xcorr2

2-D cross-correlation

Syntax

```
C = xcorr2(A,B)
A = xcorr2(A)
C = xcorr2(gpuArrayA,gpuArrayB)
```

Description

`C = xcorr2(A,B)` returns the cross-correlation of matrices `A` and `B` with no scaling. `xcorr2` is the two-dimensional version of `xcorr`.

`A = xcorr2(A)` is the autocorrelation matrix of input matrix `A`. It is identical to `xcorr2(A,A)`.

`C = xcorr2(gpuArrayA,gpuArrayB)` returns the cross-correlation of two matrices of class `gpuArray`. The output cross-correlation matrix, `C`, is also a `gpuArray` object. See Establish Arrays on a GPU for details on `gpuArray` objects. Using `xcorr2` with `gpuArray` objects requires Parallel Computing Toolbox™ software and a CUDA-enabled NVIDIA GPU with compute capability 1.3 or above. See http://www.mathworks.com/products/parallel-computing/requirements.html for details. See GPU Acceleration for Cross-Correlation Matrix Computation for an example of using a GPU to compute the cross-correlation.

2-D Cross-Correlation

The 2-D cross-correlation of an *M*-by-*N* matrix *X* and a *P*-by-*Q* matrix *H* is a matrix *C* of size *M*+*P*–1 by *N*+*Q*–1 given by

$$C(k,l) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} X(m,n)\,\bar{H}(m-k,n-l), \qquad \begin{aligned} -(P-1) \le k \le M-1, \\ -(Q-1) \le l \le N-1, \end{aligned}$$

where the bar over *H* denotes complex conjugation.

The output matrix, *C*(*k*,*l*), has negative and positive row and column indices. A negative row index corresponds to an upward shift of the rows of *H*. A negative column index corresponds to a leftward shift of the columns of *H*. A positive row index corresponds to a downward shift of the rows of *H*. A positive column index corresponds to a rightward shift of the columns. To cast the indices in MATLAB® form, simply add the size of *H*: the element *C*(*k*,*l*) corresponds to `C(k+P,l+Q)` in the workspace.

For example, consider the following 2-D cross-correlation:

```
X = ones(2,3);
H = [1 2; 3 4; 5 6];  % H is 3 by 2
C = xcorr2(X,H)
```

```
C =
     6    11    11     5
    10    18    18     8
     6    10    10     4
     2     3     3     1
```

The `C(1,1)` element in the output corresponds to *C*(1–3,1–2) = *C*(–2,–1) in the defining equation, which uses zero-based indexing. The `C(1,1)` element is computed by shifting `H` two rows upward and one column to the left. Accordingly, the only product in the cross-correlation sum is `X(1,1)*H(3,2) = 6`. Using the defining equation, you obtain

$$C(-2,-1) = \sum_{m=0}^{1}\sum_{n=0}^{2} X(m,n)\,\bar{H}(m+2,n+1) = X(0,0)\,\bar{H}(2,1) = 1\times 6 = 6,$$

with all other terms in the double sum equal to zero.

Examples

Output Matrix Size

If matrix I1 has dimensions (4,3) and matrix I2 has dimensions (2,2), the following equations determine the number of rows and columns of the output matrix:

$$C_{\text{full}_{\text{rows}}} = I1_{\text{rows}} + I2_{\text{rows}} - 1 = 4 + 2 - 1 = 5$$
$$C_{\text{full}_{\text{columns}}} = I1_{\text{columns}} + I2_{\text{columns}} - 1 = 3 + 2 - 1 = 4$$

The resulting matrix is

$$C_{\text{full}} = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \\ c_{40} & c_{41} & c_{42} & c_{43} \end{bmatrix}$$

Computing a Specific Element

$$C_{\text{valid}_{\text{columns}}} = I1_{\text{columns}} - I2_{\text{columns}} + 1 = 2$$

In cross-correlation, the value of an output element is computed as a weighted sum of neighboring elements. For example, suppose the first input matrix represents an image and is defined as

```
I1 = [17  24   1   8  15
      23   5   7  14  16
       4   6  13  20  22
      10  12  19  21   3
      11  18  25   2   9]
```

The second input matrix also represents an image and is defined as

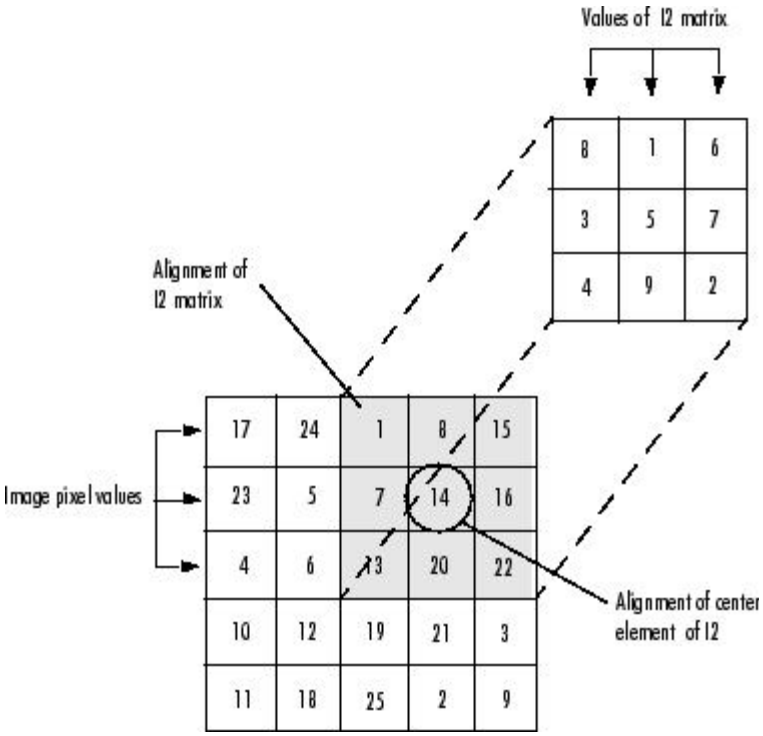
```
I2 = [8   1   6
      3   5   7
      4   9   2]
```

The following figure shows how to compute the (2,4) output element (zero-based indexing) using these steps:

1. Slide the center element of I2 so that lies on top of the (1,3) element of I1.
2. Multiply each weight in I2 by the element of I1 underneath.
3. Sum the individual products from step 2.

The (2,4) output element from the cross-correlation is

$$1 \cdot 8 + 8 \cdot 1 + 15 \cdot 6 + 7 \cdot 3 + 14 \cdot 5 + 16 \cdot 7 + 13 \cdot 4 + 20 \cdot 9 + 22 \cdot 2 = 585$$



The normalized cross-correlation of the (2,4) output element is

585/sqrt(sum(dot(I1p,I1p))*sum(dot(I2,I2))) = 0.8070

where I1p = [1 8 15; 7 14 16; 13 20 22].

Recovery of Template Shift with Cross-Correlation

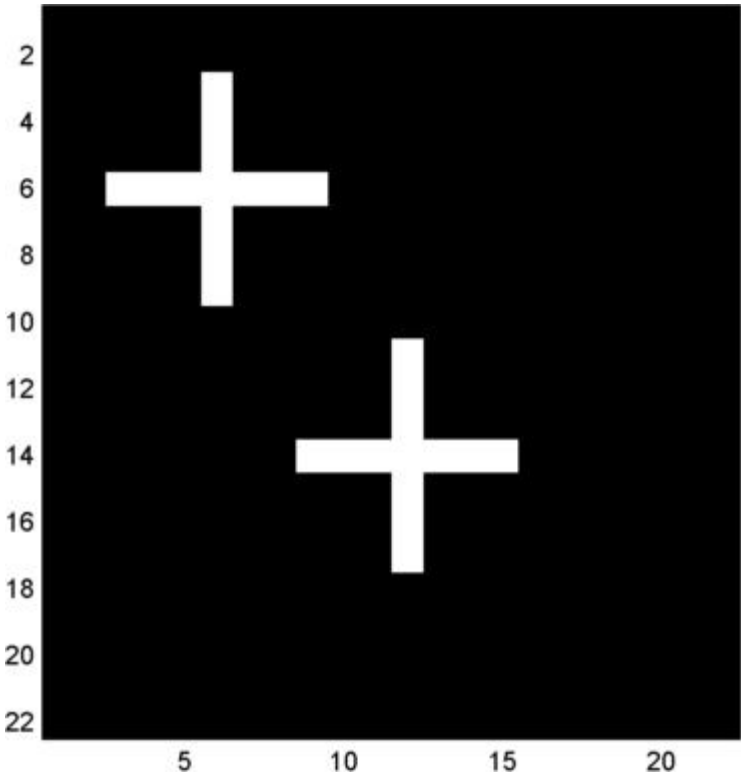
Shift a template by a known amount and recover the shift using cross-correlation.

Create a template in an 11-by-11 matrix. Create a 22-by-22 matrix and shift the original template by 8 along the row dimension and 6 along the column dimension.

```
template = .2*ones(11);
template(6,3:9) = .6;
template(3:9,6) = .6;
offsetTemplate = .2*ones(22);
offset = [8 6];
offsetTemplate( (1:size(template,1))+offset(1),...
                  (1:size(template,2))+offset(2) ) = template;
```

Plot the original and shifted templates.

```
imagesc(offsetTemplate); colormap gray;
hold on;
imagesc(template);
```

Cross-correlate the two matrices and find the maximum absolute value of the cross-correlation. Use the position of the maximum absolute value to determine the shift in the template. Check the result against the known shift.

```
cc = xcorr2(offsetTemplate,template);
[max_cc, imax] = max(abs(cc(:)));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
corr_offset = [ (ypeak-size(template,1)) (xpeak-size(template,2)) ];
isequal(corr_offset,offset)
```

The returned 1 indicates that the shift obtained the cross-correlation equals the known the template shift in both the row and column dimension.

GPU Acceleration for Cross-Correlation Matrix Computation

The following example requires Parallel Computing Toolbox software and a CUDA-enabled NVIDIA GPU with compute capability 1.3 or above. See http://www.mathworks.com/products/parallel-computing/requirements.html for details.

Repeat the example Recovery of Template Shift with Cross-Correlation. For convenience, the code to create the original and shifted templates is repeated.

```
template = .2*ones(11);
template(6,3:9) = .6;
template(3:9,6) = .6;
offsetTemplate = .2*ones(22);
offset = [8 6];
offsetTemplate( (1:size(template,1))+offset(1),...
                    (1:size(template,2))+offset(2) ) = template;
```

Put the original and shifted template matrices on your GPU using `gpuArray` objects.

```
template = gpuArray(template);
offsetTemplate = gpuArray(offsetTemplate);
```

Compute the cross-correlation on the GPU.

```
cc = xcorr2(offsetTemplate,template);
```


Return the result to the MATLAB workspace using `gather`, use the maximum absolute value of the cross-correlation to determine the shift, and compare the result with the known shift.

```
cc = gather(cc);
[max_cc, imax] = max(abs(cc(:)));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
corr_offset = [ (ypeak-size(template,1)) (xpeak-size(template,2)) ];
isequal(corr_offset,offset)
```

See Also

`conv2` | `filter2` | `xcorr`