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


> N:=4;

  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]:=0.7:beta[1]:=2:gamm[1]:=0.0:
  alpha[2]:=0.3:beta[2]:=3:gamm[2]:=0.2:#
  alpha[2]:=0.2:beta[2]:=3:gamm[2]:=0.2:#
  alpha[3]:=1.0:beta[3]:=4:gamm[3]:=0.0: #
  #alpha[4]:=0.4:beta[4]:=5:gamm[4]:=0.4:
  #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]:=0.45:gamm[N]:=0.55:
  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);
```

>

$$N := 4$$

$$\beta_4 := 1.3500000000$$

$$alpha =, \left[0.7000000000 \quad 0.2000000000 \quad 1.0000000000 \quad 0.4500000000 \right]$$

$$beta =, \left[2 \quad 3 \quad 4 \quad 1.3500000000 \right]$$

$$gamma =, \left[0.0000000000 \quad 0.2000000000 \quad 0.0000000000 \quad 0.5500000000 \right]$$

$$\beta const := 2.3500000000$$

$$b =, \left[0 \quad 0.3500000000 \quad 0.4166666667 \quad 0.6666666667 \quad 1 \right]$$

$$ag =, \left[0 \quad 1.0500000000 \quad 1.6666666667 \quad 0.9000000000 \right]$$

$$al =, \left[-0.3000000000 \quad 0.2500000000 \quad 1.6666666667 \quad 0.3500000000 \right]$$

$$a =, \left[0.0000000000 \quad 0.8500000000 \quad 1.6666666667 \quad 0.3500000000 \right]$$

$$gamma =, \left[0.0000000000 \quad 0.2000000000 \quad 0.0000000000 \quad 0.5500000000 \right]$$

>

```
> # 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]+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;
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;



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], Tc[4]], x=0..1, thickness=
[2, 1, 1, 1, 1, 1, 1]);
plot([' T(x)', x, 0, 1, Tc[1], Tc[2], Tc[3], Tc[4]], x=0..1, thickness=[2,
1, 1, 1, 1, 1, 1, 1]);
```

$$Kc =, 4$$

$$c =, \left[0.3500000000 \quad 0.3500000000 \quad 0.4166666667 \quad 0.6666666667 \quad c_5 \quad c_6 \quad c_7 \quad c_8 \right]$$

$$sidec =, \left[0 \quad 1 \quad 0 \quad 1 \quad sidec_5 \quad sidec_6 \quad sidec_7 \quad sidec_8 \right]$$

$$j\_of\_c =, \left[1 \quad 2 \quad 2 \quad 4 \quad j\_of\_c_5 \quad j\_of\_c_6 \quad j\_of\_c_7 \quad j\_of\_c_8 \right]$$

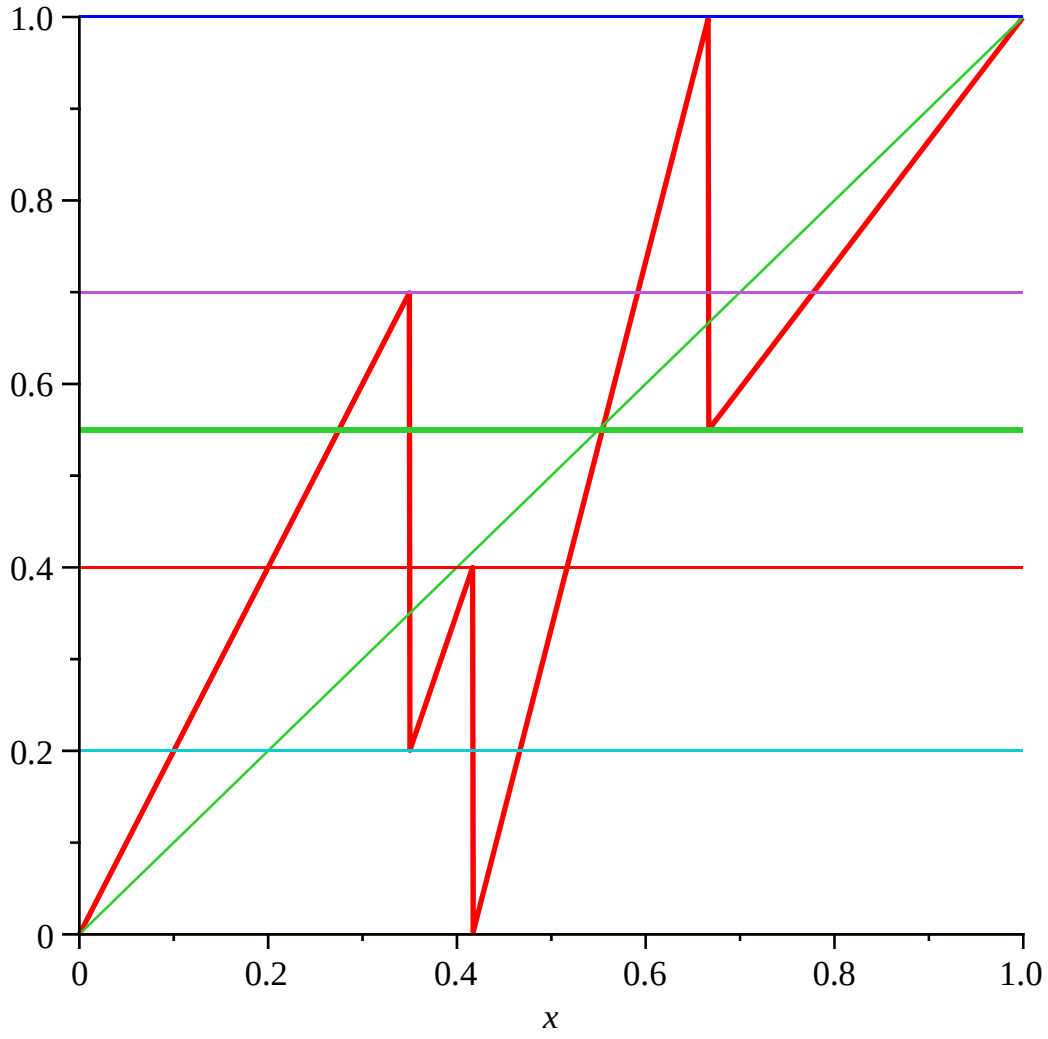
$$uint\_of\_x := x \rightarrow piecewise\left(x < b_2, 1, x < b_3, 2, x < b_4, 3, x < b_5, 4, x < b_6, 5, x < b_7, 6, x < b_8, 7, x < b_9, 8, 9 \right)$$

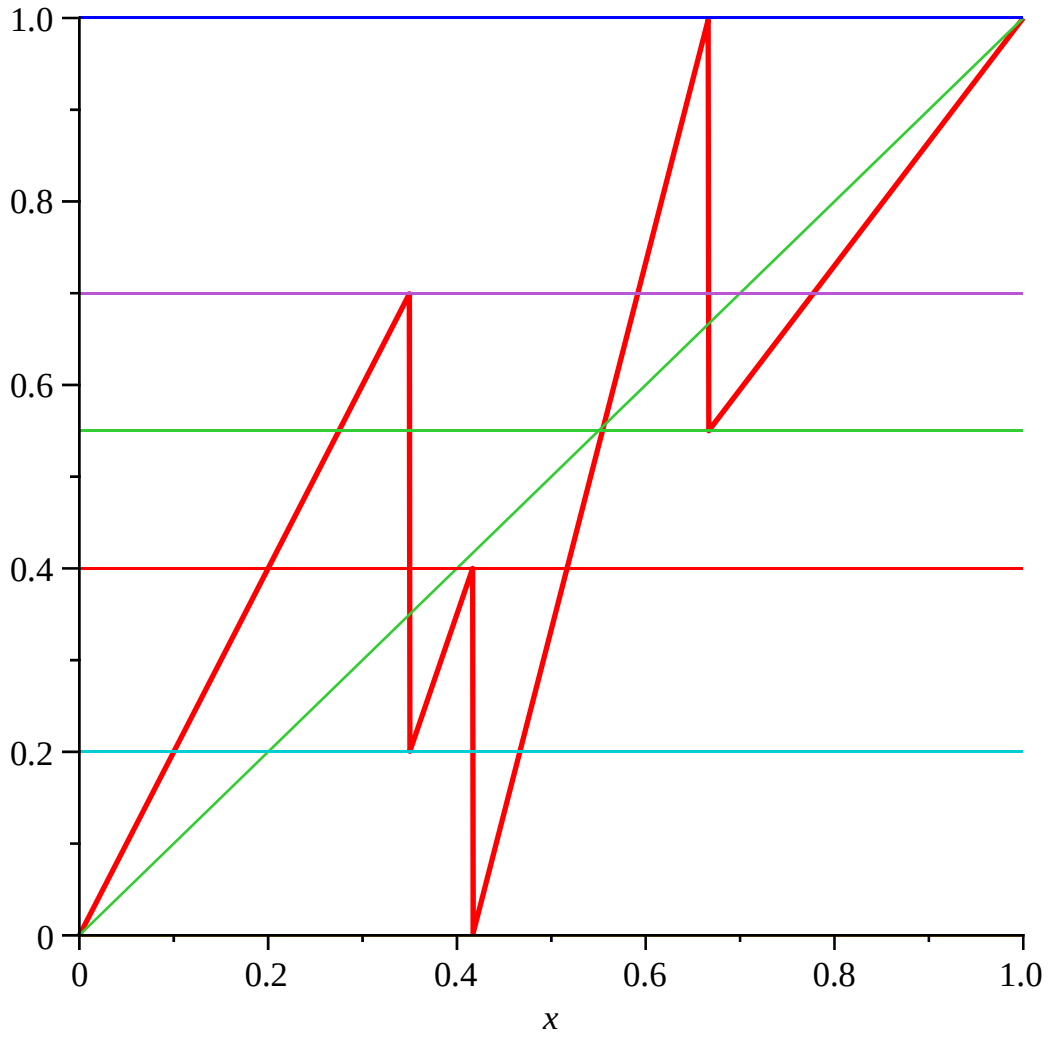
$$int\_of\_x := x \rightarrow piecewise\left(x \le b_2, 1, x \le b_3, 2, x \le b_4, 3, x \le b_5, 4, x \le b_6, 5, x \le b_7, 6, x \le b_8, 7, x \le b_9, 8, 9\right)$$

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

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

$$Tc = , \left[\begin{array}{cccccc} 0.7000000000 & 0.2000000000 & 0.4000000000 & 0.5500000000 & Tc_5 & Tc_6 \end{array}\right]$$






```
> 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
  [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
```

$$S_1 - \frac{SS_{1,3}}{bets_2} - \frac{1}{bets_1}$$

$$S_2 - \frac{SS_{2,3}}{bets_2} - \frac{1}{bets_2}$$

$$S_3 - \frac{SS_{3,3}}{bets_2}$$

$$S_4 - \frac{SS_{4,3}}{bets_2} - \frac{1}{bets_4}$$

$$1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_3} - \frac{1}{bets_4} \quad \textbf{(1)}$$

```
> # Chosing x's and actually producing the coefficients in the non
  simplified equations
  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(`xc = `,xc);
  print((CofD));
```

$$CofD := array(1..5, 1..5, [\,])$$

$$Tc_5 := 0$$

$$Tc_6 := 1$$

$$[0, 0.2000000000, 0.4000000000, 0.5500000000, 0.7000000000, 1]$$

$$xc = , \left[\begin{array}{ccccc} 0.1000000000 & 0.3000000000 & 0.4750000000 & 0.6250000000 & 0.8500000000 \end{array}\right]$$

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

$$\left.\left. - \frac{SS_{4,2}}{bets_2} - \frac{SS_{4,4}}{bets_4}, 1.0000000000 - \frac{1}{bets_1} - \frac{1}{bets_3}\right]\right],$$

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

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

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

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

```
> 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]-nvD[4]);# This to BE 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;
```

```
eval m( Cof D_n) ;
```

$$ixc = , \left[\begin{array}{cccc} 4 & 1 & 2 & 3 \end{array} \right]$$

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

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

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

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

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

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

```
>
> for i from 1 to Kc+1 do # this part works only after runnig the
program to the end and returning
vD[i]:=row(Cof D_n,i);          # Then, RUN IT TWICE
od;                          # This checks if the last equation of
EQS is dependent of the others
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]:=sum(eta[i56]*vD[i56][i],i56=1..Kc)+vD[Kc+1][i];
od;
for i from 1 to N do
bets[i]:=beta[i];
alps[i]:=alpha[i];
od:

det(Cof D_n);
```

$$vD_1 := \left[-\frac{SS_{1,1}}{bets_1} + \frac{1}{bets_1} \quad -\frac{SS_{2,1}}{bets_1} \quad -\frac{SS_{3,1}}{bets_1} \quad -\frac{SS_{4,1}}{bets_1} \quad \frac{1}{bets_1} \right]$$

$$vD_2 := \left[-\frac{SS_{1,2}}{bets_2} \quad -\frac{SS_{2,2}}{bets_2} + \frac{1}{bets_2} \quad -\frac{SS_{3,2}}{bets_2} \quad -\frac{SS_{4,2}}{bets_2} \quad \frac{1}{bets_2} \right]$$

$$vD_3 := \left[-\frac{SS_{1,3}}{bets_2} \quad -\frac{SS_{2,3}}{bets_2} \quad -\frac{SS_{3,3}}{bets_2} + \frac{1}{bets_2} \quad -\frac{SS_{4,3}}{bets_2} \quad \frac{1}{bets_2} \right]$$

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

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

1, 0.3000000000

2, 0.2000000000

3, 0.6000000000

4, 0.5500000000

$$sD_1 := -6.791384593\ 10^{-17}$$

$$sD_2 := -2.973587825\ 10^{-22}$$

$$sD_3 := -2.936432413\ 10^{-17}$$

$$sD_4 := -7.357393059\ 10^{-18}$$

$$sD_5 := 1.\ 10^{-100}$$

$$2.955838406\ 10^{-19} \qquad \textbf{(4)}$$

```
>
> 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);
uIs_it_x:=evalf(sum(udb[j1],j1=1..NN));
print(d);
Is_it_x:=evalf(sum(db[j1],j1=1..NN));
terr:=xx-uIs_it_x;
err:=xx-Is_it_x;
```

$$Digits := 100$$

$$NN := 50$$

$$xx := 0.3957188605$$

[0.8500000000, 0.0000000000, 0.3500000000, 1.6666666667, 1.6666666667, 1.6666666667,
0.3500000000, 0.3500000000, 0.3500000000, 1.6666666667, 0.3500000000,
0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000,
0.3500000000, 0.3500000000, 1.6666666667, 1.6666666667, 0.3500000000,
0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000,
0.3500000000, 1.6666666667, 0.3500000000, 0.3500000000, 0.3500000000,
0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000, 1.6666666667,
1.6666666667, 0.3500000000, 1.6666666667, 0.3500000000, 1.6666666667,
0.3500000000, 0.3500000000, 1.6666666667, 0.3500000000, 0.3500000000,
1.6666666667, 0.3500000000, 0.3500000000, 0.3500000000]

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

[0.8500000000, 0.0000000000, 0.3500000000, 1.6666666667, 1.6666666667, 1.6666666667,
0.3500000000, 0.3500000000, 0.3500000000, 1.6666666667, 0.3500000000,
0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000,
0.3500000000, 0.3500000000, 1.6666666667, 1.6666666667, 0.3500000000,
0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000,
0.3500000000, 1.6666666667, 0.3500000000, 0.3500000000, 0.3500000000,
0.3500000000, 0.3500000000, 0.3500000000, 0.3500000000, 1.6666666667,
1.6666666667, 0.3500000000, 1.6666666667, 0.3500000000, 1.6666666667,
0.3500000000, 0.3500000000, 1.6666666667, 0.3500000000, 0.3500000000,
1.6666666667, 0.3500000000, 0.3500000000, 0.3500000000]

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

$$terr := 5.708673503\ 10^{-14}$$

$$err := 5.708673503\ 10^{-14}$$

(5)

>

```
>

>


> NN:=50; 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;

              ################
                     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;

       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));

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:
```

$$NN := 50$$

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

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

$$xxt := 0.3500000000$$

$$bet := 1$$

$$Is\_it\_x := 0.3500000000$$

$$xxt := 0.3500000000$$

$$bet := 1$$

$$Is\_it\_x := 0.3500000000$$

$$xxt := 0.4166666667$$

$$bet := 1$$

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

$$xxt := 0.6666666667$$

$$bet := 1$$

$$Is\_it\_x := 0.6666666667$$

$$S_1 := 1.0540178573$$

$$S_2 := 0.7004357298$$

$$S_3 := 0.2837797619$$

$$S_4 := 0.7444201532 \qquad \textbf{(6)}$$

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

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


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;

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

print(MM);
print(ve);



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

$$MM = , \begin{bmatrix} -1.0767857144 & -0.3529411765 & -0.4529761905 & -0.0135030864 \\ -1.0767857144 & -0.3529411765 & -0.4529761905 & -0.0135030864 \\ -1.0767857144 & -0.5882352941 & -0.1196428572 & -0.0192901235 \\ -0.6267857146 & -0.5882352941 & -0.0696428572 & -0.9923804012 \end{bmatrix}$$

$$eigenvalues\ MM =, -1.8837904905, -1.750986509\ 10^{-17}, 0.3121363228, -0.9700959816$$

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

$$\begin{bmatrix} -0.0767857144 & -0.3529411765 & -0.4529761905 & -0.0135030864 \\ -1.0767857144 & 0.6470588235 & -0.4529761905 & -0.0135030864 \\ -1.0767857144 & -0.5882352941 & 0.8803571428 & -0.0192901235 \\ -0.6267857146 & -0.5882352941 & -0.0696428572 & 0.0076195988 \end{bmatrix}$$

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

$$DD := \begin{bmatrix} -0.8761081723 & -0.8761081723 & -0.8834704259 & -16.5385663803 \end{bmatrix}$$

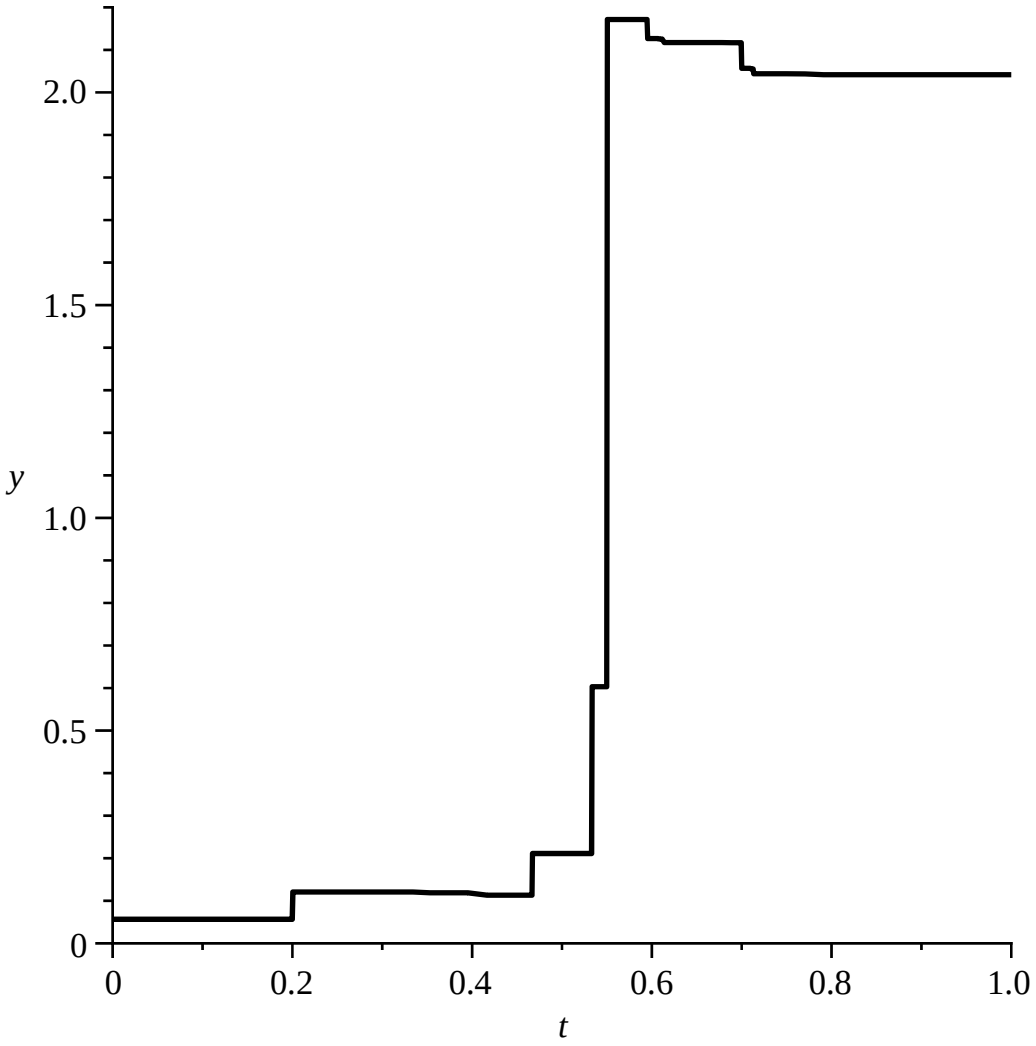
$$2.071233814\ 10^{-16} \qquad (7)$$

```
>
density:=proc(t) local j,i, den;
                i:='i':
                #den:=sum(chi(b[i],b[i+1],t)/beta[i],i=1..N);
               den:=1:
               for j from 1 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) fi;

                if sidec[j]=1 then
```

```
                    den:=den+ DD[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:=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)],t=0..1-0.000001,y=0..2.2,color=black,
thickness=2);
```

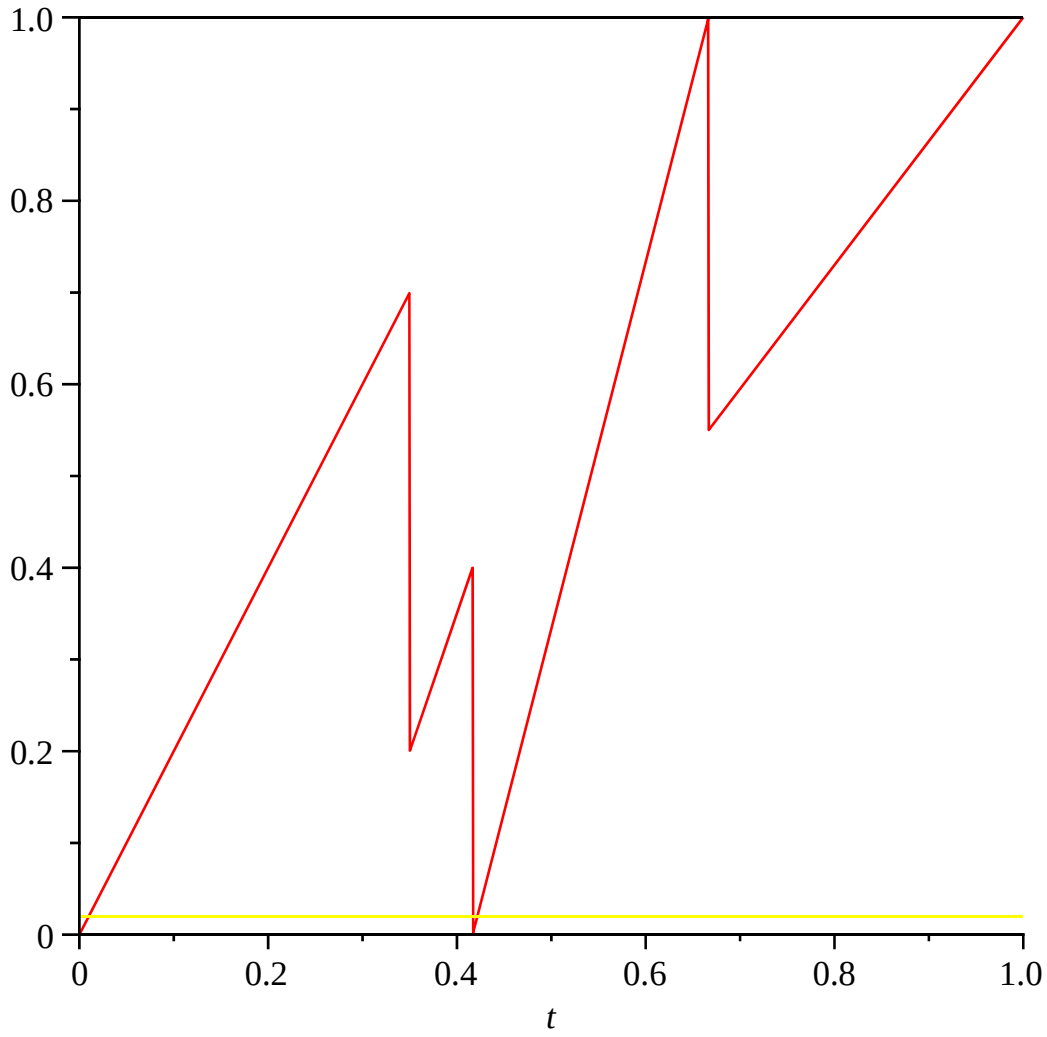
```
density := proc(t)
    local j, i, den;
    i := 'i';
    den := 1;
    for j to Kc do
        if sidec[j] = 0 then
            den := den + DD[j] * (sum(chi(0, valc[j, i1 + 1], t) * betc[j, i1 + 1], i1 = 1 ..50))
        end if;
        if sidec[j] = 1 then
            den := den + DD[j] * (sum(uchi(valc[j,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

$$NC = , -7.8118065695$$




```
> 
> 
#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;
```

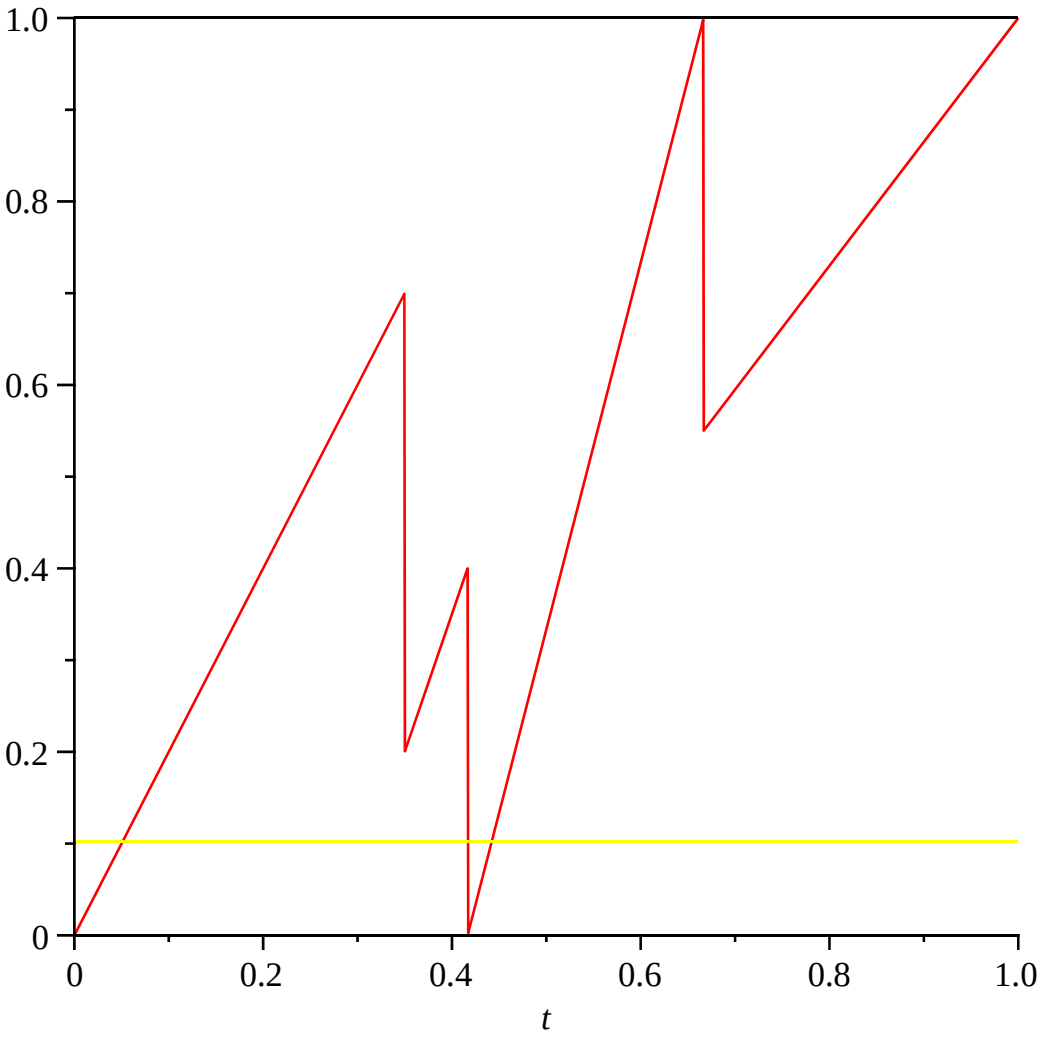


$su := 0$

1

3

$err_0 := -1.009208108\ 10^{-16}$

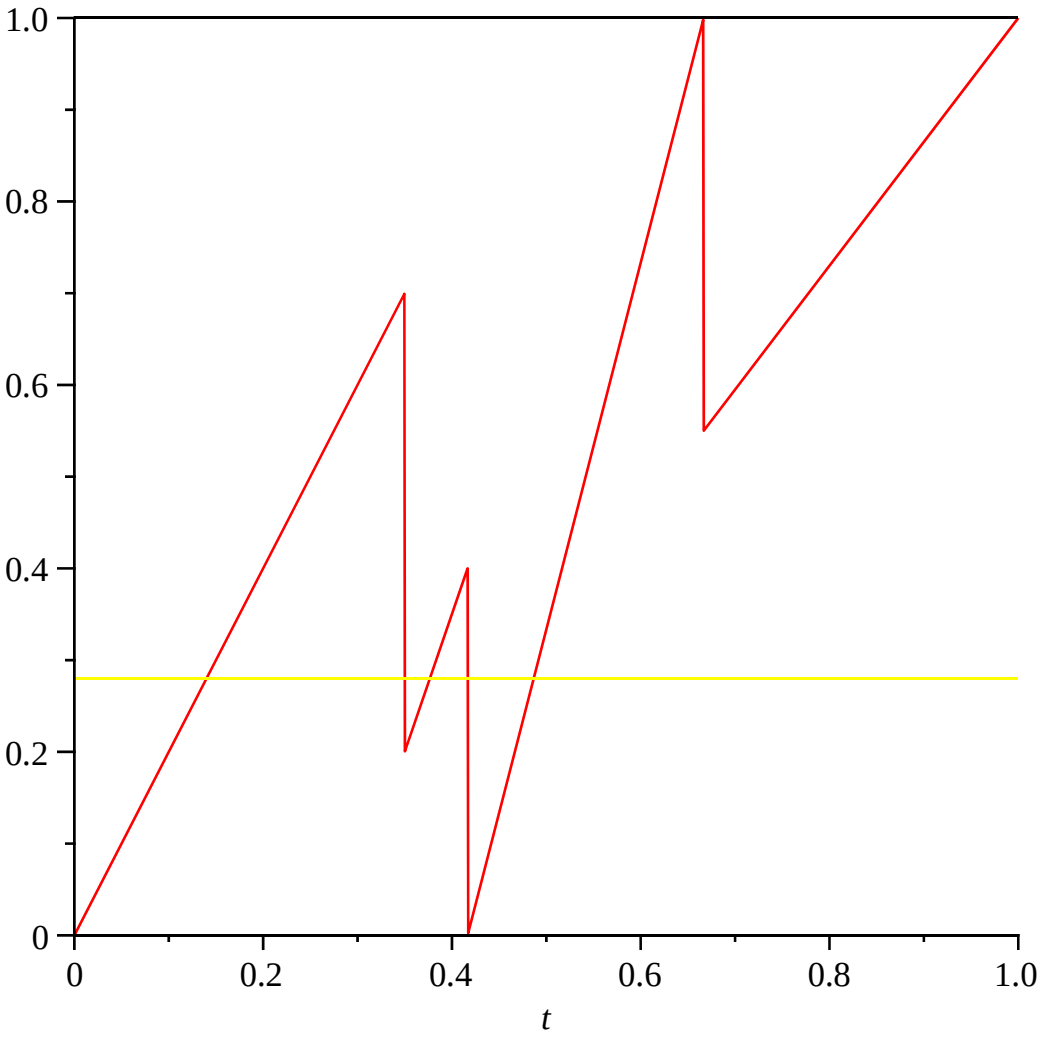


$su := 0$

1

3

$err_1 := -1.009208108\ 10^{-16}$



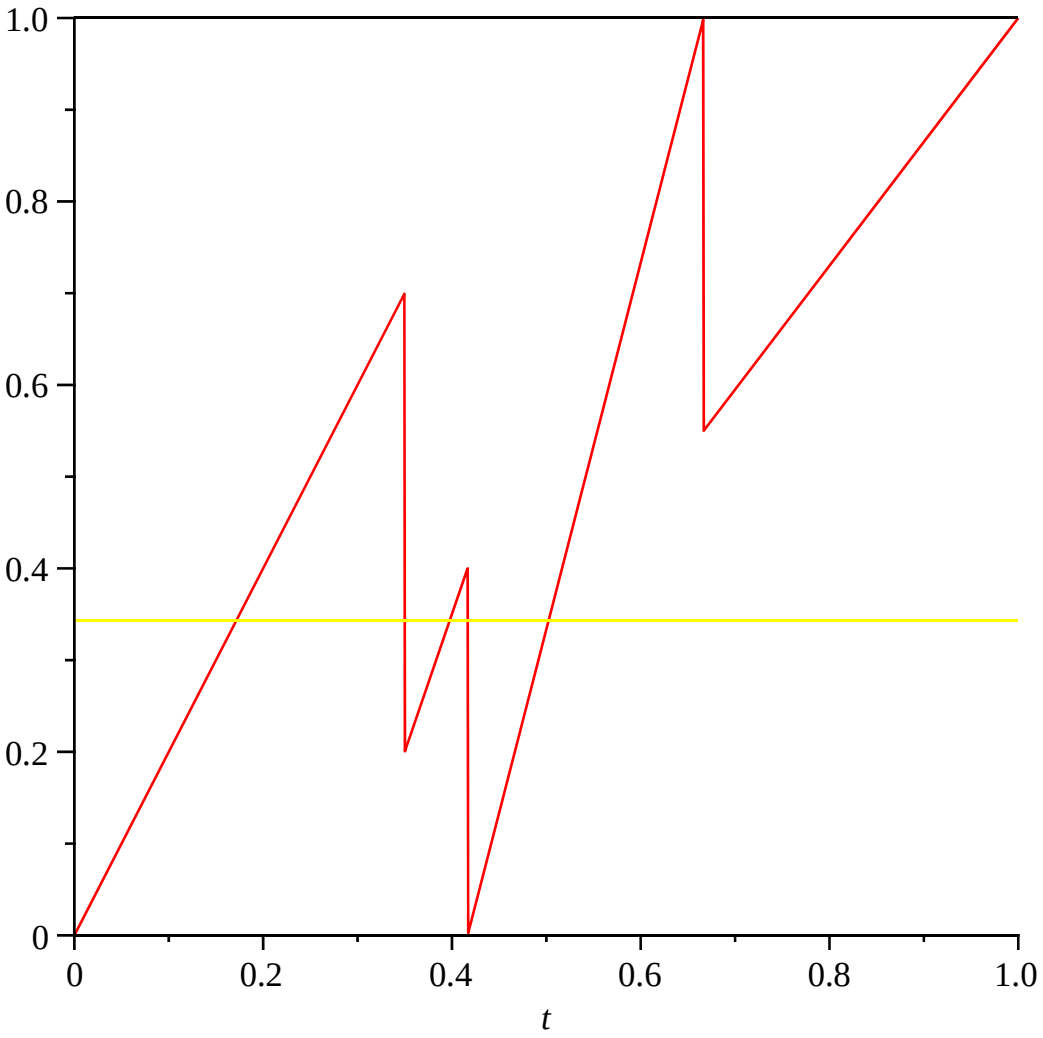
$su := 0$

1

2

3

$err_2 := -1.009208108\ 10^{-16}$



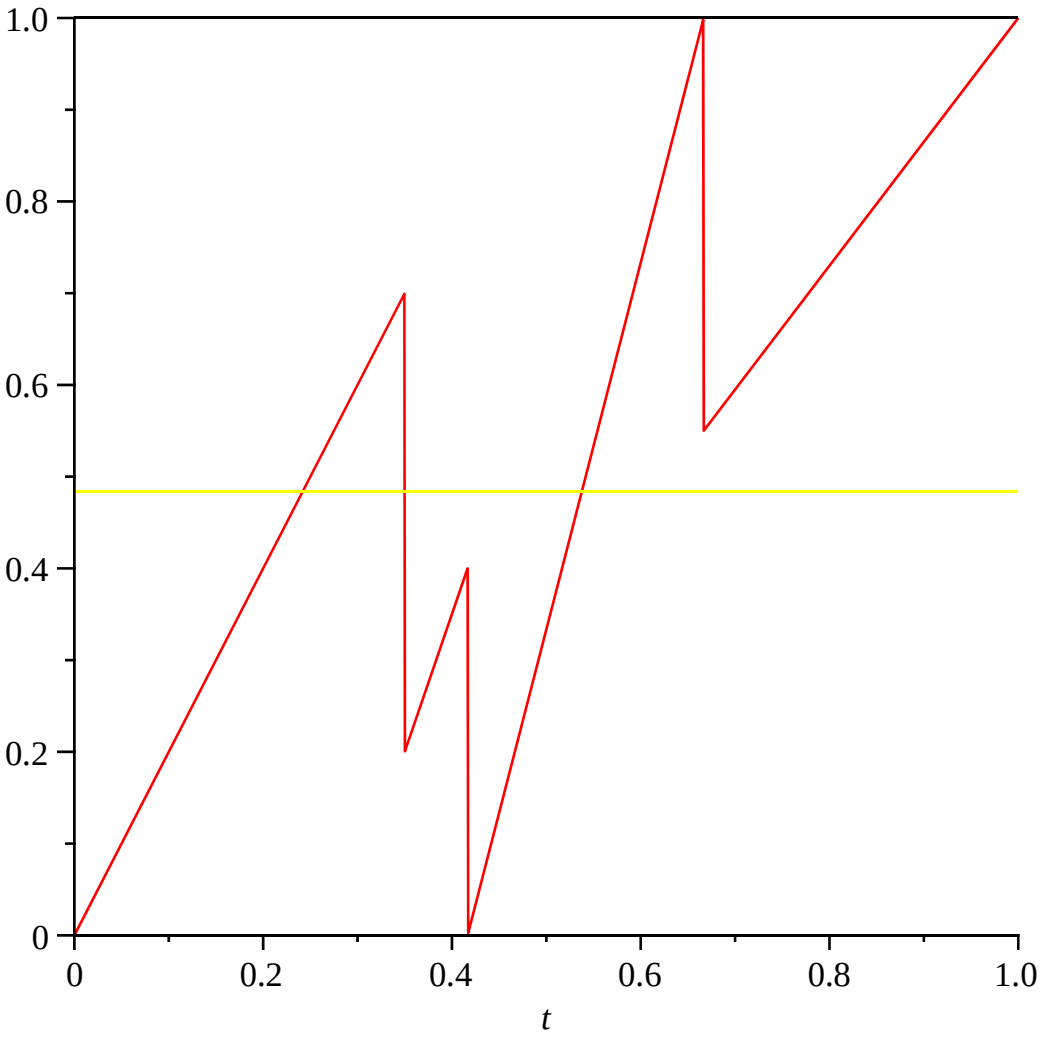
$su := 0$

1

2

3

$err_3 := -1.009208108\ 10^{-16}$

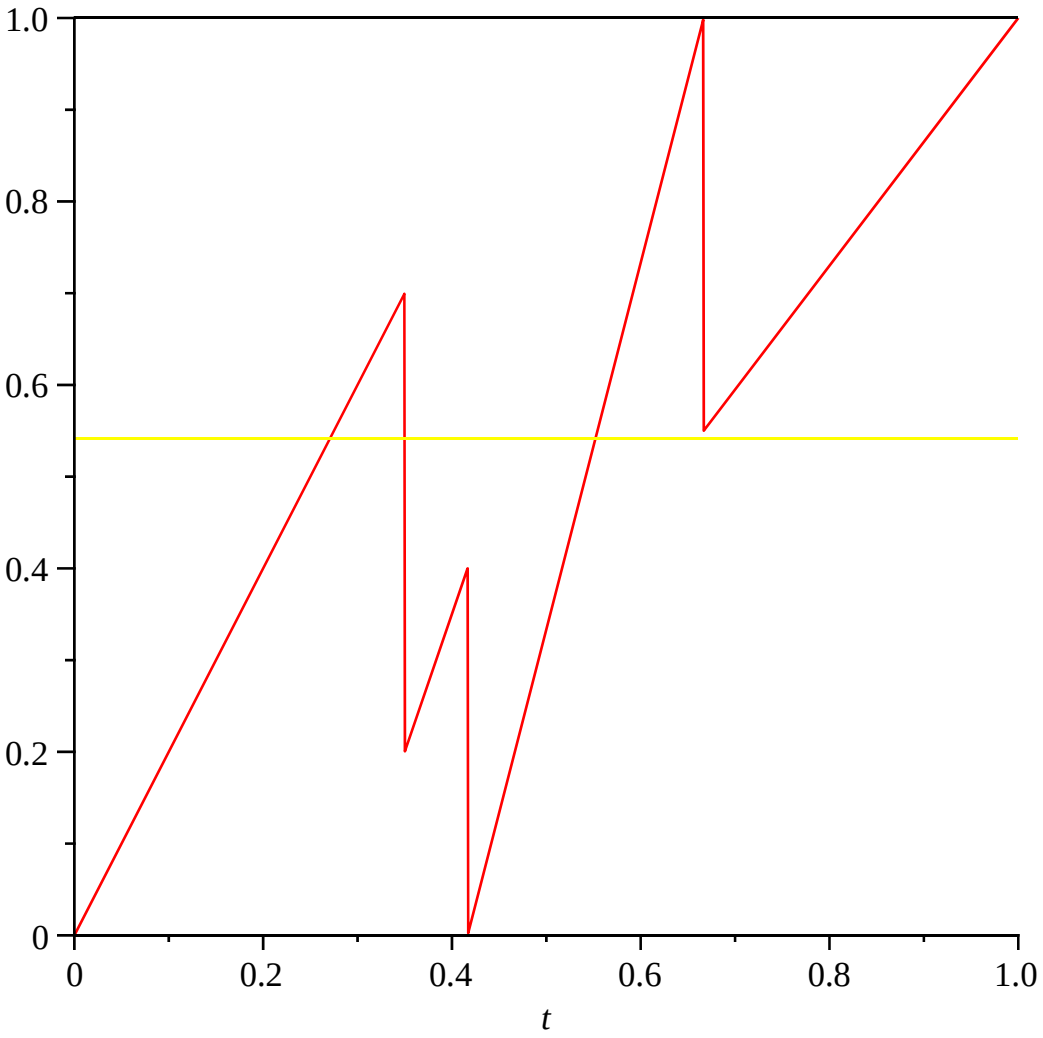


$su := 0$

1

3

$err_4 := -1.009204100\ 10^{-16}$

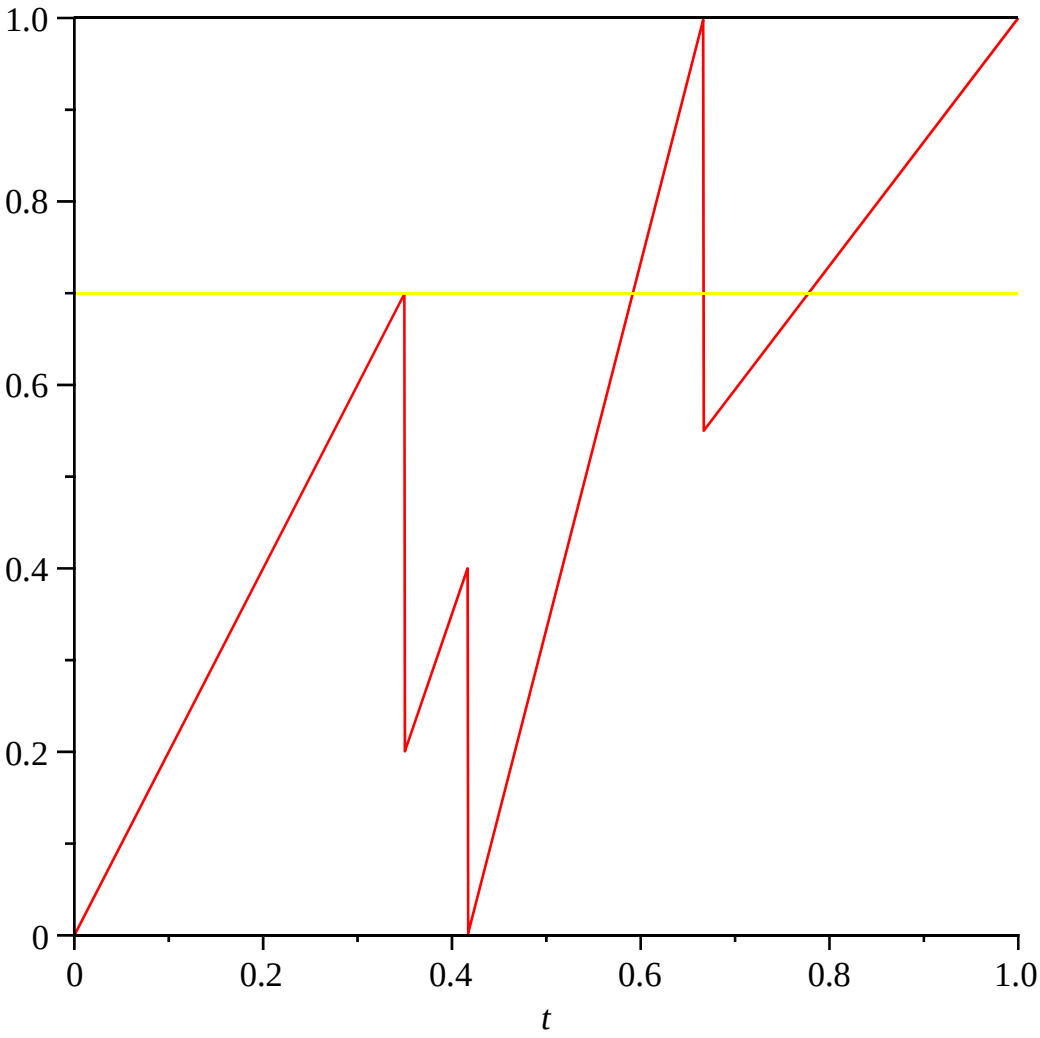


$su := 0$

1

3

$err_5 := -1.009204100\ 10^{-16}$



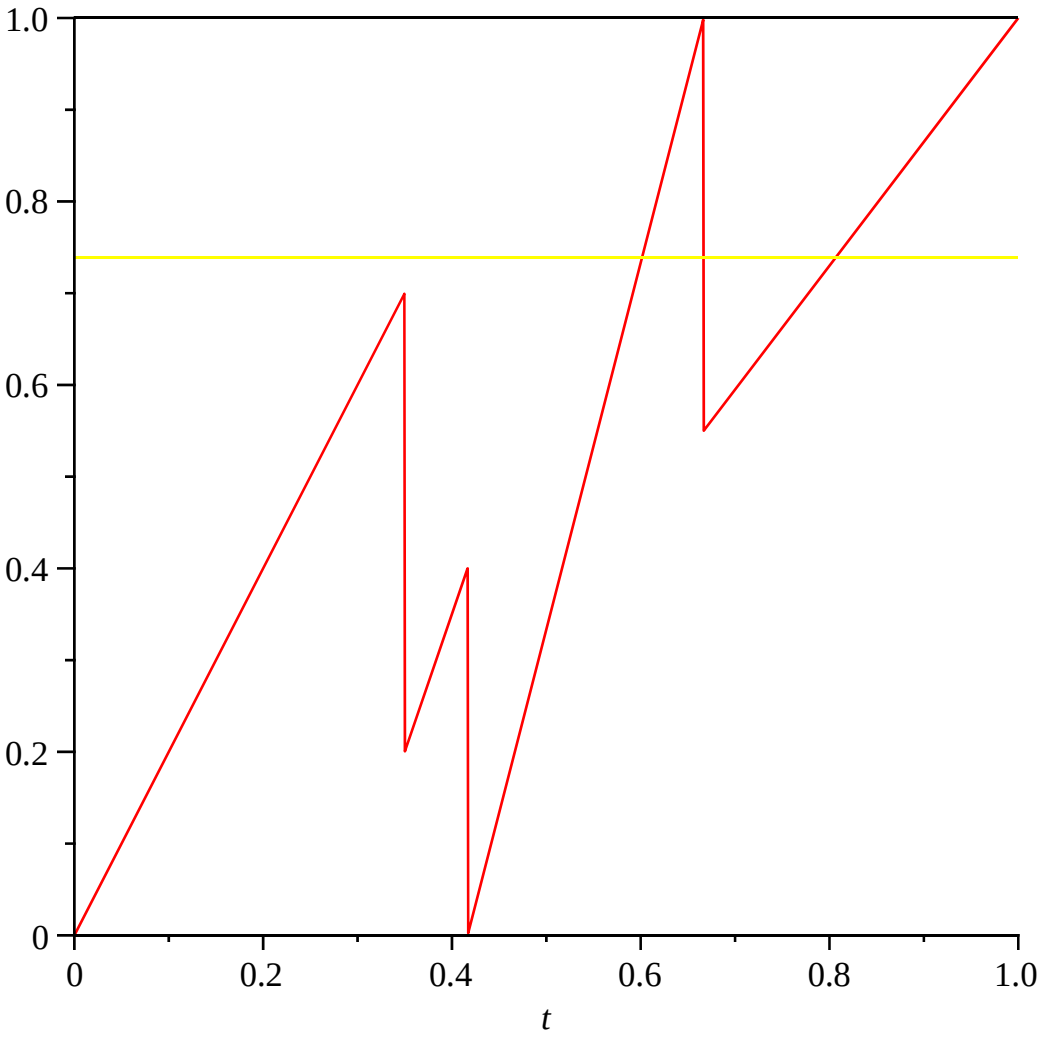
$su := 0$

1

3

4

$err_6 := 2.384518506\ 10^{-16}$

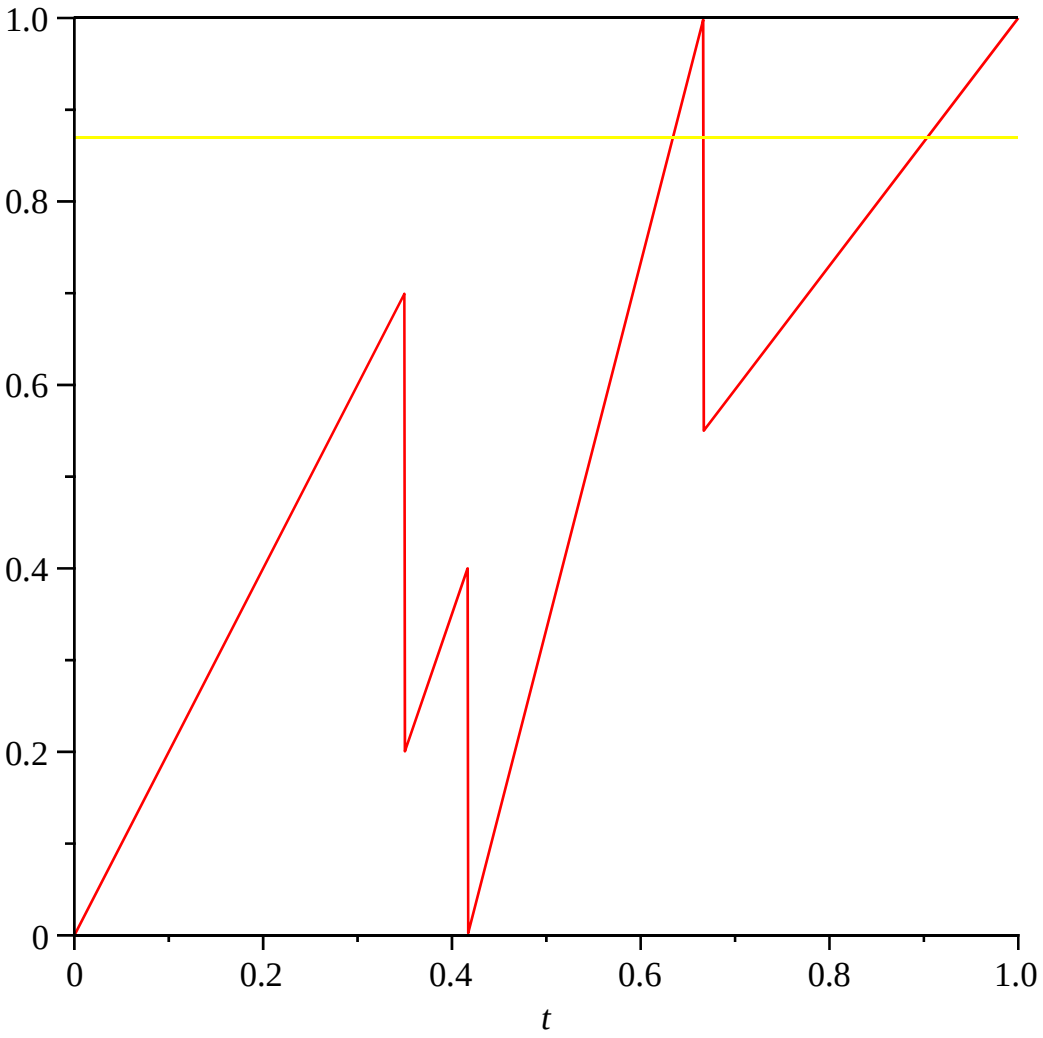


$su := 0$

3

4

$err_7 := 2.384518506\ 10^{-16}$

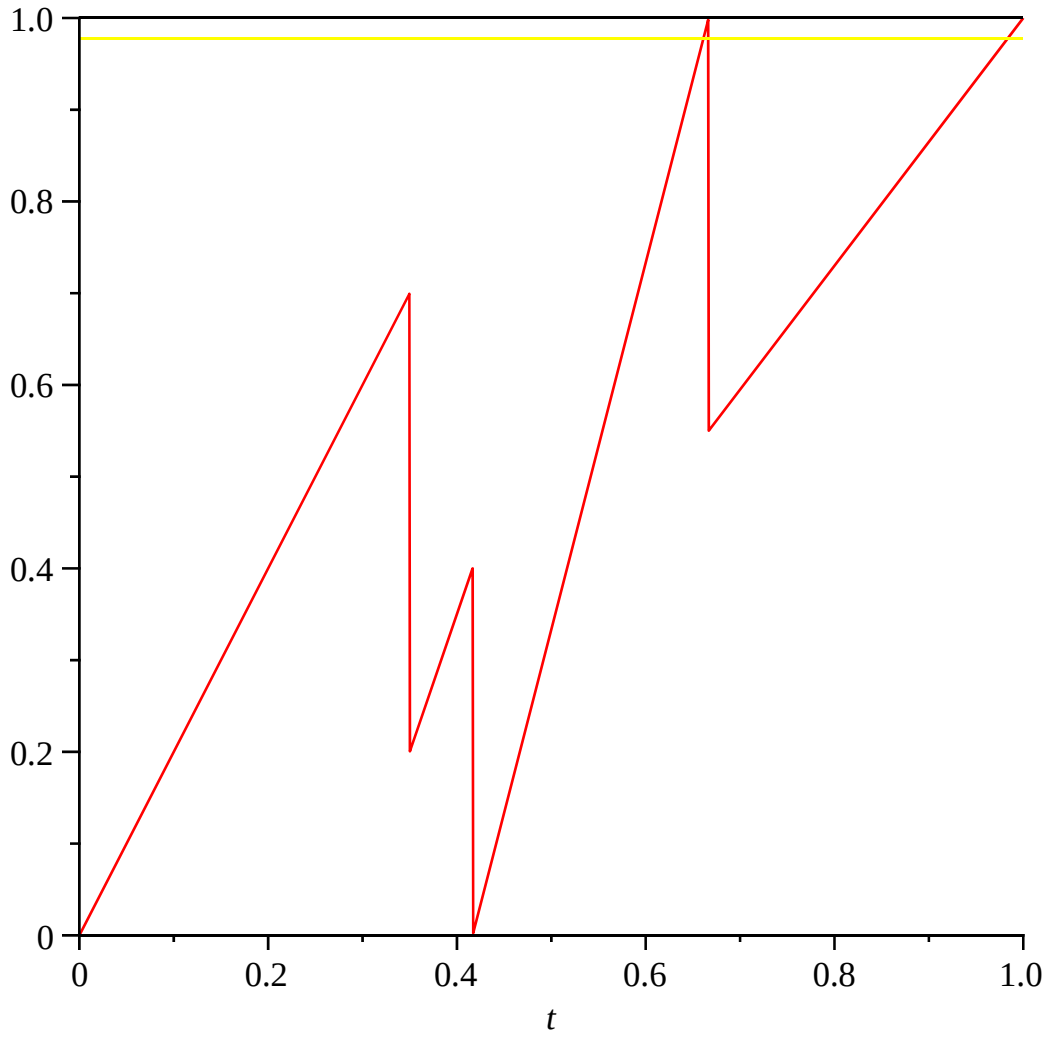


$su := 0$

3

4

$err_8 := 2.384518506\ 10^{-16}$



$su := 0$

3

4

$err_9 := 1.722837568\ 10^{-16}$

$y =, 0.0193139816$

$err[, 0,] =, -1.009208108\ 10^{-16}$

$y =, 0.1022424170$

$err[, 1,] =, -1.009208108\ 10^{-16}$

$y =, 0.2800187484$

$err[, 2,] =, -1.009208108\ 10^{-16}$

$y =, 0.3427552057$

$err[, 3,] =, -1.009208108\ 10^{-16}$

$y =, 0.4842622684$

$err[, 4,] =, -1.009204100\ 10^{-16}$

$y =, 0.5412286286$

$err[, 5,] =, -1.009204100\ 10^{-16}$

$y =, 0.6996417214$

$$err[,\ 6,\ ]=,\ 2.384518506\ 10^{-16}$$

$$y\ =,\ 0.7386408307$$

$$err[,\ 7,\ ]=,\ 2.384518506\ 10^{-16}$$

$$y\ =,\ 0.8694607189$$

$$err[,\ 8,\ ]=,\ 2.384518506\ 10^{-16}$$

$$y\ =,\ 0.9773012980$$

$$err[,\ 9,\ ]=,\ 1.722837568\ 10^{-16}$$

>