```
eps = 0.1;
k = 0.5;
xc = -3;
N=32;
t1=1e-2;

tp = linspace(0, N, N+1);
tp = -cos(pi.*tp/N);

D1 = zeros(N+1,N+1);
for i=1:N+1
    for j=1:N+1
        if i~=j
           D1(i,j) =  (-1).^(i+j)./(tp(i)-tp(j));
           if j==1 || j==N+1
              D1(i,j) = 0.5.*D1(i,j);
           end
           if i==1 || i==N+1
              D1(i,j) = 2.*D1(i,j);
           end
        else
            D1(i,j) = -tp(i)./(2.*(1-tp(i).^2));
        end
    end
end
D1(1,1) = -(2.*N.^2+1)./6;
D1(N+1,N+1) = (2.*N.^2+1)./6;

D1 = 0.2.*D1;
D2 = D1*D1;

% RKM to get initial condition at t=1. error for each step is
 O(delT^3),
% so by using delT=0.0001 and 100 steps we have error O(1e-10), which
 is
% much less than the error for Crank-Nicolson: O(delT^2) using
 delT=0.01
% and 100 steps yields truncation error O(0.01). Therefore we can use
 this
% Runge-Kutta approximate without disrupting the C-N estimate much.
delT = t1./100;
N2 = t1./delT;
ui = @(x) k.*(1-tanh(k.*(5.*x-xc)./(2.*eps)));
ui = ui(tp);
ti = 0;
f = @(u,t) eps.*(D2*u')-u'.*(D1*u');
for a=1:N2
    k1 = delT.*f(ui,ti);
    k2 = delT.*f(ui+k1'./2,ti+delT./2);
    ui = ui+k2';
    ti = ti+delT;
end
```

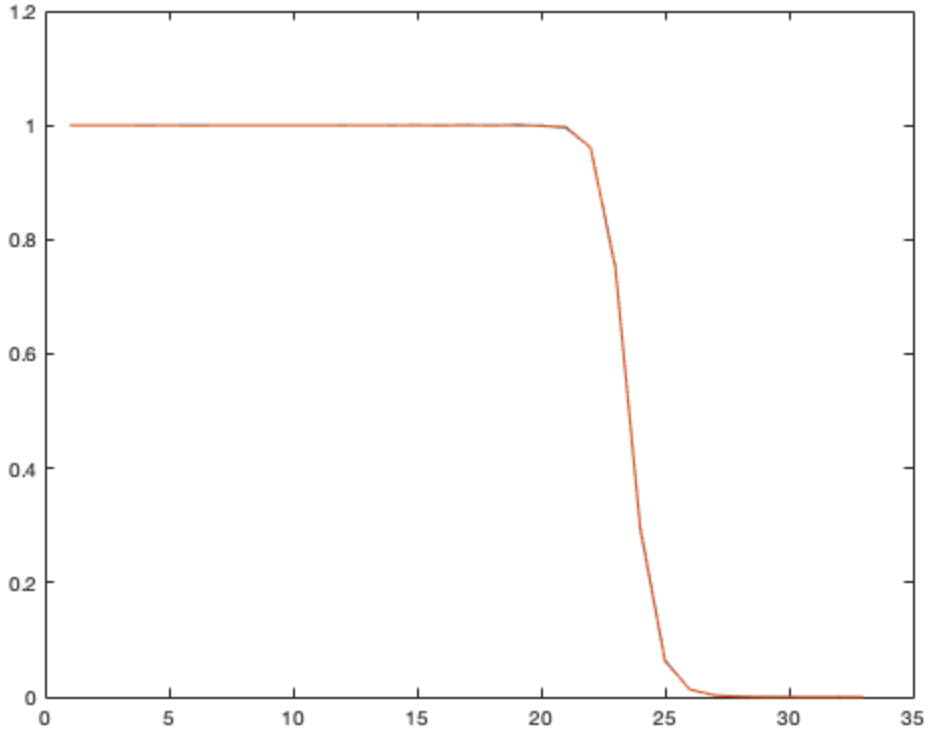
```
% end RKM. initial condition at t=1 is stored in ui

delT=t1;
ut = @(x) k.*(1-tanh(k.*(5.*x-xc)./(2.*eps)));
um1=ut(tp);
un=ui;
A=eye(N+1)-eps.*D2.*delT;
    for i=1:N+1
       A(1,i) = 0;
       A(N+1,i) = 0;
    end
A(1,1) = 1;
A(N+1,N+1) = 1;
for a=2:(12./delT)
    rhs = um1+(eps.*delT.*D2*um1')'-(2.*delT.*un'.*(D1*un'))';
    rhs(1) = k.*(1-tanh(k.*(5.*tp(1)-xc-k.*delT.*a)./(2.*eps)));
    rhs(end) = k.*(1-tanh(k.*(5.*tp(end)-xc-k.*delT.*a)./(2.*eps)));
    up1 = A\rhs';
    um1=un;
    un=up1';
end
actual = @(x) k.*(1-tanh(k.*(5.*x-xc-k.*12)./(2.*eps)));
figure
plot(actual(tp))
hold on
plot(un)

max(abs(un-actual(tp)))


ans =

    0.0027
```



Published with MATLAB® R2018a