```
> restart
> with(Statistics):
> X := RandomVariable(Uniform(0, 2*Pi)):

  f := proc()
    DEBUG():
    PDF(sin(X), y)
  end proc:

> f()
```

$$\begin{cases} 0 & y < 0 \\ \frac{3}{2\,\pi} & y = 0 \\ \frac{1}{\pi\,\sqrt{-y^2+1}} & y < 1 \\ 0 & 1 \le y \end{cases} \tag{1}$$

==========

```
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything] := proc(r::algebraic, x::name, f::{operator, D(anything)}, t::algebraic,
{inert::truefalse := false, mainbranch := ':-NoUserValue', numeric := ':-NoUserValue'})
local q, l, hd, hi, cn, cx, h0, u, si, sd, s0, support, assumptions;
  1   if type(x,':-RandomVariable') then
  2     support := Statistics:-RandomVariables:-Support:-GetValue(x,':-outputtype = :-range');
  3     assumptions := {`if`(lhs(support) <> -infinity,lhs(support) <= x,NULL), `if`(rhs(support) <> infinity,x <= rhs(support),NULL)};
  4     l := minimize(r,x = support);
  5     u := maximize(r,x = support)
      else
  6     assumptions := {}
      end if;
  7 ! if not (assigned('l') and type(l,':-algebraic')) then
  8     l := minimize(r,x)
      end if;
```

```
  9   if not (assigned('u') and type(u,':-algebraic')) then
 10     u := maximize(r,x)
       end if;
 11   q := diff(r,x);
 12   si := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
 13   sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
 14   s0 := {solve({op(assumptions), q = 0},{x})};
 15   if si = {} and sd = {} and s0 = {} then
 16     return FAIL
       end if;
 17   hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];
 18   if has(hd,FAIL) then
 19     return FAIL
       end if;
 20   hi := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,1,t),u = si)];
 21   if has(hi,FAIL) then
 22     return FAIL
       end if;
 23   h0 := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['constant'](r,x,f,u,t,_options),u = s0)];
 24   if has(h0,FAIL) then
 25     return FAIL
       end if;
 26   cn := `if`(l = -infinity,false,t < l);
 27   cx := `if`(u = infinity,true,t <= u);
 28   return simplify(`+`(op(h0))+piecewise(cn,0,cx,`+`(op(hd),op(hi)),0),'piecewise')
end proc

================================================
1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  9   if not (assigned('u') and type(u,':-algebraic')) then
        ...
       end if;

================================================
1
```

```
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  11   q := diff(r,x);

================================================
cos(_R)
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  12   si := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);

================================================
{{0 <= _R, _R < 1/2*Pi}, {_R <= 2*Pi, 3/2*Pi < _R}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  13   sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);

================================================
{{_R < 3/2*Pi, 1/2*Pi < _R}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  14   s0 := {solve({op(assumptions), q = 0},{x})};

================================================
{{_R = 1/2*Pi}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  14   s0 := {solve({op(assumptions), q = 0},{x})};

================================================
{{_R = 1/2*Pi}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  15   if si = {} and sd = {} and s0 = {} then
         ...
       end if;

================================================
{{_R = 1/2*Pi}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  17   hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];

================================================
```

```
[piecewise(y < 0,0,y < 1,1/2/Pi/(-y^2+1)^(1/2),1 <= y,0)]
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  18   if has(hd,FAIL) then
         ...
       end if;

================================================================================================
[piecewise(y < 0,0,y < 1,1/2/Pi/(-y^2+1)^(1/2),1 <= y,0)]
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  20   hi := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,1,t),u = si)];

================================================
[piecewise(y < 0,0,y < 1,1/2/Pi/(-y^2+1)^(1/2),1 <= y,0), piecewise(y = 0,1/2/Pi,0)]
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  21   if has(hi,FAIL) then
         ...
       end if;

================================================
[piecewise(y < 0,0,y < 1,1/2/Pi/(-y^2+1)^(1/2),1 <= y,0), piecewise(y = 0,1/2/Pi,0)]
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  23   h0 := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['constant'](r,x,f,u,t,_options),u = s0)];

================================================
[0]
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  24   if has(h0,FAIL) then
         ...
       end if;

================================================
[0]
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  26   cn := `if`(l = -infinity,false,t < l);

================================================
```

```
y < -1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  27   cx := `if`(u = infinity,true,t <= u);

============================================
y <= 1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  28   return simplify(`+`(op(h0))+piecewise(cn,0,cx,`+`(op(hd),op(hi)),0),'piecewise')

============================================
piecewise(y < 0,0,y = 0,3/2/Pi,y < 1,1/Pi/(-y^2+1)^(1/2),1 <= y,0)
Statistics:-RandomVariables:-PDF:-Univariate:-GetValue:
   5   return `if`(type(t,'float') or numeric = true,evalf(v),subs(x = t,v))

============================================
```

```
> kernelopts(opaquemodules=false):
> stopat(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]);
                                  [f]
```

(2

```
> f()
```

$$\begin{cases} 0 & y < 0 \\ \frac{3}{2\,\pi} & y = 0 \\ \frac{1}{\pi\,\sqrt{-y^2+1}} & y < 1 \\ 0 & 1 \le y \end{cases}$$

(3

```
f:
   2   Statistics:-PDF(sin(X),y)
============================================
f := proc()
   1   DEBUG();
```

```
   2 ! Statistics:-PDF(sin(X),y)
end proc
================================================
[f]
f:
   2   Statistics:-PDF(sin(X),y)
================================================
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
   1*  if type(x,':-RandomVariable') then
         ...
       else
         ...
       end if;
================================================
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything] := proc(r::algebraic, x::name, f::{operator, D(anything)},
t::algebraic, {inert::truefalse := false, mainbranch := ':-NoUserValue', numeric := ':-NoUserValue'})
local q, l, hd, hi, cn, cx, h0, u, si, sd, s0, support, assumptions;
   1*! if type(x,':-RandomVariable') then
   2     support := Statistics:-RandomVariables:-Support:-GetValue(x,':-outputtype = :-range');
   3     assumptions := {`if`(lhs(support) <> -infinity,lhs(support) <= x,NULL), `if`(rhs(support) <> infinity,x <= rhs(support),NULL)};
   4     l := minimize(r,x = support);
   5     u := maximize(r,x = support)
       else
   6     assumptions := {}
       end if;
   7   if not (assigned('l') and type(l,':-algebraic')) then
   8     l := minimize(r,x)
       end if;
   9   if not (assigned('u') and type(u,':-algebraic')) then
  10     u := maximize(r,x)
       end if;
  11   q := diff(r,x);
  12   si := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
  13   sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
  14   s0 := {solve({op(assumptions), q = 0},{x})};
  15   if si = {} and sd = {} and s0 = {} then
```

```
 16     return FAIL
      end if;
 17   hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];
 18   if has(hd,FAIL) then
 19     return FAIL
      end if;
 20   hi := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,1,t),u = si)];
 21   if has(hi,FAIL) then
 22     return FAIL
      end if;
 23   h0 := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['constant'](r,x,f,u,t,_options),u = s0)];
 24   if has(h0,FAIL) then
 25     return FAIL
      end if;
 26   cn := `if`(l = -infinity,false,t < l);
 27   cx := `if`(u = infinity,true,t <= u);
 28   return simplify(`+`(op(h0))+piecewise(cn,0,cx,`+`(op(hd),op(hi)),0),'piecewise')
end proc
================================================
1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  7   if not (assigned('l') and type(l,':-algebraic')) then
        ...
      end if;
================================================
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything] := proc(r::algebraic, x::name, f::{operator, D(anything)},
t::algebraic, {inert::truefalse := false, mainbranch := ':-NoUserValue', numeric := ':-NoUserValue'})
local q, l, hd, hi, cn, cx, h0, u, si, sd, s0, support, assumptions;
  1*  if type(x,':-RandomVariable') then
  2     support := Statistics:-RandomVariables:-Support:-GetValue(x,':-outputtype = :-range');
  3     assumptions := {`if`(lhs(support) <> -infinity,lhs(support) <= x,NULL), `if`(rhs(support) <> infinity,x <= rhs(support),NULL)};
  4     l := minimize(r,x = support);
  5     u := maximize(r,x = support)
      else
  6     assumptions := {}
      end if;
```

```
  7 ! if not (assigned('l') and type(l,':-algebraic')) then
  8     l := minimize(r,x)
      end if;
  9   if not (assigned('u') and type(u,':-algebraic')) then
 10     u := maximize(r,x)
      end if;
 11   q := diff(r,x);
 12   si := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
 13   sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
 14   s0 := {solve({op(assumptions), q = 0},{x})};
 15   if si = {} and sd = {} and s0 = {} then
 16     return FAIL
      end if;
 17   hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];
 18   if has(hd,FAIL) then
 19     return FAIL
      end if;
 20   hi := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,1,t),u = si)];
 21   if has(hi,FAIL) then
 22     return FAIL
      end if;
 23   h0 := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['constant'](r,x,f,u,t,_options),u = s0)];
 24   if has(h0,FAIL) then
 25     return FAIL
      end if;
 26   cn := `if`(l = -infinity,false,t < l);
 27   cx := `if`(u = infinity,true,t <= u);
 28   return simplify(`+`(op(h0))+piecewise(cn,0,cx,`+`(op(hd),op(hi)),0),'piecewise')
end proc
============================================
1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  9   if not (assigned('u') and type(u,':-algebraic')) then
        ...
      end if;
============================================
```

```
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything] := proc(r::algebraic, x::name, f::{operator, D(anything)},
t::algebraic, {inert::truefalse := false, mainbranch := ':-NoUserValue', numeric := ':-NoUserValue'})
local q, l, hd, hi, cn, cx, h0, u, si, sd, s0, support, assumptions;
  1*  if type(x,':-RandomVariable') then
  2     support := Statistics:-RandomVariables:-Support:-GetValue(x,':-outputtype = :-range');
  3     assumptions := {`if`(lhs(support) <> -infinity,lhs(support) <= x,NULL), `if`(rhs(support) <> infinity,x <= rhs(support),NULL)};
  4     l := minimize(r,x = support);
  5     u := maximize(r,x = support)
      else
  6     assumptions := {}
      end if;
  7   if not (assigned('l') and type(l,':-algebraic')) then
  8     l := minimize(r,x)
      end if;
  9 ! if not (assigned('u') and type(u,':-algebraic')) then
 10     u := maximize(r,x)
      end if;
 11   q := diff(r,x);
 12   si := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
 13   sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
 14   s0 := {solve({op(assumptions), q = 0},{x})};
 15   if si = {} and sd = {} and s0 = {} then
 16     return FAIL
      end if;
 17   hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];
 18   if has(hd,FAIL) then
 19     return FAIL
      end if;
 20   hi := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,1,t),u = si)];
 21   if has(hi,FAIL) then
 22     return FAIL
      end if;
 23   h0 := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['constant'](r,x,f,u,t,_options),u = s0)];
 24   if has(h0,FAIL) then
 25     return FAIL
      end if;
```

```
  26   cn := `if`(l = -infinity,false,t < l);
  27   cx := `if`(u = infinity,true,t <= u);
  28   return simplify(`+`(op(h0))+piecewise(cn,0,cx,`+`(op(hd),op(hi)),0),'piecewise')
end proc
==============================================
1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  11   q := diff(r,x);
==============================================
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything] := proc(r::algebraic, x::name, f::{operator, D(anything)},
t::algebraic, {inert::truefalse := false, mainbranch := ':-NoUserValue', numeric := ':-NoUserValue'})
local q, l, hd, hi, cn, cx, h0, u, si, sd, s0, support, assumptions;
   1*  if type(x,':-RandomVariable') then
   2     support := Statistics:-RandomVariables:-Support:-GetValue(x,':-outputtype = :-range');
   3     assumptions := {`if`(lhs(support) <> -infinity,lhs(support) <= x,NULL), `if`(rhs(support) <> infinity,x <= rhs(support),NULL)};
   4     l := minimize(r,x = support);
   5     u := maximize(r,x = support)
       else
   6     assumptions := {}
       end if;
   7   if not (assigned('l') and type(l,':-algebraic')) then
   8     l := minimize(r,x)
       end if;
   9   if not (assigned('u') and type(u,':-algebraic')) then
  10     u := maximize(r,x)
       end if;
  11 ! q := diff(r,x);
  12   si := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
  13   sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
  14   s0 := {solve({op(assumptions), q = 0},{x})};
  15   if si = {} and sd = {} and s0 = {} then
  16     return FAIL
       end if;
  17   hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];
  18   if has(hd,FAIL) then
  19     return FAIL
```

```
    end if;
 20   hi := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,1,t),u = si)];
 21   if has(hi,FAIL) then
 22     return FAIL
    end if;
 23   h0 := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['constant'](r,x,f,u,t,_options),u = s0)];
 24   if has(h0,FAIL) then
 25     return FAIL
    end if;
 26   cn := `if`(l = -infinity,false,t < l);
 27   cx := `if`(u = infinity,true,t <= u);
 28   return simplify(`+`(op(h0))+piecewise(cn,0,cx,`+`(op(hd),op(hi)),0),'piecewise')
end proc
================================================
cos(_R)
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
 12   si := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
================================================
solve:
  1   if nargs = 0 then
        ...
      else
        ...
      end if;
================================================
solve := proc(eqs::{`and`, `not`, `or`, algebraic, relation(algebraic), {list, set}({`and`, `not`, `or`, algebraic, relation(algebraic)})}, vars::
{algebraic, {list, set}(algebraic), {list, set}(equation)} := NULL, {[AllSolutions, allsolutions]::truefalse := FAIL, [ConditionalSolutions,
conditionalsolutions]::truefalse := FAIL, [DropMultiplicity, dropmultiplicity]::truefalse := FAIL, [Explicit, explicit]::{posint, truefalse}
:= FAIL, [MaxSols, maxsols]::{posint, pos_infinity} := FAIL, [SolveOverReals, solveoverreals, real]::truefalse := FAIL,
[SymbolicSolutions, symbolicsolutions, symbolic]::truefalse := true, [TryHard, tryhard]::truefalse := FAIL, [UseAssumptions,
useassumptions]::truefalse := false, domain::name := ':-default', parameters::{list, set} := {}, parametric::{truefalse, identical(lazy,
proviso,full)} := false, split::truefalse := false}, $)
local d, i, pw, qq, r, rt, s, sol, t, tmp, RO, T2, TT, XX, XX0, YY, YY0, Args, exArgs, newr, nosp, r_conds, r_eqs, teqns, tvars,
backsubs, discarded, lmaxsols, oldmaxsols, Solutions_are_sets, Variables_in_list, TrigRO;
global `solve/split`, _X, identity, _SolutionsMayBeLost, `solve/SubsRootOfPower`, _MaxSols;
  1 ! if nargs = 0 then
```

```
2     error "expecting at least one argument"
    else
3     Args := [`if`(type(eqs,'list'),{op(eqs)},eqs), vars]
    end if;
4   Variables_in_list := type(vars,'list');
5   oldmaxsols := _MaxSols;
6   if 2 < nargs or vars = NULL and 1 < nargs then
7     `solve/split` := split;
8     if AllSolutions <> FAIL then
9       _EnvAllSolutions := AllSolutions
      end if;
10    if DropMultiplicity <> FAIL then
11      _EnvDropMultiplicity := DropMultiplicity
      end if;
12    if Explicit <> FAIL then
13      _EnvExplicit := Explicit
      end if;
14    if TryHard <> FAIL then
15      _EnvTryHard := TryHard
      end if;
16    if ConditionalSolutions <> FAIL then
17      _EnvConditionalSolutions := ConditionalSolutions
      end if;
18    if SolveOverReals <> FAIL then
19      _EnvSolveOverReals := SolveOverReals
      end if;
20    if MaxSols <> FAIL then
21      _MaxSols := MaxSols;
22      lmaxsols := MaxSols
      else
23      if type(_MaxSols,{'posint', 'infinity'}) then
24        lmaxsols := _MaxSols
        else
25        lmaxsols := infinity
        end if
      end if
```

```
    else
26    if type(_MaxSols,{'posint', 'infinity'}) then
27      lmaxsols := _MaxSols
      else
28      lmaxsols := infinity
      end if
    end if;
29  if SymbolicSolutions <> FAIL then
30    _EnvSymbolicSolutions := SymbolicSolutions
    end if;
31  if has(Args,_X) then
32    sol := `tools/gensym`(`tools/genglobal`[1]("X_"));
33    return op(subs(sol = _X,[solve(op(subs(_X = sol,Args)),op(subsindets([_options],'anything = identical(FAIL)',() -> NULL)))]))
    elif has(Args,_Z) and member(_Z,indets(Args)) then
34    sol := `tools/gensym`(`tools/genglobal`[1]("Z_"));
35    s := subs(_Z = sol,Args);
36    for t in indets(s,('specfunc')('RootOf')) do
37      s := subs(t = subs(sol = _Z,t),s)
      end do;
38    return op(subs(sol = _Z,[solve(op(s),op(subsindets([_options],'anything = identical(FAIL)',() -> NULL)))]))
    end if;
39  if kernelopts('level') < 40 and 1 < nops(Args) and not type(Args[2],'{name, function, {list, set}(function = anything), {list, set}(
{name, function})}') then
40    WARNING("solving for expressions other than names or functions is not recommended.")
    end if;
41  _SolutionsMayBeLost := '_SolutionsMayBeLost';
42  if nops(Args) = 1 then
43    tvars := `minus`(select(type,indets(Args[1]),'name'),map2(op,2,indets(Args[1],('identity')('anything','name'))));
44    Solutions_are_sets := evalb(1 < nops(tvars))
    elif not type(Args[2],{'list', 'set'}) then
45    tvars := {Args[2]};
46    Solutions_are_sets := false
    elif type(Args[2],'list') then
47    if ListTools:-FindRepetitions(Args[2]) <> [] then
48      error "list of unknowns contains repeated elements: %1", ListTools:-MakeUnique(ListTools:-FindRepetitions(Args[2]))
      end if;
```

```
 49    tvars := Args[2];
 50    Solutions_are_sets := true
       else
 51    tvars := Args[2];
 52    Solutions_are_sets := true
       end if;
 53   t := select(type,tvars,'constant');
 54   t := `minus`({op(t)},select(proc (c) type(c,'function') and not type(c,('specfunc')('RootOf')) end proc,{op(t)}));
 55   if nops(t) <> 0 then
 56    error "a constant is invalid as a variable", op(t)
       end if;
 57   if nops(Args) = 2 and nops(Args[2]) = 2 and (typematch(Args[2][1],XX::anyfunc(TT::name)) or typematch(Args[2][1],
XX::anyfunc(TT::name) = XX0::anything)) and (typematch(Args[2][2],YY::anyfunc(T2::name)) or typematch(Args[2][2],YY::anyfunc
(T2::name) = YY0::anything)) and evalb(TT = T2) and not has(Args[1],TT) then
 58    if not (assigned(XX0) and assigned(YY0)) then
 59     r := [`solve/parametric`(Args[1],[op(0,XX), op(0,YY)],TT)]
       else
 60     r := [`solve/parametric`(Args[1],[op(0,XX) = XX0, op(0,YY) = YY0],TT)]
       end if;
 61    tvars := map(t -> `if`(type(t,'function'),op(0,t),op(0,lhs(t))),vars)
       else
 62    RO := select(x -> nops(x) = 1,indets(Args,('specfunc')('RootOf')));
 63    Args := subs(seq(rt = RootOf(op(rt),label = "dontexpand"),rt = RO),Args);
 64    `tools/ClearRememberTable`(SolveTools:-Engine:-Main);
 65    SolveTools:-Engine:-ClearCache();
 66    if 20 < nops([eval(`solve/SubsRootOfPower`,1)]) then
 67     `solve/SubsRootOfPower` := NULL
       end if;
 68    _EnvFloats := false;
 69    _EnvSimplifyRootOf_DoNotMultiply := true;
 70    Testzero := x -> evalb(normal(SolveTools:-CancelInverses(x)) = 0);
 71    if _EnvExplicit <> true and _EnvExplicit <> false and type(Args[1],set) and 1 < nops(Args[1]) then
 72     _EnvExplicit := false
       end if;
 73    if hastype(Args[1],'float') then
 74     exArgs := Args[1];
```

```
 75      exArgs := subsindets(exArgs,`^`,z -> op(1,z)^convert(op(2,z),':-rational'));
 76      exArgs := convert(exArgs,':-rational',':-exact');
 77      if exArgs <> Args[1] then
 78        _EnvFloats := true;
 79        Args[1] := exArgs
         end if
        end if;
 80     if type(Args[1],'set') then
 81       Solutions_are_sets := true;
 82       _EnvSystemOfEqs := evalb(1 < nops(Args[1]));
 83       teqns := Args[1]
        else
 84       teqns := {Args[1]}
        end if;
 85     if parametric <> false or parameters <> {} then
 86       if SolveOverReals = true or not type(teqns,{list, set}({`<>`, `=`, algebraic})) then
 87         if parametric = (':-lazy') or parametric = (':-proviso') then
 88           WARNING("the lazy and proviso modes are not supported for real parametric solving; defaulting to full parametric mode.")
          end if;
 89         r := SolveTools:-SemiAlgebraic(teqns,tvars,`if`(parameters <> {},(':-parameters') = parameters,NULL))
        else
 90         r := SolveTools:-Parametric(teqns,convert(tvars,set),`if`(parameters <> {},convert(parameters,set),NULL),`if`
(parametric::truefalse,NULL,(':-mode') = parametric))
        end if;
 91       return SolveTools:-Utilities:-UnlabelRootOfs(r,'safe')
        end if;
 92     try
 93       backsubs := {};
 94       if hastype(teqns,('specfunc')('identity')) then
 95         if not assigned(_EnvConditionalSolutions) and hastype(teqns,'{`<`, `<=`, piecewise, specfunc(abs)}') then
 96           _EnvSymbolicSolutions := false
          end if;
 97         r := SolveTools:-Identity(teqns,{},tvars)
        else
 98         if hasassumptions(teqns) then
 99           if not UseAssumptions then
```

```
100           if hasassumptions(tvars) then
101             if kernelopts('level') < 80 then
102               WARNING("solve may be ignoring assumptions on the input variables.")
              else
103               userinfo(1,solve,`Warning: solve may be ignoring assumptions on the input variables.`)
              end if;
104             userinfo(5,solve,print(getassumptions(teqns)))
            end if
          else
105           tmp, tvars := selectremove(hastype,tvars,'{identical(`limit/X`), suffixed(property)}');
106           teqns, tvars, backsubs, discarded := SolveTools:-ProcessAssumptions(teqns,tvars);
107           SolveTools:-Utilities:-Union(tvars,tmp);
108           if discarded <> {} then
109             for qq in discarded do
110               if kernelopts('level') < 90 then
111                 WARNING("solve may not respect assumed property '%1' on '%2'.",qq[2],subs(backsubs,qq[1]))
              else
112                 userinfo(1,solve,nprintf("Warning: solve may not respect assumed property '%1' on '%2'.",qq[2],subs(backsubs,qq[1]))
)
              end if;
113             if type(qq[1],'name') then
114               teqns := subs(qq[1] = subs(backsubs,qq[1]),teqns);
115               tvars := subs(qq[1] = subs(backsubs,qq[1]),tvars)
              end if
            end do
          end if
        end if
      end if;
116     if not assigned(_EnvConditionalSolutions) and hastype(teqns,'{`<`, `<=`, piecewise, specfunc(abs)}') then
117       _EnvSymbolicSolutions := false
      end if;
118     r := SolveTools:-Engine:-Main(teqns,{},tvars)
    end if;
119   if attributes(r) = ('nospec') then
120     nosp := true
    end if;
```

```
121       if UseAssumptions then
122         r := subs(backsubs,r);
123         tvars := subs(backsubs,tvars)
          end if
        catch "cannot solve", "improper use of inequality":
124       _EnvAllSolutions := '_EnvAllSolutions';
125       SolveTools:-Engine:-SolutionsLost();
126       error
        finally
127       _MaxSols := oldmaxsols
        end try;
128     r := select(x -> not has(x,FAIL),r);
129     if _EnvFloats = true then
130       newr := NULL;
131       for qq in r do
132         newr := newr, SolveTools:-Engine:-ExpandRofs(qq,'nonalgnum')
        end do;
133       r := [newr];
134       d := indets(r,fraction);
135       d := `union`(`union`(indets(r,integer),map(numer,d)),map(denom,d));
136       d := max(op(map(length,d)),0);
137       if _EnvAllSolutions = true and has(r,'RootOf') then
138         RO := indets(r,'RootOf');
139         RO := remove(has,RO,'index');
140         if select(t -> `=`(nops(t),1),RO,1) <> {} then
141           SolveTools:-Engine:-SolutionsLost()
          else
142           TrigRO, RO := selectremove(hastype,RO,trig);
143           RO := map2(op,-1,RO);
144           RO := remove(type,RO,('Or')('complexcons',('identical')('index') = ('integer')));
145           if RO <> {} or TrigRO <> {} then
146             SolveTools:-Engine:-SolutionsLost()
            end if
          end if
        end if;
147       r := evalf(evalf[Digits+d](r))
```

```
    end if;
148   if hastype(subsindets(r,('specfunc')('piecewise'),x -> subs(x = pw,x)),{'`<`', '`<=`'}) then
149     r := SolveTools:-Engine:-Unify(r)
    end if;
150   if _EnvAllSolutions <> true and hastype(r,specfunc('LambertW')) then
151     r := map(SolveTools:-Utilities:-SpecializeW,r)
    end if;
152   r := map(SolveTools:-UnwindRootOfs,r);
153   r := map(SolveTools:-Utilities:-UnlabelRootOfs,r,'safe');
154   if _EnvExplicit <> false then
155     r := map(SolveTools:-Utilities:-RecognizeCyclotomic,r)
    end if;
156   if nosp <> true then
157     r := SolveTools:-Utilities:-RemoveSpecializations(r)
    end if;
158   if lmaxsols < nops(r) then
159     r := r[1 .. lmaxsols]
    elif not assigned(_MaxSols) and 100 < nops(r) then
160     r := r[1 .. 100];
161     if kernelopts('level') < 40 then
162       WARNING("returning only the first 100 solutions, increase _MaxSols to see more solutions")
      else
163       userinfo(1,solve,`returning only the first 100 solutions, increase _MaxSols to see more solutions`)
      end if
    end if;
164   if r = [] then
165     userinfo(1,solve,`Warning: no solutions found`)
    end if;
166   if _SolutionsMayBeLost = true then
167     if kernelopts('level') < 40 then
168       WARNING("solutions may have been lost")
      else
169       userinfo(1,solve,`Warning: solutions may have been lost`)
      end if
    end if;
170   r := map(SolveTools:-MakeExplicit,r);
```

```
171     if not hastype(teqns,`<>`) then
172       r := map(s -> `if`(s::piecewise,s,map(t -> `if`(t::`<>`,NULL,t),s)),r)
        end if;
173     r := map(SolveTools:-Utilities:-AddIdentitySols,r,tvars);
174     if _EnvFloats = true then
175       r := evalf(evalf[Digits+d](r))
        end if;
176     r := subs((('label') = "dontexpand") = NULL,r);
177     if 1 < nargs then
178       tvars := vars
        end if
      end if;
179   if Variables_in_list then
180     r := SolveTools:-Utilities:-SortSolutions(r,tvars)
      end if;
181   if Variables_in_list and nops(r) = 1 and type(r,('list')('piecewise')) then
182     Variables_in_list := false
      end if;
183   if Solutions_are_sets then
184     if Variables_in_list then
185       return r
        else
186       return op(r)
        end if
      else
187     if nops(r) = 1 and type(r,('list')('piecewise')) then
188       r := PiecewiseTools:-ToList(op(r));
189       r_eqs := map2(op,2,r);
190       r_conds := map2(op,1,r);
191       try
192         r_eqs := map(x -> `if`(has(x,tvars),map(SolveTools:-Utilities:-RealRange:-FromInequalities,x),x),r_eqs)
          catch "cannot handle intervals":
193         NULL
          end try;
194       return piecewise(seq('r_conds[i], r_eqs[i]',i = 1 .. nops(r)))
        end if;
```

```
 195     try
 196       if Variables_in_list then
 197         return map(SolveTools:-Utilities:-RealRange:-FromInequalities,r)
           else
 198         return op(map(SolveTools:-Utilities:-RealRange:-FromInequalities,r))
           end if
         catch "cannot handle intervals":
 199       if Variables_in_list then
 200         return r
           else
 201         return op(r)
           end if
         end try
       end if
end proc
```

```
================================================
[{0 <= _R, _R <= 2*Pi, 0 < cos(_R)}, {_R}]
solve:
  4   Variables_in_list := type(vars,'list');
================================================
false
solve:
  5   oldmaxsols := _MaxSols;
================================================
_MaxSols
solve:
  6   if 2 < nargs or vars = NULL and 1 < nargs then
        ...
      else
        ...
      end if;
================================================
infinity
solve:
 29   if SymbolicSolutions <> FAIL then
        ...
```

```
      end if;
================================================
true
solve:
  31   if has(Args,_X) then
         ...
       elif has(Args,_Z) and member(_Z,indets(Args)) then
         ...
       end if;
================================================
true
solve:
  39   if kernelopts('level') < 40 and 1 < nops(Args) and not type(Args[2],'{name, function, {list, set}(function = anything), {list, set}(
{name, function})}') then
         ...
       end if;
================================================
true
solve:
  41   _SolutionsMayBeLost := '_SolutionsMayBeLost';
================================================
_SolutionsMayBeLost
solve:
  42   if nops(Args) = 1 then
         ...
       elif not type(Args[2],{'list', 'set'}) then
         ...
       elif type(Args[2],'list') then
         ...
       else
         ...
       end if;
================================================
solve := proc(eqs::{`and`, `not`, `or`, algebraic, relation(algebraic), {list, set}({`and`, `not`, `or`, algebraic, relation(algebraic)})}, vars::
{algebraic, {list, set}(algebraic), {list, set}(equation)} := NULL, {[AllSolutions, allsolutions]::truefalse := FAIL, [ConditionalSolutions,
conditionalsolutions]::truefalse := FAIL, [DropMultiplicity, dropmultiplicity]::truefalse := FAIL, [Explicit, explicit]::{posint, truefalse}
```

```
:= FAIL, [MaxSols, maxsols]::{posint, pos_infinity} := FAIL, [SolveOverReals, solveoverreals, real]::truefalse := FAIL,
[SymbolicSolutions, symbolicsolutions, symbolic]::truefalse := true, [TryHard, tryhard]::truefalse := FAIL, [UseAssumptions,
useassumptions]::truefalse := false, domain::name := ':-default', parameters::{list, set} := {}, parametric::{truefalse, identical(lazy,
proviso,full)} := false, split::truefalse := false}, $)
local d, i, pw, qq, r, rt, s, sol, t, tmp, RO, T2, TT, XX, XX0, YY, YY0, Args, exArgs, newr, nosp, r_conds, r_eqs, teqns, tvars,
backsubs, discarded, lmaxsols, oldmaxsols, Solutions_are_sets, Variables_in_list, TrigRO;
global `solve/split`, _X, identity, _SolutionsMayBeLost, `solve/SubsRootOfPower`, _MaxSols;
  1  if nargs = 0 then
  2    error "expecting at least one argument"
     else
  3    Args := [`if`(type(eqs,'list'),{op(eqs)},eqs), vars]
     end if;
  4  Variables_in_list := type(vars,'list');
  5  oldmaxsols := _MaxSols;
  6  if 2 < nargs or vars = NULL and 1 < nargs then
  7    `solve/split` := split;
  8    if AllSolutions <> FAIL then
  9      _EnvAllSolutions := AllSolutions
       end if;
 10    if DropMultiplicity <> FAIL then
 11      _EnvDropMultiplicity := DropMultiplicity
       end if;
 12    if Explicit <> FAIL then
 13      _EnvExplicit := Explicit
       end if;
 14    if TryHard <> FAIL then
 15      _EnvTryHard := TryHard
       end if;
 16    if ConditionalSolutions <> FAIL then
 17      _EnvConditionalSolutions := ConditionalSolutions
       end if;
 18    if SolveOverReals <> FAIL then
 19      _EnvSolveOverReals := SolveOverReals
       end if;
 20    if MaxSols <> FAIL then
 21      _MaxSols := MaxSols;
```

```
 22       lmaxsols := MaxSols
        else
 23       if type(_MaxSols,{'posint', 'infinity'}) then
 24         lmaxsols := _MaxSols
          else
 25         lmaxsols := infinity
          end if
        end if
      else
 26     if type(_MaxSols,{'posint', 'infinity'}) then
 27       lmaxsols := _MaxSols
        else
 28       lmaxsols := infinity
        end if
      end if;
 29   if SymbolicSolutions <> FAIL then
 30     _EnvSymbolicSolutions := SymbolicSolutions
      end if;
 31   if has(Args,_X) then
 32     sol := `tools/gensym`(`tools/genglobal`[1]("X_"));
 33     return op(subs(sol = _X,[solve(op(subs(_X = sol,Args)),op(subsindets([_options],'anything = identical(FAIL)',() -> NULL)))]))
    elif has(Args,_Z) and member(_Z,indets(Args)) then
 34     sol := `tools/gensym`(`tools/genglobal`[1]("Z_"));
 35     s := subs(_Z = sol,Args);
 36     for t in indets(s,('specfunc')('RootOf')) do
 37       s := subs(t = subs(sol = _Z,t),s)
        end do;
 38     return op(subs(sol = _Z,[solve(op(s),op(subsindets([_options],'anything = identical(FAIL)',() -> NULL)))]))
      end if;
 39   if kernelopts('level') < 40 and 1 < nops(Args) and not type(Args[2],'{name, function, {list, set}(function = anything), {list, set}(
{name, function})}') then
 40     WARNING("solving for expressions other than names or functions is not recommended.")
      end if;
 41   _SolutionsMayBeLost := '_SolutionsMayBeLost';
 42 ! if nops(Args) = 1 then
 43     tvars := `minus`(select(type,indets(Args[1]),'name'),map2(op,2,indets(Args[1],('identity')('anything','name'))));
```

```
 44    Solutions_are_sets := evalb(1 < nops(tvars))
      elif not type(Args[2],{'list', 'set'}) then
 45    tvars := {Args[2]};
 46    Solutions_are_sets := false
      elif type(Args[2],'list') then
 47    if ListTools:-FindRepetitions(Args[2]) <> [] then
 48      error "list of unknowns contains repeated elements: %1", ListTools:-MakeUnique(ListTools:-FindRepetitions(Args[2]))
       end if;
 49    tvars := Args[2];
 50    Solutions_are_sets := true
      else
 51    tvars := Args[2];
 52    Solutions_are_sets := true
      end if;
 53   t := select(type,tvars,'constant');
 54   t := `minus`({op(t)},select(proc (c) type(c,'function') and not type(c,('specfunc')('RootOf')) end proc,{op(t)}));
 55   if nops(t) <> 0 then
 56    error "a constant is invalid as a variable", op(t)
      end if;
 57   if nops(Args) = 2 and nops(Args[2]) = 2 and (typematch(Args[2][1],XX::anyfunc(TT::name)) or typematch(Args[2][1],
XX::anyfunc(TT::name) = XX0::anything)) and (typematch(Args[2][2],YY::anyfunc(T2::name)) or typematch(Args[2][2],YY::anyfunc
(T2::name) = YY0::anything)) and evalb(TT = T2) and not has(Args[1],TT) then
 58    if not (assigned(XX0) and assigned(YY0)) then
 59      r := [`solve/parametric`(Args[1],[op(0,XX), op(0,YY)],TT)]
       else
 60      r := [`solve/parametric`(Args[1],[op(0,XX) = XX0, op(0,YY) = YY0],TT)]
       end if;
 61    tvars := map(t -> `if`(type(t,'function'),op(0,t),op(0,lhs(t))),vars)
      else
 62    RO := select(x -> nops(x) = 1,indets(Args,('specfunc')('RootOf')));
 63    Args := subs(seq(rt = RootOf(op(rt),label = "dontexpand"),rt = RO),Args);
 64    `tools/ClearRememberTable`(SolveTools:-Engine:-Main);
 65    SolveTools:-Engine:-ClearCache();
 66    if 20 < nops([eval(`solve/SubsRootOfPower`,1)]) then
 67      `solve/SubsRootOfPower` := NULL
       end if;
```

```
 68    _EnvFloats := false;
 69    _EnvSimplifyRootOf_DoNotMultiply := true;
 70    Testzero := x -> evalb(normal(SolveTools:-CancelInverses(x)) = 0);
 71    if _EnvExplicit <> true and _EnvExplicit <> false and type(Args[1],set) and 1 < nops(Args[1]) then
 72      _EnvExplicit := false
       end if;
 73    if hastype(Args[1],'float') then
 74      exArgs := Args[1];
 75      exArgs := subsindets(exArgs,`^`,z -> op(1,z)^convert(op(2,z),':-rational'));
 76      exArgs := convert(exArgs,':-rational',':-exact');
 77      if exArgs <> Args[1] then
 78        _EnvFloats := true;
 79        Args[1] := exArgs
        end if
       end if;
 80    if type(Args[1],'set') then
 81      Solutions_are_sets := true;
 82      _EnvSystemOfEqs := evalb(1 < nops(Args[1]));
 83      teqns := Args[1]
       else
 84      teqns := {Args[1]}
       end if;
 85    if parametric <> false or parameters <> {} then
 86      if SolveOverReals = true or not type(teqns,{list, set}({`<>`, `=`, algebraic})) then
 87        if parametric = (':-lazy') or parametric = (':-proviso') then
 88          WARNING("the lazy and proviso modes are not supported for real parametric solving; defaulting to full parametric mode.")
          end if;
 89        r := SolveTools:-SemiAlgebraic(teqns,tvars,`if`(parameters <> {},(':-parameters') = parameters,NULL))
        else
 90        r := SolveTools:-Parametric(teqns,convert(tvars,set),`if`(parameters <> {},convert(parameters,set),NULL),`if`
(parametric::truefalse,NULL,(':-mode') = parametric))
        end if;
 91      return SolveTools:-Utilities:-UnlabelRootOfs(r,'safe')
       end if;
 92    try
 93      backsubs := {};
```

```
 94      if hastype(teqns,('specfunc')('identity')) then
 95        if not assigned(_EnvConditionalSolutions) and hastype(teqns,'{`<`, `<=`, piecewise, specfunc(abs)}') then
 96          _EnvSymbolicSolutions := false
          end if;
 97        r := SolveTools:-Identity(teqns,{},tvars)
        else
 98        if hasassumptions(teqns) then
 99          if not UseAssumptions then
100            if hasassumptions(tvars) then
101              if kernelopts('level') < 80 then
102                WARNING("solve may be ignoring assumptions on the input variables.")
                 else
103                userinfo(1,solve,`Warning: solve may be ignoring assumptions on the input variables.`)
                 end if;
104              userinfo(5,solve,print(getassumptions(teqns)))
               end if
             else
105            tmp, tvars := selectremove(hastype,tvars,'{identical(`limit/X`), suffixed(property)}');
106            teqns, tvars, backsubs, discarded := SolveTools:-ProcessAssumptions(teqns,tvars);
107            SolveTools:-Utilities:-Union(tvars,tmp);
108            if discarded <> {} then
109              for qq in discarded do
110                if kernelopts('level') < 90 then
111                  WARNING("solve may not respect assumed property '%1' on '%2'.",qq[2],subs(backsubs,qq[1]))
                 else
112                  userinfo(1,solve,nprintf("Warning: solve may not respect assumed property '%1' on '%2'.",qq[2],subs(backsubs,qq[1]))
)
                 end if;
113                if type(qq[1],'name') then
114                  teqns := subs(qq[1] = subs(backsubs,qq[1]),teqns);
115                  tvars := subs(qq[1] = subs(backsubs,qq[1]),tvars)
                 end if
               end do
             end if
           end if
         end if;
```

```
116       if not assigned(_EnvConditionalSolutions) and hastype(teqns,'{`<`, `<=`, piecewise, specfunc(abs)}') then
117         _EnvSymbolicSolutions := false
          end if;
118       r := SolveTools:-Engine:-Main(teqns,{},tvars)
        end if;
119     if attributes(r) = ('nospec') then
120       nosp := true
        end if;
121     if UseAssumptions then
122       r := subs(backsubs,r);
123       tvars := subs(backsubs,tvars)
        end if
      catch "cannot solve", "improper use of inequality":
124     _EnvAllSolutions := '_EnvAllSolutions';
125     SolveTools:-Engine:-SolutionsLost();
126     error
      finally
127     _MaxSols := oldmaxsols
      end try;
128   r := select(x -> not has(x,FAIL),r);
129   if _EnvFloats = true then
130     newr := NULL;
131     for qq in r do
132       newr := newr, SolveTools:-Engine:-ExpandRofs(qq,'nonalgnum')
        end do;
133     r := [newr];
134     d := indets(r,fraction);
135     d := `union`(`union`(indets(r,integer),map(numer,d)),map(denom,d));
136     d := max(op(map(length,d)),0);
137     if _EnvAllSolutions = true and has(r,'RootOf') then
138       RO := indets(r,'RootOf');
139       RO := remove(has,RO,'index');
140       if select(t -> `=`(nops(t),1),RO,1) <> {} then
141         SolveTools:-Engine:-SolutionsLost()
          else
142         TrigRO, RO := selectremove(hastype,RO,trig);
```

```
143          RO := map2(op,-1,RO);
144          RO := remove(type,RO,('Or')('complexcons',('identical')('index') = ('integer')));
145          if RO <> {} or TrigRO <> {} then
146            SolveTools:-Engine:-SolutionsLost()
           end if
         end if
       end if;
147      r := evalf(evalf[Digits+d](r))
     end if;
148    if hastype(subsindets(r,('specfunc')('piecewise'),x -> subs(x = pw,x)),{'`<`', '`<=`'}) then
149      r := SolveTools:-Engine:-Unify(r)
     end if;
150    if _EnvAllSolutions <> true and hastype(r,specfunc('LambertW')) then
151      r := map(SolveTools:-Utilities:-SpecializeW,r)
     end if;
152    r := map(SolveTools:-UnwindRootOfs,r);
153    r := map(SolveTools:-Utilities:-UnlabelRootOfs,r,'safe');
154    if _EnvExplicit <> false then
155      r := map(SolveTools:-Utilities:-RecognizeCyclotomic,r)
     end if;
156    if nosp <> true then
157      r := SolveTools:-Utilities:-RemoveSpecializations(r)
     end if;
158    if lmaxsols < nops(r) then
159      r := r[1 .. lmaxsols]
     elif not assigned(_MaxSols) and 100 < nops(r) then
160      r := r[1 .. 100];
161      if kernelopts('level') < 40 then
162        WARNING("returning only the first 100 solutions, increase _MaxSols to see more solutions")
       else
163        userinfo(1,solve,`returning only the first 100 solutions, increase _MaxSols to see more solutions`)
       end if
     end if;
164    if r = [] then
165      userinfo(1,solve,`Warning: no solutions found`)
     end if;
```

```
166     if _SolutionsMayBeLost = true then
167       if kernelopts('level') < 40 then
168         WARNING("solutions may have been lost")
        else
169         userinfo(1,solve,`Warning: solutions may have been lost`)
        end if
      end if;
170     r := map(SolveTools:-MakeExplicit,r);
171     if not hastype(teqns,`<>`) then
172       r := map(s -> `if`(s::piecewise,s,map(t -> `if`(t::`<>`,NULL,t),s)),r)
      end if;
173     r := map(SolveTools:-Utilities:-AddIdentitySols,r,tvars);
174     if _EnvFloats = true then
175       r := evalf(evalf[Digits+d](r))
      end if;
176     r := subs((('label') = "dontexpand") = NULL,r);
177     if 1 < nargs then
178       tvars := vars
      end if
    end if;
179   if Variables_in_list then
180     r := SolveTools:-Utilities:-SortSolutions(r,tvars)
    end if;
181   if Variables_in_list and nops(r) = 1 and type(r,('list')('piecewise')) then
182     Variables_in_list := false
    end if;
183   if Solutions_are_sets then
184     if Variables_in_list then
185       return r
      else
186       return op(r)
      end if
    else
187     if nops(r) = 1 and type(r,('list')('piecewise')) then
188       r := PiecewiseTools:-ToList(op(r));
189       r_eqs := map2(op,2,r);
```

```
190       r_conds := map2(op,1,r);
191       try
192         r_eqs := map(x -> `if`(has(x,tvars),map(SolveTools:-Utilities:-RealRange:-FromInequalities,x),x),r_eqs)
          catch "cannot handle intervals":
193         NULL
          end try;
194       return piecewise(seq('r_conds[i], r_eqs[i]',i = 1 .. nops(r)))
        end if;
195     try
196       if Variables_in_list then
197         return map(SolveTools:-Utilities:-RealRange:-FromInequalities,r)
          else
198         return op(map(SolveTools:-Utilities:-RealRange:-FromInequalities,r))
          end if
        catch "cannot handle intervals":
199       if Variables_in_list then
200         return r
          else
201         return op(r)
          end if
        end try
      end if
end proc
================================================
true
solve:
  53   t := select(type,tvars,'constant');
================================================
{}
solve:
  54   t := `minus`({op(t)},select(proc (c) type(c,'function') and not type(c,('specfunc')('RootOf')) end proc,{op(t)}));
================================================
{}
solve:
  55   if nops(t) <> 0 then
         ...
```

```
      end if;
================================================
{}
solve:
  57   if nops(Args) = 2 and nops(Args[2]) = 2 and (typematch(Args[2][1],XX::anyfunc(TT::name)) or typematch(Args[2][1],
XX::anyfunc(TT::name) = XX0::anything)) and (typematch(Args[2][2],YY::anyfunc(T2::name)) or typematch(Args[2][2],YY::anyfunc
(T2::name) = YY0::anything)) and evalb(TT = T2) and not has(Args[1],TT) then
        ...
      else
        ...
      end if;
================================================
{_R}
solve:
 179   if Variables_in_list then
        ...
      end if;
================================================
{_R}
solve:
 181   if Variables_in_list and nops(r) = 1 and type(r,('list')('piecewise')) then
        ...
      end if;
================================================
{_R}
solve:
 183   if Solutions_are_sets then
        ...
      else
        ...
      end if
================================================
{_R}
solve:
 184     if Variables_in_list then
          ...
```

```
        else
          ...
        end if
================================================
{_R}
solve:
 186       return op(r)
================================================
Statistics:-RandomVariables:-MonotonicIsReal:
   1   return is(Im(eval(expr,x = Statistics:-RandomVariables:-GetPoint(r,x))) = 0)
================================================
{{0 <= _R, _R < 1/2*Pi}, {_R <= 2*Pi, 3/2*Pi < _R}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  13   sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
================================================
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything] := proc(r::algebraic, x::name, f::{operator, D(anything)},
t::algebraic, {inert::truefalse := false, mainbranch := ':-NoUserValue', numeric := ':-NoUserValue'})
local q, l, hd, hi, cn, cx, h0, u, si, sd, s0, support, assumptions;
   1*  if type(x,':-RandomVariable') then
   2     support := Statistics:-RandomVariables:-Support:-GetValue(x,':-outputtype = :-range');
   3     assumptions := {`if`(lhs(support) <> -infinity,lhs(support) <= x,NULL), `if`(rhs(support) <> infinity,x <= rhs(support),NULL)};
   4     l := minimize(r,x = support);
   5     u := maximize(r,x = support)
       else
   6     assumptions := {}
       end if;
   7   if not (assigned('l') and type(l,':-algebraic')) then
   8     l := minimize(r,x)
       end if;
   9   if not (assigned('u') and type(u,':-algebraic')) then
  10     u := maximize(r,x)
       end if;
  11   q := diff(r,x);
  12   si := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
  13 ! sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
  14   s0 := {solve({op(assumptions), q = 0},{x})};
```

```
  15   if si = {} and sd = {} and s0 = {} then
  16     return FAIL
       end if;
  17   hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];
  18   if has(hd,FAIL) then
  19     return FAIL
       end if;
  20   hi := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,1,t),u = si)];
  21   if has(hi,FAIL) then
  22     return FAIL
       end if;
  23   h0 := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['constant'](r,x,f,u,t,_options),u = s0)];
  24   if has(h0,FAIL) then
  25     return FAIL
       end if;
  26   cn := `if`(l = -infinity,false,t < l);
  27   cx := `if`(u = infinity,true,t <= u);
  28   return simplify(`+`(op(h0))+piecewise(cn,0,cx,`+`(op(hd),op(hi)),0),'piecewise')
end proc
============================================
{{_R < 3/2*Pi, 1/2*Pi < _R}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  14   s0 := {solve({op(assumptions), q = 0},{x})};
============================================
{{_R = 1/2*Pi}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  15   if si = {} and sd = {} and s0 = {} then
         ...
       end if;
============================================
{{_R = 1/2*Pi}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  17   hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];
============================================
`type/operator`:
   1   if type(f,'complex(numeric)') then
```

```
        ...
      end if;
=============================================
`type/operator` := proc(f)
local g;
  1 ! if type(f,'complex(numeric)') then
  2     return true
      end if;
  3   g := `if`(f::name,eval(f,1),f);
  4   if type(g,'{`*`, `+`}') then
  5     foldl(`and`,true,op(map(type,[op(g)],'operator')))
      elif type(g,`^`) then
  6     type(g,'operator^numeric')
      else
  7     type([g],['procedure']) and member('operator',[op(3,eval(g))])
      end if
end proc
=============================================

f:
  2   Statistics:-PDF(sin(X),y)
=============================================

f:
  2   Statistics:-PDF(sin(X),y)
=============================================
[f]
f:
  2   Statistics:-PDF(sin(X),y)
=============================================
[f]
f:
  2   Statistics:-PDF(sin(X),y)
=============================================
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
```

```
  1*  if type(x,':-RandomVariable') then
        ...
      else
        ...
      end if;
==========================================
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything] := proc(r::algebraic, x::name, f::{operator, D(anything)},
t::algebraic, {inert::truefalse := false, mainbranch := ':-NoUserValue', numeric := ':-NoUserValue'})
local q, l, hd, hi, cn, cx, h0, u, si, sd, s0, support, assumptions;
  1*! if type(x,':-RandomVariable') then
   2    support := Support:-GetValue(x,':-outputtype = :-range');
   3    assumptions := {`if`(lhs(support) <> -infinity,lhs(support) <= x,NULL), `if`(rhs(support) <> infinity,x <= rhs(support),NULL)};
   4    l := minimize(r,x = support);
   5    u := maximize(r,x = support)
      else
   6    assumptions := {}
      end if;
   7  if not (assigned('l') and type(l,':-algebraic')) then
   8    l := minimize(r,x)
      end if;
   9  if not (assigned('u') and type(u,':-algebraic')) then
  10    u := maximize(r,x)
      end if;
  11  q := diff(r,x);
  12  si := select(MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
  13  sd := select(MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
  14  s0 := {solve({op(assumptions), q = 0},{x})};
  15  if si = {} and sd = {} and s0 = {} then
  16    return FAIL
      end if;
  17  hd := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,-1,t),u = sd)];
  18  if has(hd,FAIL) then
  19    return FAIL
      end if;
  20  hi := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['monotonic'](r,x,f,u,1,t),u = si)];
  21  if has(hi,FAIL) then
```

```
  22     return FAIL
       end if;
  23    h0 := [seq(Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab['constant'](r,x,f,u,t,_options),u = s0)];
  24    if has(h0,FAIL) then
  25     return FAIL
       end if;
  26    cn := `if`(l = -infinity,false,t < l);
  27    cx := `if`(u = infinity,true,t <= u);
  28    return simplify(`+`(op(h0))+piecewise(cn,0,cx,`+`(op(hd),op(hi)),0),'piecewise')
end proc
================================================
1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
   7   if not (assigned('l') and type(l,':-algebraic')) then
         ...
       end if;
================================================
1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
   9   if not (assigned('u') and type(u,':-algebraic')) then
         ...
       end if;
================================================
1
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  11   q := diff(r,x);
================================================
cos(_R)
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  12   si := select(MonotonicIsReal,{solve({op(assumptions), 0 < q},{x})},r,x);
================================================
{{0 <= _R, _R < 1/2*Pi}, {_R <= 2*Pi, 3/2*Pi < _R}}
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  13   sd := select(Statistics:-RandomVariables:-MonotonicIsReal,{solve({op(assumptions), q < 0},{x})},r,x);
================================================
{{_R < 3/2*Pi, 1/2*Pi < _R}}
```

```
Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[anything]:
  14   s0 := {solve({op(assumptions), q = 0},{x})};
================================================
```

Further on:

```
================================================

Statistics:-RandomVariables:-PDF:-Univariate:-GetValueTab[monotonic] := proc(r::algebraic, x::name, f::{operator, D(anything)}, j::anything,
s::{-1, 1}, t::algebraic, {inert::truefalse := false})
local g, h, k, zz, ret;
   1 ! h := GetInverse(r,x,j,t);
   2   if h = FAIL then
   3     return FAIL
       end if;
   4   k := GetRange(r,x,j,s,t);
   5   if type(t,':-name') then
   6     g := eval(h,t = zz)
       else
   7     g := GetInverse(r,x,j,zz)
       end if;
   8   if inert then
   9     g := Diff(g,zz)
       else
  10     g := diff(g,zz)
       end if;
  11   g := eval(g,zz = t);
  12   g := f(h)*abs(g);
  13   return `assuming`([simplify(piecewise(k,g,0),'piecewise')],[op(j)])
end proc

================================================
```

```
Statistics:-RandomVariables:-GetInverse:
  1   _EnvExplicit := true;

================================================


Statistics:-RandomVariables:-GetInverse := proc(f::algebraic, x::name, r, y::algebraic)
local sols, g, h, t, a, b, c, xx;
  1 ! _EnvExplicit := true;
  2   a, b := Statistics:-RandomVariables:-GetBounds(r,x);
  3   g := `assuming`([eval(f,x = xx)],[a < xx, xx < b]);
  4   g := eval(g,xx = x);
  5   sols := {solve(g = y,{x})};
  6   if numelems(sols) <> 1 and not type(y,':-name') then
  7     return eval(thisproc(f,x,r,xx),xx = y)
      end if;
  8   sols := evalindets(sols,`^`(('dependent')(y),'rational'),t -> convert(t,'surd'));
  9   c := Statistics:-RandomVariables:-GetPoint(r,x);
 10   if numelems(sols) = 1 then
 11     return eval(x,sols[1])
      end if;
 12   for t in sols do
 13     h := eval(x,t);
 14     try
 15       if is(simplify(c-eval(h,eval(y = eval(g,x = c)))) = 0) then
 16         return h
          end if
        catch :
 17       if `assuming`([(testeq or is)(x = eval(h,eval(y = g)))],[a < x, x < b]) then
 18         return h
          end if
        end try
      end do;
 19   return FAIL
end proc

================================================
```

stopat Statistics:-RandomVariables:-GetInverse

```
=============================================
# "Third branch"
3/2*Pi,
2*Pi
Statistics:-RandomVariables:-GetInverse:
   3   g := `assuming`([eval(f,x = xx)],[a < xx, xx < b]);

=============================================
sin(xx)
Statistics:-RandomVariables:-GetInverse:
   4   g := eval(g,xx = x);

=============================================
sin(_R)
Statistics:-RandomVariables:-GetInverse:
   5   sols := {solve(g = y,{x})};

=============================================
{{_R = arcsin(y)}}                    # Should have been  _R = arcsin(y) + 2*Pi
Statistics:-RandomVariables:-GetInverse:
   6   if numelems(sols) <> 1 and not type(y,':-name') then
         ...
       end if;

=============================================
```

THE CORRECT ANSWER

```
> restart
> with(plots):
  with(Statistics):
```

```
> X := RandomVariable(Uniform(0, 2*Pi)):

  phi := x -> sin(x):
```

```
> solve(phi(x)=y, x, allsolutions)
```

$$\pi \_B1\sim + 2\,\pi \_Z1\sim - 2\arcsin(y)\ \_B1\sim + \arcsin(y) \qquad \textbf{(4}$$

```
> indets((4), name) minus {y, Pi}:
  about~(%):
Originally _B1, renamed _B1~:
  is assumed to be: OrProp(0,1)

Originally _Z1, renamed _Z1~:
  is assumed to be: integer
```

```
> reciprocal := eval((4), _Z1=0);
```

$$reciprocal := \pi \_B1\sim - 2\arcsin(y)\ \_B1\sim + \arcsin(y) \qquad \textbf{(5}$$

```
> # Note that the expressions of the reciprocal function of ϕ depend on the branch
  # we consider (obvious) but that the algorithm Maple uses to construf the PDF of
  # Y finds similar expressions for all the branches.

  phi__1 := unapply( eval(reciprocal, _B1=0), y);
  phi__2 := unapply( eval(reciprocal, _B1=1), y);
  phi__3 := unapply( eval(reciprocal, _B1=0) + 2*Pi, y);
```

$$\phi_1 := y \to \arcsin(y)$$

$$\phi_2 := y \to \pi - \arcsin(y)$$

$$\phi_3 := y \to \arcsin(y) + 2\,\pi \qquad \textbf{(6}$$

```
> # Branch domains

  phi__domains := solve(diff(sin(x), x) = 0, allsolutions);
  about(indets(phi__domains)[1])
```

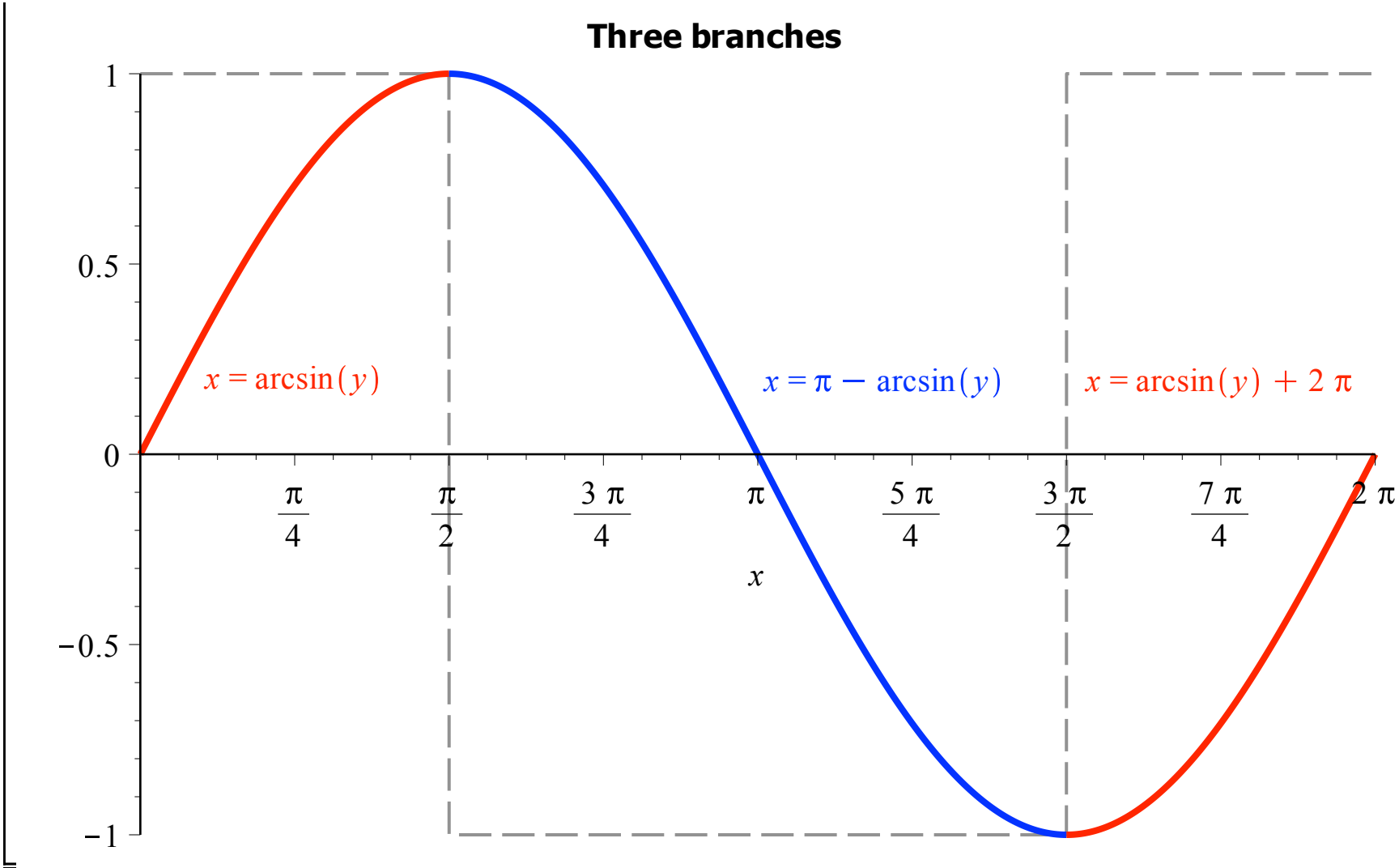
$$\phi_{domains} := \frac{1}{2}\,\pi + \pi \_Z2\sim$$

```
Originally _Z2, renamed _Z2~:
  is assumed to be: integer
```

```
> d := unapply(phi__domains, indets(phi__domains)[1]):

  phi__domains := [0..d(0), d(0)..d(1), d(1)..2*Pi];
```

$$\phi_{domains} := \left[0 .. \frac{1}{2}\,\pi,\ \frac{1}{2}\,\pi .. \frac{3}{2}\,\pi,\ \frac{3}{2}\,\pi .. 2\,\pi\right] \tag{7}$$

```
> display(
    plot(signum(diff(phi(x), x)), x=0..2*Pi, color="Gray", linestyle=3)
    , seq(
        plot(
          phi(x)
          , x=op(1, phi__domains[i])..op(2, phi__domains[i])
          , color=`if`(i::odd, red, blue)
          , thickness=3
        )
        , i=1..3
      )
    , seq(
        textplot(
          [`+`(op(phi__domains[i]))/2, 0.2, typeset(x=(phi__||i)(y))]
          , color=`if`(i::odd, red, blue)
          , `if`(i=2, align=right, NULL)
        )
        , i=1..3
      )
    , title="Three branches"
    , titlefont=[Tahoma, bold, 12]
    , size=[700, 400]
  )
```




```
> f__X := unapply(PDF(X, x), x)
```

$$f_X := x \to piecewise\left(x < 0, 0, x < 2\,\pi, \frac{1}{2\,\pi}, 0\right)$$

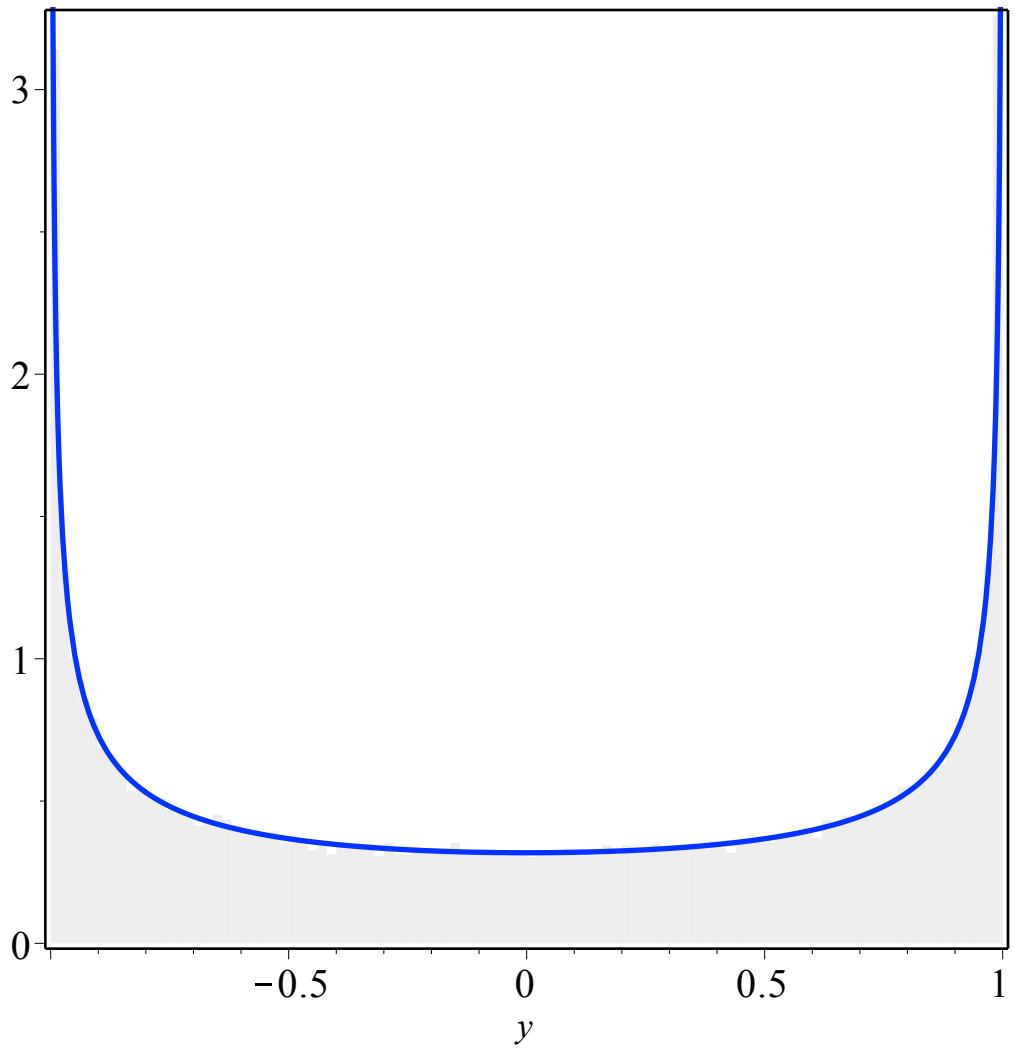
(8

```
> g__X := (a, b) -> x -> piecewise(x < a, 0, x > b, 0, f__X(x)/(b-a)):
```

```
> for b from 1 to numelems(phi__domains) do
    lb := op(1, phi__domains[b]):
```

```
  rb := op(2, phi__domains[b]):
  f__Y__||b := g__X(lb, rb)((phi__||b)(y))
               *
               abs(diff((phi__||b)(y), y))

end do:

f__Y := simplify~( add(f__Y__||b, b=1..numelems(phi__domains)) );


S := phi~(Sample(X, 10^5)):


display(
  plot(f__Y(y), y=-1..1, color=blue, thickness=2)
  ,
  Histogram(S, minbins=100, color="LightGray", transparency=0.5, style=polygon)
)
```

$$f_Y := \begin{cases} 0 & y < -1 \\ \dfrac{1}{\pi\sqrt{y^2-1}} & y = -1 \\ \dfrac{1}{\pi\sqrt{-y^2+1}} & y < 1 \\ \dfrac{1}{\pi\sqrt{y^2-1}} & y = 1 \\ 0 & 1 < y \end{cases}$$



THE INCORRECT MAPLE ANSWER

```
> restart
> with(plots):
  with(Statistics):
```

```
> X := RandomVariable(Uniform(0, 2*Pi)):

  phi := x -> sin(x):
```

```
> # What comes from debugging

  phi__1 := y ->   arcsin(y);
  phi__2 := y -> - arcsin(y);        # instead of y -> Pi - arcsin(y))
  phi__3 := y ->   arcsin(y);        # instead of y -> arcsin(y) + 2*Pi)...
                                     # even if this does not matter here because f_X has a
  constant over Support(X)
```

$$\phi_1 := y \to \arcsin(y)$$

$$\phi_2 := y \to -\arcsin(y)$$

$$\phi_3 := y \to \arcsin(y) \qquad \textbf{(9}$$

```
> phi__domains := [0 .. (1/2)*Pi, (1/2)*Pi .. (3/2)*Pi, (3/2)*Pi .. 2*Pi]
```

$$\phi_{domains} := \left[0 .. \frac{1}{2}\,\pi,\ \frac{1}{2}\,\pi .. \frac{3}{2}\,\pi,\ \frac{3}{2}\,\pi .. 2\,\pi\right] \qquad \textbf{(10}$$

```
> f__X := unapply(PDF(X, x), x)
```

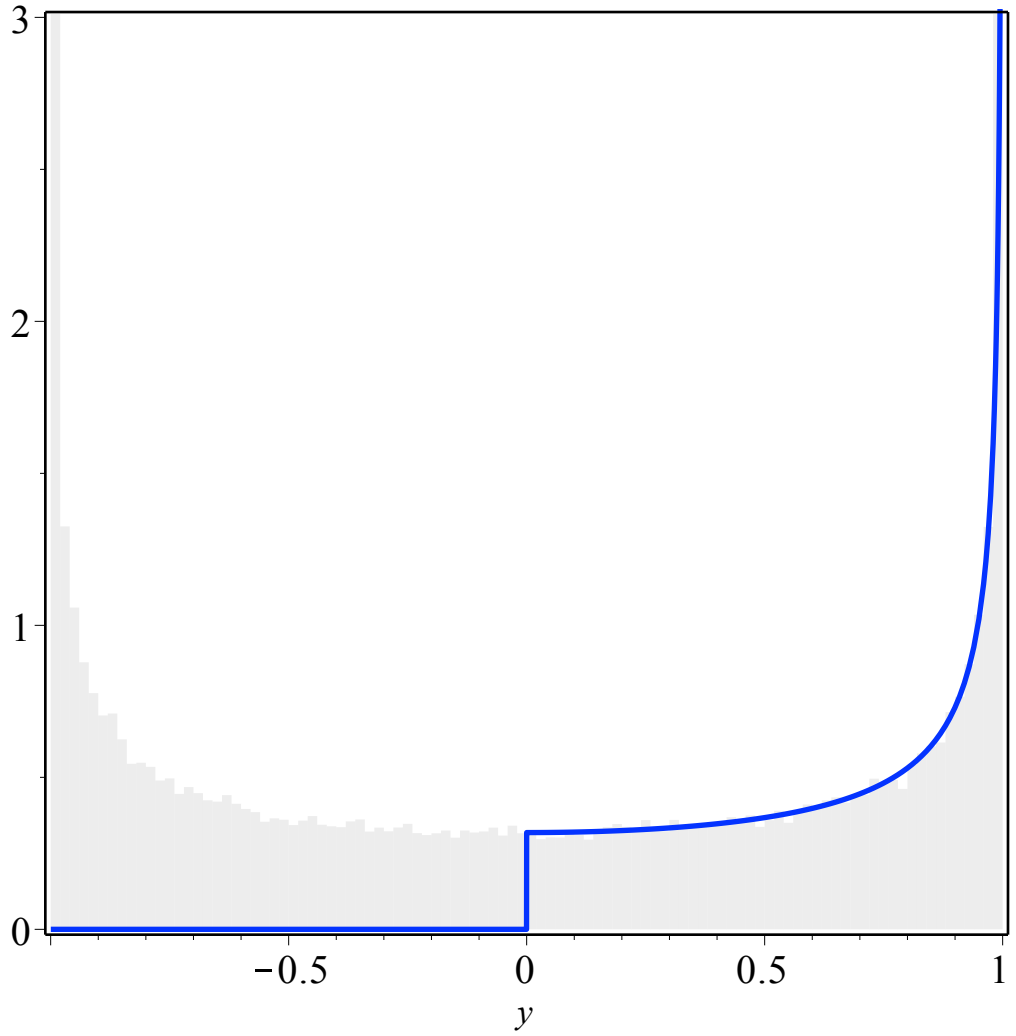
$$f_X := x \to piecewise\left(x < 0, 0, x < 2\,\pi, \frac{1}{2\,\pi}, 0\right) \qquad \textbf{(11}$$

```
> # True random variables as components of the mixture (see main text in the post)

  g__X := (a, b) -> x -> piecewise(x < a, 0, x > b, 0, f__X(x)/(b-a)):
```

```
> f__Y__1 := 0:
  f__Y__2 := 0:
  f__Y__3 := 0:

  for b from 1 to numelems(phi__domains) do
    lb := op(1, phi__domains[b]):
    rb := op(2, phi__domains[b]):
    f__Y__||b := g__X(lb, rb)((phi__||b)(y))
                 *
                 abs(diff((phi__||b)(y), y))
                 /
                 ( (rb-lb) / (Pi) )
```

```
end do:

f__Y := simplify~( add(f__Y__||b, b=1..numelems(phi__domains)) );


S := phi~(Sample(X, 10^5)):


display(
  plot(f__Y, y=-1..1, color=blue, thickness=2)
  ,
  Histogram(S, minbins=100, color="LightGray", transparency=0.5, style=polygon)
  , view=[default, 0..3]
)
```

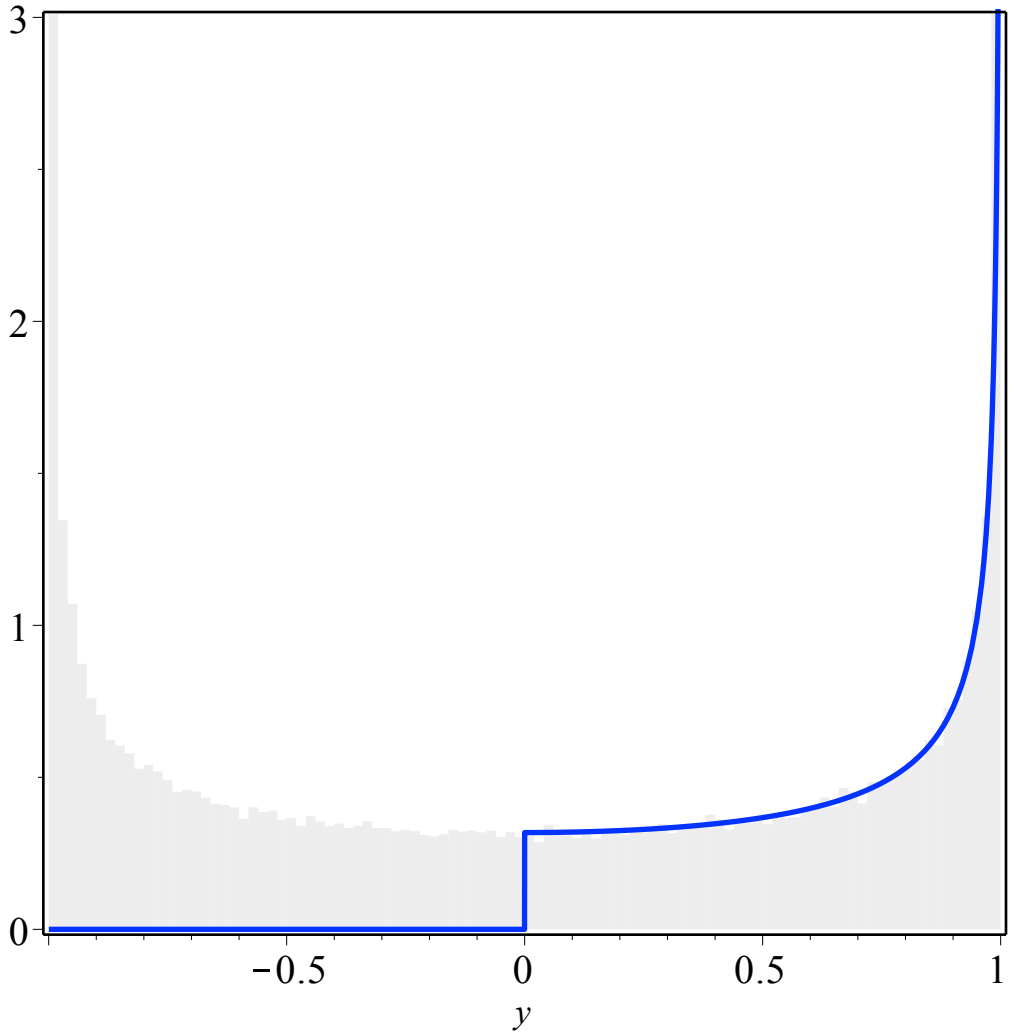
$$f_Y := \begin{cases} 0 & y < -1 \\ \dfrac{1}{2\,\pi\,\sqrt{y^2-1}} & y = -1 \\ 0 & y < 0 \\ \dfrac{1}{\pi\,\sqrt{-y^2+1}} & y < 1 \\ \dfrac{1}{\pi\,\sqrt{y^2-1}} & y = 1 \\ 0 & 1 < y \end{cases}$$




```
> # Alternative point of view

  g__X := (a, b) -> x -> piecewise(x < a, 0, x > b, 0, f__X(x)):

> f__Y__1 := 0:
  f__Y__2 := 0:
  f__Y__3 := 0:

  for b from 1 to numelems(phi__domains) do
```

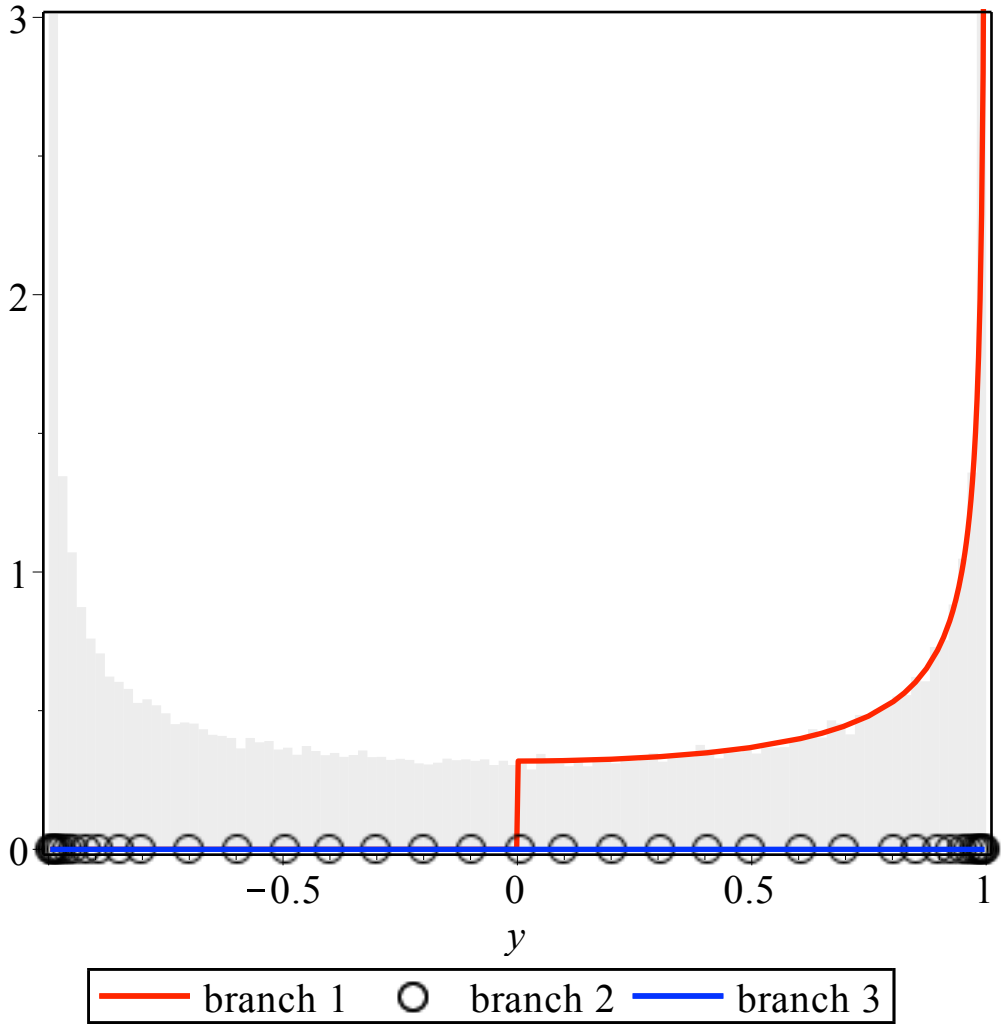
```
  lb := op(1, phi__domains[b]):
  rb := op(2, phi__domains[b]):
  f__Y__||b := g__X(lb, rb)((phi__||b)(y))
               *
               abs(diff((phi__||b)(y), y))

end do:

f__Y := simplify~( add(f__Y__||b, b=1..numelems(phi__domains)) );


S := phi~(Sample(X, 10^5)):


display(
  plot(f__Y*2, y=-1..1, color=blue, thickness=2)
  ,
  Histogram(S, minbins=100, color="LightGray", transparency=0.5, style=polygon)
  , view=[default, 0..3]
)
```

$$f_Y := \begin{cases} 0 & y < -1 \\ \dfrac{1}{2\,\pi\,\sqrt{y^2-1}} & y = -1 \\ 0 & y < 0 \\ \dfrac{1}{2\,\pi\,\sqrt{-y^2+1}} & y < 1 \\ \dfrac{1}{2\,\pi\,\sqrt{y^2-1}} & y = 1 \\ 0 & 1 < y \end{cases}$$




```
> # In fact only the first branch matters given the way Maple does the computations

  co := [red, black, blue]:
  st := [line, point, line]:
  sy := [circle$3]:
  sz := [20$3]:

  i := 'i':
  opt := i -> [color=co[i], style=st[i], symbol=sy[i], symbolsize=sz[i], thickness=2, legend=
```

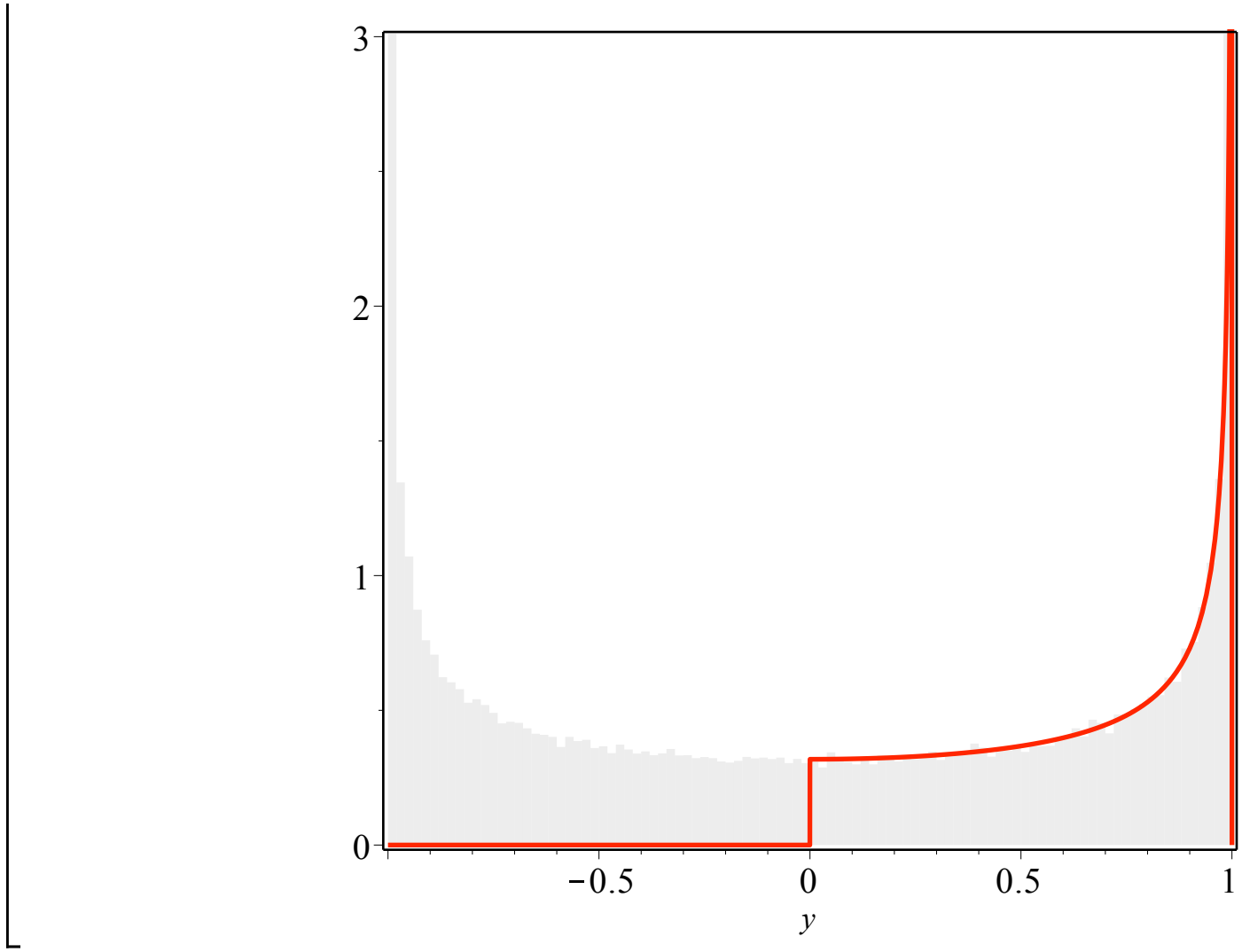
```
cat("branch ", i)]:

display(
  seq(
    plot(2*f__Y__||b, y=-1..1, op(opt(b)), numpoints=21)
    , b=1..numelems(phi__domains)
  )
  ,
  Histogram(S, minbins=100, color="LightGray", transparency=0.5, style=polygon)
  , view=[default, 0..3]
)
```




```
> f__Maple := PDF(phi(X), y)
```

$$f_{Maple} := \begin{cases} 0 & y < 0 \\ \frac{3}{2\pi} & y = 0 \\ \frac{1}{\pi\sqrt{-y^2+1}} & y < 1 \\ 0 & 1 \le y \end{cases} \quad \textbf{(12}$$

```
> display(
    plot(f__Maple, y=-1..1, color=red, thickness=2)
    ,
    Histogram(S, minbins=100, color="LightGray", transparency=0.5, style=polygon)
    , view=[default, 0..3]
  )
```


3
2
1
0
−0.5
0
0.5
1
y