

```
> with(plot.s): Digits:=100: interface(dsplypreci=10): with
(Linalg):
```

```
>
```

```
N:=4;
```

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

```
beta:=vector(N,[]):
```

```
alpha:=vector(N,[]):
```

```
gamma:=vector(N,[]): #heights of lower ends of hanging branches
```

```
    # alpha[i]+gamma[i]<1  !!!!!!!
```

```
    #if gamma[i]>0
```

```
alpha[1]:=0.5: beta[1]:=2: gamma[1]:=0.5:
```

```
alpha[2]:=0.5: beta[2]:=2: gamma[2]:=0.5: #
```

```
alpha[3]:=0.5: beta[3]:=2: gamma[3]:=0.0: #
```

```
#alpha[4]:=0.4: beta[4]:=5: gamma[4]:=0.4:
```

```
#alpha[5]:=1.0: beta[5]:=9: gamma[5]:=0:
```

```
#alpha[6]:=0.6: beta[6]:=7: gamma[6]:=0.2: #
```

```
#alpha[7]:=1.0: beta[7]:=6: gamma[7]:=0.0: #
```

```
alpha[N]:=0.5: gamma[N]:=0.0:
```

```
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 =`, gamma);
```

```
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] := bet a[j] * b[j];
al[j] := -1 + bet a[j] * b[j+1];
od:
bb[N+1] := 1:
for j from 1 to N do
a[j] := ag[j] - gam[j];
od:

print(`b =`, bb);
print(`ag =`, ag);
print(`al =`, al);
print(`a =`, a);
print(`gamma =`, gam);
>
>
> # 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 arbitrarily between minimum and maximum
and then put 2 into vector U

# Now we will name points c[i] (there is K + number of 2's in U
points c[i])
# and create a vectors si dec[], 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 (gam[j]>0 and alpha[j]+gam[j]<1) then
Kc:=Kc+1 fi od:
print(`Kc =`, Kc);
c:=vector(2*N, []):
si dec:=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 gam[j]=0) then  c[cj]:=b[j+1]; si dec[cj]:=
0; j_of_c[cj]:=j; cj:=cj+1 fi;
if (alpha[j]<1 and gam[j]+alpha[j]=1) then  c[cj]:=b[j]; si dec
[cj]:=1; j_of_c[cj]:=j; cj:=cj+1 fi;
if (gam[j]>0 and gam[j]+alpha[j]<1) then  c[cj]:=b[j]; si dec
[cj]:=1; j_of_c[cj]:=j; cj:=cj+1 ;
                                c[cj]:=b[j+1]; si dec[cj]:=0; j_of_c[cj]:=j;
cj:=cj+1 fi;

```

```

od:
print(`c =`, c);
print(`si dec =`, si dec);
print(`j_of_c =`, j_of_c);

```

>

>

>

```

ui nt_of_x:=x->piecewise(x<b[2], 1, # This functi on needs addi ti ons
by hand for

```

```

# N>9 . Automat hi c procedur e

```

```

causes plott ing probl ems

```

```

# but i s used i n other

```

```

pr ogr am s

```

```

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);

```

```

i nt_of_x:=x->piecewise(x<=b[2], 1, # This functi on needs addi ti ons
by hand for

```

```

# N>9 . Automat hi c procedur e

```

```

causes plott ing probl ems

```

```

# but i s used i n other

```

```

pr ogr am s

```

```

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->bet a[ ui nt _of _x( x) ] * x- a[ ui nt _of _x( x) ] ;
T: =x->bet a[ i nt _of _x( x) ] * x- a[ i nt _of _x( x) ] ;
Tc: =vect or( Kc+2, [ ] ) :
for j from 1 to Kc do
if si dec[ j ] =0 then Tc[ j ] :=T( c[ j ] ) ;
else Tc[ j ] :=uT( c[ j ] ) fi ;
od:
Tc[ 1 ] :=0. 500000;
Tc[ 2 ] :=0. 50000000;
pri nt ( ` Tc = ` , Tc ) ;

#pl ot ( [ ' uT( x) ' , x, 0, 1, Tc[ 1] , Tc[ 2] , Tc[ 3] ] , x=0. . 1, t hi ckness=[ 2, 1, 1,
1, 1, 1, 1] ) ;
pl ot ( [ ' T( x) ' , x, 0, 1, Tc[ 1] , Tc[ 2] , Tc[ 3] ] , x=0. . 1, t hi ckness=[ 2, 1, 1, 1,
1, 1, 1, 1] ) ;

```

```

N := 4
β4 := 2.0000000000
alpha =, [ 0.5000000000 0.5000000000 0.5000000000 0.5000000000 ]
beta =, [ 2 2 2 2.0000000000 ]
gamma =, [ 0.5000000000 0.5000000000 0.0000000000 0.0000000000 ]
βconst := 2.0000000000
b =, [ 0 0.2500000000 0.5000000000 0.7500000000 1 ]
ag =, [ 0 0.5000000000 1.0000000000 1.5000000000 ]
al =, [ -0.5000000000 0.0000000000 0.5000000000 1.0000000000 ]
a =, [ -0.5000000000 0.0000000000 1.0000000000 1.5000000000 ]
gamma =, [ 0.5000000000 0.5000000000 0.0000000000 0.0000000000 ]
Kc =, 4
c =, [ 0 0.2500000000 0.7500000000 1 c5 c6 c7 c8 ]
sidec =, [ 1 1 0 0 sidec5 sidec6 sidec7 sidec8 ]
j_of_c =, [ 1 2 3 4 j_of_c5 j_of_c6 j_of_c7 j_of_c8 ]

```

$uint_of_x := x \rightarrow \text{piecewise}(x < b_2, 1, x < b_3, 2, x < b_4, 3, x < b_5, 4, x < b_6, 5, x < b_7, 6, x < b_8, 7, x < b_9, 8, 9)$

$int_of_x := x \rightarrow \text{piecewise}(x \leq b_2, 1, x \leq b_3, 2, x \leq b_4, 3, x \leq b_5, 4, x \leq b_6, 5, x \leq b_7, 6, x \leq b_8, 7, x \leq b_9, 8, 9)$

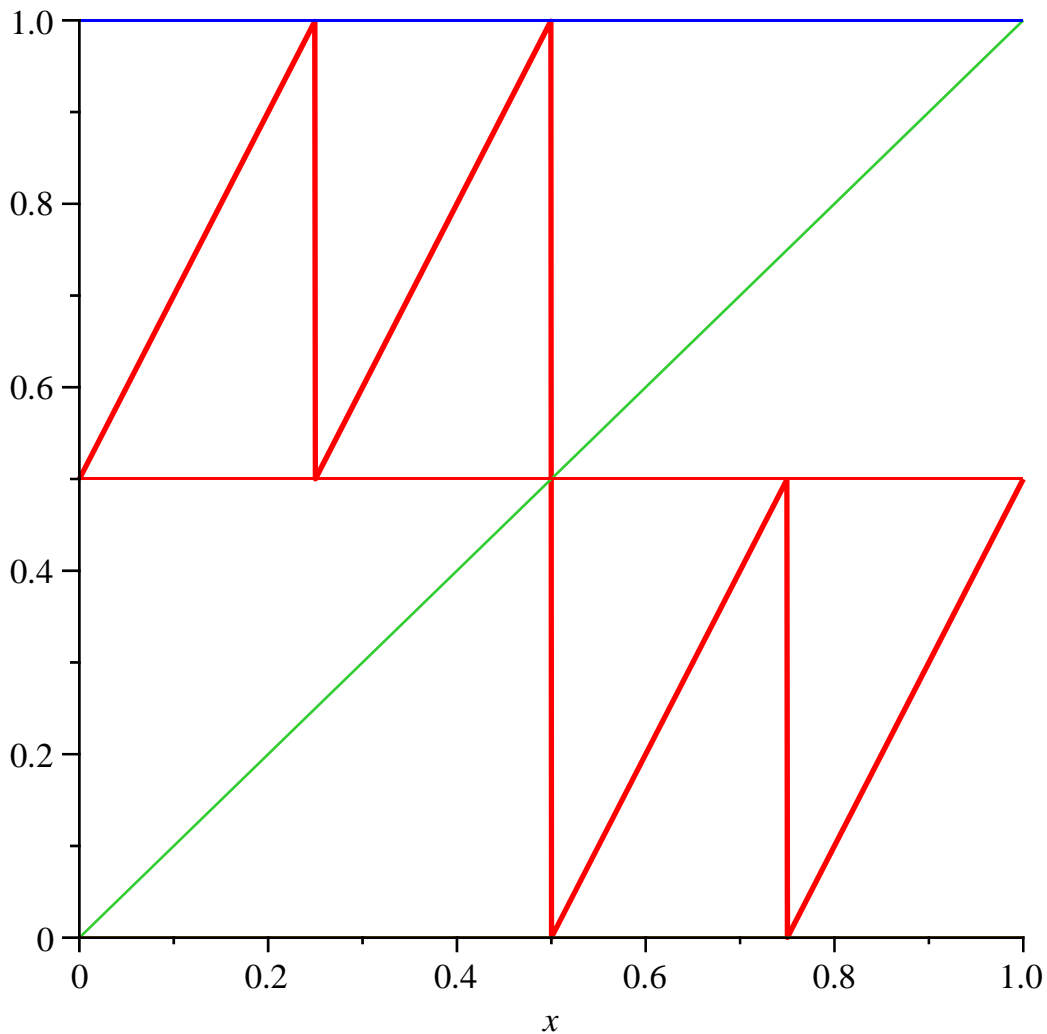
$uT := x \rightarrow \beta_{uint_of_x(x)} x - a_{uint_of_x(x)}$

$T := x \rightarrow \beta_{int_of_x(x)} x - a_{int_of_x(x)}$

$Tc_1 := 0.5000000000$

$Tc_2 := 0.5000000000$

$Tc = , [0.5000000000 0.5000000000 0.5000000000 0.5000000000 Tc_5 Tc_6]$



>

> del ta1:=(xw,yw)->piecewise(xw=yw,0,1):

del ta2:=(xw,yw)->piecewise(xw<yw,0,1):

symbolic versions of alpha, beta

alph:=vector(N,[]):

betas:=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[ii4], 0.5) / 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(0.5, sidec[ii4]) / bets[j_of_c[ii4]], ii4=1..Kc);
```

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

```
val:=val + sum(delta1(Tc[ii4], x) * delta1(sidec[ii4], 0.5) / bets[j_of_c[ii4]], ii4=1..Kc);
```

```
end if;
```

```
return 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

$$\begin{aligned}
S_1 &= \frac{SS_{1,3}}{bets_3} - \frac{SS_{1,4}}{bets_4} - \frac{1}{bets_1} \\
S_2 &= \frac{SS_{2,3}}{bets_3} - \frac{SS_{2,4}}{bets_4} - \frac{1}{bets_2} \\
S_3 &= \frac{SS_{3,3}}{bets_3} - \frac{SS_{3,4}}{bets_4} \\
S_4 &= \frac{SS_{4,3}}{bets_3} - \frac{SS_{4,4}}{bets_4} \\
1.0000000000 &= \frac{1}{bets_1} - \frac{1}{bets_2}
\end{aligned} \tag{1}$$

> # Chosing x's and actually producing the coefficients in the non simplified equations ES

Cof D: =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**

Cof D[i , k]: =Dc(k, x);

od:

od:

```
print(`x's = `, xc);
print((Cof D));
```

>

$Tc_5 := 0$

$Tc_6 := 1$

[0, 0.5000000000, 0.5000000000, 0.5000000000, 0.5000000000, 1]

$x's = , [0.2500000000 \ 0.5000000000 \ 0.5000000000 \ 0.5000000000 \ 0.7500000000]$

$$\left[\left[S_1 - \frac{SS_{1,1}}{bets_1} - \frac{SS_{1,2}}{bets_2}, S_2 - \frac{SS_{2,1}}{bets_1} - \frac{SS_{2,2}}{bets_2}, S_3 - \frac{SS_{3,1}}{bets_1} - \frac{SS_{3,2}}{bets_2} - \frac{1}{bets_3}, S_4 - \frac{SS_{4,1}}{bets_1} - \frac{SS_{4,2}}{bets_2} - \frac{1}{bets_4}, 1.0000000000 - \frac{1}{bets_3} - \frac{1}{bets_4} \right], \right. \quad (2)$$

$$\left[S_1 - \frac{1}{bets_1}, S_2 - \frac{1}{bets_2}, S_3 - \frac{1}{bets_3}, S_4 - \frac{1}{bets_4}, 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} - \frac{1}{bets_3} - \frac{1}{bets_4} \right],$$

$$\left[S_1 - \frac{1}{bets_1}, S_2 - \frac{1}{bets_2}, S_3 - \frac{1}{bets_3}, S_4 - \frac{1}{bets_4}, 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} - \frac{1}{bets_3} - \frac{1}{bets_4} \right],$$

$$\left[S_1 - \frac{1}{bets_1}, S_2 - \frac{1}{bets_2}, S_3 - \frac{1}{bets_3}, S_4 - \frac{1}{bets_4}, 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} - \frac{1}{bets_3} - \frac{1}{bets_4} \right],$$

$$\left[S_1 - \frac{SS_{1,3}}{bets_3} - \frac{SS_{1,4}}{bets_4} - \frac{1}{bets_1}, S_2 - \frac{SS_{2,3}}{bets_3} - \frac{SS_{2,4}}{bets_4} - \frac{1}{bets_2}, S_3 - \frac{SS_{3,3}}{bets_3} - \frac{SS_{3,4}}{bets_4}, S_4 - \frac{SS_{4,3}}{bets_3} - \frac{SS_{4,4}}{bets_4}, 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} \right]$$

> # 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 (si dec[i]=1 and xc[j]<level) then ixc[i]:=j fi;
```

```
if (si dec[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( Cof D, i );
od;
for i from 1 to Kc do
if si dec[i]=0 then nvD[i] := eval m( vD[ i xc[i]+1] - vD[ i xc[i]] ) fi;
if si dec[i]=1 then nvD[i] := eval m( vD[ i xc[i]] - vD[ i xc[i]+1] ) fi;
od;

```

```

nvD[ Kc+1] := eval m( vD[ 2] );
Cof D_n := matrix( Kc+1, Kc+1, [] );
for i from 1 to Kc+1 do
for j from 1 to Kc+1 do
Cof D_n[ i, j ] := nvD[ i ][ j ];
od; od;
eval m( Cof D_n );

```

$$ixc = , [1 \ 1 \ 4 \ 4]$$

$$nvD_5 := \left[S_1 - \frac{1}{bets_1}, S_2 - \frac{1}{bets_2}, S_3 - \frac{1}{bets_3}, S_4 - \frac{1}{bets_4}, 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} - \frac{1}{bets_3} - \frac{1}{bets_4} \right]$$

$$\left[\left[-\frac{SS_{1,1}}{bets_1} - \frac{SS_{1,2}}{bets_2} + \frac{1}{bets_1}, -\frac{SS_{2,1}}{bets_1} - \frac{SS_{2,2}}{bets_2} + \frac{1}{bets_2}, -\frac{SS_{3,1}}{bets_1} - \frac{SS_{3,2}}{bets_2}, -\frac{SS_{4,1}}{bets_1} - \frac{SS_{4,2}}{bets_2}, \frac{1}{bets_1} + \frac{1}{bets_2} \right], \right. \quad (3)$$

$$\left[-\frac{SS_{1,1}}{bets_1} - \frac{SS_{1,2}}{bets_2} + \frac{1}{bets_1}, -\frac{SS_{2,1}}{bets_1} - \frac{SS_{2,2}}{bets_2} + \frac{1}{bets_2}, -\frac{SS_{3,1}}{bets_1} - \frac{SS_{3,2}}{bets_2}, -\frac{SS_{4,1}}{bets_1} - \frac{SS_{4,2}}{bets_2}, \frac{1}{bets_1} + \frac{1}{bets_2} \right],$$

$$\left[-\frac{SS_{1,3}}{bets_3} - \frac{SS_{1,4}}{bets_4}, -\frac{SS_{2,3}}{bets_3} - \frac{SS_{2,4}}{bets_4}, -\frac{SS_{3,3}}{bets_3} - \frac{SS_{3,4}}{bets_4} + \frac{1}{bets_3}, -\frac{SS_{4,3}}{bets_3} - \frac{SS_{4,4}}{bets_4} + \frac{1}{bets_4}, \frac{1}{bets_3} + \frac{1}{bets_4} \right],$$

$$\left[-\frac{SS_{1,3}}{bets_3} - \frac{SS_{1,4}}{bets_4}, -\frac{SS_{2,3}}{bets_3} - \frac{SS_{2,4}}{bets_4}, -\frac{SS_{3,3}}{bets_3} - \frac{SS_{3,4}}{bets_4} + \frac{1}{bets_3}, -\frac{SS_{4,3}}{bets_3} - \frac{SS_{4,4}}{bets_4} + \frac{1}{bets_4}, \frac{1}{bets_3} + \frac{1}{bets_4} \right],$$

$$\left[S_1 - \frac{1}{bets_1}, S_2 - \frac{1}{bets_2}, S_3 - \frac{1}{bets_3}, S_4 - \frac{1}{bets_4}, 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} \right]$$

$$\left. \left. \left. -\frac{1}{bets_3} - \frac{1}{bets_4} \right] \right] \right]$$

>

> for i from 1 to Kc+1 do # this part works only after running the program to the end and returning

vD[i]:=row(Cof D_n, i); # Then, RUN IT TWICE

od;

for i from 1 to Kc do

if si dec[i]=0 then

et a[i]:=1- al pha[j_of_c[i]]- gam[j_of_c[i]];

el se

et a[i]:=gam[j_of_c[i]];

end if;

print(i, et a[i]);

od;

for i from 1 to Kc+1 do

sD[i]:=et a[1]*vD[1][i]+et a[3]*vD[3][i]+vD[Kc+1][i]; # This line may need to be changed by hand

od;

for i from 1 to N do

bet s[i]:=bet a[i];

al ps[i]:=al pha[i];

od:

det (Cof D_n);

>

>

>

$$vD_1 := \left[\begin{array}{l} -\frac{SS_{1,1}}{bets_1} - \frac{SS_{1,2}}{bets_2} + \frac{1}{bets_1}, -\frac{SS_{2,1}}{bets_1} - \frac{SS_{2,2}}{bets_2} + \frac{1}{bets_2}, -\frac{SS_{3,1}}{bets_1} - \frac{SS_{3,2}}{bets_2}, -\frac{SS_{4,1}}{bets_1} \\ -\frac{SS_{4,2}}{bets_2}, \frac{1}{bets_1} + \frac{1}{bets_2} \end{array} \right]$$

$$vD_2 := \left[\begin{array}{l} -\frac{SS_{1,1}}{bets_1} - \frac{SS_{1,2}}{bets_2} + \frac{1}{bets_1}, -\frac{SS_{2,1}}{bets_1} - \frac{SS_{2,2}}{bets_2} + \frac{1}{bets_2}, -\frac{SS_{3,1}}{bets_1} - \frac{SS_{3,2}}{bets_2}, -\frac{SS_{4,1}}{bets_1} \\ -\frac{SS_{4,2}}{bets_2}, \frac{1}{bets_1} + \frac{1}{bets_2} \end{array} \right]$$

$$vD_3 := \left[\begin{array}{l} -\frac{SS_{1,3}}{bets_3} - \frac{SS_{1,4}}{bets_4}, -\frac{SS_{2,3}}{bets_3} - \frac{SS_{2,4}}{bets_4}, -\frac{SS_{3,3}}{bets_3} - \frac{SS_{3,4}}{bets_4} + \frac{1}{bets_3}, -\frac{SS_{4,3}}{bets_3} - \frac{SS_{4,4}}{bets_4} \\ + \frac{1}{bets_4}, \frac{1}{bets_3} + \frac{1}{bets_4} \end{array} \right]$$

$$vD_4 := \left[-\frac{SS_{1,3}}{bets_3} - \frac{SS_{1,4}}{bets_4}, -\frac{SS_{2,3}}{bets_3} - \frac{SS_{2,4}}{bets_4}, -\frac{SS_{3,3}}{bets_3} - \frac{SS_{3,4}}{bets_4} + \frac{1}{bets_3}, -\frac{SS_{4,3}}{bets_3} - \frac{SS_{4,4}}{bets_4} + \frac{1}{bets_4}, \frac{1}{bets_3} + \frac{1}{bets_4} \right]$$

$$vD_5 := \left[S_1 - \frac{1}{bets_1}, S_2 - \frac{1}{bets_2}, S_3 - \frac{1}{bets_3}, S_4 - \frac{1}{bets_4}, 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_2} - \frac{1}{bets_3} - \frac{1}{bets_4} \right]$$

1, 0.5000000000

2, 0.5000000000

3, 0.5000000000

4, 0.5000000000

$sD_1 := -2.220446049 \cdot 10^{-16}$

$sD_2 := -2.220446049 \cdot 10^{-16}$

$sD_3 := -2.220446049 \cdot 10^{-16}$

$sD_4 := -2.220446049 \cdot 10^{-16}$

$sD_5 := 0.0000000000$

0.0000000000

(4)

>

> **ud:=vect or (50) : Di gi t s: =100; NN: =50; #Expansion with variable slopes**

d:=vect or (50) :

xx:=eval f (rand() / 10^12) ;

xxt :=xx:

bet :=1:

for i from 1 to NN do

bet :=bet / bet a[ui nt _of _x(xxt)] ;

ud[i] :=a[ui nt _of _x(xxt)] ;

udb[i] :=a[ui nt _of _x(xxt)] * bet ;

xxt :=uT(xxt) ;

od:

xxt :=xx:

bet :=1:

for i from 1 to NN do

bet :=bet / bet a[ui nt _of _x(xxt)] ;

d[i] :=a[ui nt _of _x(xxt)] ;

db[i] :=a[ui nt _of _x(xxt)] * bet ;

xxt :=T(xxt) ;

od:

pr i nt (ud) ;

```

ul s_it_x := eval f ( sum( udb[ j 1 ] , j 1=1. . NN ) ) ;
pr i nt ( d ) ;
l s_it_x := eval f ( sum( db[ j 1 ] , j 1=1. . NN ) ) ;
t err := xx- ul s_it_x ;
err := xx- l s_it_x ;

```

Digits := 100

NN := 50

xx := 0.3957188605

```

[0.0000000000, 1.5000000000, -0.5000000000, 1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, -0.5000000000,
1.5000000000, 0.0000000000, 1.0000000000, 0.0000000000, 1.5000000000,
0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, -0.5000000000, 1.5000000000, 0.0000000000, 1.0000000000,
-0.5000000000, 1.5000000000, 0.0000000000, 1.0000000000, -0.5000000000,
1.0000000000, 0.0000000000, 1.5000000000, -0.5000000000, 1.0000000000,
0.0000000000, 1.5000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.5000000000, -0.5000000000, 1.0000000000, 0.0000000000, 1.5000000000,
-0.5000000000, 1.5000000000, 0.0000000000, 1.5000000000]

```

uIs_it_x := 0.3957188605

```

[0.0000000000, 1.5000000000, -0.5000000000, 1.0000000000, 0.0000000000, 1.0000000000,
0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, -0.5000000000,
1.5000000000, 0.0000000000, 1.0000000000, 0.0000000000, 1.5000000000,
0.0000000000, 1.0000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.0000000000, -0.5000000000, 1.5000000000, 0.0000000000, 1.0000000000,
-0.5000000000, 1.5000000000, 0.0000000000, 1.0000000000, -0.5000000000,
1.0000000000, 0.0000000000, 1.5000000000, -0.5000000000, 1.0000000000,
0.0000000000, 1.5000000000, 0.0000000000, 1.0000000000, 0.0000000000,
1.5000000000, -0.5000000000, 1.0000000000, 0.0000000000, 1.5000000000,
-0.5000000000, 1.5000000000, 0.0000000000, 1.5000000000]

```

Is_it_x := 0.3957188605

terr := 3.720957775 10⁻¹⁶

err := 3.720957775 10⁻¹⁶

(5)

>

>

```

> NN:=50; chi :=( x1, x2, t ) ->pi ecewi se( t <x1, 0, t <=x2, 1, 0 ) ;
uchi :=( x1, x2, t ) ->pi ecewi se( 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 si dec[i]=1 then int x:=ui nt_of _x(xxt) el se int x:=
int_of _x(xxt) fi;
      bet _real :=bet;
      bet :=bet / bet a[i nt x];
      dcb[i , n]:=a[i nt x] * bet;

      if si dec[i]=0 then
        for ii from 1 to Kc do
          if xxt>c[ii] then cc[i,ii,n]:=1*bet _real el se
cc[i,ii,n]:=0 fi;
          od;
          if int x=1 then Sc[i , n]:= 0
          el se Sc[i , n]:=sum( 1/ ( bet a[j 7] ) , j 7=1..
int x- 1) * bet _real fi;
          #bc[i , n]:=b[i nt x] * bet _real;
          #####
          el se
          for ii from 1 to Kc do
            if xxt<c[ii] then cc[i,ii,n]:=1*bet _real el se
cc[i,ii,n]:=0 fi;
            od;
            if int x=N then Sc[i , n]:= 0
            el se Sc[i , n]:=sum( 1/ ( bet a[j 8] ) , j 8=
int x+1.. N) * bet _real fi;
            #bc[i , n]:=b[i nt x+1] * bet _real;
          fi;
          val c[i , n]:=xxt;
          bet c[i , n]:=bet _real;
          if si dec[i]=1 then xxt:=uT(xxt) el se xxt:=T(xxt) fi;
          od:
ls_it_x:=sum( dcb[i , j 1] , j 1=1.. NN);
od;
for i from 1 to Kc do
S[i]:=eval f ( sum( Sc[i , j 2+1] , j 2=1.. NN) );
#sum_b[i]:=eval f ( sum( bc[i , j 2+1] , j 2=1.. NN) );
od;
for i from 1 to Kc do
for j from 1 to Kc do
SS[i , j]:=eval f ( sum( cc[i , j , j 1+1] , j 1=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] * bet a[ 1] , val c[ 1 , n] );
#print ( 3 , 2 , cc[ 3 , 2 , n] * bet a[ 2] , val c[ 3 , n] );

```

#od:

$NN := 50$

$\chi := (x1, x2, t) \rightarrow \text{piecewise}(t < x1, 0, t \leq x2, 1, 0)$

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

$xxt := 0$

$bet := 1$

$Is_it_x := 0.0000000000$

$xxt := 0.2500000000$

$bet := 1$

$Is_it_x := 0.2500000000$

$xxt := 0.7500000000$

$bet := 1$

$Is_it_x := 0.7500000000$

$xxt := 1$

$bet := 1$

$Is_it_x := 1.0000000000$

$S_1 := 0.8333333333$

$S_2 := 0.8333333333$

$S_3 := 0.8333333333$

$S_4 := 0.8333333333$

(6)

>

>

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

```
MM[ i , j ] := - SS[ j , i ] ;  
od ; od ;  
print ( ` MM = ` , MM ) ;
```

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

```
MM[ i , i ] := MM[ i , i ] + 1. 0 ;  
od :
```

```

print ( MM );
print ( `det MM = ` , det ( MM ) );
print ( ve );

```

```

DD:=| i nsol ve( MM, ve );
sum( ( S[ i i 7] - 1/ bet a[ j _of _c[ i i 7] ] ) * DD[ i i 7] , i i 7=1. . Kc) - ( 1- sum
( 1/ bet a[ i 8] , i 8=1. . N) );

```

$$MM = \begin{bmatrix} -0.0000000000 & -0.0000000000 & -1.0000000000 & -1.0000000000 \\ -0.3333333333 & -0.3333333333 & -1.0000000000 & -1.0000000000 \\ -1.0000000000 & -1.0000000000 & -0.3333333333 & -0.3333333333 \\ -1.0000000000 & -1.0000000000 & -0.0000000000 & -0.0000000000 \end{bmatrix}$$

eigenvalues MM =, -2.3333333333, 1.6666666667, 1.007251347 10⁻¹⁵, 0.0000000000

$$1/ \text{average beta} = , \frac{1}{\text{ave_beta}}$$

$$\begin{bmatrix} 1.0000000000 & -0.0000000000 & -1.0000000000 & -1.0000000000 \\ -0.3333333333 & 0.6666666667 & -1.0000000000 & -1.0000000000 \\ -1.0000000000 & -1.0000000000 & 0.6666666667 & -0.3333333333 \\ -1.0000000000 & -1.0000000000 & -0.0000000000 & 1.0000000000 \end{bmatrix}$$

$$\det MM = , -3.5555555556$$

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

$$DD := \begin{bmatrix} -0.5000000000 & -1.0000000000 & -1.0000000000 & -0.5000000000 \end{bmatrix}$$

$$6.661338148 10^{-16}$$

(7)

>

```

> density:=proc(t) local j,i, den;
    i:='i':
    #den:=sum( chi ( b[ i ] , b[ i +1] , t) / bet a[ i ] , i=1. . N) ;
    den:=1:
    for j from 1 to Kc do
    if si dec[ j]=0 then
        den:=den+ DD[ j ] * sum( ( chi ( 0, val c[ j , i 1+1] , t) ) *
bet c[ j , i 1+1] , i 1=1. . 50) fi ;

        if si dec[ j]=1 then
            den:=den+ DD[ j ] * sum( ( uchi ( val c[ j , i 1+1] , 1, t) ) *
bet c[ j , i 1+1] , i 1=1. . 50) fi ;

```

```

od;
return den;
end proc;

```

#Normalizing factor

```

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

od;

print(`NC = `, NC);

plot([(1/NC)*density(t), (1/NC)*density_t(t)], t=0..1-0.000001, y=
0..1.6, color=black, thickness=2);

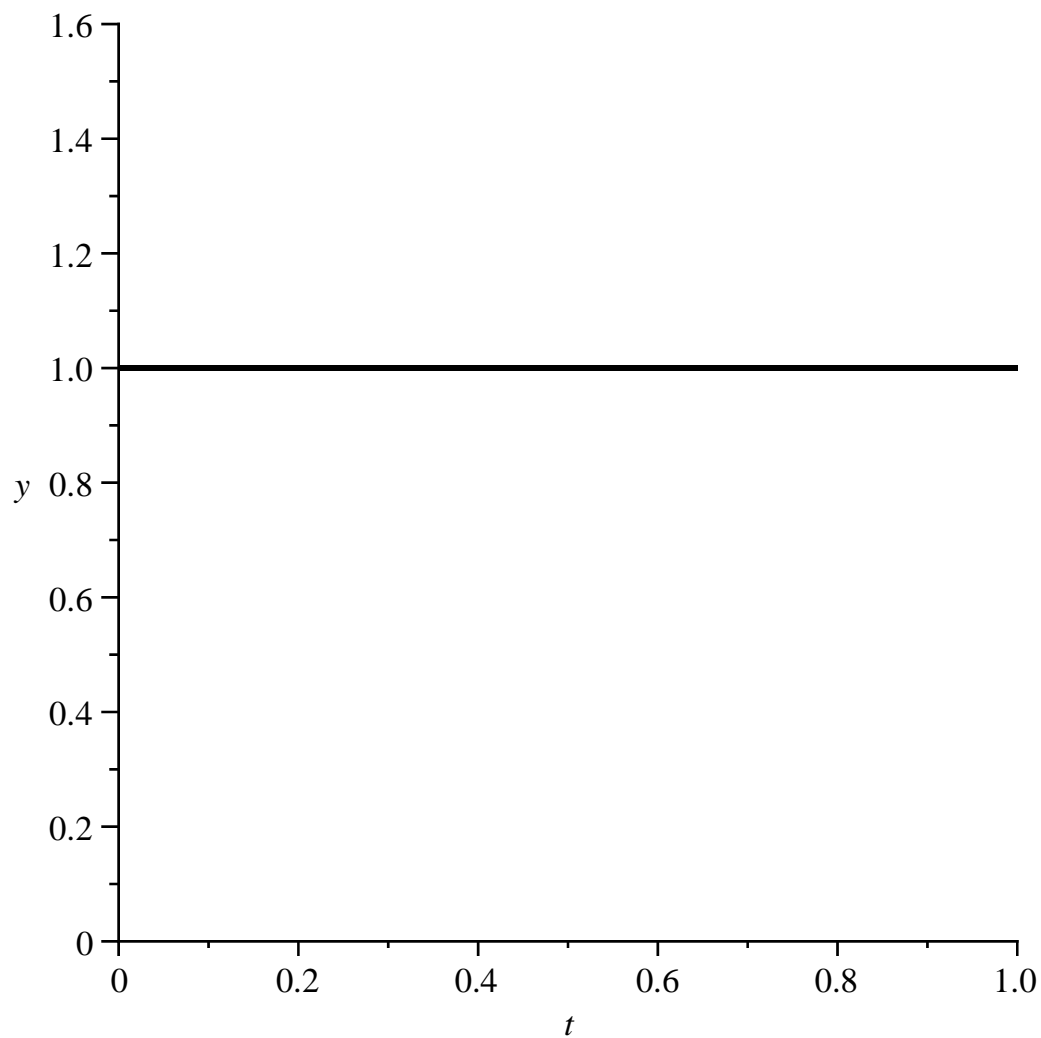
```

```

density := proc(t)
local j, i, den;
i := 'i';
den := 1;
for j to Kc do
if sidec[j]=0 then
den := den + DD[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 + DD[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 = , -1.0000000000



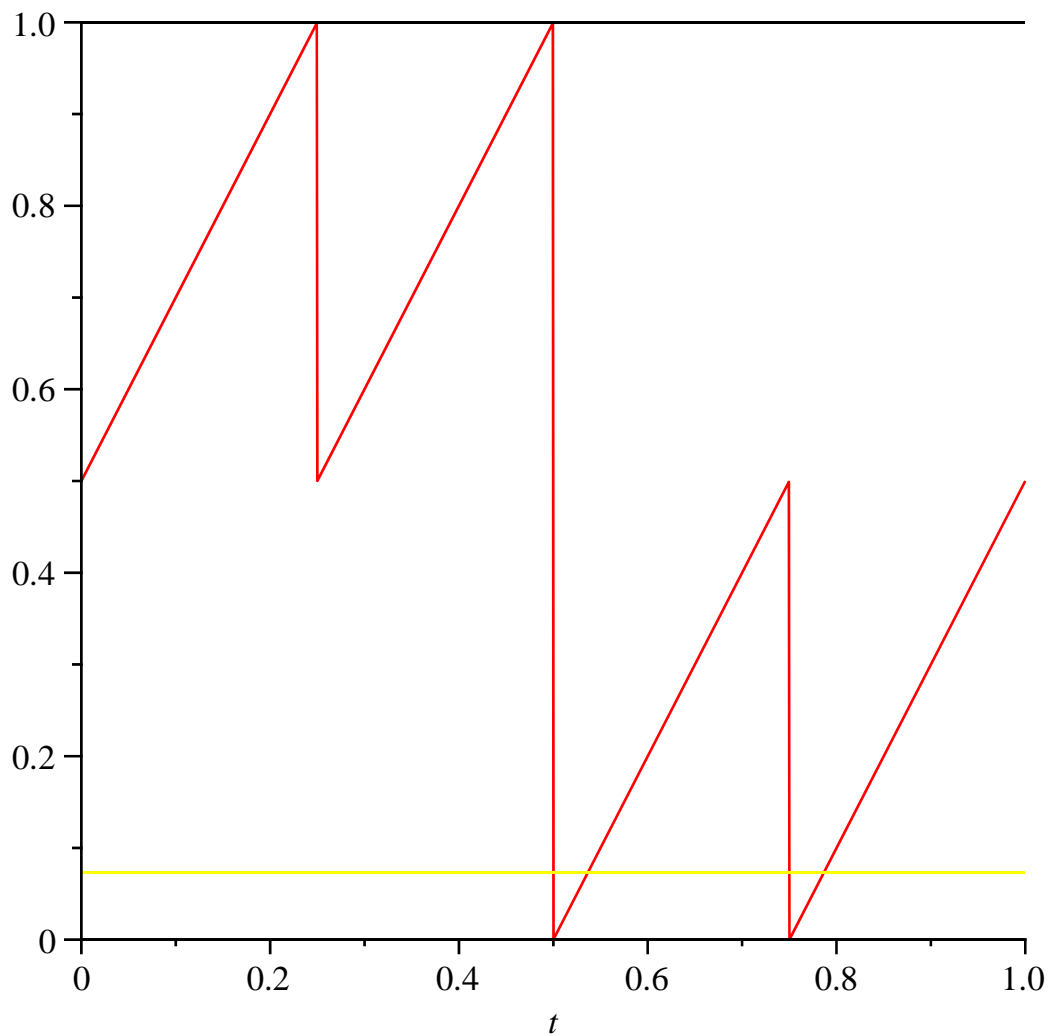
>
>

```

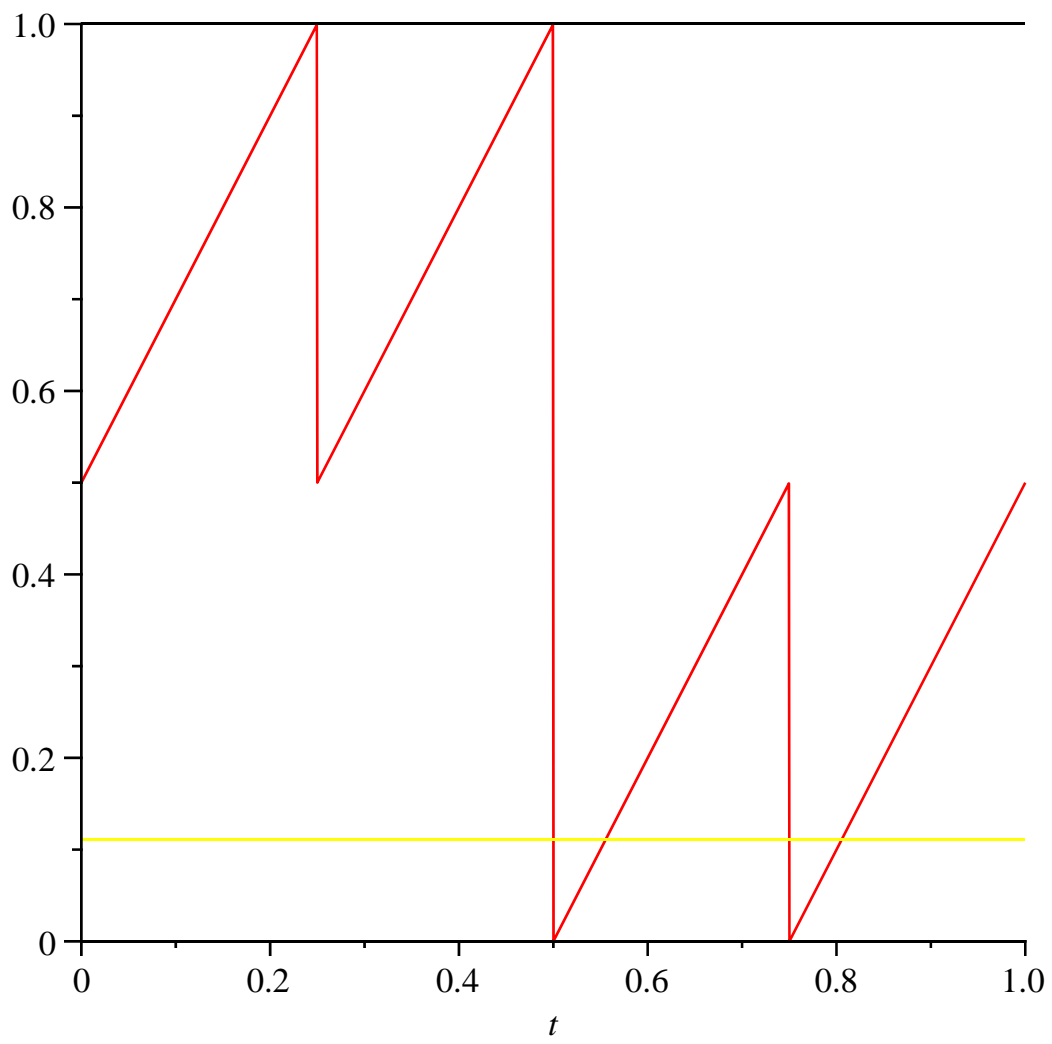
#check density
#preimages
for j6 from 0 to 9 do
y[j6] := j6/10 + (0.1) * rand() / 10^12;
od;
for j6 from 0 to 9 do
for i3 from 1 to N do
pre[i3] := (y[j6] + a[i3]) / beta[i3];
od;
plot([T(t), 0, 1, y[j6]], t=0..1,
color=[red, black, black, yellow]);
su:=0;
for i3 from 1 to N do
if (pre[i3] >= b[i3] and 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
3
4
*err*₀ := 1. 10⁻⁹⁹

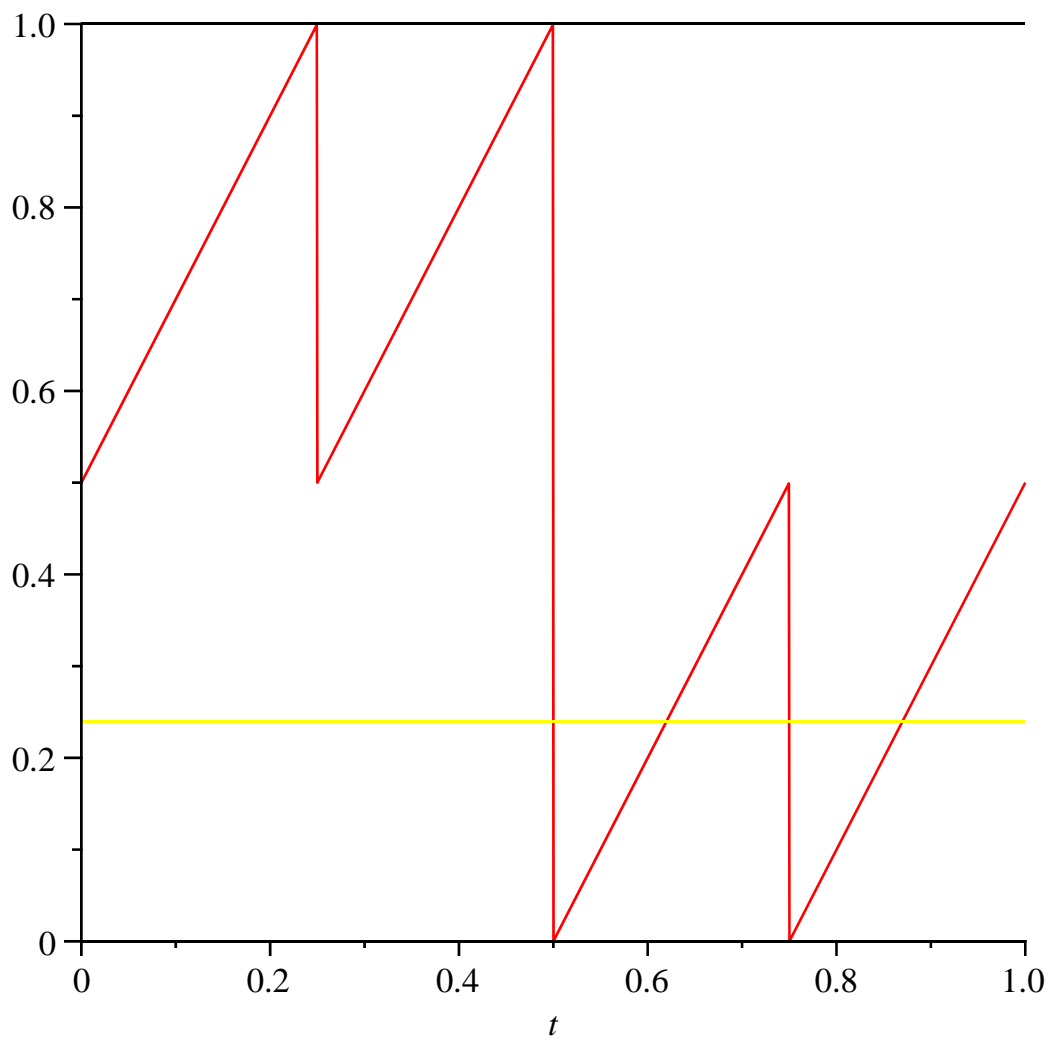


$su := 0$

3

4

$err_1 := 1.10^{-99}$

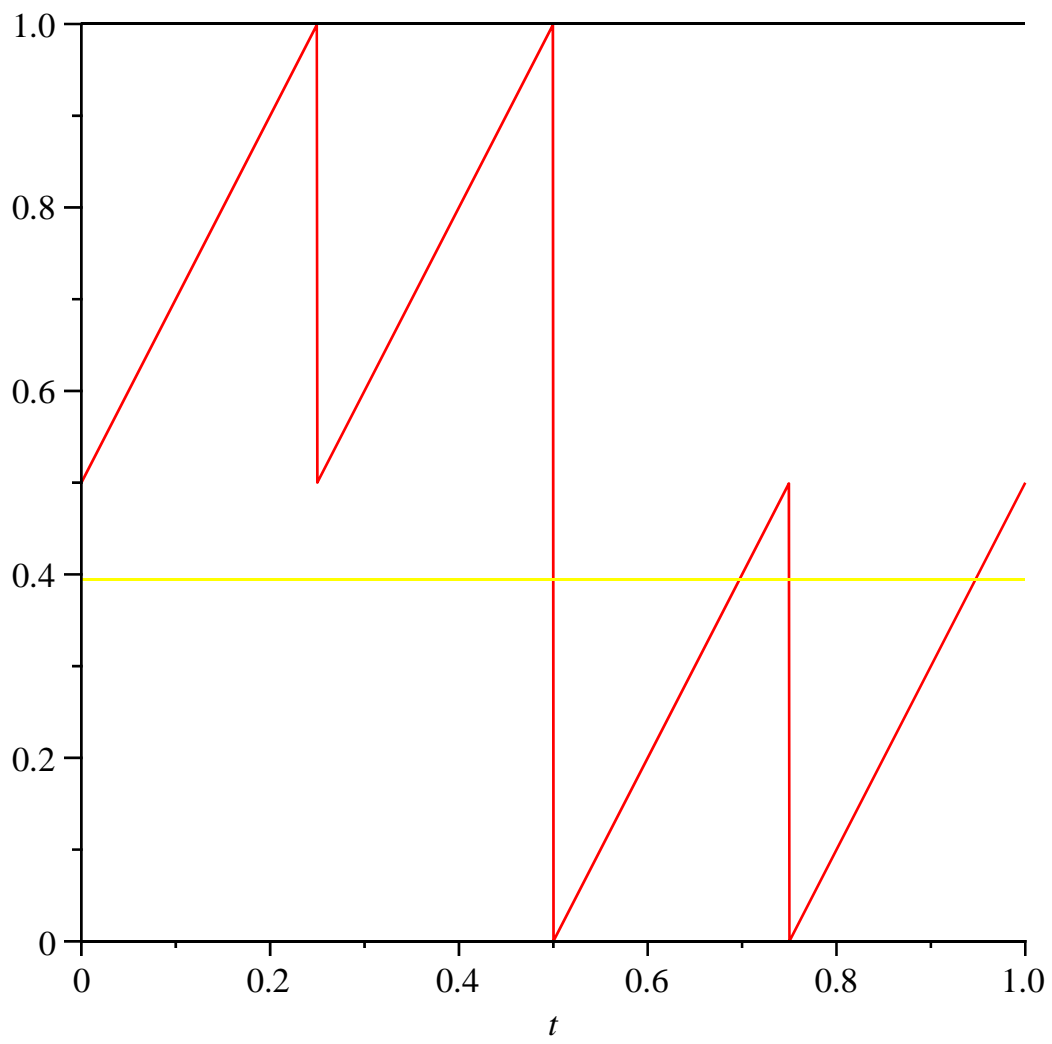


$su := 0$

3

4

$err_2 := 1.10^{-99}$

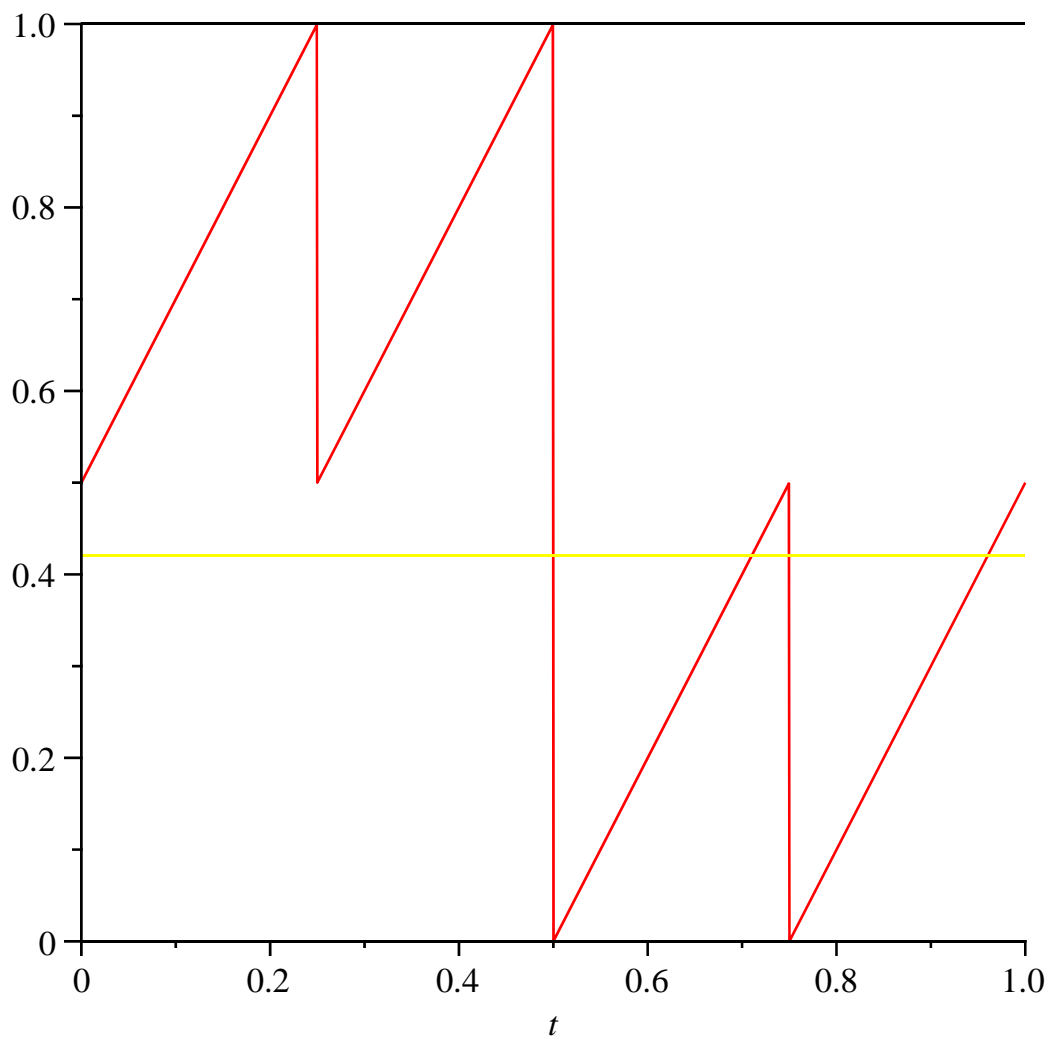


$su := 0$

3

4

$err_3 := 1. 10^{-99}$

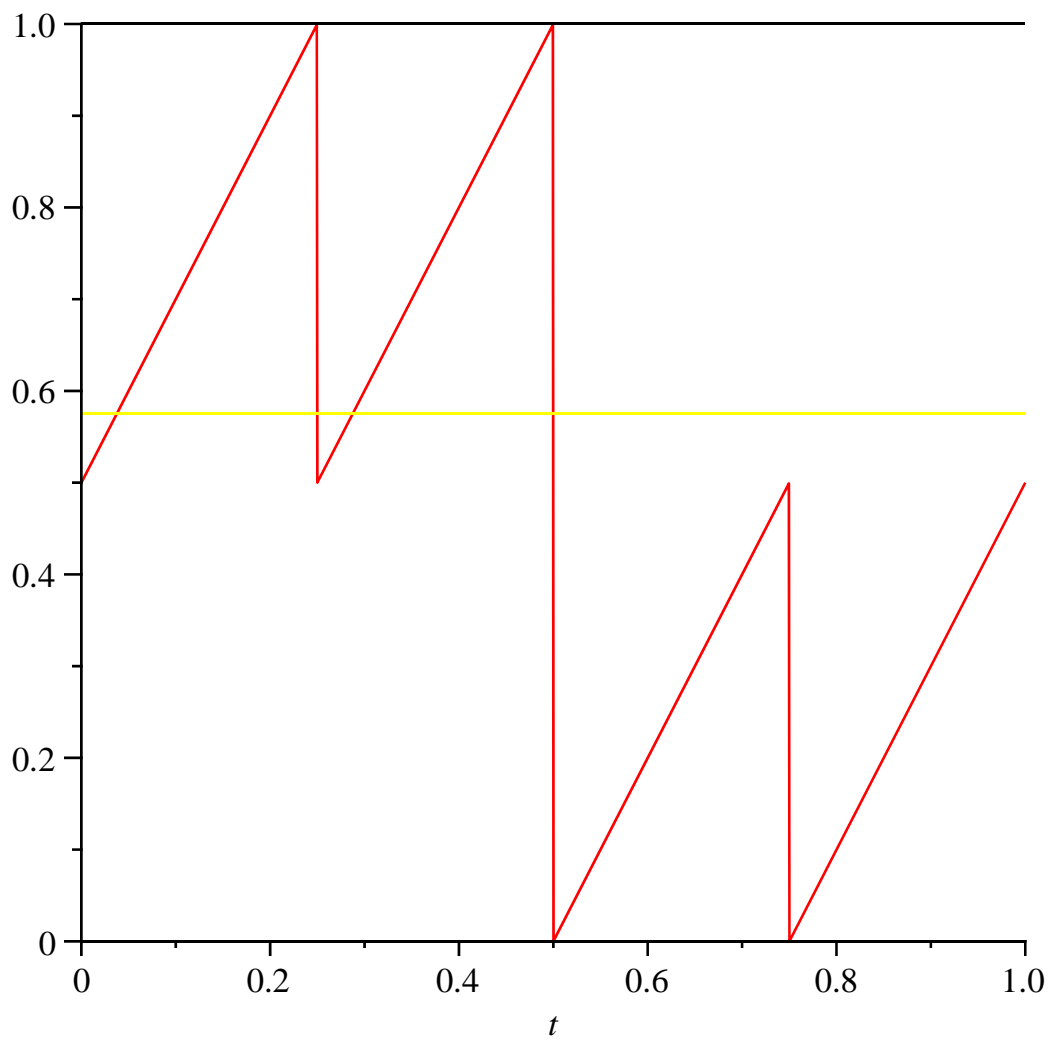


$su := 0$

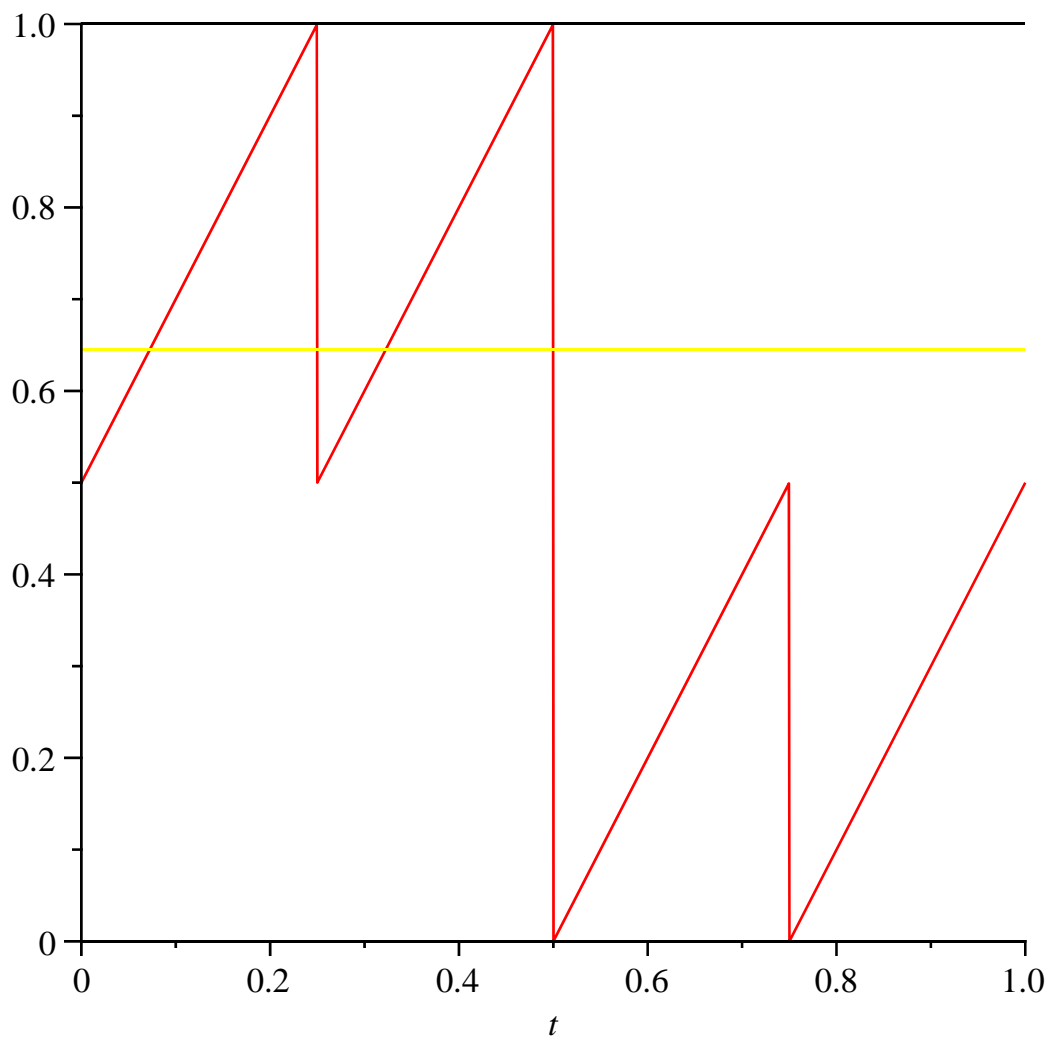
3

4

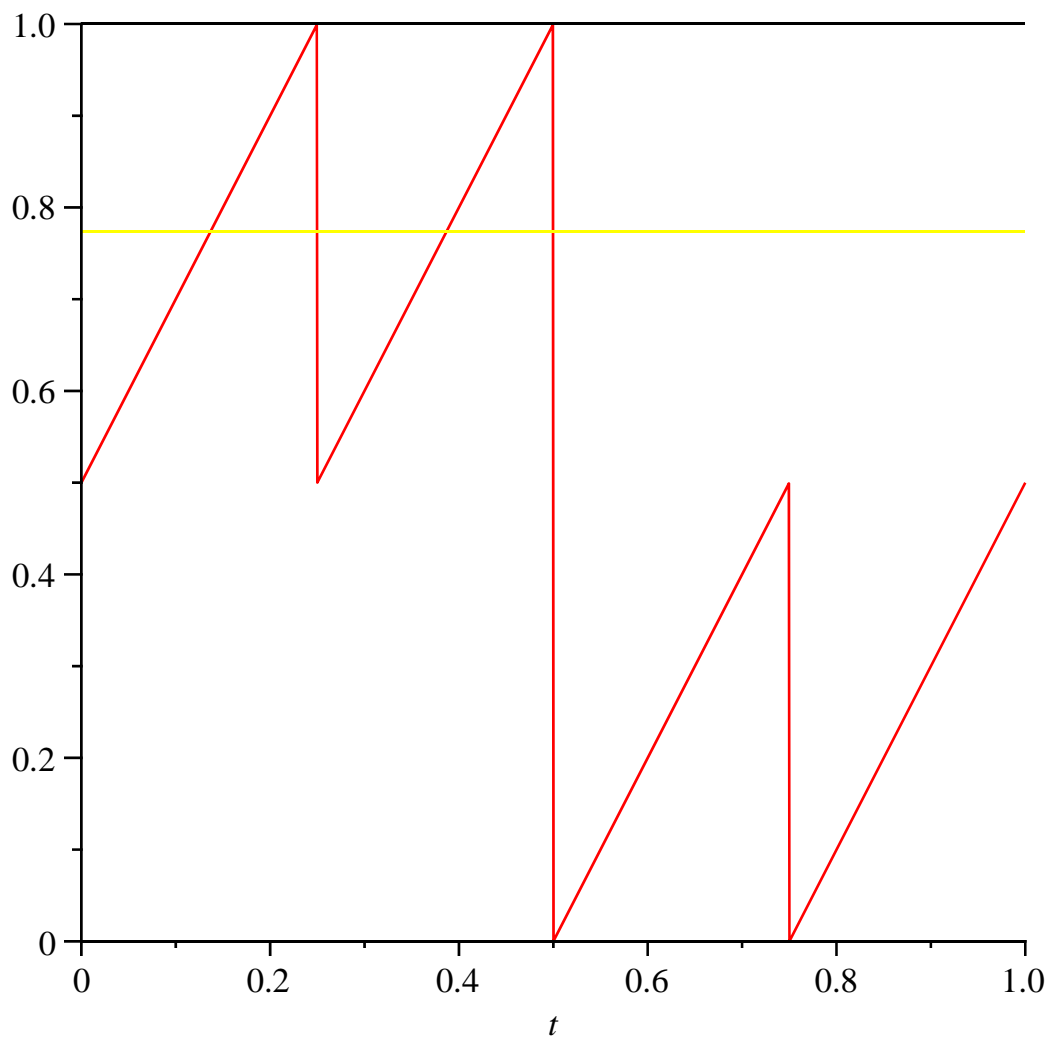
$err_4 := 1.10^{-99}$



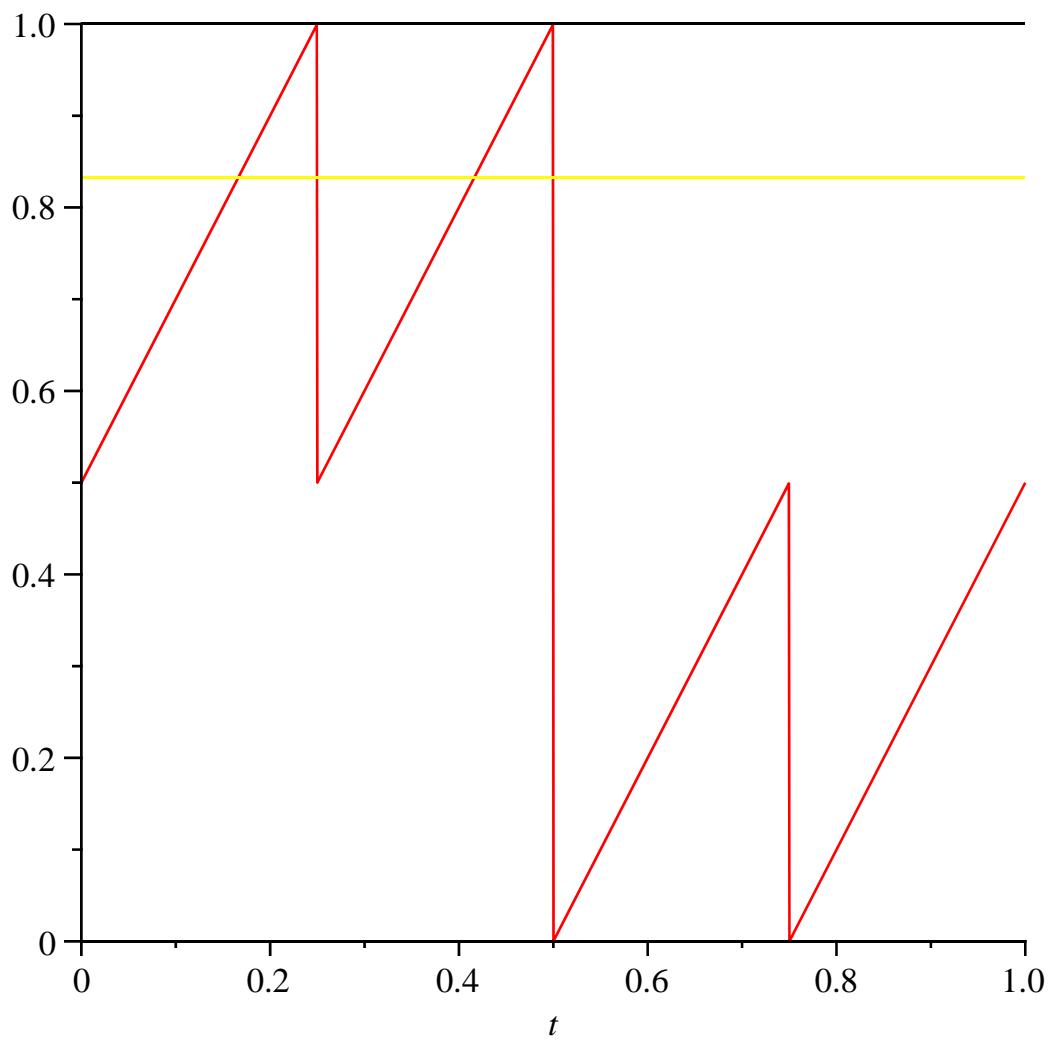
$su := 0$
1
2
 $err_5 := -1.10^{-99}$



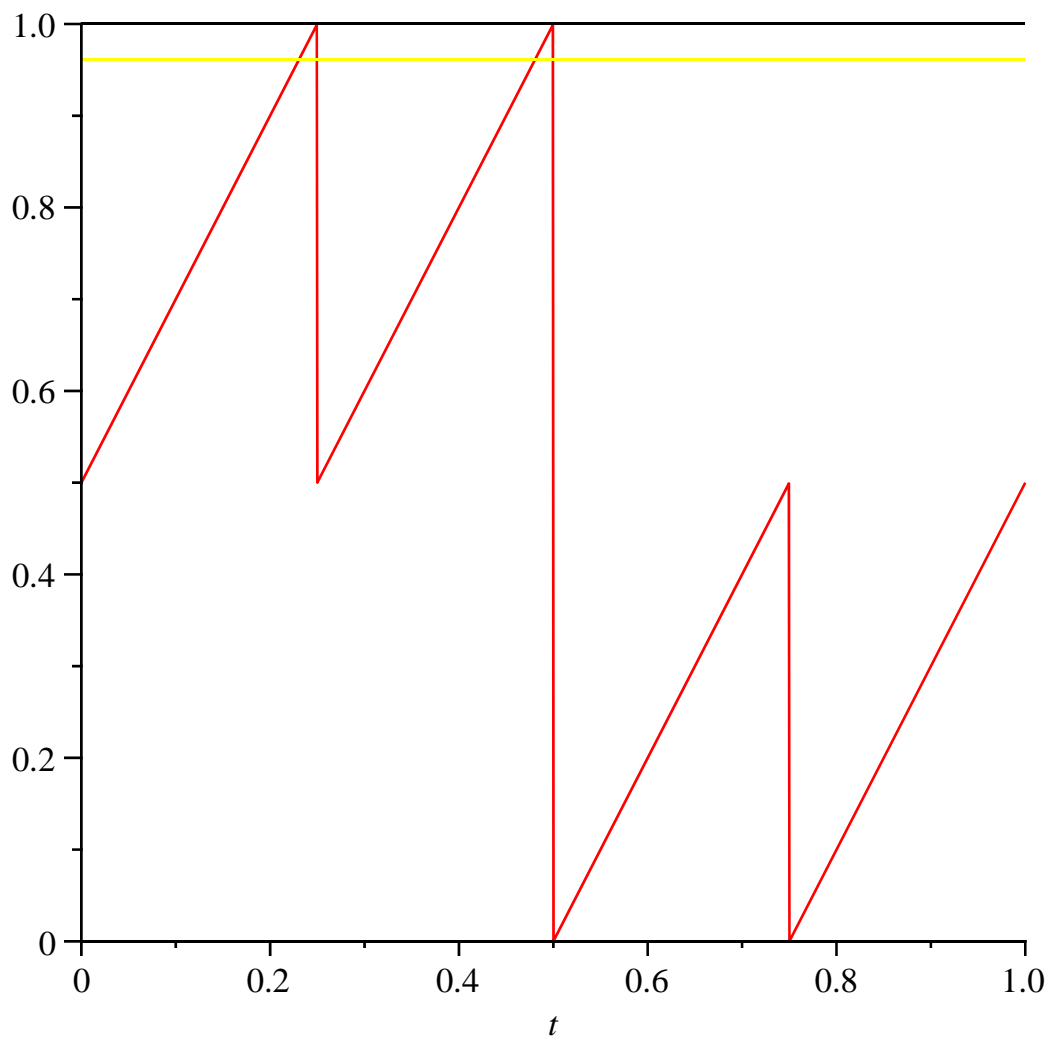
$su := 0$
1
2
 $err_6 := -1.10^{-99}$



$su := 0$
1
2
 $err_7 := -1.10^{-99}$



$su := 0$
1
2
 $err_8 := -1.10^{-99}$



$su := 0$

1

2

$err_0 := -1. 10^{-99}$

$y =, 0.0730616293$

$err[0,] =, 1. 10^{-99}$

$y =, 0.1106507054$

$err[1,] =, 1. 10^{-99}$

$y =, 0.2396412723$

$err[2,] =, 1. 10^{-99}$

$y =, 0.3944913350$

$err[3,] =, 1. 10^{-99}$

$y =, 0.4210936429$

$err[4,] =, 1. 10^{-99}$

$y =, 0.5750072072$

$err[5,] =, -1. 10^{-99}$

$y =, 0.6454744397$

$err[6, j] = -1.10^{-99}$

$y = 0.7736602622$

$err[7, j] = -1.10^{-99}$

$y = 0.8329844592$

$err[8, j] = -1.10^{-99}$

$y = 0.9615732069$

$err[9, j] = -1.10^{-99}$

