```
function [u,q] = AdaptiveSmoothingUpwind(f, lambda, sigma)
  %This function implements an alteration of the upwind TV scheme that
  %smooths less near edges as described in "Variable Exponent, Linear Growth
  %Functionals in Image Restoration" by Yunmei Chen, Stacey Levine, Murali Rao
  %This matlab code was written by Katie Heaps

  %Input is noisy image, lambda (optional), and standard deviation (optional)
  %Output is improved image

  if(nargin<1)
      error('requires input image');
  end
  if(nargin<2)
      lambda = 35;
  end
  if(nargin<3)
      sigma = 20;
  end

  %initializations
  f = double(f);
  u = f;
  tau = 1/32;           %step size
  scale = .05;         %scale for Gaussian
  k = .0075;           %parameter edge detector
  [M,N] = size(f);     %M,N = size
  gauss = size(M,N);

  %boundary condition = zero padding
  g=zeros(M+2,N+2);
  g(2:M+1,2:N+1) = f;


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %Calculate q
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  %Calculate Gaussian
  for i=1:M
      for j=1:N
          gauss(i,j) = sqrt(2*pi)*scale*exp(-2*pi^2*scale^2*(i^2+j^2));
      end
  end

  %%perform convolution

  %calculate and shift FFT
  F = fftshift(fft2(f));
  GAUSS = fftshift(fft2(gauss));

  %performs multiplication
  R = F.*GAUSS;

  %calculate and shift inverse of result
  conv = ifft2(R);
  conv = real(conv);

  for i=1:M
      for j=1:N
          conv(i,j) = conv(i,j)*(-1)^(i+j);
      end
  end


  %calculate q and q*
  q1 = zeros(M+2,N+2);
  q1(2:M+1,2:N+1) = conv;
  gradq1 = zeros(M+2,N+2,2);
  q = zeros(M+2,N+2);
  qstar = zeros(M+2,N+2);


  gradq1(2:M+1,2:N+1,1) = q1(3:M+2,2:N+1)-q1(1:M,2:N+1);
```

```
  gradq1(2:M+1,2:N+1,2) = q1(2:M+1,3:N+2)-q1(2:M+1,1:N);
  magq1(:,:) = sqrt(gradq1(:,:,1).^2+gradq1(:,:,2).^2);

  q = 1+1./(1+k*magq1.^2);
  qstar = q./(q-1);


  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %Perform Gradient Descent
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


  w = zeros(M+2,N+2);
  mag = zeros(M+2,N+2);
  divp = zeros(M+2,N+2);
  grad = zeros(M+2,N+2,4);
  p = zeros(M+2,N+2,4);
  pn = zeros(M+2,N+2,4);

  for iter = 1:200

      %calculate gradient
      w = divp - g/lambda;

      grad(2:M+1,2:N+1,1) = w(2:M+1,2:N+1) - w(3:M+2,2:N+1);
      grad(2:M+1,2:N+1,2) = w(2:M+1,2:N+1) - w(1:M,2:N+1);
      grad(2:M+1,2:N+1,3) = w(2:M+1,2:N+1) - w(2:M+1,3:N+2);
      grad(2:M+1,2:N+1,4) = w(2:M+1,2:N+1) - w(2:M+1,1:N);

      mag = (p(:,:,1).^2+p(:,:,2).^2+p(:,:,3).^2+p(:,:,4).^2).^.5;

      %calculate p^(n+1/2)
      pn(:,:,1) = p(:,:,1) - tau*(grad(:,:,1)+1/lambda.*mag.^(qstar-2).*p(:,:,1));
      pn(:,:,2) = p(:,:,2) - tau*(grad(:,:,2)+1/lambda.*mag.^(qstar-2).*p(:,:,2));
      pn(:,:,3) = p(:,:,3) - tau*(grad(:,:,3)+1/lambda.*mag.^(qstar-2).*p(:,:,3));
      pn(:,:,4) = p(:,:,4) - tau*(grad(:,:,4)+1/lambda.*mag.^(qstar-2).*p(:,:,4));



      %take only positive differences
      pn = max(pn,0);

      %calculate magnitue of pn1
      mag = (pn(:,:,1).^2+pn(:,:,2).^2+pn(:,:,3).^2+pn(:,:,4).^2).^.5;

      %if magnitute of p is greater than 1, divide by magnitude
      p(:,:,1) = pn(:,:,1)./max(1,mag);
      p(:,:,2) = pn(:,:,2)./max(1,mag);
      p(:,:,3) = pn(:,:,3)./max(1,mag);
      p(:,:,4) = pn(:,:,4)./max(1,mag);

      %calculate divp
      divp(2:M+1,2:N+1) = (p(2:M+1,2:N+1,1) - p(1:M,2:N+1,1)) + (p(2:M+1,2:N+1,2)-p(
3:M+2,2:N+1,2)) + (p(2:M+1,2:N+1,3) - p(2:M+1,1:N,3)) + (p(2:M+1,2:N+1,4)- p(2:M+1,3
:N+2,4));

      %calculate new lambda (as described on page 93 of An Algorithm for
      %Total Minimization and Applications
      %lambda =  ((M*N)^.5*sigma / (sum(sum(divp(2:M+1,2:N+1).^2)))^.5 );


  end

  %calculate denoised image
  u = u - lambda*divp(2:M+1,2:N+1);



end
```