Bibliography

[Federer 1996] FEDERER, H.. **Geometric Measure Theory**. Springer, 1996.

[Langevin 2006] LANGEVIN, R.. **Integral geometry from Buffon to the use of twentieth century mathematics**, 1996.

[Lebeau 1999] LEBEAU, G.. **Théorie des distributions et analyse de Fourier**. École Polytechnique, 1999.

[Morgan 2000] MORGAN, F.. **Geometric Measure Theory: A Beginner's Guide**. Academic Press, London, UK, 2000.

[Santaló 1953] SANTALÓ, L. A.. **Introduction to integral geometry**. Université de Nancago, 1953.

[Schwartz 1997] SCHWARTZ, L.. **Un mathématicien aux prises avec le siècle**. Editions Odile Jacob, Paris, France, 1997.

[do Carmo 2005] DO CARMO, M. P.. **Geometria Diferencial de Curvas e Superfícies**. Sociedade Brasilieira de Matemática, Rio de Janeiro, Brazil, 2005.

A
Maple worksheet for the 2d case

A.0.1
Global parameters

Accuracy

Digits := 5 :

Epsilon

epsilon := 0.1 :

Overlapping

trans := 9/10 :

Graphics viewport

xm := -1.1 - trans : xM := +1.1 + trans : ym := -1.1 : yM := +3.1 :

Immersion

X := t -> t ; Y := t -> sqrt(abs(t)) ;

$$X := t \mapsto t$$
$$Y := t \mapsto \sqrt{|t|}$$

A.0.2
Test functions

```
> phi_unnorm := t-> piecewise( -1<t and t<1, exp((-t^2)/(1-t^2)),
0 );
```

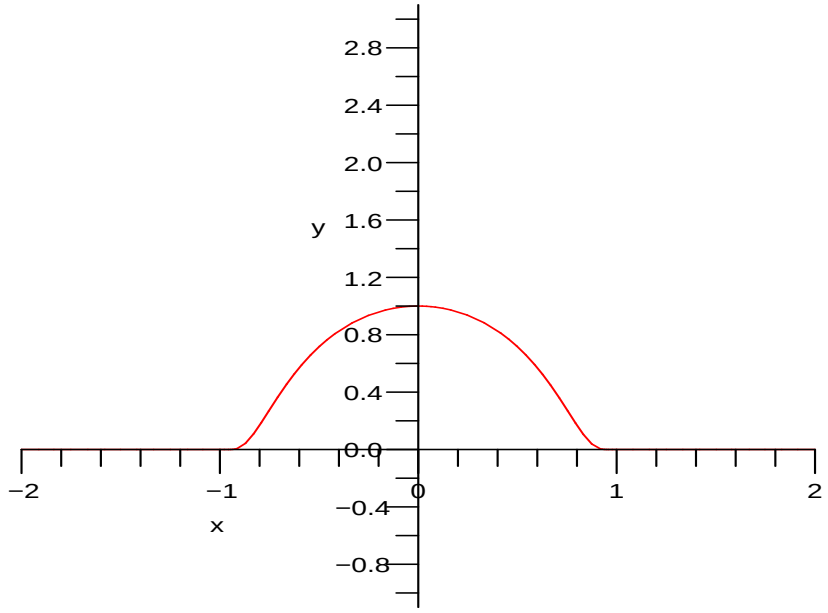
$$phi\_unnorm := (s,t) \mapsto \begin{cases} e^{-\frac{s^2+t^2}{1-s^2-t^2}} & s^2+t^2<1 \\ 0 & otherwise \end{cases}$$

```
> Digits_cpy := Digits :
> Digits := max( Digits, 20 ) :
> norm_phi := evalf( Int( phi_unnorm(t), t=-1..1 ) ) ;
> Digits := Digits_cpy :
```

$$norm\_phi := 1.2069003224378761753$$

```
> phi := t -> phi_unnorm(t) / norm_phi :
> plot( phi_unnorm(x), x=xm..xM, y=ym..yM, scaling=constrained )
;
```

$$\phi := t \mapsto \frac{phi\_unnorm(t)}{norm\_phi}$$




```
>  TFonction := proc( a,m )
>    local A, f, fe, dfe, ddfe ;
>    global phi, epsilon, trans ;
>
>    A := sum( a[i], i=1..nops(a) ) ;
>
>    f    := simplify( t -> sum( a[i]*phi(t+m[i]*trans),
i=1..nops(a) ) / A ) ;
>    fe   := t -> f(t/epsilon) / epsilon ;
>    dfe  := simplify( t -> subs( u=t, diff(  fe(u),u ) ) ) ;
>    ddfe := simplify( t -> subs( u=t, diff( dfe(u),u ) ) ) ;
>
>    print( plot( [ epsilon * fe( epsilon * x ), epsilon^2 *
dfe( epsilon * x ), epsilon^3 * ddfe( epsilon * x ) ], x=xm..xM,
y=ym..yM, scaling=constrained, color=[red, navy, green] ) ) ;
>    return fe, dfe, ddfe ;
>  end proc :
```

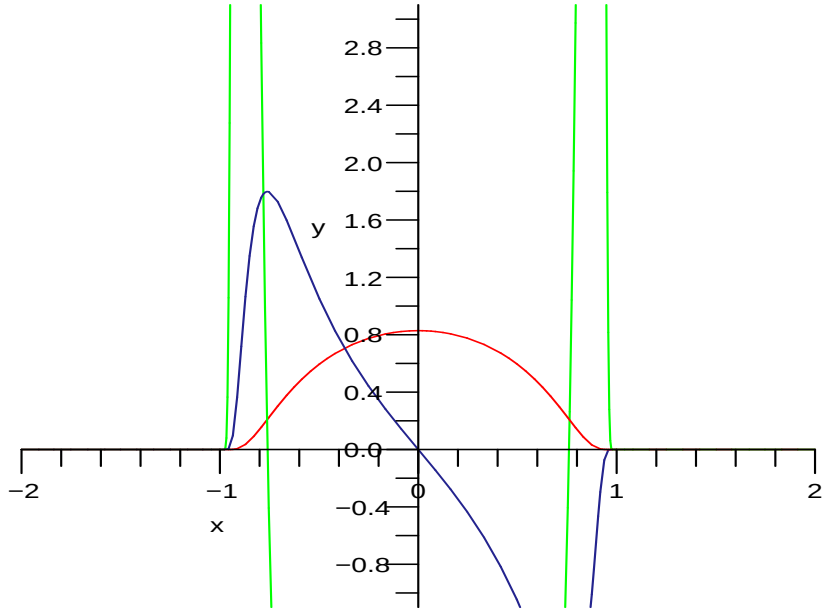
A.0.3
Convolutions

```
>  convole := proc( phi, dphi, ddphi, q )
>    global X,Y, trans, xm,xM,ym,yM, epsilon ;
>    local x,y, dx,dy, ddx, ddy, x0,y0, dx0,dy0, ddx0, ddy0, tg,
ntg, crv, tm,tM ;
>
>    tm   := epsilon * (-1.0-trans) ;
>    tM   := epsilon * ( 1.0+trans) ;
>
>    x    := z -> evalf( Int( X(z-t) * phi  (t), t=tm..tM ) ) ;
>    y    := z -> evalf( Int( Y(z-t) * phi  (t), t=tm..tM ) ) ;
>    dx   := z -> evalf( Int( X(z-t) * dphi (t), t=tm..tM ) ) ;
>    dy   := z -> evalf( Int( Y(z-t) * dphi (t), t=tm..tM ) ) ;
>    ddx  := z -> evalf( Int( X(z-t) * ddphi(t), t=tm..tM ) ) ;
>    ddy  := z -> evalf( Int( Y(z-t) * ddphi(t), t=tm..tM ) ) ;
>
>    x0   := x  (q) ;
>    y0   := y  (q) ;
>    dx0  := dx (q) ;
>    dy0  := dy (q) ;
>    ddx0 := ddx(q) ;
>    ddy0 := ddy(q) ;
>
>    tg   := evalf( dy0 / dx0 ) ;
>    ntg  := sqrt( dx0^2 + dy0^2 ) ;
>    crv  := evalf( ( dx0*ddy0 - dy0*ddx0 ) / ntg^3 ) ;
>
>    print( "At ", x0,y0, " tg=", tg, " crv=", crv ) ;
>
>    plot( [ [x(z),y(z),z=-1..1], tg*(z-x0) + y0, [x0-dy0/crv/ntg
+ sin(theta)/crv,y0+dx0/crv/ntg - cos(theta)/crv,theta=0..2*Pi]
], x=epsilon*xm..epsilon*xM, y=epsilon*ym..epsilon*yM, scaling=constrained,
color=[red,navy,green], thickness=[3,5,2] ) ;
>   end proc :
```

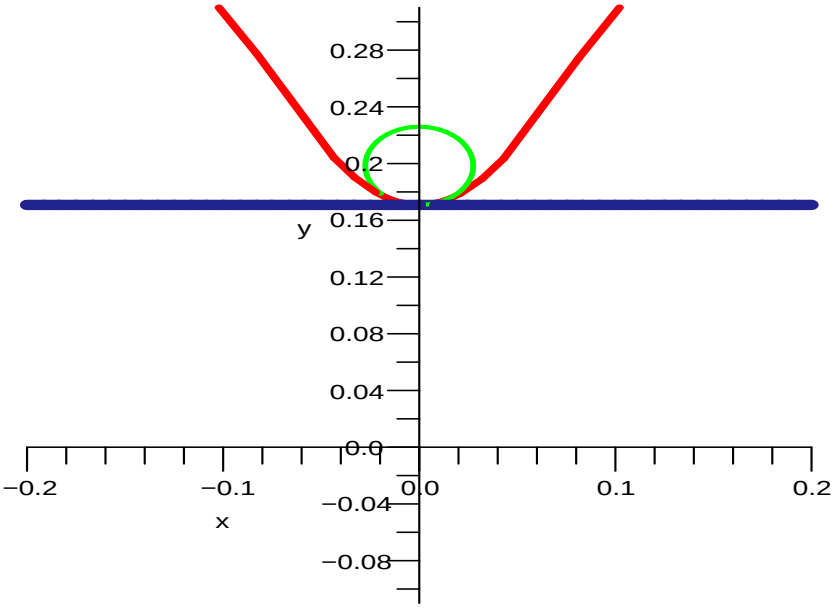
A.0.4
Results

Basic test function

phie, dphie, ddphie := TFonction([3],[0]) : convole(phie, dphie, ddphie, 0) ;

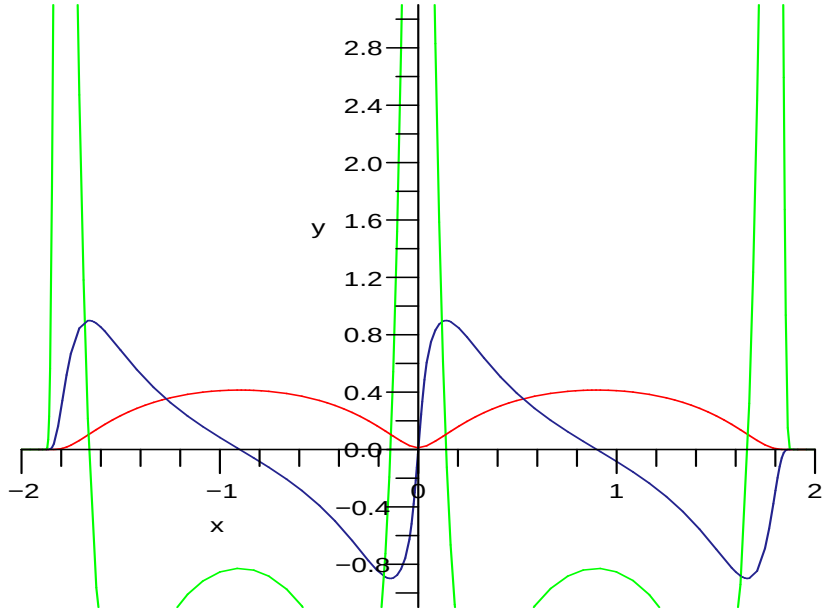


"At'', 0.0, 0.17085, "$tg =''$, 0.0, "$crv =''$, 36.299

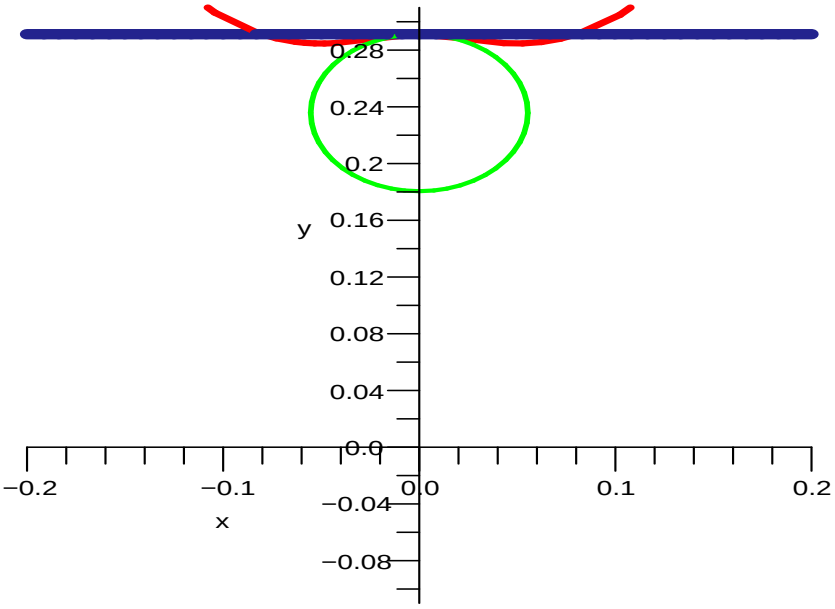


Positive and symmetric double bump test function

phie_2sym, dphie_2sym, ddphie_2sym := TFonction([1,1], [-1,+1]) : convole(phie_2sym, dphie_2sym, ddphie_2sym, 0) ;

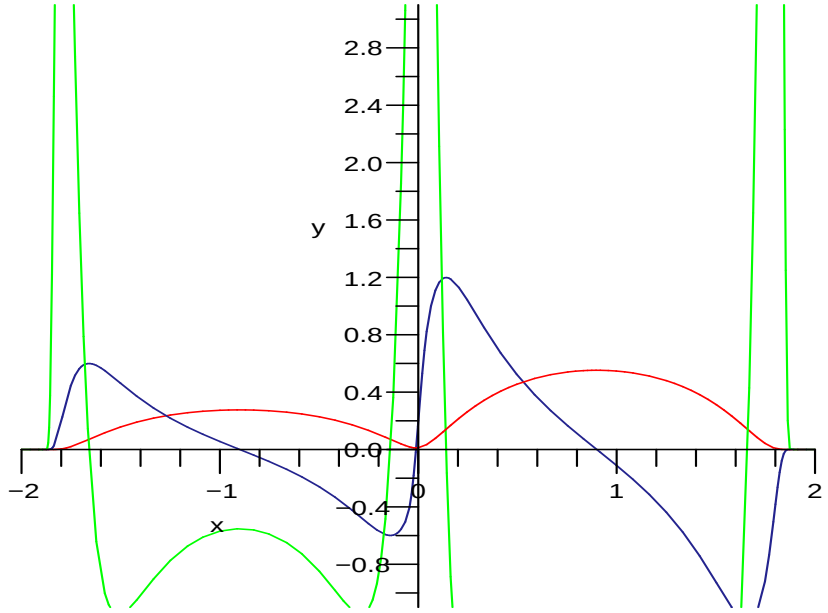


"At'', 0.0, 0.29122, "$tg =''$, 0.0, "$crv =''$, -18.065



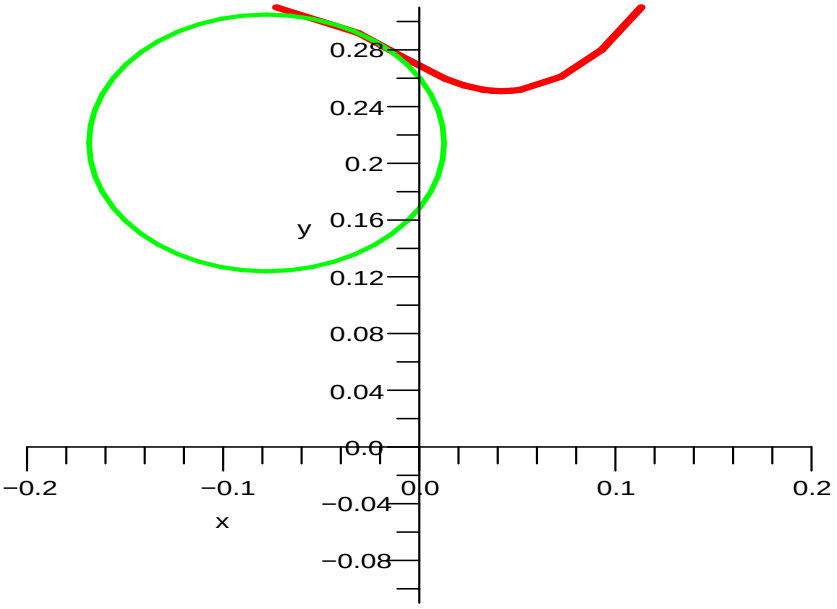
Positive and non-symmetric double bump test function

phie_2dis, dphie_2dis, ddphie_2dis := TFonction([2,1], [-1,+1]) :
convole(phie_2dis, dphie_2dis, ddphie_2dis, 0) ;



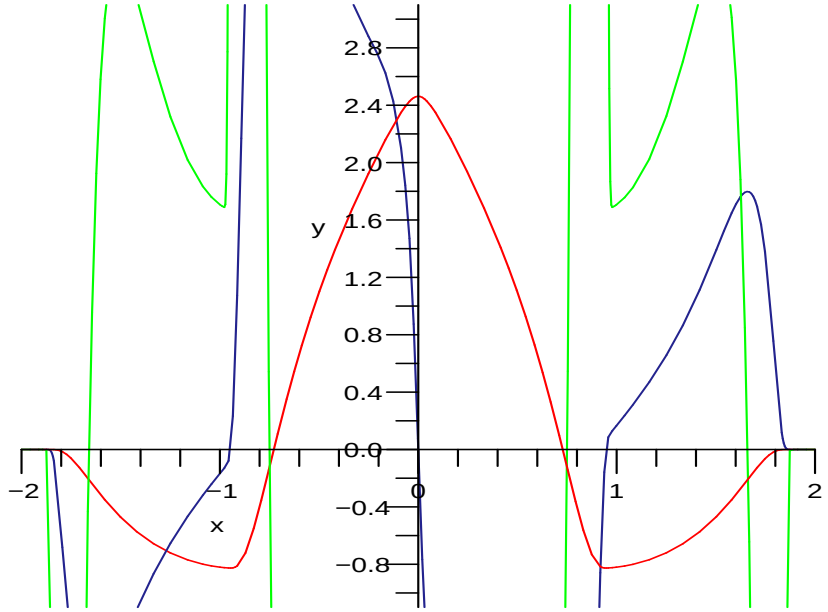
"At'', -0.029999, 0.29122, "$tg =''$, -0.62308, "$crv =''$, -11.045

```
Warning, unable to evaluate 1 of the 3 functions to numeric
values in the region; see the plotting command's help page to
ensure the calling sequence is correct
```

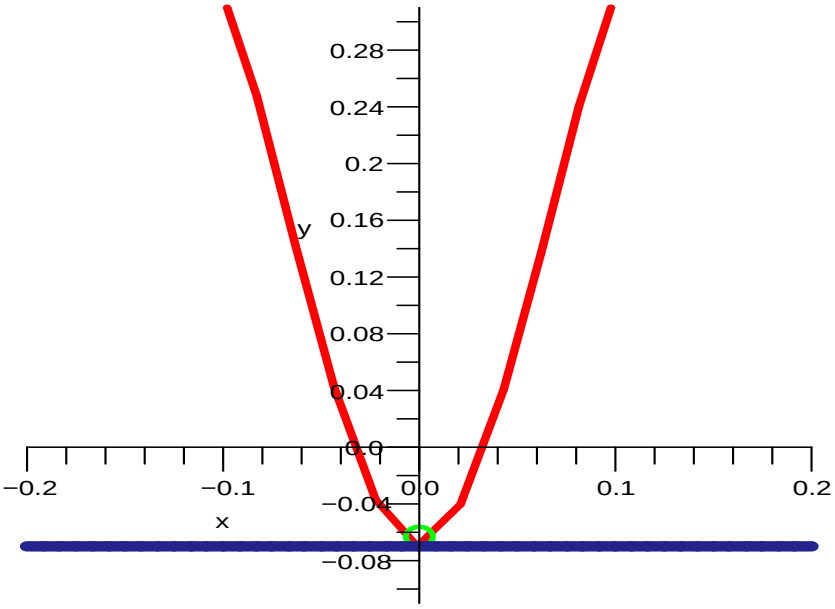


Non-positive and symmetric triple bump test function

phie_nsym, dphie_nsym, ddphie_nsym := TFonction([-1,3,-1], [-1,0,+1]) : convole(phie_nsym, dphie_nsym, ddphie_nsym, 0) ;

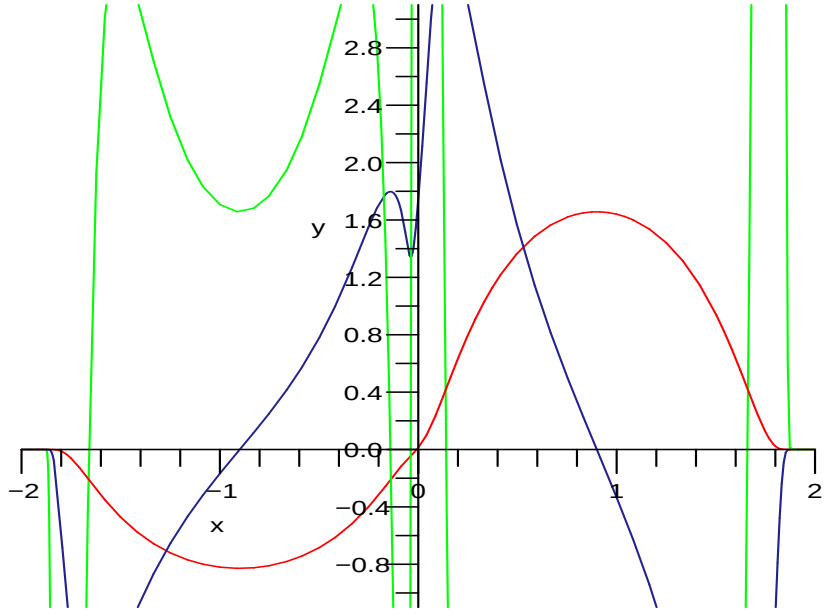


"At'', 0.0, -0.069909, "$tg =''$, 0.0, "$crv =''$, 145.03



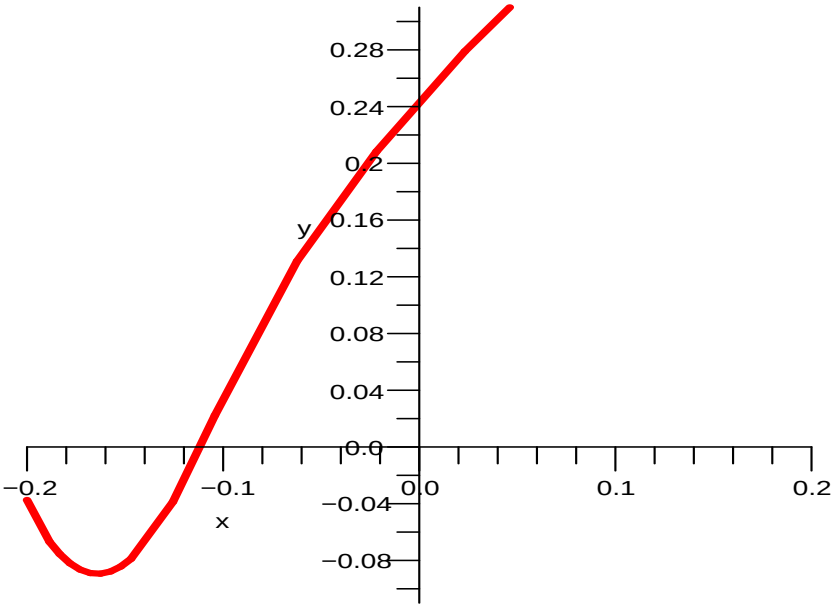
Non-positive and non-symmetric double bump test function

phie_ndis, dphie_ndis, ddphie_ndis := TFonction([2,-1], [-1,+1]) :
convole(phie_ndis, dphie_ndis, ddphie_ndis, 0) ;



"At'', -0.27000, 0.29121, "$tg =''$, -5.6081, "$crv =''$, -0.097733

```
Warning, unable to evaluate 1 of the 3 functions to numeric
values in the region; see the plotting command's help page to
ensure the calling sequence is correct
```



B
Maple worksheet for the 3d case

B.0.5
Global parameters

Accuracy

Digits := 10 :

Epsilon

epsilon := 0.1 :

Overlapping

trans := 9/10 :

Graphics viewport

xm := -1.1 - trans : xM := +1.1 + trans : ym := -1.1 - trans : yM := +1.1 + trans : zm := -1.1 : zM := +3.1 :

Immersion

X := (s,t) -> s ; Y := (s,t) -> t ; Z := (s,t) -> s^2 + t^2 ;

$$X := (s,t) \mapsto s$$
$$Y := (s,t) \mapsto t$$
$$Z := (s,t) \mapsto s^2 + t^2$$

B.0.6
Test functions

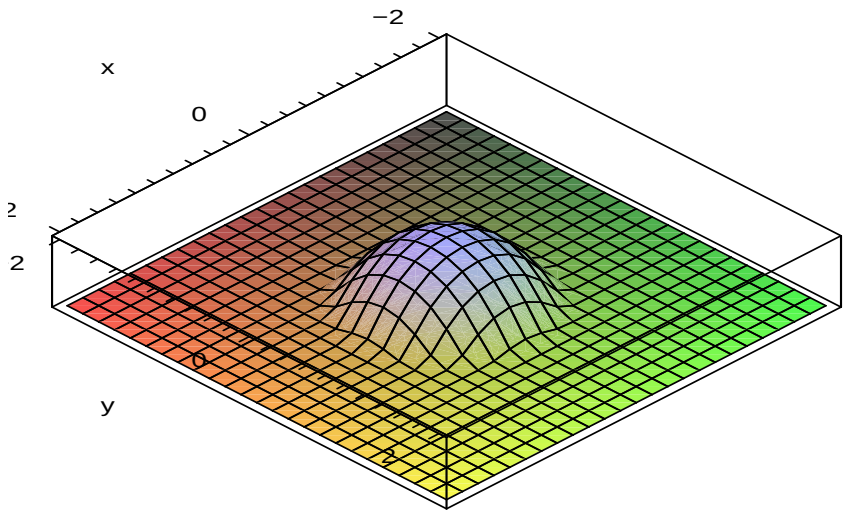
```
> phi_unnorm := (s,t)-> piecewise( (s^2) + (t^2) <1, exp(-(s^2 +
t^2)/(1-s^2 - t^2)), 0 );
```

$$phi\_unnorm := (s,t) \mapsto \begin{cases} e^{-\frac{s^2+t^2}{1-s^2-t^2}} & s^2+t^2<1 \\ 0 & otherwise \end{cases}$$

```
> Digits_cpy := Digits :
> Digits := max( Digits, 10 ) :
> norm_phi := evalf( Int(Int( phi_unnorm(s,t), s=-1..1), t=-1..1
) ) ;
> Digits := Digits_cpy :
```

$$norm\_phi := 1.268112161$$

```
> phi := (s,t) -> phi_unnorm(s,t) / norm_phi :
> plot3d( phi_unnorm(x,y), x=xm..xM, y=ym..yM, scaling=constrained,
axes=boxed ) ;
```




```
> TFonction := proc( a,m,n )
>   local A, f, fe, dfeu, dfev, ddfeu, ddfev, ddfeuv ;
>   global phi, epsilon, trans ;
>
>   A := sum( a[i], i=1..nops(a) ) ;
>
>   f    := simplify( (s,t) -> sum( a[i]*phi(s+m[i]*trans,
t+n[i]*trans), i=1..nops(a) ) / A ) ;
>   fe   := (s,t) -> f(t/epsilon, s/epsilon) / epsilon ;
>   dfeu  := simplify( (s,t) -> subs( u=s, v=t, diff(  fe(u,v),u
) ) ) ;
>   dfev  := simplify( (s,t) -> subs( u=s, v=t, diff(  fe(u,v),v
) ) ) ;
>   ddfeu := simplify( (s,t) -> subs( u=s, v=t, diff( fe(u,v),u,u
) ) ) ;
>   ddfev := simplify( (s,t) -> subs( u=s, v=t, diff( fe(u,v),v,v
) ) ) ;
>   ddfeuv := simplify( (s,t) -> subs( u=s, v=t, diff( fe(u,v),u,v
) ) ) ;
>
>   print( plot3d( [ epsilon * fe ( epsilon * x, epsilon * y )
], x=-2..2, y=-2..2, scaling=constrained ) ) ;
> return fe, dfeu, dfev, ddfeu, ddfev, ddfeuv ;
> end proc :
```

B.0.7
Convolutions

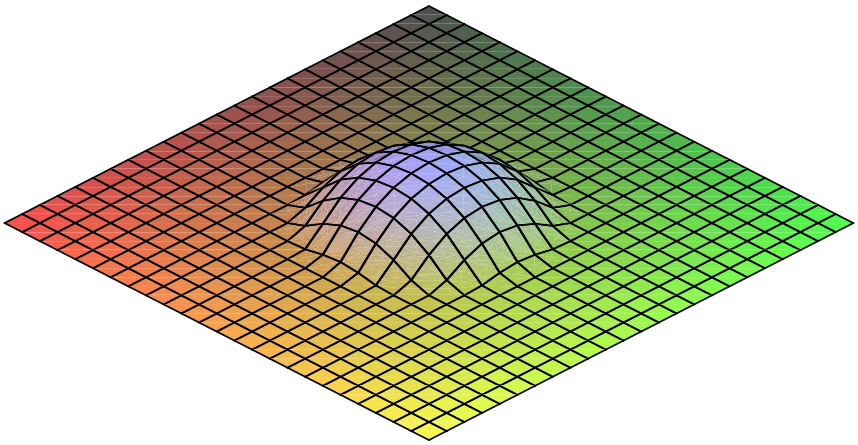
```
> convole := proc( phi, dphiu, dphiv, ddphiu, ddphiv, ddphiuv,
q, p )
>   global X,Y,Z, trans, xm,xM,ym,yM, epsilon ;
>   local x,y,z, dxu,dxv,dyu,dyv,dzu,dzv, ddxu,ddxv,ddxuv,ddyu,ddyv,ddyuv,ddzu,ddzv
x0,y0,z0, dxu0,dxv0,dyu0,dyv0,dzu0,dzv0, ddxu0,ddxv0,ddxuv0,ddyu0,ddyv0,ddyuv0,ddzu0
tg, tm, tM, crv ;
>
>   tm     := epsilon * (-1.0-trans) ;
>   tM     := epsilon * ( 1.0+trans) ;
>
>   x      := (u,v) -> evalf( int(int( X(u-s,v-t) * phi  (s,t),
s=tm..tM), t=tm..tM ) ) ;
>   y      := (u,v) -> evalf( int(int( Y(u-s,v-t) * phi  (s,t),
s=tm..tM), t=tm..tM ) ) ;
>   z      := (u,v) -> evalf( int(int( Z(u-s,v-t) * phi  (s,t),
s=tm..tM), t=tm..tM ) ) ;
>
>   dxu    := (u,v) -> evalf( int(int( X(u-s,v-t) * dphiu
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   dxv    := (u,v) -> evalf( int(int( X(u-s,v-t) * dphiv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   dyu    := (u,v) -> evalf( int(int( Y(u-s,v-t) * dphiu
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   dyv    := (u,v) -> evalf( int(int( Y(u-s,v-t) * dphiv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   dzu    := (u,v) -> evalf( int(int( Z(u-s,v-t) * dphiu
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   dzv    := (u,v) -> evalf( int(int( Z(u-s,v-t) * dphiv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>
>   ddxu   := (u,v) -> evalf( int(int( X(u-s,v-t) * ddphiu
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   ddxv   := (u,v) -> evalf( int(int( X(u-s,v-t) * ddphiv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   ddxuv  := (u,v) -> evalf( int(int( X(u-s,v-t) * ddphiuv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   ddyu   := (u,v) -> evalf( int(int( Y(u-s,v-t) * ddphiu
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   ddyv   := (u,v) -> evalf( int(int( Y(u-s,v-t) * ddphiv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   ddyuv  := (u,v) -> evalf( int(int( Y(u-s,v-t) * ddphiuv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   ddzu   := (u,v) -> evalf( int(int( Z(u-s,v-t) * ddphiu
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   ddzv   := (u,v) -> evalf( int(int( Z(u-s,v-t) * ddphiv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>   ddzuv  := (u,v) -> evalf( int(int( Z(u-s,v-t) * ddphiuv
(s,t), s=tm..tM), t=tm..tM ) ) ;
>
```

```
>    x0     := x  (q,p) ;
>    y0     := y  (q,p) ;
>    z0     := z  (q,p) ;
>
>    dxu0   := dxu (q,p) ;
>    dxv0   := dxv (q,p) ;
>    dyu0   := dyu (q,p) ;
>    dyv0   := dyv (q,p) ;
>    dzu0   := dzu (q,p) ;
>    dzv0   := dzv (q,p) ;
>
>    ddxu0  := ddxu(q,p) ;
>    ddxv0  := ddxv(q,p) ;
>    ddxuv0 := ddxuv(q,p) ;
>    ddyu0  := ddyu(q,p) ;
>    ddyv0  := ddyv(q,p) ;
>    ddyuv0 := ddyuv(q,p) ;
>    ddzu0  := ddzu(q,p) ;
>    ddzv0  := ddzv(q,p) ;
>    ddzuv0 := ddzuv(q,p) ;
>
>    tg     := linalg[matrix](2,2,[evalf (dzu0 / dxu0), evalf(dzv0
/ dxv0), evalf(dzu0 / dyu0), evalf(dzv0 / dyv0)]);
>
>    crv    := linalg[matrix](3,3,[ddxu0, ddxuv0, ddxv0, ddyu0,
ddyuv0, ddyv0,ddzu0, ddzuv0, ddzv0]);
>
>
>    print( "At ", x0,y0,z0, " tg=", tg , "crv=", crv) ;
>
>   end proc :
```

B.0.8
Results

Basic test function

phie, dphieu, dphiev, ddphieu, ddphiev, ddphieuv := TFonction([1],[0],[0]) ; convole(phie, dphieu, dphiev, ddphieu, ddphiev, ddphieuv, 0, 0) ;

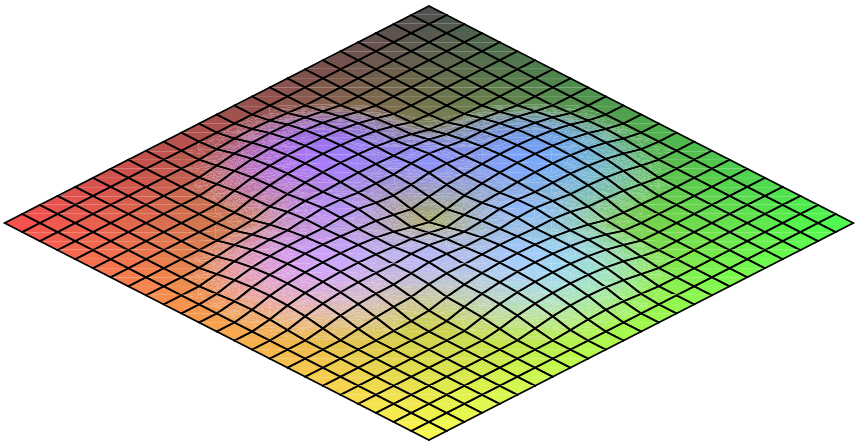


$phie,\ dphieu,\ dphiev,\ ddphieu,\ ddphiev,\ ddphieuv := fe,\ dfeu,\ dfev,\ ddfeu,\ ddfev,\ ddfeuv$

Error, (in evalf/int) unable to convert to pwlist

Positive and axis symmetric test function

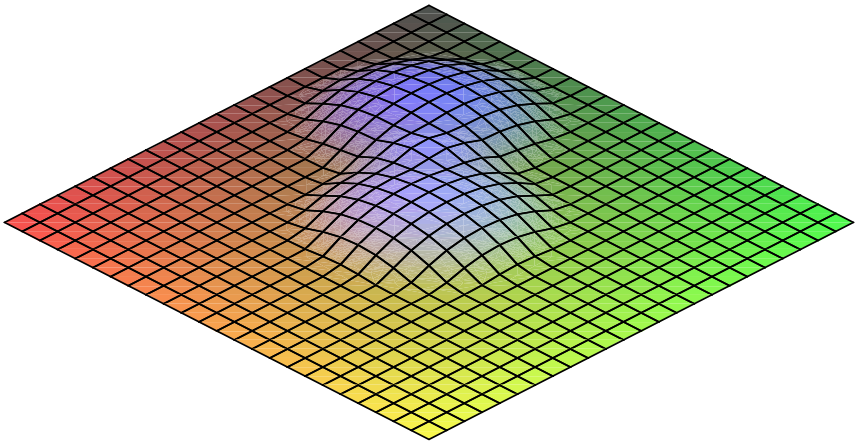
phie_sym, dphieu_sym, dphiev_sym, ddphieu_sym, ddphiev_sym, ddphieuv_sym := TFonction([1,1,1,1],[1,0,-1,0],[0,1,0,-1]) ; convole(phie_2sym, dphie_2sym, ddphie_2sym,phie_2sym, dphie_2sym, ddphie_2sym,0, 0) ;



$e\_sym,\ dphieu\_sym,\ dphiev\_sym,\ ddphieu\_sym,\ ddphiev\_sym,\ ddphieuv\_sym := fe,\ dfeu,\ dfev,\ ddfeu,\ ddfev,\ ddfeuv$

Positive and non-symmetric test function

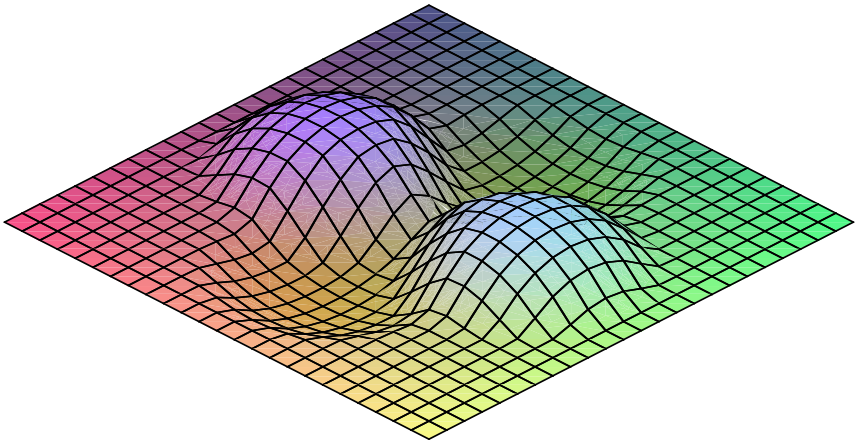
phie_dis, dphieu_dis, dphiev_dis, ddphieu_dis, ddphiev_dis, ddphieuv_dis := TFonction([1,1], [0,+1], [0,1]) : convole(phie_dis, dphieu_dis, dphiev_dis, ddphieu_dis, ddphiev_dis, ddphieuv_dis, 0, 0) ;



Error, (in evalf/int) unable to convert to pwlist

Non-positive and axis symmetric test function

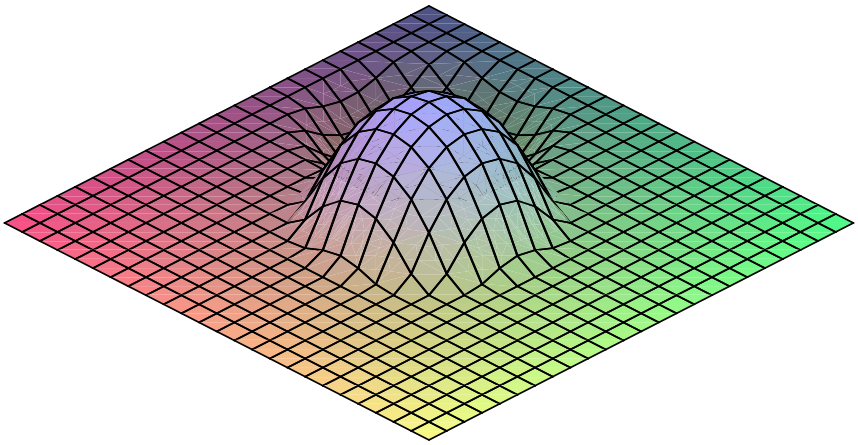
phie_nsym, dphieu_nsym, dphiev_nsym, ddphieu_nsym, ddphiev_nsym, ddphieuv_nsym := TFonction([2,2,-1,-1],[1,-1,0,0],[0,0,1,-1]) ; convole(phie_nsym, dphieu_nsym, dphiev_nsym, ddphieu_nsym, ddphiev_nsym, ddphieuv_nsym,0, 0) ;



Error, (in evalf/int) unable to convert to pwlist

Non positive and non-symmetric test function

phie_ndis, dphieu_ndis, dphiev_ndis, ddphieu_ndis, ddphiev_ndis, ddphieuv_ndis := TFonction([2,-1], [0,+1],[0,1]) ; convole(phie_ndis, dphieu_ndis, dphiev_ndis, ddphieu_ndis, ddphiev_ndis, ddphieuv_ndis,0, 0) ;



e_ndis hieu_ndis, dphiev_ndis, ddphieu_ndis, ddphiev_ndis, ddphieuv_ndis := fe, dfeu, dfev, ddfeu, ddfev, ddfeuv

Error, (in evalf/int) unable to convert to pwlist

>