Procesamiento Digital de Imágenes (PDI)

Universidad Nacional de Asunción

M. Sc. José Luis Vázquez Noguera

jlvazquez@pol.una.py

3era Clase de Laboratorio

Capitulo 2 (Cont.)

Fundamentos

Contenido

- Optimización de Código

1. Vectorización
2. Matrices con valores preasignados

- Entrada y Salida de Datos
- Arreglos de Celdas
- Estructuras
- Filtrado en Matlab

Vectorización

El proceso de convertir un bucle for en una operación matricial o vectorial se llama vectorizar, en MATLAB es importante evitar los bucles for si se usan para hacer operaciones sobre elementos de un vector, en este caso la programación es ineficiente debido a que los bucles for son interpretados en Matlab lo cual hace lento el proceso, este bucle se debe utilizar como última opción

Un ejemplo sencillo

Supongamos que queremos generar una función 1-D de la forma

$$f(x) = A\sin(x/2\pi)$$

para $x = 0, 1, 2, \ldots, M-1.$

Un ciclo for para implementar este calculo es

```
for x=1:M %Array indices in MATLAB cannot be 0.
    f(x)=A*sin((x-1)/(2*pi));
end
```

El código vectorizado

```
x=0:M-1;
f=A*sin(x/(2*pi));
```

Indexación 2D

```
[C,R]=meshgrid(c,r)
```

```
>> c=[0 1];
>> r=[0 1 2];
>> [C,R]=meshgrid(c,r)
C =
     0     1
     0     1
     0     1
R =
     0     0
     1     1
     2     2
>> h=R.^2+C.^2
h =
     0     1
     1     2
     4     5
```

Comparación entre ciclos for y vectorización

```
function [rt,f,g]=twodsin(A,u0,v0,M,N)
%TWODSIN Compares for loops vs. vectorization.
%  The comparison is based on implementing the function
%  f(x,y)=Asin(u0x+v0y) for x=0,1,2,...,M-1 and
%  y=0,1,2,...,N-1. The inputs to the function are
%  M and N and the constants in the function.
```

Comparación entre ciclos for y vectorización

```
% First implement using for loops.

tic  %Start timing.

for r=1:M
    u0x=u0*(r-1);
    for c=1:N
        v0y=v0*(c-1);
        f(r,c)=A*sin(u0x+v0y);
    end
end

t1=toc;  %End timing.
```

Comparación entre ciclos for y vectorización

```
%Now implement using vectorization. Call the image g.

tic  %Start timing;

r=0:M-1;
c=0:N-1;
[C,R]=meshgrid(c,r);
g=A*sin(u0*R+v0*C);

t2=toc;  %End timing

% Compute the ratio of the two times.

rt=t1/(t2+eps);  % Use eps in case t2 is close to 0.
```

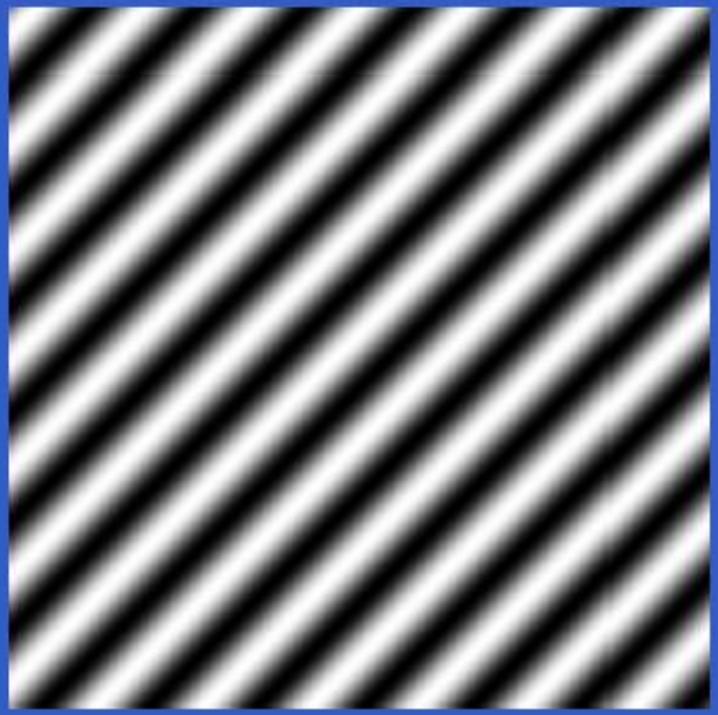
Comparación entre ciclos for y vectorización

```
>> [rt,f,g]=twodsin(1,1/(4*pi),1/(4*pi),512,512);
>> rt

rt =

    19.5833

>> g=mat2gray(g);
>> imshow(g)
```



Matrices con valores preasignados

```
tic
for i=1:1024
    for j=1:1024
        f(i,j)=i+2*j;
    end
end
toc
Elapsed time is 30.484000 seconds.

tic
g=zeros(1024);  %Preallocation
for i=1:1024
    for j=1:1024
        g(i,j)=i+2*j;
    end
end
toc
Elapsed time is 0.221000 seconds.
```

Entrada y Salida de Datos

`disp(argument)`

```
>> A=[1 2;3 4];
>> disp(A)
     1     2
     3     4

>> sc='Digital Image Processing.';
>> disp(sc)
Digital Image Processing.

>> disp('This is another way to display text.')
This is another way to display text.
```

Entrada y Salida de Datos

```
t=input('message')
```

```
t=input('messages','s')
```

```
>> t=input('Enter your data: ','s')
Enter your data: 1, 2, 4
t =
1, 2, 4
>> class(t)
ans =
char
>> size(t)
ans =
     1     7
```

Entrada y Salida de Datos

```
>> n=str2num(t)

n =
     1     2     4

>> size(n)

ans =
     1     3

>> class(n)

ans =
double
```

Entrada y Salida de Datos

```
[a,b,c,...]=strread(cstr,'format,...
                        'param','value')
```

```
>> t='12.6, x2y, z';
>> [a,b,c]=strread(t,'%f%q%q','delimiter',',')
a =
   12.6000
b =
    'x2y'
c =
    'z'
>> d=char(b)
d =
x2y
```

Arreglos de Celdas

En Matlab, los arreglos de celdas son arreglos multidimensionales cuyos elementos son copias de otros arreglos

```
>> C={'gauss',[1 0;1 0],3}
C =
    'gauss'    [2x2 double]    [3]
>> C{1}
ans =
gauss
>> C{2}
ans =
     1     0
     1     0
>> C{3}
ans =
     3
```

Estructuras

Estructuras permiten la agrupación de una colección de datos diferentes en una sola variable

```
>> S.char_string='gauss';
>> S.matrix=[1 0;1 0];
>> S.scalar=3;
>> S
S =
    char_string: 'gauss'
         matrix: [2x2 double]
         scalar: 3
>> S.matrix
ans =
     1     0
     1     0
```

Estructuras

Estructuras permiten la agrupación de una colección de datos diferentes en una sola variable

```
>> S.char_string='gauss';
>> S.matrix=[1 0;1 0];
>> S.scalar=3;
>> S
S =
    char_string: 'gauss'
         matrix: [2x2 double]
         scalar: 3
>> S.matrix
ans =
     1     0
     1     0
```

Filtrado en Matlab

La función filter 2 hace el filtrado lineal

```
filter2(filter,image,shape)
```

y el resultado es una matriz de tipo double. El parametro shape es opcional que describe el método para encontrar los bordes.

```
filter2(filter,image,'same')
```

es el valor predeterminado, produce una matriz del mismo tamaño que la matriz original. Rellena los bordes con ceros para realizar el filtrado

Filtrado en Matlab

```
>> x=uint8(10*magic(5))

x =

  170  240   10   80  150
  230   50   70  140  160
   40   60  130  200  220
  100  120  190  210   30
  110  180  250   20   90

>> a=ones(3,3)/9

a =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>> filter2(a,x,'same')

ans =

   76.6667   85.5556   65.5556   67.7778   58.8889
   87.7778  111.1111  108.8889  128.8889  105.5556
   66.6667  110.0000  130.0000  150.0000  106.6667
   67.7778  131.1111  151.1111  148.8889   85.5556
   56.6667  105.5556  107.7778   87.7778   38.8889
```

Filtrado en Matlab

```
filter2(filter,image,'valid')
```

la máscara se aplica solamente a aquellos píxeles donde la máscara recaerá totalmente en la imagen

```
>> filter2(a,x,'valid')

ans =

  111.1111  108.8889  128.8889
  110.0000  130.0000  150.0000
  131.1111  151.1111  148.8889
```

Filtrado en Matlab

El resultado de arriba se puede obtener rellenando con ceros y usando ‘valid’

```
>> x2=zeros(7,7)

x2 =

     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0

>> x2(2:6,2:6)=x

x2 =

     0     0     0     0     0     0     0
     0   170   240    10    80   150     0
     0   230    50    70   140   160     0
     0    40    60   130   200   220     0
     0   100   120   190   210    30     0
     0   110   180   250    20    90     0
     0     0     0     0     0     0     0

>> filter2(a,x2,'valid')
```

Filtrado en Matlab

```
filter2(filter,image,'full')
```

retorna una matriz más grande que la original, rellenando con ceros y aplicando el filtro en todos los lugares donde la imagen se intersecta con la mascara

```
>> filter2(a,x,'full')

ans =

   18.8889   45.5556   46.6667   36.6667   26.6667   25.5556   16.6667
   44.4444   76.6667   85.5556   65.5556   67.7778   58.8889   34.4444
   48.8889   87.7778  111.1111  108.8889  128.8889  105.5556   58.8889
   41.1111   66.6667  110.0000  130.0000  150.0000  106.6667   45.5556
   27.7778   67.7778  131.1111  151.1111  148.8889   85.5556   37.7778
   23.3333   56.6667  105.5556  107.7778   87.7778   38.8889   13.3333
   12.2222   32.2222   60.0000   50.0000   40.0000   12.2222   10.0000
```

Filtrado en Matlab

Podemos crear nuestro filtro utilizando la función especial

```
>> fspecial('average',[5,7])
```

que retornará filtro promedio de tamaño 5 x 7, o más simplemente

```
>> fspecial('average',11)
```

que retornará un filtro de tamaño 11 x 11

Filtrado en Matlab

Por ejemplo si queremos aplicar el filtro promedio de tamaño 3 x 3 a la imagen:

```
>> c=imread('cameraman.tif');
>> f1=fspecial('average');
>> cf1=filter2(f1,c);
```

Ahora tenemos una matriz de tipo double. Para mostrar esto, podemos hacer lo siguiente

- transformarlo en una matriz de tipo uint8, para su uso con imshow
- dividimos sus valores por 255 para obtener una matriz en el rango 0.1 – 1.0, para usar imshow
- usar mat2gray para escalar el resultado (lo discutiremos luego)

```
>> figure,imshow(c),figure,imshow(cf1/255)
```

Filtrado en Matlab

Filtros separables

Algunos filtros pueden ser implementados aplicando fltros más simples sucesivamente

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

el filtro promedio puede ser implementado primero aplicando un filtro promedio 3 x 1, y luego aplicando un filtro promedio 1 x 3 para llegar al resutlado . El filtro promedio 3 x 3 es separable en dos filtros pequeños. La separabilidad puede ayudarnos una gran cantidad de tiempo. Suponemos que un filtro *n x n* es separable en dos pequeños filtros de *n x 1* y *1 x n*. La aplicación de un filtro *n x n* requiere n^2 multiplicaciones y $n^2 - 1$ sumas por cada pixel. Pero la aplicación de un filtro de tamaño *n x 1* solo requiere *n* multiplicaciones y *2n – 2* sumas. Si *n* es grande el ahorro de tiempo puede ser dramático

Filtros separables

Todos los filtros promedios son separables. Otro filtro separable es el laplaciano

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}.$$

Filtros Pasa Altos

```
>> f=fspecial('laplacian')

f =

    0.1667    0.6667    0.1667
    0.6667   -3.3333    0.6667
    0.1667    0.6667    0.1667

>> cf=filter2(f,c);
imshow(cf/100)
```