------)
                                                                 2          2
(%i6) prod(unit_step(x+i)*unit_step(x-i),i,1,10)$

(%i7) simpunitstep(%,x);
(%o7)                                                          unit_step(x - 10)
(%o8) pulliniif(iif(x>0,a,b)+iif(x<0,c,d));
(%o8                                                 iif(x < 0, c + b, iif(x > 0, d + a, d + b))
```

Functions pwlimit(expr, x, minf), pwlimit(expr, x, inf).
These functions try to compute the limit at + or - infinity of an expression involving signum(). Only ‘inf and ‘minf are allowed in the third argument. With a little creativity you can find limits anywhere even given this restriction. (Hint: To compute at 3 consider the expression y=3+1/x)

```
(%i1) pwlimit(signum(x^9-c),x,inf);
(%o1) 1

(%i2) simp_given(pwlimit(x^2*(signum(a*x^9-c)-signum(a*x^3-b)),x,inf),notequal(a,0));
(%o2) 0

(%i3) pwlimit((signum(a*x^9-c)-signum(a*x^3-b)),x,inf);
(%o3) unit_spike(a) signum(b) - unit_spike(a) signum(c)

EOF
```