CURSO DE DOCTORADO: Procesamiento de Imágenes Biomédicas - 2010
UTN / FRBA
Examen Final

Pablo Slavkin

Suavizado de imágenes por difusión inhomogenea

El método de difusión inhomogenea es comúnmente utilizado en el pre-procesado de imágenes médicas para reducir el ruido preservando los bordes. A diferencia del suavizado mediante kernels Gaussianos, se busca que la difusión de intensidades desde o hacia píxeles vecinos se produzca fuertemente en regiones de bajo contraste, pero se haga prácticamente nula en regiones de alto contraste, usualmente asociadas con bordes o interfaces entre tejidos o estructuras diferenciadas.

Se provee una imagen 2D de un aneurisma cerebral obtenida a partir de una imagen 3DRA (*Three-Dimensional Rotational Angiography*) utilizando la técnica de MIP (*Maximum Intensity Projection*). La imagen consiste de 120x120 píxeles de dimensión isotrópica de 0.04 cm. La intensidad de cada píxel está almacenada en un archivo binario sin formato (*raw data*) y es representada por números enteros sin signo de 16 bits (*unsigned short*). La intensidad de la imagen puede ser pensada como un campo escalar ***θ(x,y).***

a) Implementar un código mediante diferencias finitas para resolver la siguiente ecuación escalar en dos dimensiones:

$$\frac{\partial \theta}{\partial t} = \nabla.(D\ \nabla\theta)$$

donde

$$D = D(|\nabla\theta|)$$

es el coeficiente inhomogeneo de difusión que en general es función del gradiente del campo escalar. Use un esquema de integración temporal explícito (forward Euler o Runge-Kutta) y un esquema de diferencias centrales de segundo orden para la discretización espacial. Para los bordes, imponga condiciones de contorno tipo Neumann dadas por:

$$\frac{\partial \theta}{\partial n} = 0$$

donde n es la normal a la frontera del dominio.

Elija un criterio para finalizar la integración, por ejemplo, que el promedio de los errores cuadráticos entre dos soluciones sucesivas sea menor que una dada tolerancia (por ejemplo, *tol = 0.005*).

Respuestas:

a) Aunque no se demuestra, la ecuacion numerica para resolver lo solicitado en el apartado es la siguiente:

$$\frac{\Phi_{i,j}^{k+1} - \Phi_{i,j}^{k}}{\Delta t} = \Delta\left[D(\Phi) * \overline{\nabla}\Phi \right]$$

$$\Phi_{i,j}^{k+1} = \Phi_{i,j}^{k} + \frac{\Delta t}{(\Delta x)^2}\left[D^{k}_{i+\frac{1}{2},j}\left(\Phi^{k}_{i+1,j} - \Phi^{k}_{i,j}\right) - D^{k}_{1-\frac{1}{2},j}\left(\Phi^{k}_{i,j} - \Phi^{k}_{i-1,j}\right)\right] +$$

$$\frac{\Delta t}{(\Delta y)^2}\left[D^{k}_{i,j+\frac{1}{2}}\left(\Phi^{k}_{i,j+1} - \Phi^{k}_{i,j}\right) - D^{k}_{1,j-\frac{1}{2}}\left(\Phi^{k}_{i,j} - \Phi^{k}_{i,1-1}\right)\right]$$

Los coeficientes D dependen del modulo del gradiente de la imagen, pero evaluados en puntos intermedios en los cuales no se tiene informacion directa. Para calcular el gradiente en los puntos intermedios de la imagen se usan 2 artilugios en funcion de la direccion de la derivada:

a) En el caso de las derivadas en i+1/2 en direccion x, se usan diferencias centrales y se obtiene:

$$\frac{\vartheta \phi^k}{\vartheta x}_{i+\frac{1}{2},j} = \frac{\phi^k_{i+1,j} - \phi^k_{i,j}}{\Delta x}$$

b) para calcular los coeficientes en i+1/2 pero en direccin y se hace:

$$\frac{\vartheta \phi^k}{\vartheta y}_{i+\frac{1}{2},j} = \frac{\phi^k_{i+\frac{1}{2},j+\frac{1}{2}} - \phi^k_{i+\frac{1}{2},j-\frac{1}{2}}}{\Delta y}$$

pero como no se cuentan con los puntos intermedios de la imagen se los estima usando interpolacion bilineal y se obtiene:

$$\phi^k_{i\pm\frac{1}{2},j\pm\frac{1}{2}} = \frac{1}{4}\left[\phi^k_{i,j} + \phi^k_{i\pm1,j} + \phi^k_{i,j\pm1} + \phi^k_{i\pm1,j\pm1}\right]$$

Eligiendo siempre los 4 vecinos al punto intermedio en cuestion.
Se realza el mismo procedimiento para los otros tres puntos de la ecuacion, (i-1/2,j), (i,j+1/2(y (i,j-1/2) para obtener todos los datos necesarios para la codificacion.
Se codifico la ecuacion para Octave V4.2.4 con el siguiente conjunto de funciones:

```
function [Out]=Heat(Image)
In=Load_Image(Image);                   %elige una imagen en funcion de In.

L=[min(min(In)),max(max(In))];          %setea los limites de colores de la imagen.
subplot(2,4,1:8);plot(0);
subplot(2,4,1);imagesc(In,L);           %muestra la imagen original
p=2;k=1;                                %p para pictures, hasta 8. k para el tiempo
Last_T=T=0;
 do                                     %procesa avanzando en el tiempo 'k' hasta Tol
  Out=One_Step(In);                     %ejecuta un paso en el tiempo
  k,k++;                                %simplemente indica el k en cuestion seguido por...
  Last_T=T;
  T=Tol(Out,In)                         %tolerancia usada para definir el stop.
  In=Out;                               %inicializa la entrada como la salida anterior...
  if (T<Params(5)*(9-p))                %cada tanto muestra como va.. hasta 7 fotos
  subplot(2,4,p);imagesc(Out,L);        %muestra la foto
   p,p++;                               %muestra que se imprimio
  end;
  until (T<Params(5) || \               %cuando se hace menor que la tolerancia o
          (k>2 && T>Last_T));           %la tolerancia comenzo a subir, para...
end
```

```
function [Out]=Load_Aneurysm()          %carga la imagen de una aneurisma en formato raw
X1=Params(7);X2=Params(8);
Y1=Params(9);Y2=Params(10);             %usado para fraccionar la imagen y procesar parte de esta
nx=ny=Params(11);                        %tamanio de la imagen fija en raw
[fileID , message] = fopen ('aneurysm.raw');          %handler del archivo
A = fread (fileID,'ushort');            %carga la imagen
                                                % se convierte el vector raw en matriz
Out = zeros (X2-X1,Y2-Y1);                      %inicializa una imagen vacia
for i=X1:X2;                                    %recorre la imagen en X
   for j=Y1:Y2;                         %recorre la imagen en Y
      Out (i-X1+1,j-Y1+1) = A ((i-1)*ny+j);%ubica los datos en forma matricial
   end;
end;
```

```
---------------------------------------------------------------------------------------------------
function [Out]=Load_Image(Image)          %devuelve 1 imagenes o patrones de test
Out=zeros(9,9);
if        Image==1 Out=Load_Aneurysm();
elseif    Image==2 Out(5,5)=1;
elseif  Image==3 Out(5,:)=1;
elseif  Image==4 Out(:,5)=1;
elseif  Image==5 Out(5,:)=1; Out(:,5)=1;
end;
end
```

```
function [Out] = One_Step(In)
[nx,ny]=size(In); %calcula el tamanio de la imagen para recorrerla
Out=zeros(nx,ny);          %inicializa una imagen del mismo tamanio que la entrada para operar
x=y=Params(6);             %espaciado entre pixeles; dato de la imagen.
t=Params(1);               %espaciado temporal, se establece para asegurar la estabilidad

 for i=2:nx-1              %saltea la 1era y ultima columna porque se usa Newman contorno
  for j=2:ny-1             %saltea la 1era y ultima fila porque se usa Newman en contorno
   Out(i,j)=In(i,j) + t/x^2 * \
                           (D(Grad_Half_X(In,i,j))   * (In(i+1,j)-In(i,j))    - \
                            D(Grad_Half_X(In,i-1,j)) * (In(i,j)-In(i-1,j)))   + \
                      t/y^2 * \
                           (D(Grad_Half_Y(In,i,j))   * (In(i,j+1)-In(i,j))  - \
                            D(Grad_Half_Y(In,i,j-1)) * (In(i,j)-In(i,j-1)));
  end;
 end;
Out(1,:)=Out(2,:);Out(nx,:)=Out(nx-1,:);Out(:,1)=Out(:,2);Out(:,ny)=Out(:,ny-1); %Newman

end
```

```
function [Out]=Params(In)
 if       In==1 Out=0.0002;          %delta t. :especificado para asegurar la estabilidad
 elseif In==2 Out=5.3;               %gama     :parametro de la funcino D
 elseif In==3 Out=5;                 %alfa     :parametro de la funcion D
 elseif In==4 Out=42000;             %c        :parametro de la funcion D
 elseif In==5 Out=50;                %TOL      :tolerancia, determina el fin del metodo.
 elseif In==6 Out=0.04;              %delta xy :espaciado entre pixeles dato de la iagen
 elseif In==7 Out=1;                 %X1       :limite ingerior  en X de la imagen
 elseif In==8 Out=120;               %X2       :lmimte superior  en X de la imagen
 elseif In==9 Out=1;                 %Y1       :liminte inferior en Y de la imagen
 elseif In==10 Out=120;              %Y2       :limite superior  en Y de la imagen
 elseif In==11 Out=120;              %nx ny    :tamanio de la imagen de aneurisma, Fijo
 elseif In==12 Out=3;                %D        :tipo de funcion D, 1, 2 o 3 parametros
end;
end
```

```
function [Out] = Grad_Half_Y(In,i,j)      %gradiente en puntos intermedios de Y + 1/2
x=y=Params(6);                            %carga el espaciado entre pixeles
Out(2)=(In(i,j+1) - In(i,j))/y;           %derivada en Y en j+1/2
Out(1)=1/((4*x)*(\                        %derivada en X en j+1/2
          (In(i,j)+In(i+1,j)+In(i,j+1)+In(i+1,j+1)) - \
          (In(i,j)+In(i-1,j)+In(i,j+1)+In(i-1,j+1)))+1e-20);
end
```

```
function [Out] = Grad_Half_X(In,i,j)      %gradiente en puntos intermedios de X + 1/2
```

```
x=y=Params(6);                          %carga el espaciado entre pixeles
Out(1)=(In(i+1,j) - In(i,j))/x;         %derivada en X en 1+1/2
Out(2)=1/((4*y)*(\                      %derivada en Y en 1+1/2
        (In(i,j)+In(i+1,j)+In(i,j+1)+In(i+1,j+1)) - \
        (In(i,j)+In(i+1,j)+In(i,j-1)+In(i+1,j-1)))+1e-20);
end
```

```
function [Out]=Tol(Actual,Last)
[nx,ny]=size(Actual);
Out=sqrt(sum(sum((Last-Actual).^2)/(nx*ny)));    %raiz de la suma de las diferencias cuadrada
                                                 % de los elementos dividido la dimension de
                                                 %la matrz
end
```

```
function [Out]=D(In)
   if(Params(12)==1) Out=D1(In);
elseif(Params(12)==2) Out=D2(In);
elseif(Params(12)==3) Out=D3(In);

end
```

```
function [Out]=D3(In)
Mod=sqrt((In(1)^2+In(2)^2));                    %raiz de la suma de los cuadrados del gradiente
Out=1-exp(-Params(2)*(Params(4)/Mod)^Params(3));  %Modelo de 3 parametros c, alfa y gama...
end
```

```
function [Out]=D2(In)
Mod=sqrt((In(1)^2+In(2)^2));                    %raiz de la suma de los cuadrados del gradiente
Out=1/(1+((Mod/Params(4))^(1+Params(3))));  %Modelo de 2 parametros = c y alfa...
end
```

```
function [Out]=D1(In)
Mod=sqrt((In(1)^2+In(2)^2));        %raiz de la suma de los cuadrados del gradiente
Out=exp(-Mod^2/Params(4)^2);    %Modelo de 1 parametro = c...
end
```

Para ejecutarla se tipea Heat(1). En el archivo Patams.m se establecen todos los parametros del sistema de manera que modificando solo ese archivo se puede controlar el modo de procesamiento..

Permite elegir entre modelos para D, de 1, 2 y 3 parametros, permite cargar varios tipos de imagenes o patrones de prueba, se puede procesar solo un trozo de la imagen para agilizar los tiempos de procesamoiento en la etapa de pruebas, entre otras cosas.

Para determinar el final de la simulacion, ademas de usar la diferenca cuadratica entre dos pasos suscesivos, se agrega la condicion de que esta diferencia sea monotona decreciante, ya que se encontro que llegado a cierto punto, la diferencia comienza a incrementarse y/o oscilar y nunca alcanza el valor solicitado.

c) Suavice la imagen utilizando difusión inhomogenea con un coeficiente de difusión dependiente del gradiente de intensidades de acuerdo a:

$$D(|\nabla\theta|) = 1 - e^{-\lambda\left(\frac{c}{|\nabla\theta|}\right)^{\alpha}}$$

con λ=3.3, α=4 y c=10.000.

Si grafica ***D*** como función del gradiente verá que es una función asintótica con ***D(0)=1***. El valor de ***c*** es un valor característico para el gradiente, para el cual la difusión cayó xa $e^{-\lambda}$. Cambios en el valor de ***c*** producirán mayor o menor difusión en bordes originalmente más difusos. Investigue cómo se altera la solución para distintos valores de ***c***.

Para suavizar la imagen primero se setea en el archivo params.m los siguientes valores: y se smula solo un trozo de la imagen. Luego de comprobar que los resultados sean satisfactorios se procede con la imagen completa

---

```
function [Out]=Params(In)
 if     In==1 Out=0.0002;          %delta t. :especificado para asegurar la estabilidad
 elseif In==2 Out=3.3;             %gama     :parametro de la funcino D
 elseif In==3 Out=4;               %alfa     :parametro de la funcion D
 elseif In==4 Out=19000;           %c        :parametro de la funcion D
 elseif In==5 Out=10;              %TOL      :tolerancia, determina el fin del metodo.
 elseif In==6 Out=0.04;            %delta xy :espaciado entre pixeles dato de la iagen
 elseif In==7 Out=35;              %X1       :limite ingerior  en X de la imagen
 elseif In==8 Out=55;              %X2       :lmimte superior  en X de la imagen
 elseif In==9 Out=35;              %Y1       :liminte inferior en Y de la imagen
 elseif In==10 Out=55;             %Y2       :limite superior  en Y de la imagen
 elseif In==11 Out=120;            %nx ny    :tamanio de la imagen de aneurisma, Fijo
 elseif In==12 Out=3;              %D        :tipo de funcion D, 1, 2 o 3 parametros
end;
end
```

---

se tipea
Heatt(1)

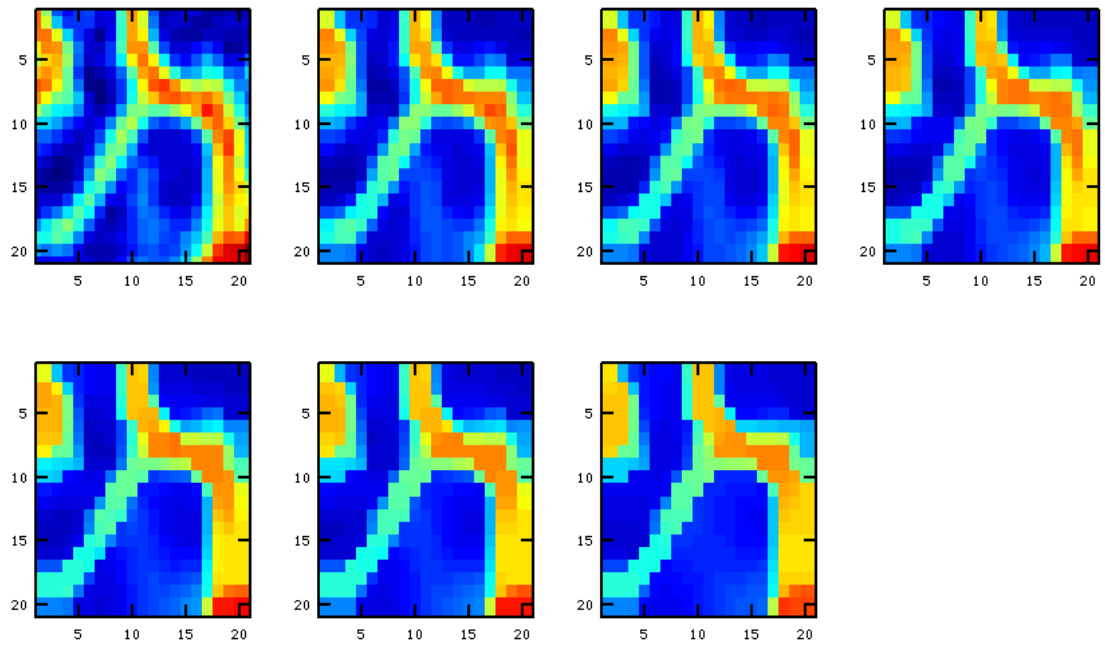
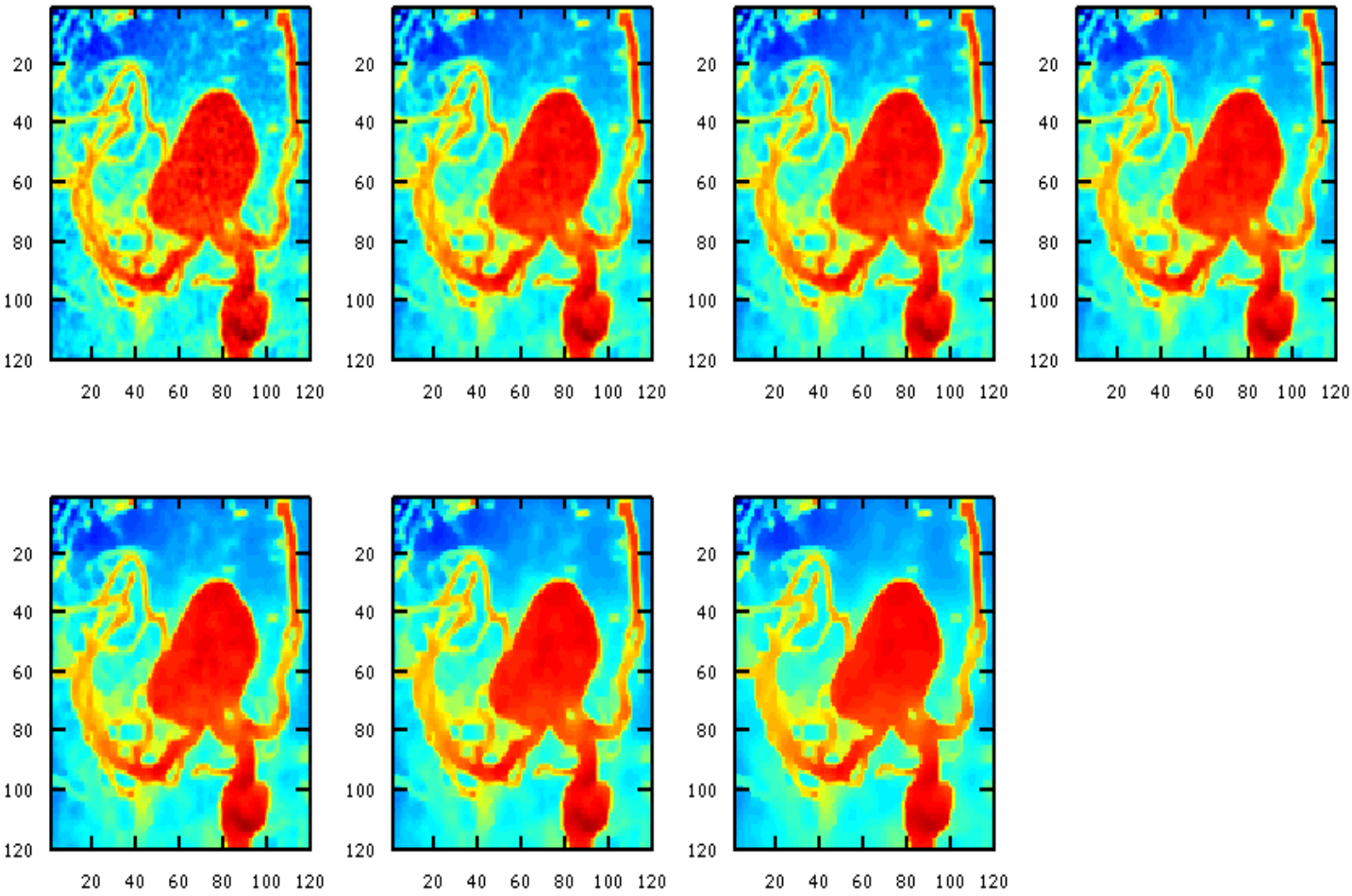
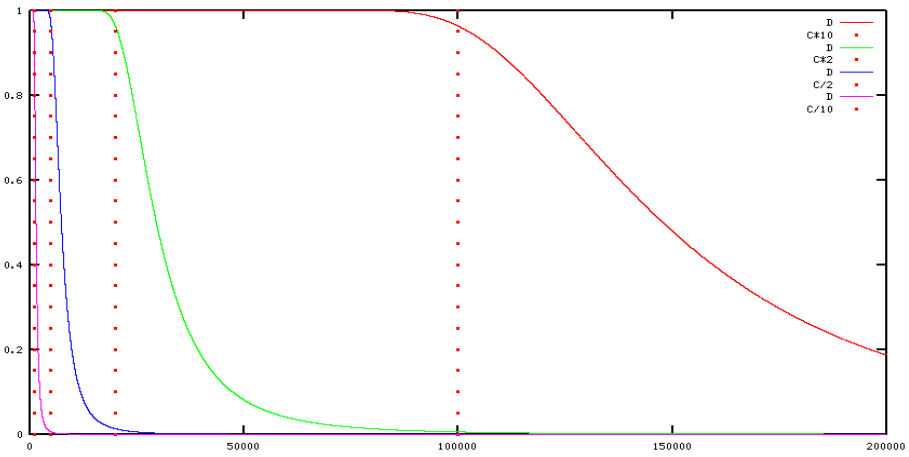


Imagen completa:



Grafica de D para distintos valores de C:

C=1000
C=5000
C=20000
C=100000



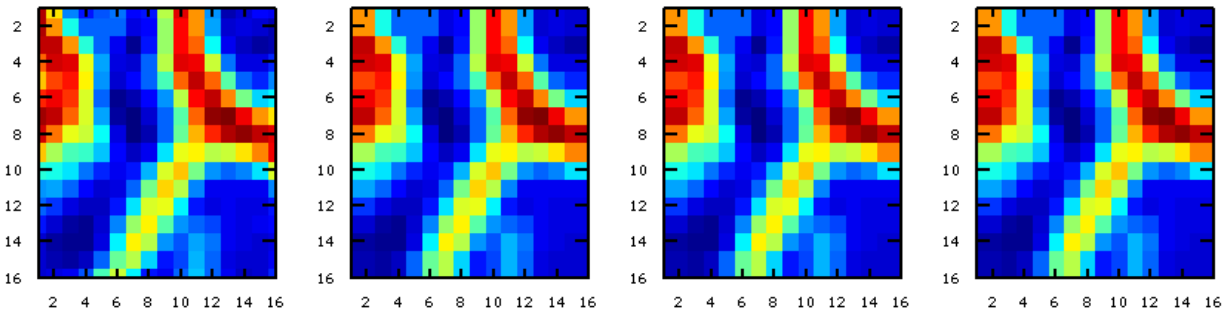
Codigo utilizado:

```
function [Out]=Plot_D3(a,g,c)
In=0:20*c;
Out1=1-exp(-g*(10*c./In).^a);
Out2=1-exp(-g*(2*c./In).^a);
```

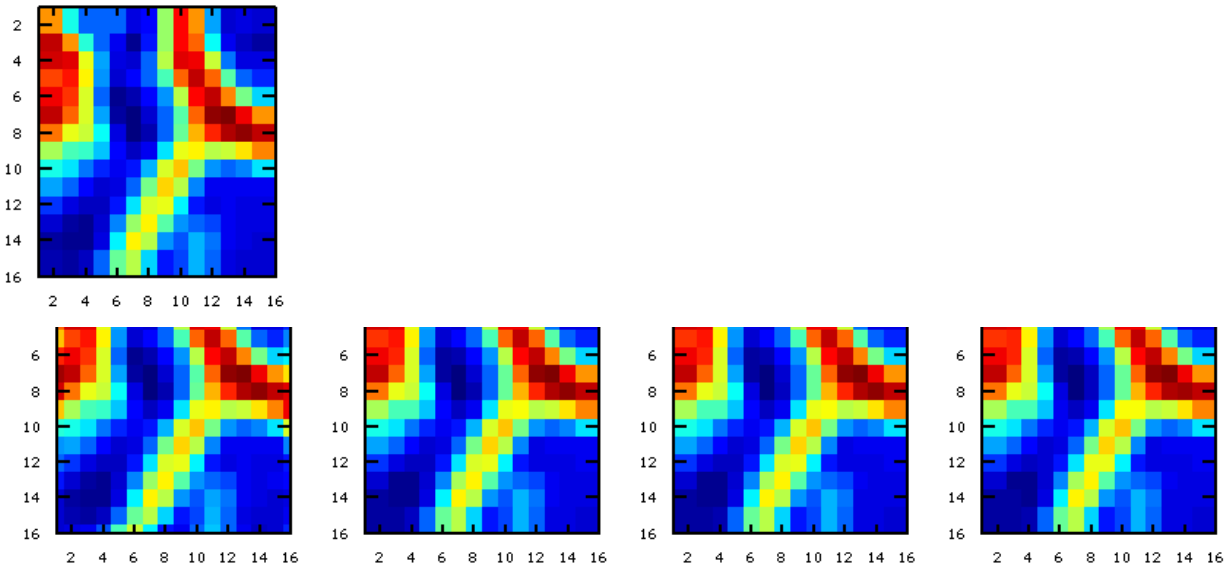
```
Out3=1-exp(-g*(c/2./In).^a);
Out4=1-exp(-g*(c/10./In).^a);

Vert=0:0.05:1;
C=zeros(size(Vert));
C(:)=c;

subplot(1,1,1);
plot(In,Out1,"-1;D;",C.*10,Vert,"*1;C*10;","markersize",2,\
    In,Out2,"-2;D;",C.*2, Vert,"*1;C*2; ","markersize",2,\
    In,Out3,"-3;D;",C./2, Vert,"*1;C/2; ","markersize",2,\
    In,Out4,"-4;D;",C./10,Vert,"*1;C/10;","markersize",2);

end
```

Resultados de procesamiento con estos valores de C:

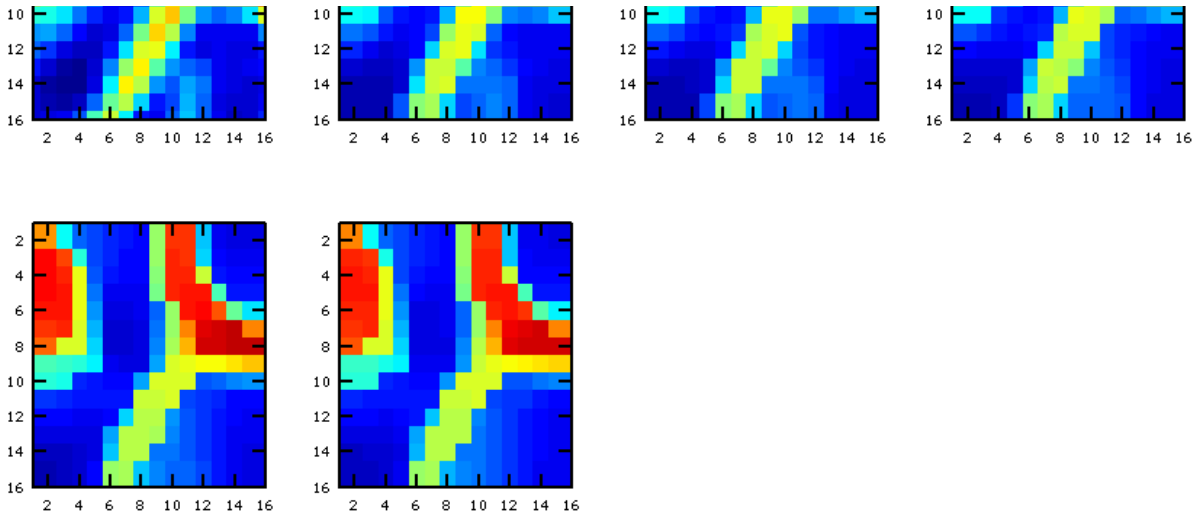
C=1000



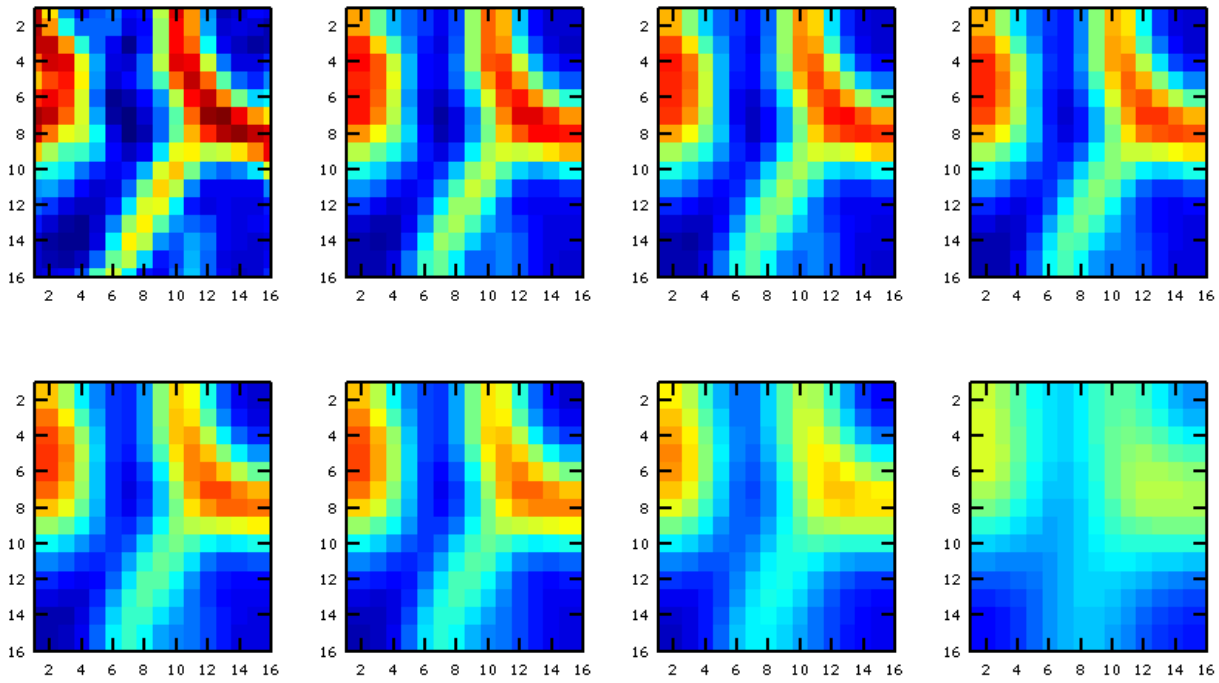
C=5000



C=20000



C=100000

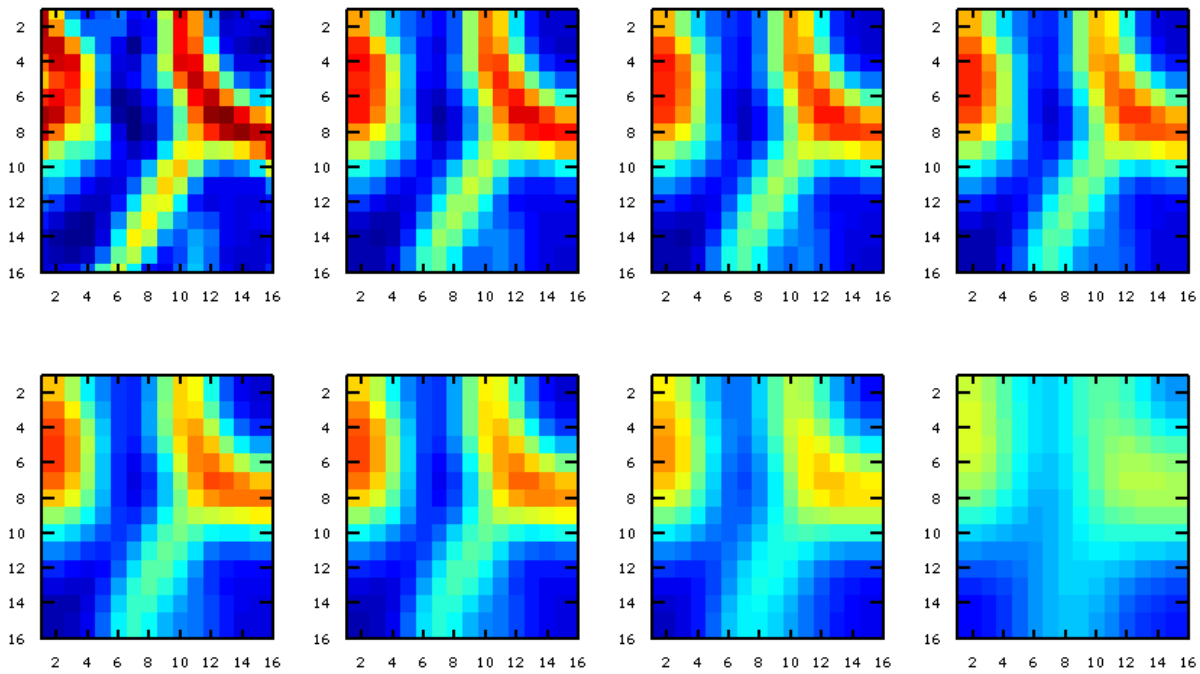


Se puede ver que a medida que se elige un C mayor, el procesamiento pierde la capacidad de discriminar los bordes y termina comportandose como una difusion homogenea, mientras que para un C muy bajo directamente no se realiza filtrado alguno

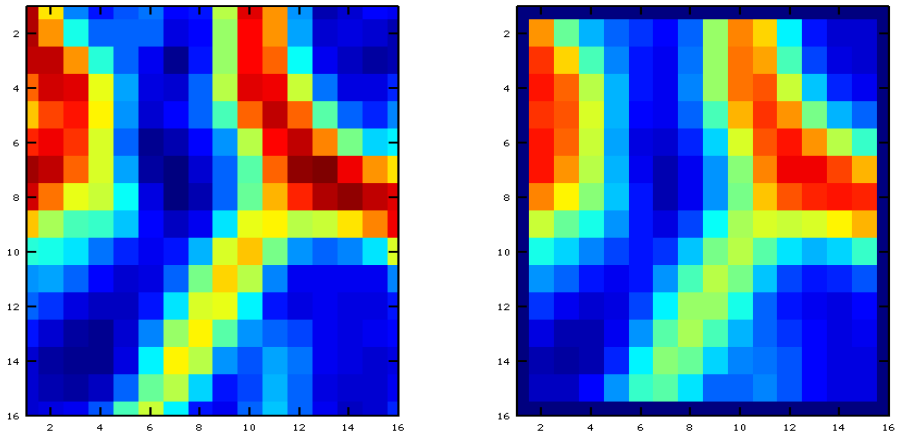
b) Para el caso *D=1*, se tiene una difusión homogenea. Demuestre que para un dado número de iteraciones, se obtiene una solución similar a aquélla obtenida usando un kernel Gaussiano de dimensión 3x3. Discuta las ventajas de la difusión homogenea por sobre el suavizado por kernels Gaussianos.

Usando el siguiente codigo se simula un filtrado con D=1 y se lo compara con un filtrado gaussiando con un kernel de 3s3

```
function [Out]=Gaussian(In)
Kernel=fspecial('gaussian',3,1);
Out=filter2(Kernel,In);
L=[min(min(In)),max(max(In))];
subplot(1,2,1);
imagesc(In,L);
subplot(1,2,2);
imagesc(Out,L);
end;
```



Filtrado Gaussiano



Se puede ver que el resultado del fltro gaussiano es comparable con la foto 3 y 4 del filtrado por difusion homogenea.. Luego la difusion seguira filtrando hasta convertir la imagen en un unico color.

La ecuacion que aplica a cada pixel en la difusion isotropica es una combinacion lineal del pixel en cuestion y sus vecinos y el factor de escala es proporcional a las derivadas parciales asociadas. Mientras que el kernel gausiano es exactamente igual pero con un sigma dado fijo para toda la imagen. Se puede pensar que en cada pixel de la difusion se esta aplicando un kernel con un nuevo sigma. Dependiendo de la complejidad de la imagen puede que sea una ventaja ya que permite ajustar punto a punto el grado de difusion mientras que con el kernel se debe elegir de partida un valor.

A favor del kernel estara la simplicidad y agilidad del calculo comparado con la iteracion de calculos bastante mas complejos para la difusion.

Apendice A

Se agrega la funcion 'fspecial' debido a que no esta incluida en Octave pero es O.Source

---

```
function f = fspecial(varargin)

  [filtertype, sze, sigma, radius, len, angle] = checkargs(varargin(:));
```

```
    rows = sze(1); cols = sze(2);
    r2 = (rows-1)/2; c2 = (cols-1)/2;

    if strcmpi(filtertype, 'average')
            f = ones(sze)/(rows*cols);

    elseif strcmpi(filtertype, 'disk')
            [x,y] = meshgrid(-c2:c2, -r2:r2);
            rad = sqrt(x.^2 + y.^2);
            f = rad <= radius;
            f = f/sum(f(:));

    elseif strcmpi(filtertype, 'gaussian')
            [x,y] = meshgrid(-c2:c2, -r2:r2);
            radsqrd = x.^2 + y.^2;
            f = exp(-radsqrd/(2*sigma^2));
            f = f/sum(f(:));

    elseif strcmpi(filtertype, 'log')
            [x,y] = meshgrid(-c2:c2, -r2:r2);
            radsqrd = x.^2 + y.^2;
            f = -1/(pi*sigma^4)*(1-radsqrd/(2*sigma^2))...
                .*exp(-radsqrd/(2*sigma^2));
            f = f-mean(f(:));    # Ensure 0 DC

    elseif strcmpi(filtertype, 'laplacian')
            f = [1  1  1
                 1 -8  1
                 1  1  1];

    elseif strcmpi(filtertype, 'unsharp')
            f = -fspecial('log') + [0 0 0
                                    0 1 0
                                    0 0 0];

    elseif strcmpi(filtertype, 'sobel')
            f = [1 2 1; 0 0 0; -1 -2 -1];

    elseif strcmpi(filtertype, 'prewitt')
            f = [1 1 1; 0 0 0; -1 -1 -1];

    elseif strcmpi(filtertype, 'motion')
            # First generate a horizontal line across the middle
            f = zeros(sze);
            f(floor(len/2)+1,1:len) = 1;

            # Then rotate to specified angle
            f = imrotate(f,angle,'bilinear','loose');
            f = f/sum(f(:));

    else
            error('Unrecognized filter type');

    end

##---------------------------------------------------------------------
##
## Sort out input arguments, setting defaults and checking for errors.

function [filtertype, sze, sigma, radius, len, angle] = checkargs(arg)

# Set defaults
```

```
sze = [3 3];
sigma = 0.5;
radius = 5;
len = 9;
angle = 0;

narg = length(arg);

filtertype = arg{1};
if ~ischar(filtertype)
        error('filtertype must be specified as a string');
end

if strcmpi(filtertype, 'log')
  if narg == 1
          sze = [5 5];
  end
end

if strcmpi(filtertype, 'average')   | ...
   strcmpi(filtertype, 'gaussian')  | ...
   strcmpi(filtertype, 'log')
        if narg >= 2
           sze = arg{2};
           if isscalar(sze)
                   sze = [sze sze];
           end
        end
end

if strcmpi(filtertype, 'gaussian')  | ...
   strcmpi(filtertype, 'log')
        if narg >= 3
           sigma = arg{3};
        end
end

if strcmpi(filtertype, 'disk')
        if narg >= 2
           radius = arg{2};
        end
        sze = [2*radius+1 2*radius+1];
end

if strcmpi(filtertype, 'motion')
        if narg >= 2
           len = arg{2};
        end

        if narg >= 3
           angle = arg{3};
        end

        # Ensure size is odd so that there is a middle point
        # about which to rotate the filter by angle.
        if mod(len,2)
           sze = [len len];
        else
           sze = [len+1 len+1];
        end
end

if ~isscalar(len) | len < 1
```

```
        error('length must be a scalar >= 1');
end

if ~isscalar(angle)
        error('angle must be a scalar');
end

if ~isscalar(radius) | radius < 1
        error('radius must be a scalar >= 1');
end

if ~isscalar(sigma) | sigma < 0.5
        error('sigma must be a scalar >= 0.5');
end

if length(sze) > 2
        error('filter size must be a scalar or 2-vector');
end

if any(fix(sze) ~= sze)
        error('filter size must be integer');
end
```
