```
> with(plots):Digits:=100:interface(displayprecision=10):with
  (linalg):


>
  N:=3;

  bb:=vector(N+1,[]):#for printing only = b[]
  beta:=vector(N,[]):

  alpha:=vector(N,[]):
  gamm=vector(N,[]):#heiths of lower ends of hanging branches
                          # alpha[i]+gamma[i]<1   !!!!!!!
                          #if gamma[i]>0

  alpha[1]:=1:beta[1]:=2.4:gamm[1]:=0:

  alpha[2]:=0.4:gamm[2]:=(1-alpha[2])/2:#
  #alpha[3]:=1:beta[3]:=2.6:gamm[3]:=0.0:  #
  #alpha[4]:=0.3:beta[4]:=2.6:gamm[4]:=0.6:
  #alpha[5]:=1.0:beta[5]:=9:gamm[5]:=0:
  #alpha[6]:=0.6:beta[6]:=7:gamm[6]:=0.2:#
  #alpha[7]:=1.0:beta[7]:=6:gamm[7]:=0.0:  #
  alpha[N]:=1:  beta[N]:=2.4:              gamm[N]:=0.0:
  beta[2]:=alpha[2]/(1-alpha[1]/beta[1]-alpha[3]/beta[3]);
  i:='i':beta[N]:=alpha[N]/(1-sum(alpha[i]/beta[i],i=1..N-1));
  if beta[N] < 0 then print(`ERROR - make beta's larger `) fi;

  print(`alpha =`,alpha);
  print(`beta  =`,beta);
  print(`gamma =`,gamm);
  i:='i':
  beta_const:=sum(alpha[i],i=1..N);
  i:='i':
  #for j from 1 to N do
  #beta[j]:=beta_const;
  #od:


  b[1]:=0:
  for j from 1 to N do
  b[j+1]:=b[j]+alpha[j]/beta[j]:
   od:  i:='i':
  b[N+1]:=1:
  ag:=vector(N,[]):
  al:=vector(N,[]):
  a:=vector(N,[]):
```

```
   c:=vector(N,[]):
   for j from 1 to N do
   bb[j]:=b[j];
   ag[j]:=beta[j]*b[j];
   al[j]:=-1+beta[j]*b[j+1];
   od:
   bb[N+1]:=1:
   for j from 1 to N do
   a[j]:=ag[j]-gamm[j];
    od:

   print(`b =`,bb);
   print(`ag =`,ag);
   print(`al =`,al);
   print(`a =`,a);
   print(`gamma =`,gamm);
>
>
> # ag shows maximal digit (greedy)
   #  al shows minimal digit (lazy)  #### if ag[j]=al[j] then  j is
   onto branch and there is
                                                                  #
   no choice there
   # a shows digits assigned automatically using the vector U: U(j)
   =1 lazy
   #                                                          U(j)=
   0 greedy
   # we can assign digit a arbitrarily between minimum and maximum
   and then put 2 into vector U



   # Now we will name points c[i] (there is KK + number of 2's in U
   points c[i])
   # and create a vectors sidec[], ineqc[],signc[] which shows the
   character of the point c[i]
   Kc:=0:# new number of c points
   for j from 1 to N do if alpha[j]<1 then Kc:=Kc+1 fi od:
   for j from 1 to N do if (gamm[j]>0 and alpha[j]+gamm[j]<1) then
   Kc:=Kc+1 fi od:
   print(`Kc =`,Kc);
   c:=vector(2*N,[]):
   sidec:=vector(2*N,[]):# 1 left (use uT), 0 right (use T)
   j_of_c:=vector(2*N,[]):# shows the index of the interval
   associated with c

   cj:=1:# this is the new index for c points
```

```
for j from 1 to N do
if (alpha[j]<1 and gamm[j]=0) then   c[cj]:=b[j+1]; sidec[cj]:=
0;j_of_c[cj]:=j;cj:=cj+1 fi;
if (alpha[j]<1 and gamm[j]+alpha[j]=1) then  c[cj]:=b[j]; sidec
[cj]:=1;j_of_c[cj]:=j;cj:=cj+1 fi;
if (gamm[j]>0 and gamm[j]+alpha[j]<1) then  c[cj]:=b[j]; sidec
[cj]:=1;j_of_c[cj]:=j;cj:=cj+1 ;
                      c[cj]:=b[j+1]; sidec[cj]:=0;j_of_c[cj]:=j;
cj:=cj+1 fi;




od:
print(`c =`,c);
print(`sidec =`,sidec);
print(`j_of_c =`,j_of_c);


>
>


>
uint_of_x:=x->piecewise(x<b[2],1,# This function needs additions
by hand for
                                  # N>9 . Automathic procedure
causes plotting problems
                                  # but is used in other
programs
                        x<b[3],2,
                        x<b[4],3,
                        x<b[5],4,
                        x<b[6],5,
                        x<b[7],6,
                        x<b[8],7,
                        x<b[9],8,
                             9);
int_of_x:=x->piecewise(x<=b[2],1,# This function needs additions
by hand for
                                  # N>9 . Automathic procedure
causes plotting problems
                                  # but is used in other
programs
                        x<=b[3],2,
                        x<=b[4],3,
                        x<=b[5],4,
```

```
                                x<=b[6], 5,
                                x<=b[7], 6,
                                x<=b[8], 7,
                                x<=b[9], 8,
                                      9);
x:='x':
uT:=x->beta[uint_of_x(x)]*x-a[uint_of_x(x)];
T:=x->beta[int_of_x(x)]*x-a[int_of_x(x)];
Tc:=vector(Kc+2,[]):
for j from 1 to Kc do
if sidec[j]=0 then    Tc[j]:=T(c[j]);
              else    Tc[j]:=uT(c[j]) fi;
od:

print(`Tc = `, Tc);

#plot([`uT(x)`, x, 0, 1, Tc[1], Tc[2], Tc[3]], x=0..1, thickness=[2, 1, 1, 1, 1, 1, 1]);
plot([`T(x)`, x, 0, 1, Tc[1], Tc[2]], x=0..1, thickness=[2, 1, 1, 1, 1, 1, 1, 1]);
```

$$N := 3$$

$$\beta_2 := 2.4000000000$$

$$\beta_3 := 2.4000000000$$

$$alpha =, \left[\begin{array}{ccc} 1 & 0.4000000000 & 1 \end{array}\right]$$

$$beta =, \left[\begin{array}{ccc} 2.4000000000 & 2.4000000000 & 2.4000000000 \end{array}\right]$$

$$gamma =, \left[\begin{array}{ccc} 0 & 0.3000000000 & 0.0000000000 \end{array}\right]$$

$$\beta const := 2.4000000000$$

$$b =, \left[\begin{array}{cccc} 0 & 0.4166666667 & 0.5833333333 & 1 \end{array}\right]$$

$$ag =, \left[\begin{array}{ccc} 0.0000000000 & 1.0000000000 & 1.4000000000 \end{array}\right]$$

$$al =, \left[\begin{array}{ccc} 0.0000000000 & 0.4000000000 & 1.4000000000 \end{array}\right]$$

$$a =, \left[\begin{array}{ccc} 0.0000000000 & 0.7000000000 & 1.4000000000 \end{array}\right]$$

$$gamma =, \left[\begin{array}{ccc} 0 & 0.3000000000 & 0.0000000000 \end{array}\right]$$

$$Kc =, 2$$

$$c =, \left[\begin{array}{cccccc} 0.4166666667 & 0.5833333333 & c_3 & c_4 & c_5 & c_6 \end{array}\right]$$

$$sidec =, \left[\begin{array}{cccccc} 1 & 0 & sidec_3 & sidec_4 & sidec_5 & sidec_6 \end{array}\right]$$

$$j\_of\_c = , \left[\begin{array}{cccccc} 2 & 2 & j\_of\_c_3 & j\_of\_c_4 & j\_of\_c_5 & j\_of\_c_6 \end{array} \right]$$

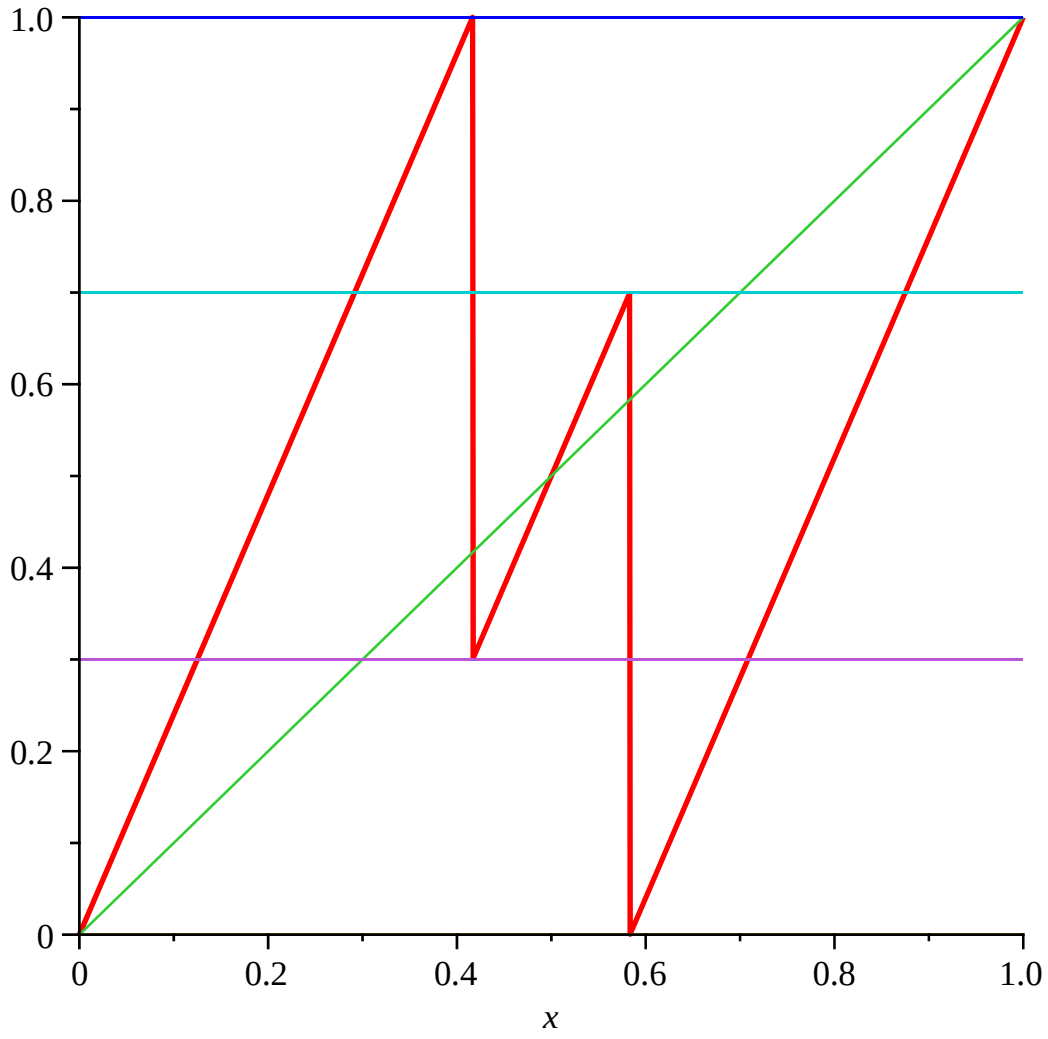
$$uint\_of\_x := x \rightarrow piecewise\left(x < b_2, 1, x < b_3, 2, x < b_4, 3, x < b_5, 4, x < b_6, 5, x < b_7, 6, x < b_8, 7, x < b_9, 8, 9\right)$$

$$int\_of\_x := x \rightarrow piecewise\left(x \leq b_2, 1, x \leq b_3, 2, x \leq b_4, 3, x \leq b_5, 4, x \leq b_6, 5, x \leq b_7, 6, x \leq b_8, 7, x \leq b_9, 8, 9\right)$$

$$uT := x \rightarrow \beta_{uint\_of\_x(x)}\, x - a_{uint\_of\_x(x)}$$

$$T := x \rightarrow \beta_{int\_of\_x(x)}\, x - a_{int\_of\_x(x)}$$

$$Tc = , \left[\begin{array}{cccc} 0.3000000000 & 0.7000000000 & Tc_3 & Tc_4 \end{array} \right]$$

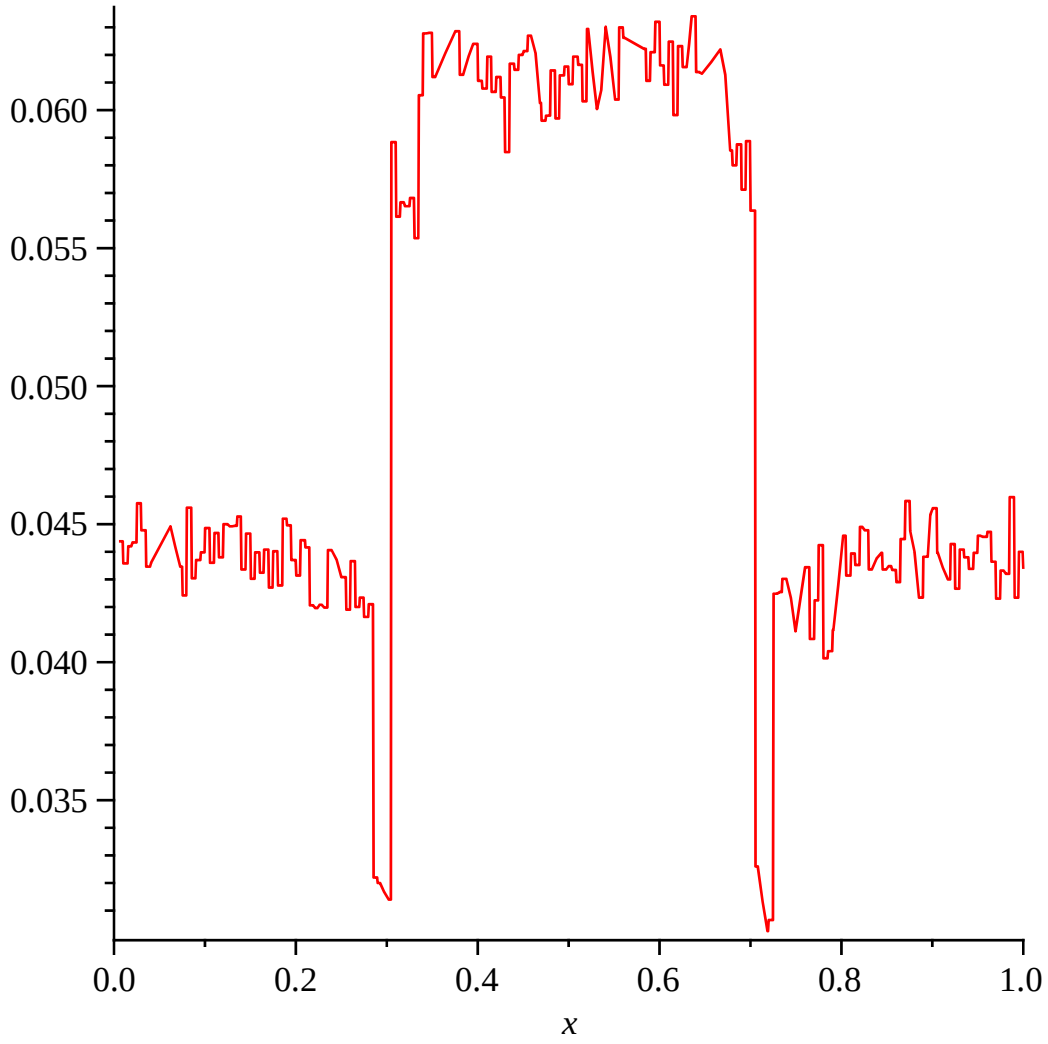



```
> niter:=500000; Nb:=200: v := array(1..Nb):
  AA:=0: BB:=1:
  for i from 1 to Nb do v[i]:=0; od:
  y:=0.2347777777333333337:
  for j from 1 to niter do
       y:=T(y):
       iy:=floor(Nb*(y-AA)/(BB-AA))+1;
       if iy>Nb then iy:=Nb fi;
       v[iy]:=v[iy]+1
```

```
  od:
g:=x->10*v[floor(Nb*x)]/niter;
plot(g(x),x=0..1);
```

$$niter := 500000$$

$$g := x \rightarrow \frac{10\, v_{\mathrm{floor}(Nb\, x)}}{niter}$$




```
>
```

```
> delta1:=(xw,yw)->piecewise(xw<=yw,0,1):
  delta2:=(xw,yw)->piecewise(xw<yw,0,1):
  # symbolic versions of alpha, beta
  alps:=vector(N,[]):
  bets:=vector(N,[]):
  # Procedure producing coefficients in the non simplified
  equations
  Dc:=proc(k,x) local val,ii4;

  if (sidec[k]=0 and k<=Kc) then
              val:=S[k];
            val:=val-sum(delta1(x,Tc[ii4])*SS[k,ii4]*delta1(0.5,
  sidec[ii4])/bets[j_of_c[ii4]],ii4=1..Kc);
            val:=val-sum(delta1(Tc[ii4],x)*SS[k,ii4]*delta1(sidec
```

```
[i i 4], 0.5) / bet s[j _of _c[i i 4]], i i 4=1..Kc);
            val :=val - del t a2( Tc[ k], x) / bet s[ j _of _c[ k]];
end if;
if (si dec[ k]=1 and k<= Kc) then
             val :=S[ k];
            val :=val - sum( del t a1( x, Tc[ i i 4]) * SS[ k, i i 4] * del t a1( 0. 5,
si dec[ i i 4]) / bet s[ j _of _c[ i i 4]], i i 4=1..Kc);
            val :=val - sum( del t a1( Tc[ i i 4], x) * SS[ k, i i 4] * del t a1( si dec
[i i 4], 0.5) / bet s[j _of _c[ i i 4]], i i 4=1..Kc);
            val :=val - del t a2( x, Tc[ k]) / bet s[ j _of _c[ k]];
end if;
if ( k = Kc+1) then
             val :=1- sum( 1/ bet s[ i i 4], i i 4=1..N);
            val :=val +sum( del t a1( x, Tc[ i i 4]) * del t a1( 0. 5, si dec[ i i 4])
/ bet s[ j _of _c[ i i 4]], i i 4=1..Kc);
            val :=val +sum( del t a1( Tc[ i i 4], x) * del t a1( si dec[ i i 4], 0. 5)
/ bet s[ j _of _c[ i i 4]], i i 4=1..Kc);

end if;
            ret urn val;
      end proc;
for k from 1 to Kc+1 do
Dc( k, 0. 6);
od;
#
```

```
Dc := proc(k, x)
    local val, ii4;
    if sidec[k] = 0 and k <= Kc then
        val := S[k];
        val := val − (sum(delta1(x, Tc[ii4]) * SS[k, ii4] * delta1(0.5000000000, sidec[ii4])
        / bets[j_of_c[ii4]], ii4 = 1..Kc));
        val := val − (sum(delta1(Tc[ii4], x) * SS[k, ii4] * delta1(sidec[ii4], 0.5000000000)
        / bets[j_of_c[ii4]], ii4 = 1..Kc));
        val := val − delta2(Tc[k], x) / bets[j_of_c[k]]
    end if;
    if sidec[k] = 1 and k <= Kc then
        val := S[k];
        val := val − (sum(delta1(x, Tc[ii4]) * SS[k, ii4] * delta1(0.5000000000, sidec[ii4])
        / bets[j_of_c[ii4]], ii4 = 1..Kc));
        val := val − (sum(delta1(Tc[ii4], x) * SS[k, ii4] * delta1(sidec[ii4], 0.5000000000)
        / bets[j_of_c[ii4]], ii4 = 1..Kc));
        val := val − delta2(x, Tc[k]) / bets[j_of_c[k]]
    end if;
    if k = Kc + 1 then
```

```
        val := 1 − (sum(1/bets[ii4], ii4 = 1..N));
        val := val + sum(delta1(x, Tc[ii4]) * delta1(0.5000000000, sidec[ii4])/bets[j_of_c[ii4
        ]], ii4 = 1..Kc);
        val := val + sum(delta1(Tc[ii4], x) * delta1(sidec[ii4], 0.5000000000)/bets[j_of_c[ii4
        ]], ii4 = 1..Kc)
    end if;
    return val
end proc
```

$$S_1 - \frac{1}{bets_2}$$

$$S_2 - \frac{1}{bets_2}$$

$$1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} - \frac{1}{bets_3} \qquad \textbf{(1)}$$

```
> # Chosing x's and actually producing the coefficients in the non
  simplified equations ES
  CofD:=matrix(Kc+1,Kc+1,[]):
  xc:=vector(Kc+1,[]):
  Tc[Kc+1]:=0;
  Tc[Kc+2]:=1;
  Tcl:=convert(Tc,list):
  Tc_s:=sort(Tcl,`<`):
  print(Tc_s);

  for i from 1 to Kc+1 do
  x:=(Tc_s[i]+Tc_s[i+1])/2;
  xc[i]:=x;
  for k from 1 to Kc+1 do
  CofD[i,k]:=Dc(k,x);
  od:
  od:
  print(` x's = `,xc);
  print((CofD));

>
```

$$Tc_3 := 0$$

$$Tc_4 := 1$$

$$[0,\ 0.3000000000,\ 0.7000000000,\ 1]$$

$$x's = , \left[\begin{array}{ccc} 0.1500000000 & 0.5000000000 & 0.8500000000 \end{array}\right]$$

(2)

$$\begin{bmatrix} S_1 - \frac{SS_{1,1}}{bets_2} & S_2 - \frac{SS_{2,1}}{bets_2} - \frac{1}{bets_2} & 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_3} \\ S_1 - \frac{1}{bets_2} & S_2 - \frac{1}{bets_2} & 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} - \frac{1}{bets_3} \\ S_1 - \frac{SS_{1,2}}{bets_2} - \frac{1}{bets_2} & S_2 - \frac{SS_{2,2}}{bets_2} & 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_3} \end{bmatrix} \quad (2)$$

```
> #  producing the coefficients in the  simplified equations EQS
  # finding where Tc[i] is (between which xc[i]'s)

  ixc:=vector(Kc,[]):
  for i from 1 to Kc do
  level:=Tc[i];
  for j from 1 to Kc+1 do
  if (sidec[i]=1 and xc[j]<level) then ixc[i]:=j fi;
  if (sidec[i]=0 and xc[j]<=level) then ixc[i]:=j fi;
  od:
  od:
  print(`ixc = `,ixc);

  for i from 1 to Kc+1 do
  vD[i]:=row(CofD,i);
  od:
  for i from 1 to Kc do
  if sidec[i]=0 then nvD[i]:=evalm(vD[ixc[i]+1]-vD[ixc[i]]) fi;
  if sidec[i]=1 then nvD[i]:=evalm(vD[ixc[i]]-vD[ixc[i]+1]) fi;
  od:



  nvD[Kc+1]:=evalm(vD[2]);# NEED TO ADJUSTED BY HAND
  CofD_n:=matrix(Kc+1,Kc+1,[]):
  for i from 1 to Kc+1 do
  for j from 1 to Kc+1 do
  CofD_n[i,j]:=nvD[i][j];
  od;od;
  evalm(CofD_n);
```

$$ixc = , \begin{bmatrix} 1 & 2 \end{bmatrix}$$

$$nvD_3 := \begin{bmatrix} S_1 - \frac{1}{bets_2} & S_2 - \frac{1}{bets_2} & 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} - \frac{1}{bets_3} \end{bmatrix}$$

$$\begin{bmatrix} -\frac{SS_{1,1}}{bets_2}+\frac{1}{bets_2} & -\frac{SS_{2,1}}{bets_2} & \frac{1}{bets_2} \\ -\frac{SS_{1,2}}{bets_2} & -\frac{SS_{2,2}}{bets_2}+\frac{1}{bets_2} & \frac{1}{bets_2} \\ S_1-\frac{1}{bets_2} & S_2-\frac{1}{bets_2} & 1.0000000000-\frac{1}{bets_1}-\frac{1}{bets_2}-\frac{1}{bets_3} \end{bmatrix} \quad (3)$$

>

```
> for i from 1 to Kc+1 do # this part works only after runnig the
  program to the end and returning
  vD[i]:=row(CofD_n,i);            # Then, RUN IT TWICE
  od;
  for i from 1 to Kc do
  if sidec[i]=0 then
                  eta[i]:=1-alpha[j_of_c[i]]-gamm[j_of_c[i]];
                      else
                  eta[i]:=gamm[j_of_c[i]];
  end if;
  print(i,eta[i]);
  od;
  for i from 1 to Kc+1 do
  sD[i]:=eta[1]*vD[1][i]+vD[Kc+1][i];# This line may need to be
  changed by hand
  od;
  for i from 1 to N do
  bets[i]:=beta[i];
  alps[i]:=alpha[i];
  od:

  detCofD_n);
```

>
>
>

$$vD_1 := \begin{bmatrix} -\frac{SS_{1,1}}{bets_2}+\frac{1}{bets_2} & -\frac{SS_{2,1}}{bets_2} & \frac{1}{bets_2} \end{bmatrix}$$

$$vD_2 := \begin{bmatrix} -\frac{SS_{1,2}}{bets_2} & -\frac{SS_{2,2}}{bets_2}+\frac{1}{bets_2} & \frac{1}{bets_2} \end{bmatrix}$$

$$vD_3 := \begin{bmatrix} S_1-\frac{1}{bets_2} & S_2-\frac{1}{bets_2} & 1.0000000000-\frac{1}{bets_1}-\frac{1}{bets_2}-\frac{1}{bets_3} \end{bmatrix}$$

1, 0.3000000000

2, 0.3000000000

$sD_1 := 0.0637482373$

$sD_2 := -0.0637482373$

$sD_3 := -0.1250000000$

$2.177925413\ 10^{-28}$ **(4)**

```
> 
> 
> ud:=vector(50):Digits:=100;NN:=50;#Expansion with variable
  slopes
  d:=vector(50):
  xx:=evalf(rand()/10^12);
  xxt:=xx:
  bet:=1:
  for i from 1 to NN do
  bet:=bet/beta[uint_of_x(xxt)];
  ud[i]:=a[uint_of_x(xxt)];
  udb[i]:=a[uint_of_x(xxt)]*bet;
  xxt:=uT(xxt);
  od:

  xxt:=xx:
  bet:=1:
  for i from 1 to NN do
  bet:=bet/beta[uint_of_x(xxt)];
  d[i]:=a[uint_of_x(xxt)];
  db[i]:=a[uint_of_x(xxt)]*bet;
  xxt:=T(xxt);
  od:
  print(ud);
  uls_it_x:=evalf(sum(udb[j1],j1=1..NN));
  print(d);
  ls_it_x:=evalf(sum(db[j1],j1=1..NN));
  terr:=xx-uls_it_x;
  err:=xx-ls_it_x;
```

$Digits := 100$

$NN := 50$

$xx := 0.3957188605$

[0.0000000000, 1.4000000000, 1.4000000000, 1.4000000000, 0.0000000000, 1.4000000000,
0.0000000000, 1.4000000000, 1.4000000000, 0.0000000000, 0.0000000000,
1.4000000000, 1.4000000000, 1.4000000000, 1.4000000000, 1.4000000000,
0.7000000000, 0.7000000000, 1.4000000000, 0.0000000000, 0.0000000000,
0.7000000000, 1.4000000000, 0.0000000000, 0.0000000000, 1.4000000000,
0.7000000000, 1.4000000000, 0.0000000000, 0.0000000000, 1.4000000000,
0.7000000000, 0.0000000000, 1.4000000000, 1.4000000000, 1.4000000000,

0.0000000000, 0.0000000000, 1.4000000000, 0.0000000000, 1.4000000000,
0.0000000000, 1.4000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 1.4000000000, 1.4000000000, 1.4000000000]

$$uIs\_it\_x := 0.3957188605$$

[0.0000000000, 1.4000000000, 1.4000000000, 1.4000000000, 0.0000000000, 1.4000000000,
0.0000000000, 1.4000000000, 1.4000000000, 0.0000000000, 0.0000000000,
1.4000000000, 1.4000000000, 1.4000000000, 1.4000000000, 1.4000000000,
0.7000000000, 0.7000000000, 1.4000000000, 0.0000000000, 0.0000000000,
0.7000000000, 1.4000000000, 0.0000000000, 0.0000000000, 1.4000000000,
0.7000000000, 1.4000000000, 0.0000000000, 0.0000000000, 1.4000000000,
0.7000000000, 0.0000000000, 1.4000000000, 1.4000000000, 1.4000000000,
0.0000000000, 0.0000000000, 1.4000000000, 0.0000000000, 1.4000000000,
0.0000000000, 1.4000000000, 0.0000000000, 0.0000000000, 0.0000000000,
0.0000000000, 1.4000000000, 1.4000000000, 1.4000000000]

$$Is\_it\_x := 0.3957188605$$

$$terr := 3.295582500\ 10^{-20}$$

$$err := 3.295582500\ 10^{-20} \quad \textbf{(5)}$$

```
>
>
> NN:=70; chi:=(x1,x2,t)->piecewise(t<x1,0,t<=x2,1,0);
  uchi:=(x1,x2,t)->piecewise(t<x1,0,t<x2,1,0);

  #Expansion of c1, c2 ... and all the S's

  for i from 1 to Kc do
  xxt:=c[i];
  bet:=1:
       for n from 1 to NN+1 do
        if sidec[i]=1 then intx:=uint_of_x(xxt) else intx:=
  int_of_x(xxt) fi;
          bet_real:=bet;
          bet:=bet/beta[intx];
          dcb[i,n]:=a[intx]*bet;

         if sidec[i]=0 then
              for ii from 1 to Kc do
                   if xxt>c[ii] then cc[i,ii,n]:=1*bet_real else
  cc[i,ii,n]:=0 fi;
              od;
              if intx=1 then Sc[i,n]:= 0
                            else Sc[i,n]:=sum(1/(beta[j7]),j7=1..
  intx-1)*bet_real fi;
              #bc[i,n]:=b[intx]*bet_real;
```

```
                ################
                        else
                for ii from 1 to Kc do
                     if xxt<c[ii] then cc[i,ii,n]:=1*bet_real else
cc[i,ii,n]:=0 fi;
                od;
                 if intx=N then Sc[i,n]:= 0
                              else Sc[i,n]:=sum(1/(beta[j8]),j8=
intx+1..N)*bet_real fi;
                #bc[i,n]:=b[intx+1]*bet_real;
        fi;
        valc[i,n]:=xxt;
        betc[i,n]:=bet_real;
        if sidec[i]=1 then xxt:=uT(xxt) else xxt:=T(xxt)fi;
        od:
Is_it_x:=sum(dcb[i,j1],j1=1..NN);
od;
for i from 1 to Kc do
S[i]:=evalf(sum(Sc[i,j2+1],j2=1..NN));
#sum_b[i]:=evalf(sum(bc[i,j2+1],j2=1..NN));
od;
for i from 1 to Kc do
for j from 1 to Kc do
SS[i,j]:=evalf(sum(cc[i,j,j1+1],j1=1..NN));

#print(`SS[`,i,j,`] =`,SS[i,j]):
od;od:
#for n from 1 to 20 do

#print(1,2,cc[1,2,n]*beta[1],valc[1,n]);
#print(3,2,cc[3,2,n]*beta[2],valc[3,n]);
#od:
```

$$NN := 70$$

$$\chi := (x1, x2, t) \rightarrow piecewise(t < x1, 0, t \le x2, 1, 0)$$

$$uchi := (x1, x2, t) \rightarrow piecewise(t < x1, 0, t < x2, 1, 0)$$

$$xxt := 0.4166666667$$

$$bet := 1$$

$$Is\_it\_x := 0.4166666667$$

$$xxt := 0.5833333333$$

$$bet := 1$$

$$Is\_it\_x := 0.5833333333$$

$$S_1 := 0.4166666667$$

$$S_2 := 0.4166666667$$

```
>
>
MM:=matrix(Kc,Kc,[]):
MMM:=matrix(Kc,Kc,[]):
for i from 1 to Kc do
for j from 1 to Kc do

MM[i,j]:=-SS[j,i];
MMM[i,j]:=-SS[j,i];
od;od;
print(`MM = `,MM);
print(`det MM = `,det(MM));
print(1/beta[j_of_c[5]]);


print(`eigenvalues MM =`,eigenvalues(MM));
print(`1/ average beta = `,1/ave_beta);
ve:=vector(Kc,[]):
for i from 1 to Kc do
ve[i]:=1;

MMM[i,i]:=MMM[i,i]+1.0;
od:

print(MMM);
print(`det MMM = `,det(MMM));
print(ve);_t:='_t';




DD:=linsolve(MMM,ve);
sum((S[ii7]-1/beta[j_of_c[ii7]])*DD[ii7],ii7=1..Kc)-(1-sum
(1/beta[i8],i8=1..N));
eigenvectors(MM);
```

$$MM = , \begin{bmatrix} -0.4900141016 & -0.5099858984 \\ -0.5099858984 & -0.4900141016 \end{bmatrix}$$

$$det\ MM = , -0.0199717969$$

$$\frac{1}{\beta_{j\_of\_c_5}}$$

$$eigenvalues\ MM =, -1.0000000000, 0.0199717969$$

$$1/\ average\ beta = , \frac{1}{ave\_beta}$$

$$\begin{bmatrix} 0.5099858984 & -0.5099858984 \\ -0.5099858984 & 0.5099858984 \end{bmatrix}$$

$$det\ MMM = , 2.150545779\ 10^{-27}$$

$$\begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$\_t := \_t$$

$$DD := \begin{bmatrix} 4.742850893\ 10^{26} & 4.742850893\ 10^{26} \end{bmatrix}$$

$$-0.5833333333$$

$$\left[-1.0000000000, 1, \left\{\begin{bmatrix} 0.7071067812 & 0.7071067812 \end{bmatrix}\right\}\right], \left[0.0199717969, 1, \left\{\begin{bmatrix} -0.7071067812 & 0.7071067812 \end{bmatrix}\right\}\right] \quad \textbf{(7)}$$

```
> DDv:=vector(2,[0.7071067812,0.7071067812]);
```

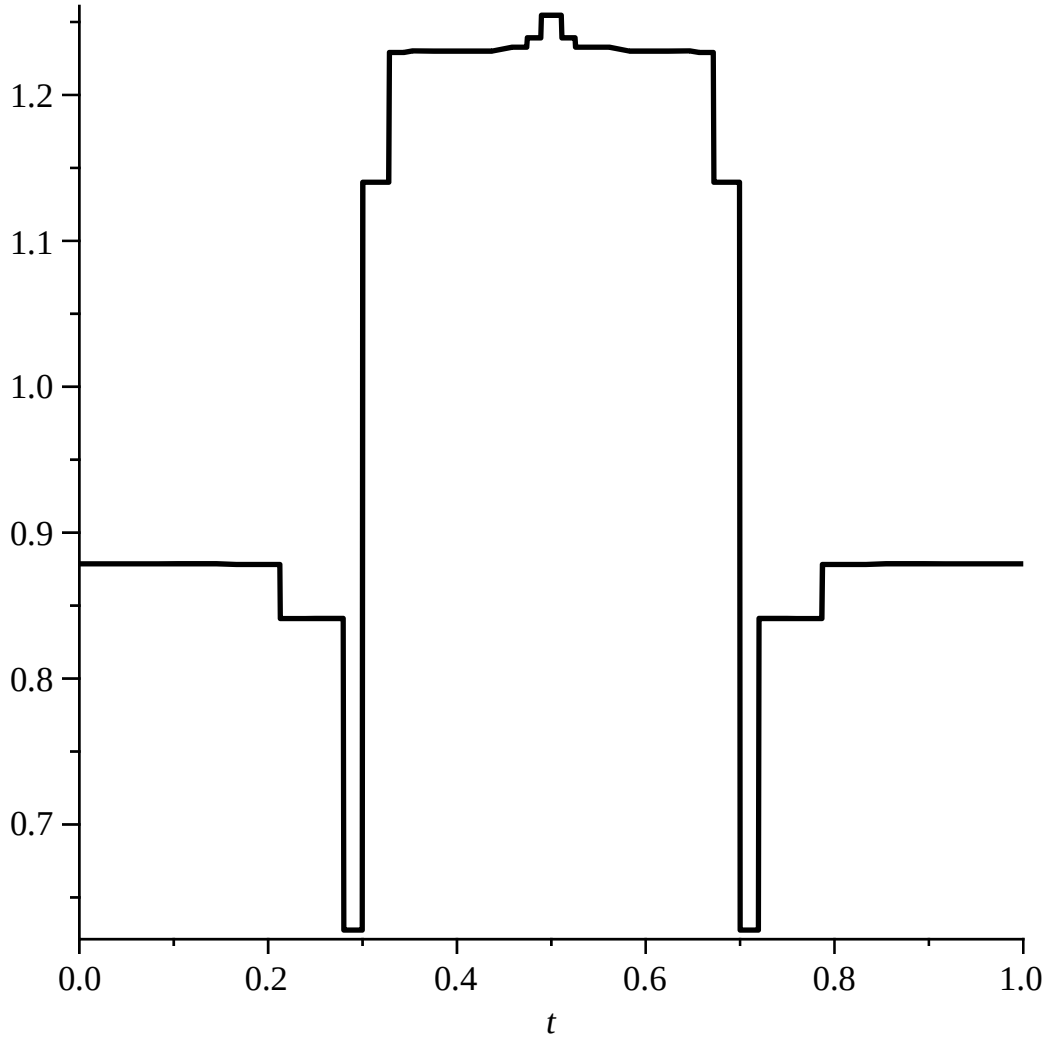
$$DDv := \begin{bmatrix} 0.7071067812 & 0.7071067812 \end{bmatrix} \quad \textbf{(8)}$$

```
>
> density:=proc(t) local j,i, den;
                    i:='i':

                  den:=0:
                  for j from 1 to Kc do
                  if sidec[j]=0 then
                    den:=den+ DDv[j]*sum((chi(0,valc[j,i1+1],t))*
betc[j,i1+1],i1=1..50) fi;

                   if sidec[j]=1 then
                    den:=den+ DDv[j]*sum((uchi(valc[j,i1+1],1,t))*
betc[j,i1+1],i1=1..50) fi;


                   od;
                  return den;
            end proc;

#Normalizing factor

NC:=0:
for j from 1 to Kc do
if sidec[j]=0 then
 NC:=NC+DDv[j]*sum((valc[j,i1+1])*betc[j,i1+1],i1=1..50) fi;
if sidec[j]=1 then
NC:=NC+DDv[j]*sum((1-valc[j,i1+1])*betc[j,i1+1],i1=1..50) fi;

 od:
```

```
print(`NC = `, NC);

plot([(1/NC)*density(t)], t=0..1-0.000001, color=black, thickness=2);
```

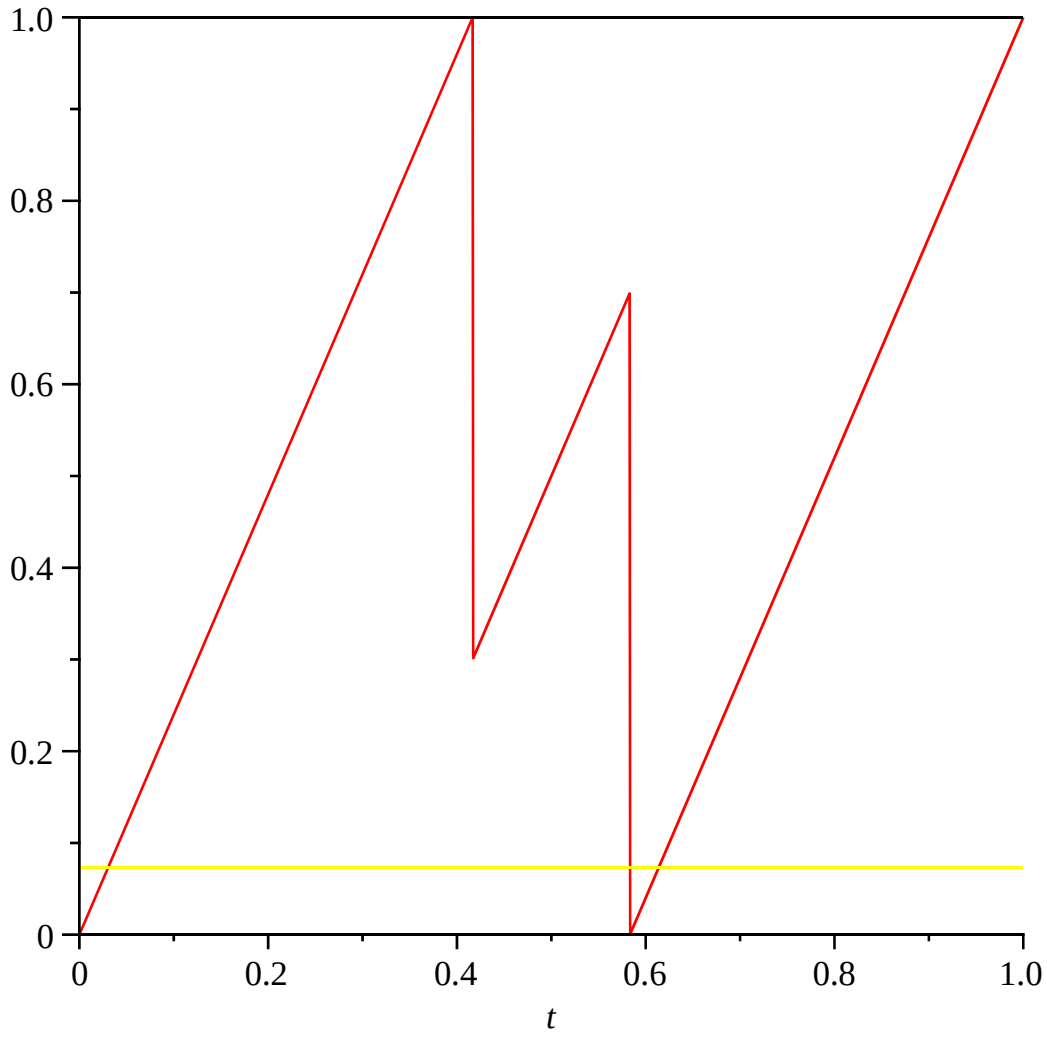
```
density := proc(t)
    local j, i, den;
    i := 'i';
    den := 0;
    for j to Kc do
        if sidec[j] = 0 then
            den := den + DDv[j]*(sum(chi(0, valc[j, i1 + 1], t)*betc[j, i1 + 1], i1 = 1..50))
        end if;
        if sidec[j] = 1 then
            den := den + DDv[j]*(sum(uchi(valc[j, i1 + 1], 1, t)*betc[j, i1 + 1], i1 = 1
            ..50))
        end if
    end do;
    return den
end proc
```

$$NC = , 0.5748648649$$




```
> 
> 
#check density
#preimages
for j6 from 0 to 9 do
y[j6]:=j6/10+(0.1)*rand()/10^12;
od:
for j6 from 0 to 9 do
for i3 from 1 to N do
pre[i3]:=(y[j6]+a[i3])/beta[i3];
od;
plot([T(t),0,1,y[j6]],t=0..1,
color=[red,black,black,yellow]);
su:=0:
for i3 from 1 to N do
if (pre[i3]>=b[i3] and pre[i3]<=b[i3+1]) then
 su:=su+evalf(density(pre[i3])/beta[i3]);
print(i3);
fi;
od;
err[j6]:=evalf(density(y[j6])-su);
od;
```

```
for j6 from 0 to 9 do
print(`y =`,y[j6]);
print(`err[`,j6,`]=`,err[j6]);
od;
```

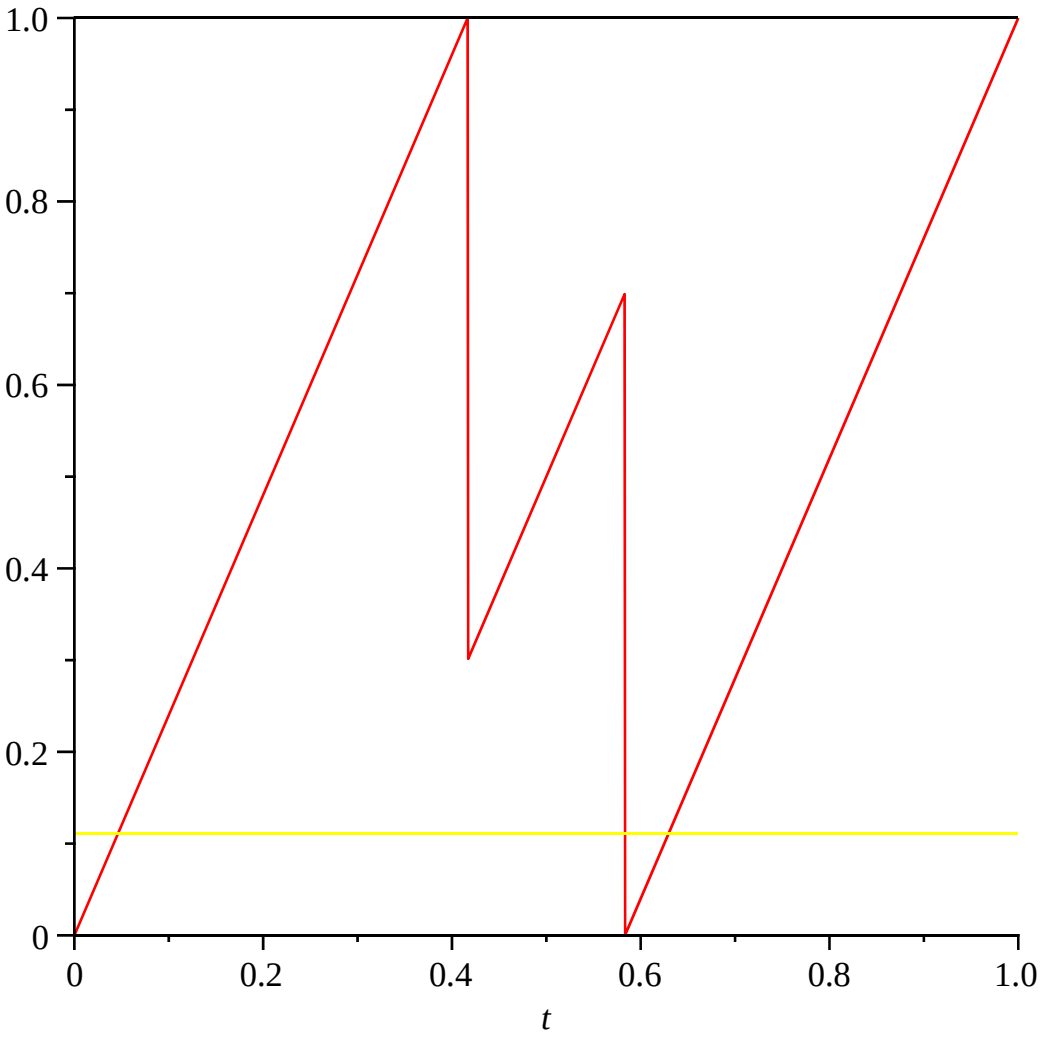


$su := 0$

1

3

$err_0 := 3.384702092\ 10^{-22}$

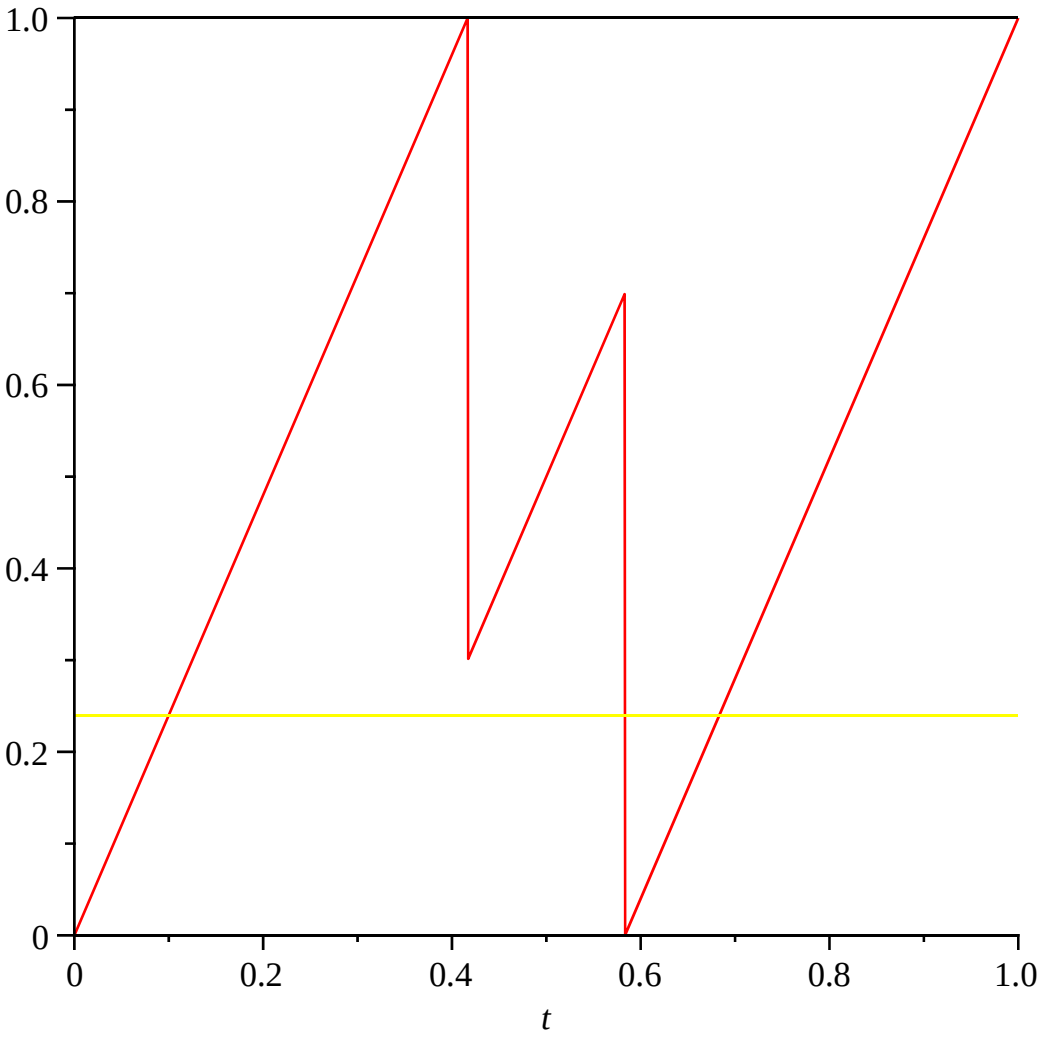


$su := 0$

1

3

$err_1 := 3.384702092\ 10^{-22}$

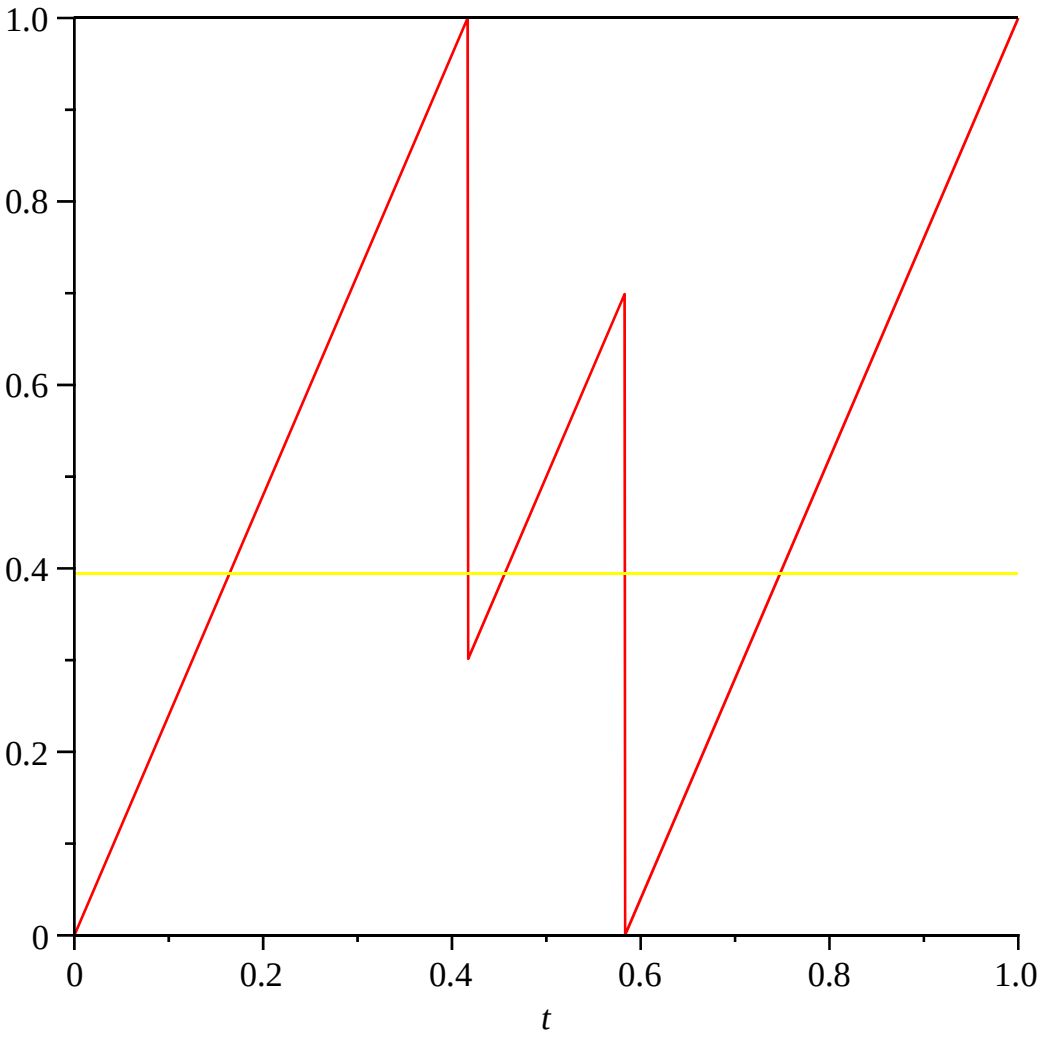


$su := 0$

1

3

$err_2 := 3.384702092\ 10^{-22}$



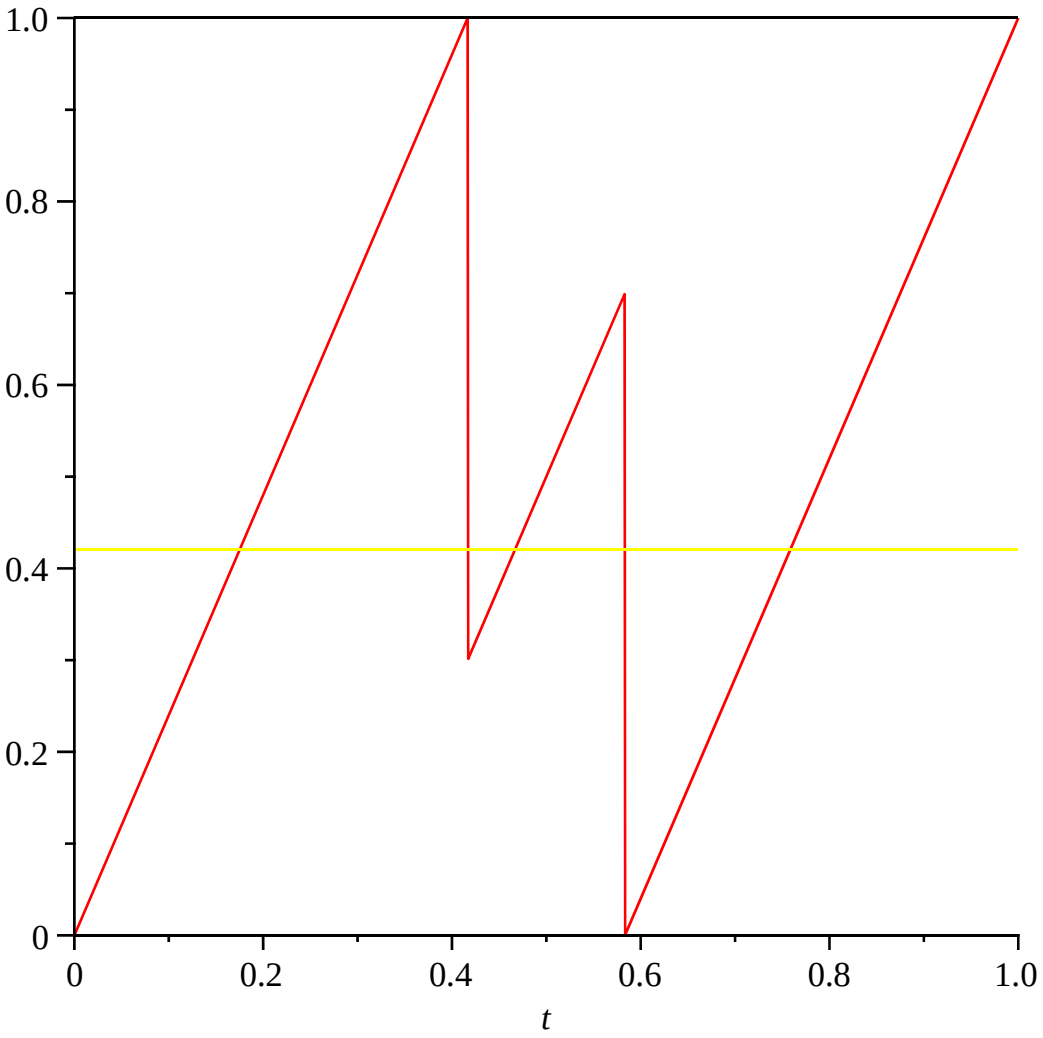
$su := 0$

1

2

3

$err_3 := 6.769404183\ 10^{-22}$



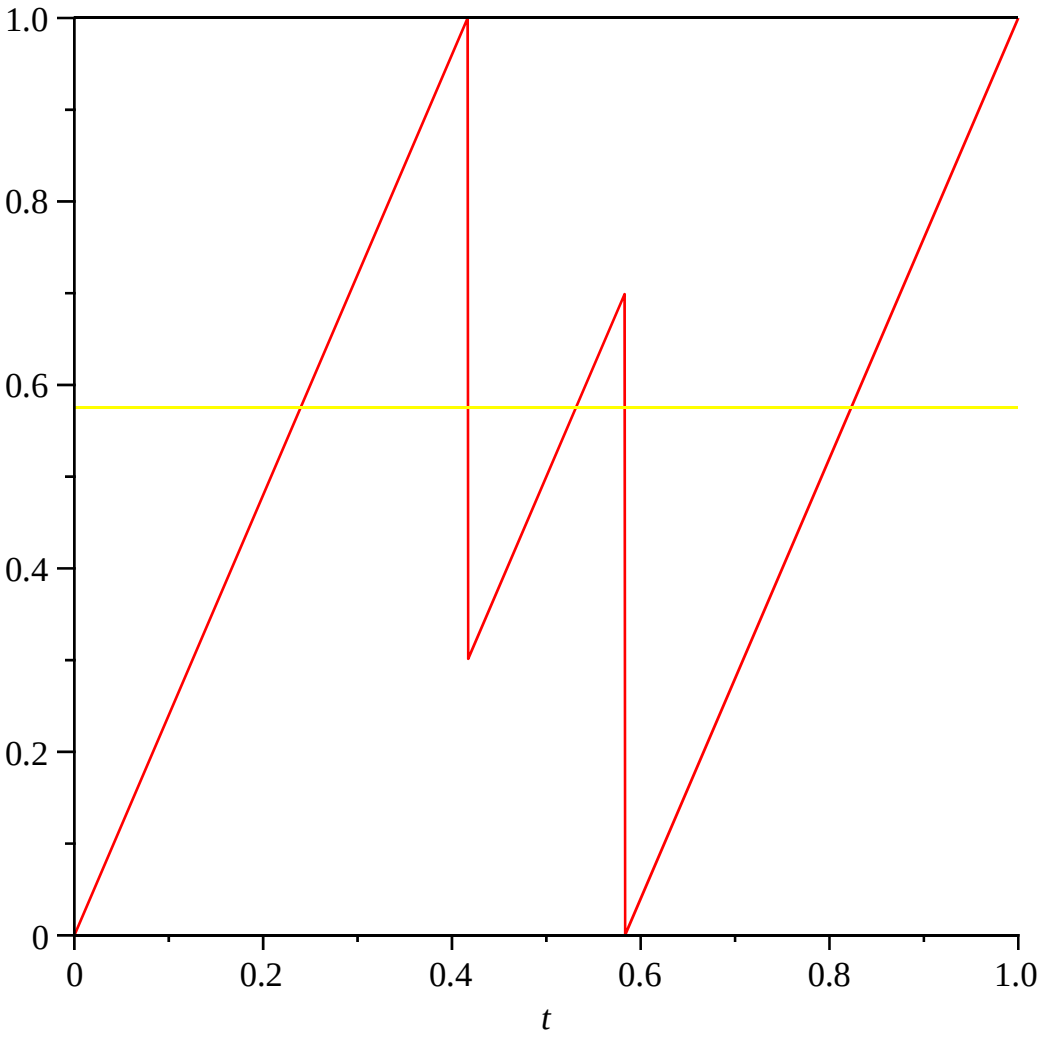
$su := 0$

1

2

3

$err_4 := 6.769404183\ 10^{-22}$



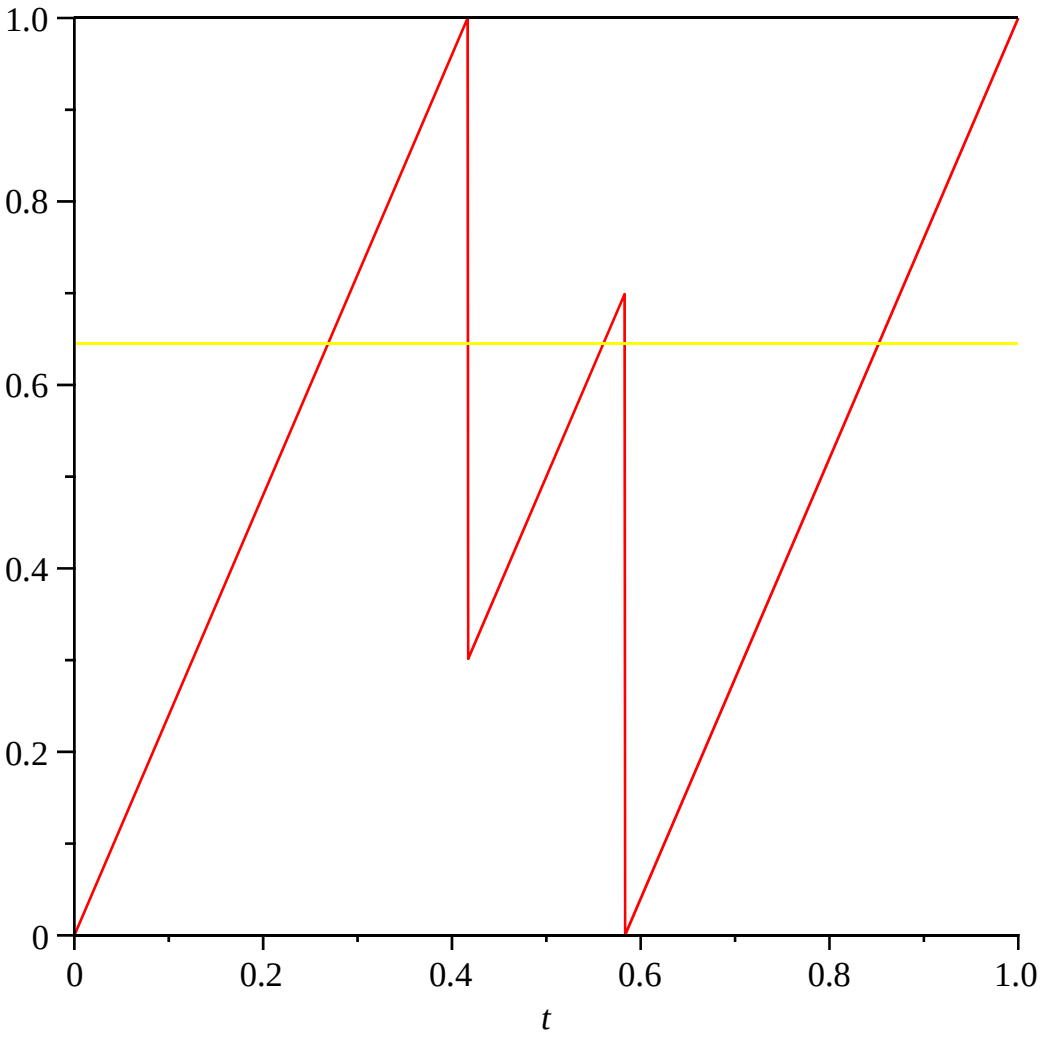
$su := 0$

1

2

3

$err_5 := 6.769404183\ 10^{-22}$



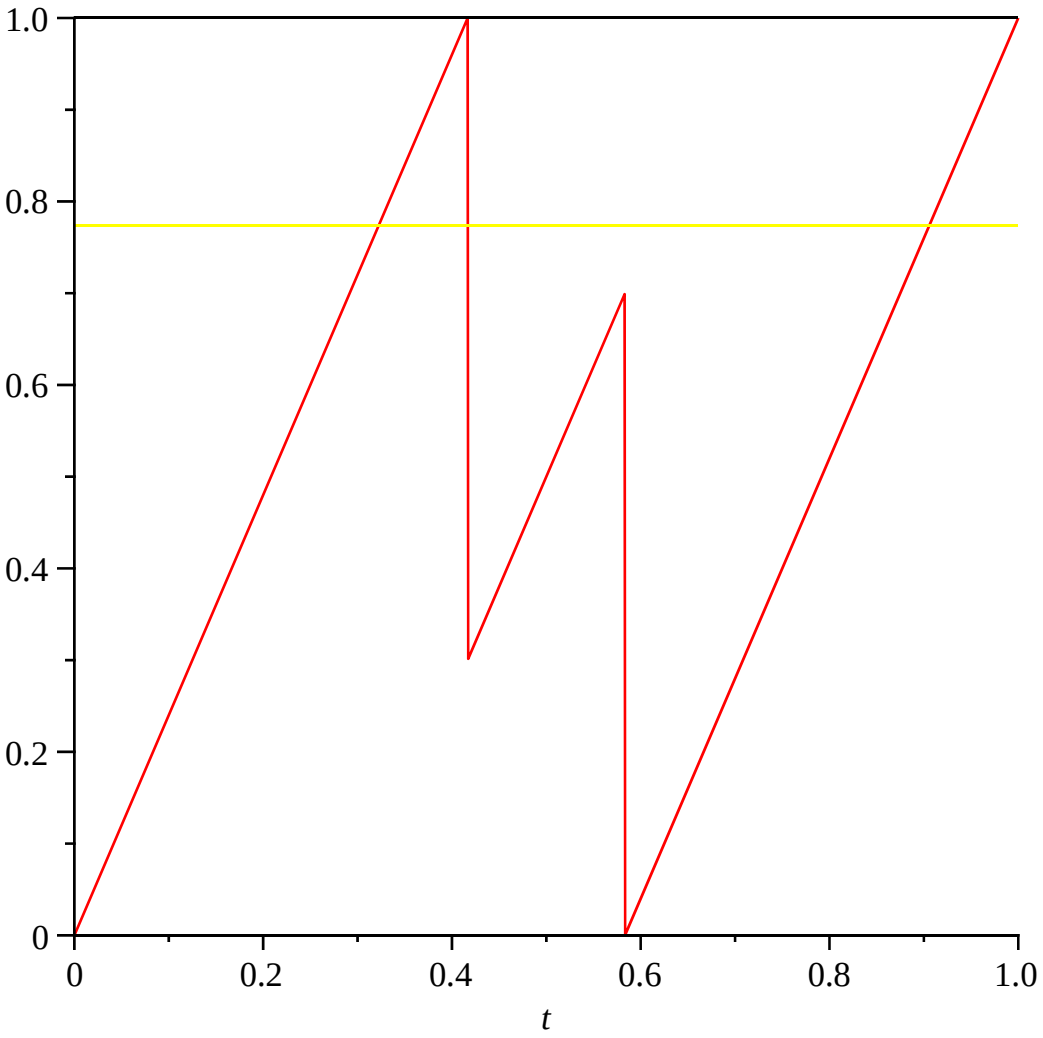
$su := 0$

1

2

3

$err_6 := 6.769404183\ 10^{-22}$

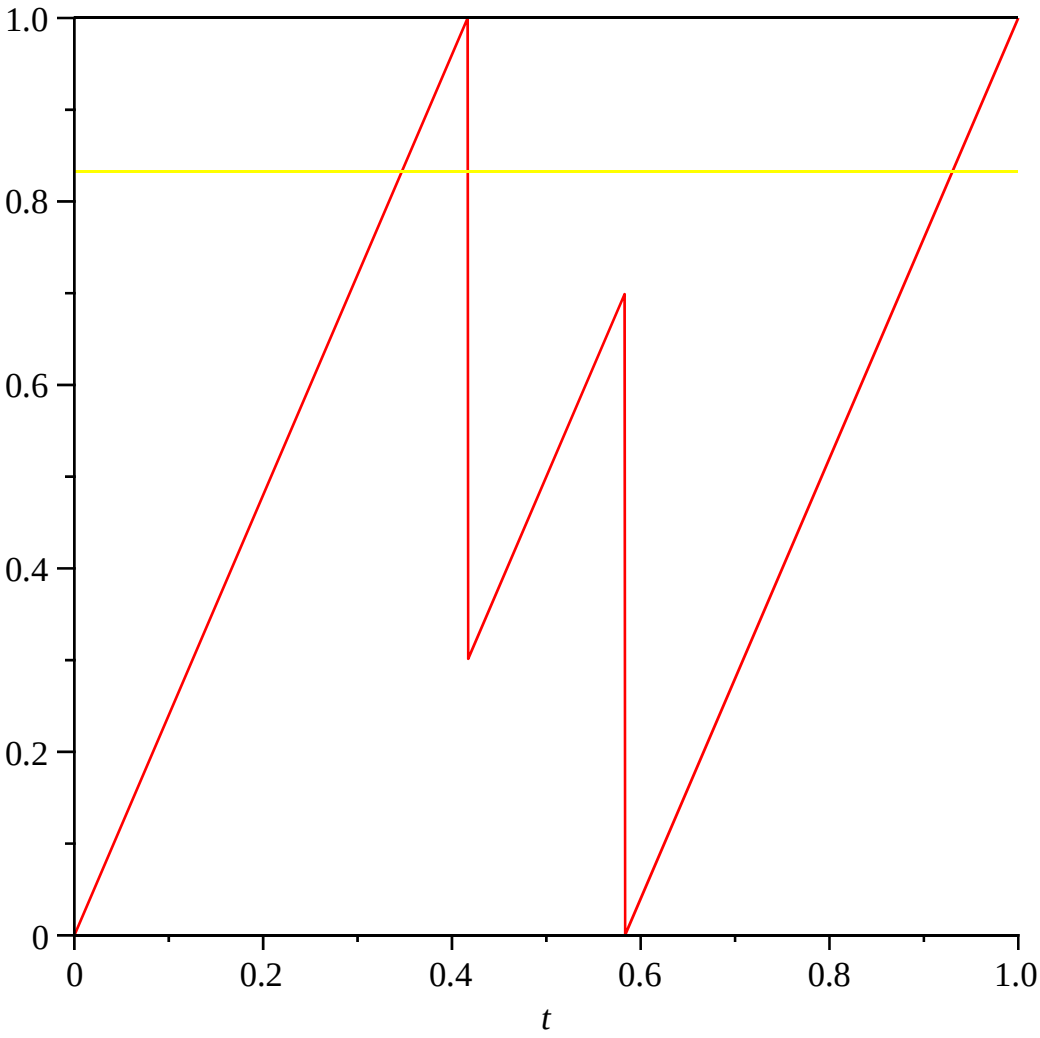


$su := 0$

1

3

$err_7 := 3.384702092\ 10^{-22}$

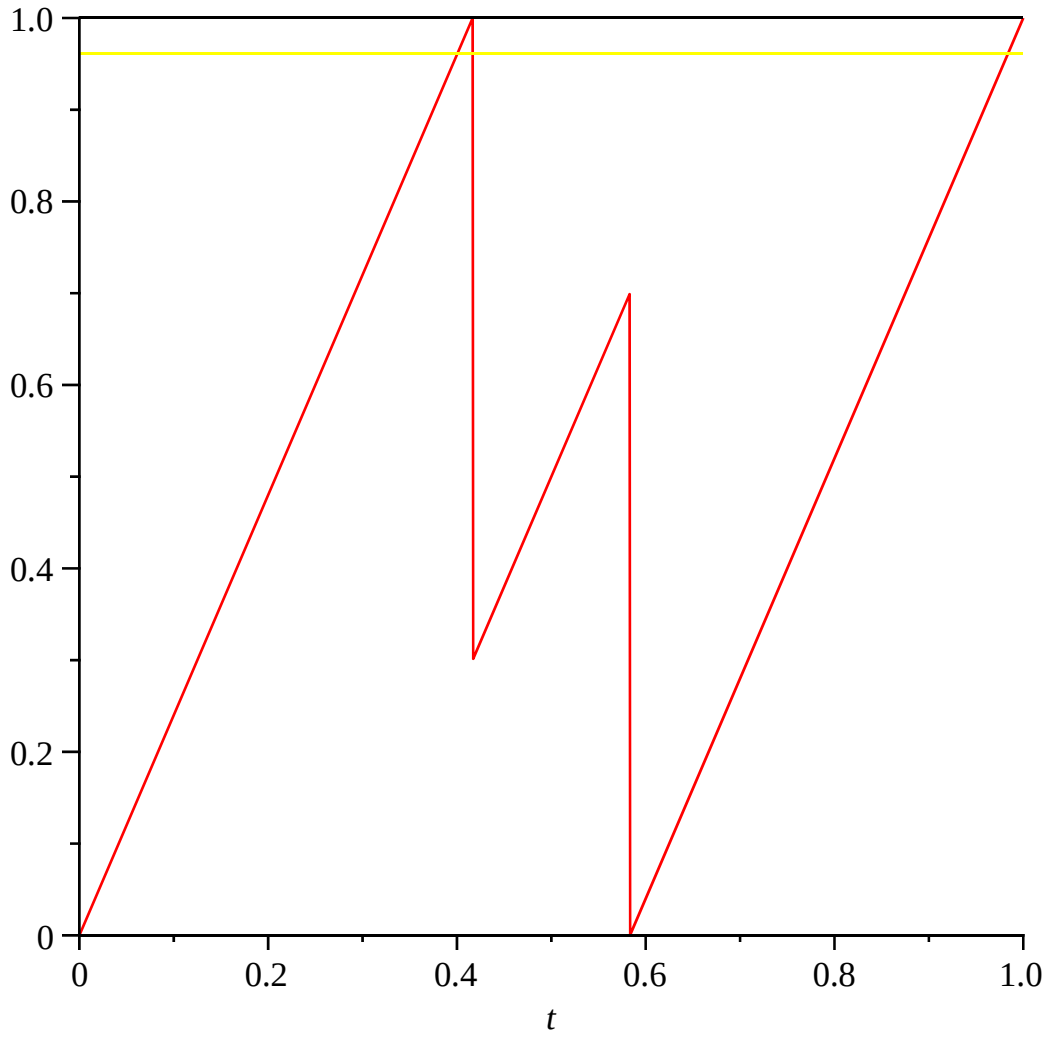


$su := 0$

1

3

$err_8 := 3.384702092\ 10^{-22}$



$su := 0$

1

3

$err_9 := 3.384702092\ 10^{-22}$

$y =, 0.0730616293$

$err[, 0,]=, 3.384702092\ 10^{-22}$

$y =, 0.1106507054$

$err[, 1,]=, 3.384702092\ 10^{-22}$

$y =, 0.2396412723$

$err[, 2,]=, 3.384702092\ 10^{-22}$

$y =, 0.3944913350$

$err[, 3,]=, 6.769404183\ 10^{-22}$

$y =, 0.4210936429$

$err[, 4,]=, 6.769404183\ 10^{-22}$

$y =, 0.5750072072$

$err[, 5,]=, 6.769404183\ 10^{-22}$

$y =, 0.6454744397$

$$err[,\ 6,\ ]=,\ 6.769404183\ 10^{-22}$$

$$y\ =,\ 0.7736602622$$

$$err[,\ 7,\ ]=,\ 3.384702092\ 10^{-22}$$

$$y\ =,\ 0.8329844592$$

$$err[,\ 8,\ ]=,\ 3.384702092\ 10^{-22}$$

$$y\ =,\ 0.9615732069$$

$$err[,\ 9,\ ]=,\ 3.384702092\ 10^{-22}$$

>