Special Function Implementation in IDL
A Case Study

Special Functions
• You have been asked or will be asked to implement certain special functions that is commonly used in imaging and image system analysis
• Examples
  – Rect
  – Sinc
  – Gaus

Implementation
• While the definitions of these functions are mathematically unambiguous, the implementation of these functions are as varied as writing styles and skill are between two individuals
• We will study the case of the Rect function from definition to implementation and point out the various forms used in traditional programming languages and IDL.

Mathematical Definition of One-Dimensional Rect Function
\[
\text{Rect}\left(\frac{x-x_0}{b}\right) = \begin{cases} 
0 & \text{if } \frac{x-x_0}{b} < \frac{1}{2}, \\
\frac{1}{2} & \text{if } \frac{1}{2} \leq \frac{x-x_0}{b} < 1 \\
1 & \text{if } \frac{x-x_0}{b} \geq 1 
\end{cases}
\]

Uses of Rect function
• Used to describe the profile across an aperture.
• 2D Rect function can be used as a filter or mask in both the spatial and frequency domain

Plot of One-Dimensional Rect Function
Common Implementation Pitfall

• Function construct not used
• Printing out results from within the Rect function
• Looping to generate the range of x values is embedded into the Rect function.
• Use of a three parameter function for the arguments x, x0, and b

Implementation Styles

• Scalar method
  – More Generalized and Traditional Style
  – Applicable to most programming languages
• Vector method
  – More Efficient in Vectorized Languages

Sample Implementation

function rect, x, x0, b

abs_x_x0_b = abs((x-x0)/b)
if (abs_x_x0_b gt 0.5 ) then return, 0.0
if (abs_x_x0_b lt 0.5 ) then return, 1.0
if( abs_x_x0_b eq 0.5 ) then return, 0.5
end

Quick Test of Rect Function

IDL> print, rect(0.0,0.0,1.0)
1.00000
IDL> print, rect(0.5,0.0,1.0)
0.500000
IDL> print, rect(1.0,0.0,1.0)
0.00000

Additional Test of Rect Function

IDL> for x=-2.0,2.0,0.5 do_
  print,x,rect(x,0.0,1.0)

0.00000  1.00000
0.500000  0.500000
1.000000  0.000000
1.500000  0.000000
2.000000  0.000000

Simplifying the Parameter List

• Through the use of variable substitution, the parameter list can be simplified from
Rect(x, x0, b) to Rect((x-x0)/b)

\[ x' = \frac{x - x_0}{b} \]
Rect Function Implementation (Single if statement)

```idl
function rect, x
  abs_x = abs(x)
  if (abs_x gt 0.5) then answer = 0.0
  else if (abs_x lt 0.5) then answer = 1.0
  else answer = 0.5
  return, answer
end
```

Rect Function Implementation Blocked if statements

```idl
function rect, x
  abs_x = abs(x)
  if (abs_x gt 0.5) then begin
    answer = 0.0
  endif else if (abs_x lt 0.5) then begin
    answer = 1.0
  endif else begin
    answer = 0.5
  endelse
  return, answer
end
```

Rect Function Test

```
IDL> print, rect(0.0)
1.00000
IDL> print, rect(0.5)
0.500000
IDL> print, rect(1.0)
0.00000
```

Additional Rect Function Test

```
IDL> for x=-2.0,2.0,0.5 do
  print,x,rect((x-x0)/b)
-2.00000 0.00000
-1.50000 0.00000
-1.00000 0.00000
-0.500000 0.500000
 0.000000 1.00000
 0.500000 0.500000
 1.00000 0.00000
 1.500000 0.00000
 2.00000 0.00000
```

2D Rect Definition

```
Rect2d(x-x0, y-y0, b, d) = Rect((x-x0)/b, (y-y0)/d)
```

Error-prone Implementation

```idl
function rect2d, x, x0, b, y, y0, d
  answer = rect((x-x0)/b) * rect((y-y0)/d)
  return, answer
end
```
Error-prone Interface

- This implementation provides too many possibilities of transposing parameter values which the routine cannot trap.

```
IDL> print, rect2d(0.0, 0.0, 1.0, 0.0, 0.0, 1.0)
```

- By using the simplified parameter rect function, the above interface becomes less confusing.

2D Rect Function

```
function rect2d, x, y
    answer = rect(x) * rect(y)
    return, answer
end
```

Test of 2D Rect Function

```
IDL> print, rect2d(0.0, 0.0)    1.00000
IDL> print, rect2d(0.5, 0.0)    0.500000
IDL> print, rect2d(0.0, 0.5)    0.500000
IDL> print, rect2d(0.5, 0.5)    0.250000
IDL> print, rect2d(1.0, 1.0)    0.000000
```

Program to Test rect2d function

```
pro test_rect2d
    answer = fltarr(5, 5) & j = 0
    for y = -1.0, 1.0, 0.5 do begin
        i = 0
        for x = -1.0, 1.0, 0.5 do begin
            answer(i, j) = rect2d(x, y)
            i = i + 1
        endfor
        j = j + 1
    endfor
    plot, answer
end
```

The Vectorized Approach

- In IDL, there is a better way of dealing with this problem.
- But first...

Advanced Array Processing in IDL
Accesing Array Values

IDL> a = [0,2,4,1]
IDL> b = [6,5,1,8,4,3]
IDL> print, b(a)
6      1      4      5

- Use the values of array a as indices to “look up” or “point to” the values of array b.

Simple Image Example

IDL> image=bindgen(5,5)
IDL> print, image
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24

Subsample Image using for loop

pro subsample_loop
a=bindgen(5,5) & answer=bytarr(3,3)
i=0
for m=0,4,2 do begin
  j=0
  for n=0,4,2 do begin
    answer(i,j)=a(m,n)
    j=j+1
  endfor
  i=i+1
endfor
print,answer
end

Result of subsample_loop

IDL> subsample_loop
0 2 4
10 12 14
20 22 24

Analyzing the pattern of indices

a[0,0]  a[2,0]  a[4,0]
a[0,2]  a[2,2]  a[4,2]
a[0,4]  a[2,4]  a[4,4]

Looking at the row indices

a[0,0]  a[2,0]  a[4,0]
a[0,2]  a[2,2]  a[4,2]
a[0,4]  a[2,4]  a[4,4]
Looking at the column indices

\[
\begin{align*}
  a[0,0] & \quad a[2,0] & \quad a[4,0] \\
  a[0,2] & \quad a[2,2] & \quad a[4,2] \\
  a[0,4] & \quad a[2,4] & \quad a[4,4] 
\end{align*}
\]

Defining a new variable \texttt{row}

IDL> row=[[0,2,4],[0,2,4],[0,2,4]]
IDL> print,row
\[
\begin{align*}
  0 & \quad 2 & \quad 4 \\
  0 & \quad 2 & \quad 4 \\
  0 & \quad 2 & \quad 4 
\end{align*}
\]

Defining a new variable \texttt{column}

IDL> column=[[0,0,0],[2,2,2],[4,4,4]]
IDL> print,column
\[
\begin{align*}
  0 & \quad 0 & \quad 0 \\
  2 & \quad 2 & \quad 2 \\
  4 & \quad 4 & \quad 4 
\end{align*}
\]

IDL can use arrays/matrices as indices into other arrays

IDL> b=a[row,column]
IDL> print,b
\[
\begin{align*}
  0 & \quad 2 & \quad 4 \\
  10 & \quad 12 & \quad 14 \\
  20 & \quad 22 & \quad 24 
\end{align*}
\]

Creating the row and column index matrices

- Currently, we created the row and column index matrices by hand.
- We could use a for loop to fill the values, but there is a more convenient way in IDL using the \# operator

The \# operator

- For a given array \(a\) and \(b\), \(a \# b\) will have the effect of multiplying the columns of the first array by the rows of the second array.
- Suppose \(a = [0,2,4]\) and \(b = [1,1,1]\) then
  IDL> a=[0,2,4]
  IDL> b=[1,1,1]
  IDL> print,a#b
\[
\begin{align*}
  0 & \quad 2 & \quad 4 \\
  0 & \quad 2 & \quad 4 \\
  0 & \quad 2 & \quad 4 
\end{align*}
\]
The # operator

• Likewise

```idl```
IDL> print,b#a
0  0  0
2  2  2
4  4  4
```idl```

The General Form of Creating Index Matrices

• For the Row Index Matrix (rim) given a vector row containing the indices for a given row

```idl```
IDL> rim=row#replicate(1,n_elements(column))
```idl```

• Where replicate(1,n_elements(column)) is creating a column of ones (1) equal to the number of elements in the vector column.

The General Form of Creating Index Matrices

• For the Column Index Matrix (cim) given a vector column containing the indices for a given column

```idl```
IDL> cim=replicate(1,n_elements(row))#columns
```idl```

• Where replicate(1,n_elements(row)) is creating a row of ones (1) equal to the number of elements in the vector row.

Vectorized rect function

```idl```
function rect, x
   answer = x
   abs_x = abs(x)
   gt_half = where(abs_x gt 0.5, count_gt)
   eq_half = where(abs_x eq 0.5, count_eq)
   lt_half = where(abs_x lt 0.5, count_lt)
   if (count_gt ne 0) then answer(gt_half)=0.0
   if (count_eq ne 0) then answer(eq_half)=0.5
   if (count_lt ne 0) then answer(lt_half)=1.0
   return, answer
end
```idl```

Sample Use of Vectorized rect

```idl```
IDL> x=[-1.0,-0.5,0.0,0.5,1.0]
OR
IDL> x=findgen(5)*0.5-1.0
IDL> print,rect(x)
0.00000  0.500000  1.00000
0.50000  0.00000
```idl```

What about Vectorized rect2d function?

- We don’t need to do anything!
Testing Procedure of the Vectorized rect2d

- Define the extents of the x and y values you want to calculate
- Create the x and y index matrix (or more appropriately x and y value matrix)
- Pass the x and y value matrix into the function

Define the extents of the x and y values

IDL> x=[-1.0, -0.5, 0.0, 0.5, 1.0]
IDL> y=[-2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0]

Create the x and y value matrices

IDL> xm=x#replicate(1.0,n_elements(y))
IDL> print,xm

Pass the x and y value matrix into the function

IDL> print,rect2d(xm,ym)

Statistics Function Implementation (Revisited)

Vector Programming Approach
sum.pro

function sum, x  
n = n_elements( x )  
answer = 0.0  
for i = 0L, n-1 do begin  
  answer = answer + x(i)  
endfor  
return, answer  
end

sum.pro

function sum, x  
n = n_elements( x )  
array_size = size( x )  
array_dimensions = array_size(1:array_size(0))  
x = reform( x, n, /overwrite )  
array_size = array_size, /overwrite  
x = reform( x, array_dimensions, /overwrite )  
return, answer  
end

sum_squares.pro

function sum_squares, x  
n = n_elements( x )  
array_size = size( x )  
array_dimensions = array_size(1:array_size(0))  
x = reform( x, n, /overwrite )  
array_size = array_size, /overwrite  
x = reform( x, array_dimensions, /overwrite )  
return, answer  
end

Midterm Topics

- UNIX Commands
- Shell Programming
- IDL Programs  
  - Statistics, Image I/O, Simple Widgets  
  - Special Functions
- PBM Image Formats and Utilities
- Mystery Image Techniques