Pansharpening algorithms

Component Substitution methods: BT, GS, GSA

Brolich Nina, Carbone Alessia, Mansi Paolo

Overview

The issue

Spatial

Temporal

There are **trade-offs** among different resolutions

Radiometric

Spectral

High-quality images can be obtained by **fusion** between **images** with **complementary resolutions**

What is Pansharpening ?

High spatial resolution + High spectral resolution

1

Pansharpening

Panchromatic high spatial resolution

+

High spectral resolution

2

Multispectral pansharpening

Monochromatic high spatial resolution

+

Multispectral image

Multispectral pansharpening

Fuse a panchromatic high spatial resolution image with an High spectral resolution image

Panchromatic

Multispectral

Pansharpened

Pansharpening algorithms

We can distinguish **four** main **classes** of algorithms for pansharpening

We will present only **component substitution** because it has a more **efficient implementation** in spite of the **lower quality** of the fused image

Component substitution

Component substitution can be considered a unique **process** divided into **two** sequential **phases**

The **extraction** of the spatial details from the PAN image:

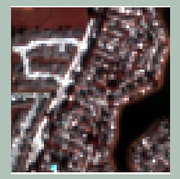
$$D = PAN - f(MS)$$

The **injection** of D in the unsampled MS to PAN size:

$$MS = \underline{MS} + G*D$$

Component Substitution in detail

MS



Interpolation

\+

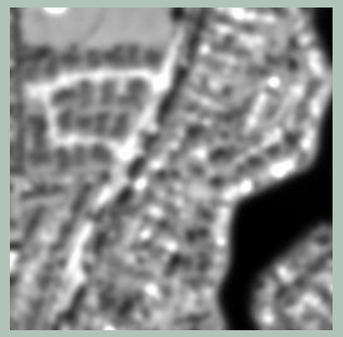
Pansharpened

Gains

PAN

CS

I



Spatial details

P_i

Histogram Matching

What differentiates all the component substitution approaches?

Component substitution approaches are different from one another in how they compute **I** and **G**

1

Brovey Transform

2

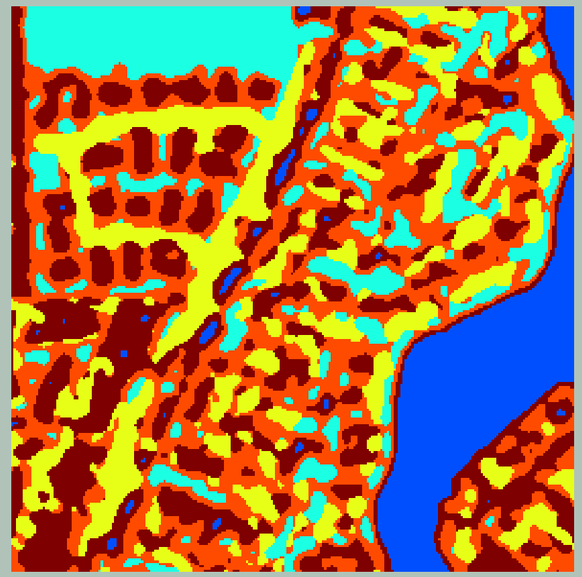
Gram–Schmidt

3

Gram–Schmidt Adaptive

What is segmentation?

In these algorithms the **Gains** are **constant for each band** so it is possible to partition the image in **clusters** of **homogeneous pixels**



Quality assessment

The fused data have to satisfy two **properties**

Consistency

We compare the original MS image to a degraded version of the PAN image obtained through decimation filters

Synthesis

The original MS image acts as the reference image to be compared to the fusion (**reduced resolution protocol**)

Quality indices

The main **indexes** used for quality assessment between the fused image and the reference image are:

1. Spectral Angle Mapper (SAM) index.
2. Erreur Relative Globale Adimensionnelle de Synthèse (ERGAS).
3. Q index.
4. Spatial Correlation Coefficient (SCC).
5. Q2n index.

Overview

1. Introduction
2. **Brovey Transform (BT)**
3. Gram-Schmidt (GS)
4. Gram-Schmidt Adaptive (GSA)
5. Segmentation
6. Gram-Schmidt with Segmentation (GS_Segm)

The Brovey Transform

Extraction

$$w_i = argmin\left[MSE(PAN_{smoothed}, MS)\right]$$

$$I_{L,BTH} = \sum_{i=1}^{N} w_i * (\tilde{MS}_i - L_H)$$

Injection

$$G_k = \frac{\tilde{MS}_k}{I_{L,BT}}$$

BT– H

$$\hat{MS}_k = (\tilde{MS}_k - L_H) + G_k * (P^I - I_{L,BTH}) = \frac{\tilde{MS}_k - L_H}{I_{L,BTH} - L_H} * (P^I - L_H)$$

IDL Implementation : Prototype

```
pro Brovey, PAN, MS, BT, RATIO=ratio, HAZE=haze
```

- PAN : The pachromatic image
- MS: The multispectral image
- BT: The output Pansharpened image with Brovey Transform
- RATIO: Optional parameter indicating the ratio between the PAN image and the original MS at low resolution
- HAZE: Optional Keyword. If set, enables the Haze Correction

IDL Implementation : Details Extraction

$$D = P^i - I_{L,BTH}$$

```
; ---------- P_i Calculation: ---------------------------
  ; Histogram Matching of P over I_L
  P_i = histogram_matching(PAN, I_L)
```

```
; -------- I_L Calculation ----------------------------------------
  ; 1) Generate a Smoothed version of PAN
  PAN_smooth = gaussian_smoothing(PAN, ratio)

  ; 2) Coefficients generated by minimizing MSE between smoothed
  ;    PAN and MS
  num_elements = sizes_PAN[0] * sizes_PAN[1]
  PAN_flatten = reform(PAN_smooth, 1, num_elements)
  MS_flatten = reform(MS, channels, num_elements)
  w_k = LA_LEAST_SQUARES(MS_flatten, PAN_flatten)

  ; 3) I_L = Average Mean of MS scaled by coefficients w
  W = fltarr(sizes_MS)
  for i=0,channels-1 do W[i,*,*] = replicate(w_k[i], [1, sizes_PAN])
  I_L = total((MS-L) * W, 1) ;
```

IDL Implementation: Haze correction and Fusion

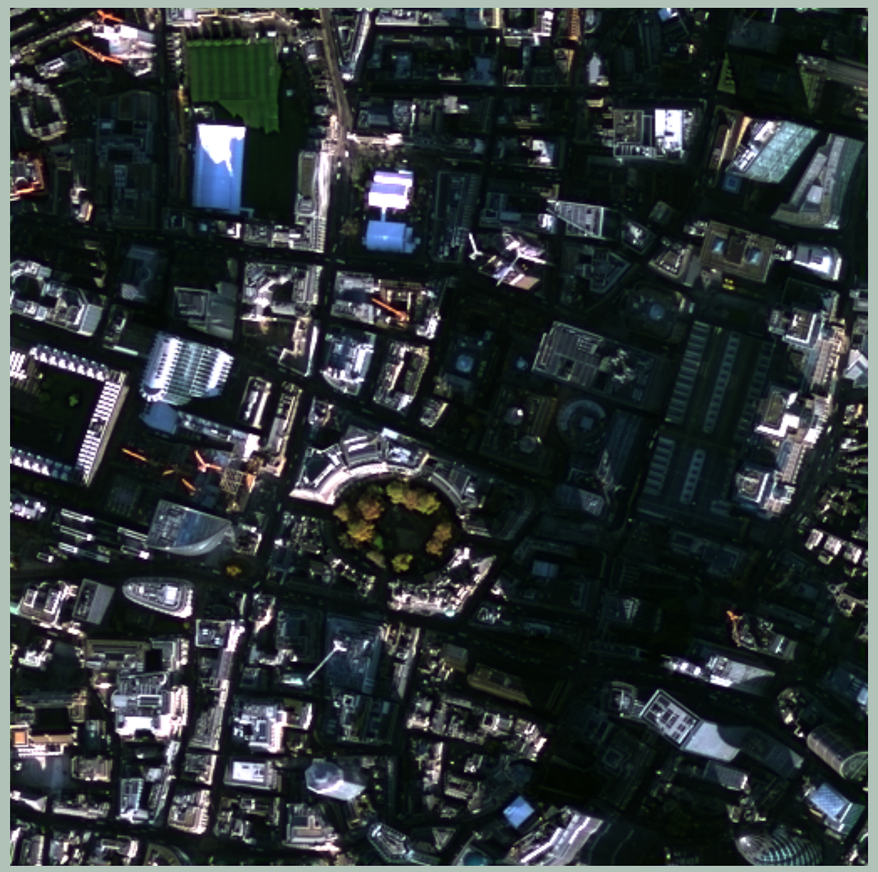
```
;-----  Haze Correction -----------------------------------
L = fltarr(sizes_MS)
if KEYWORD_SET(HAZE) then begin
  l_k = [0.95*cgPercentiles(MS[0,*,*], Percentiles=.01),$
    0.45*cgPercentiles(MS[1,*,*], Percentiles=.01),$
    0.40*cgPercentiles(MS[2,*,*], Percentiles=.01),$
    0.05*cgPercentiles(MS[3,*,*], Percentiles=.01)]

  for i=0,3 do L[i,*,*] = replicate(l_k[i], [1, sizes])
endif
```

The effect of Haze is present only at lower bands (R - G - B - NIR)

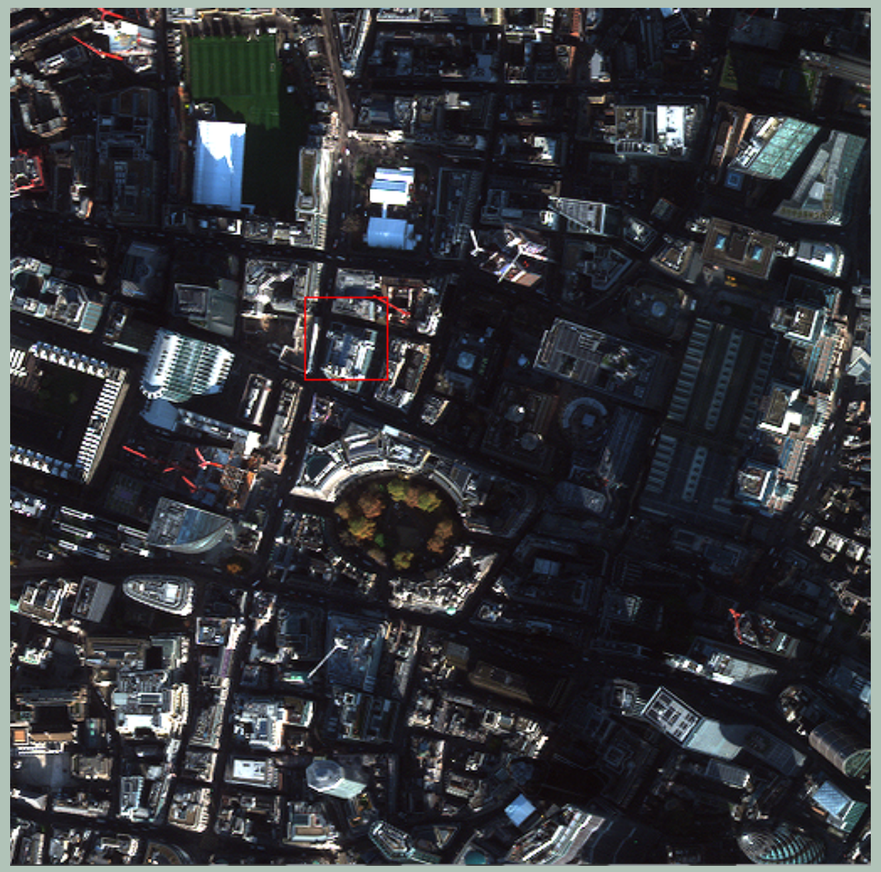
$$\hat{MS}_k = \tilde{MS}_k * \frac{P^I}{I_{L,BTH}}$$

```
; --------- Fusion ------------------------
; MS with Haze Correction
I_MS_L = MS - L;

D = P_I / I_L
D = reform(D, 1, sizes_PAN)

BT = fltarr(sizes_MS)
for i = 0,channels-1 do BT[i,*,*] = I_MS_L[i,*,*] * D
BT = BT + L
```

IDL implementation: the results



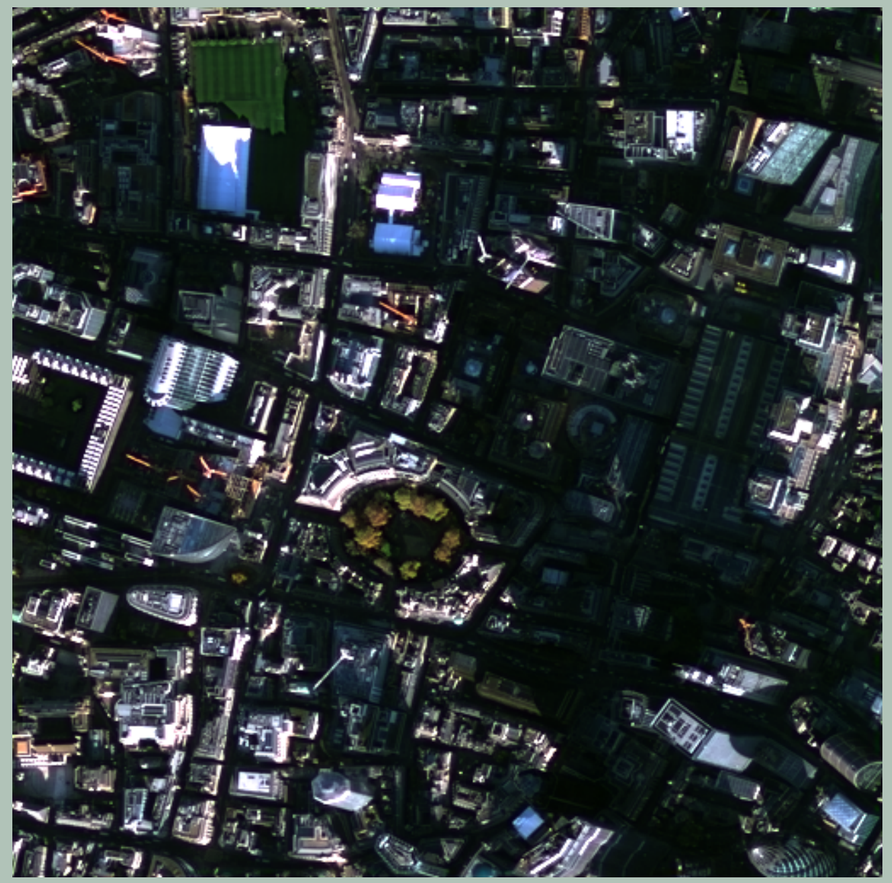
BT- H



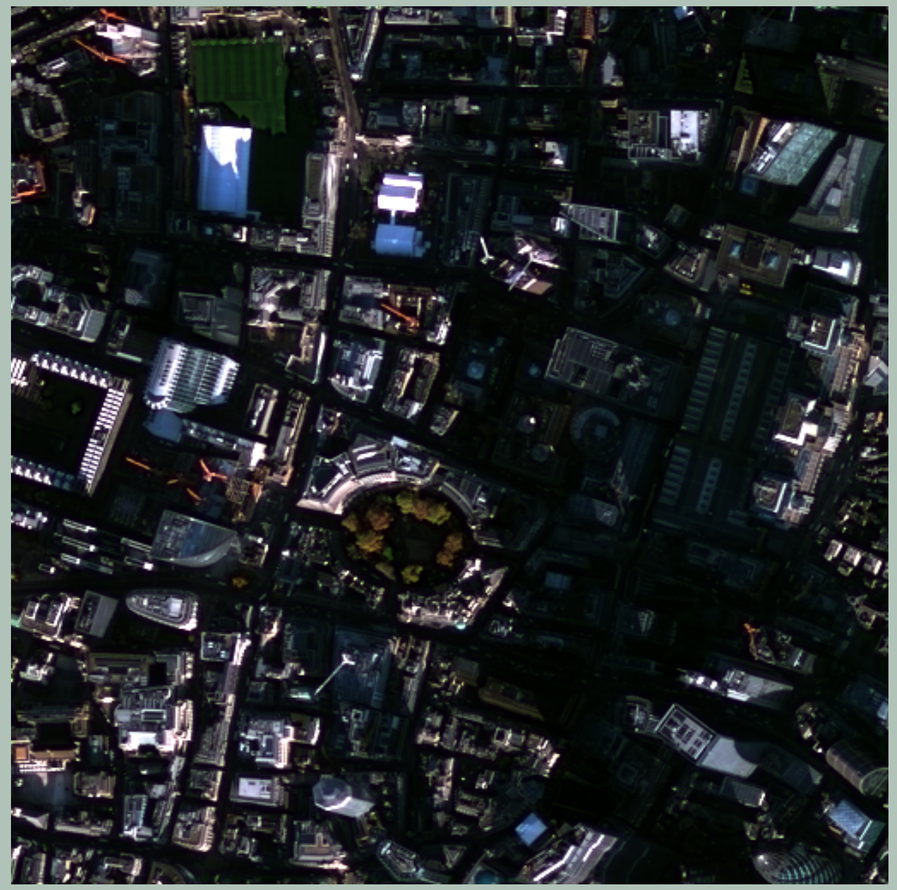
GT

BT-H in IDL MATLAB

We compute the **fusion** through the **available code**

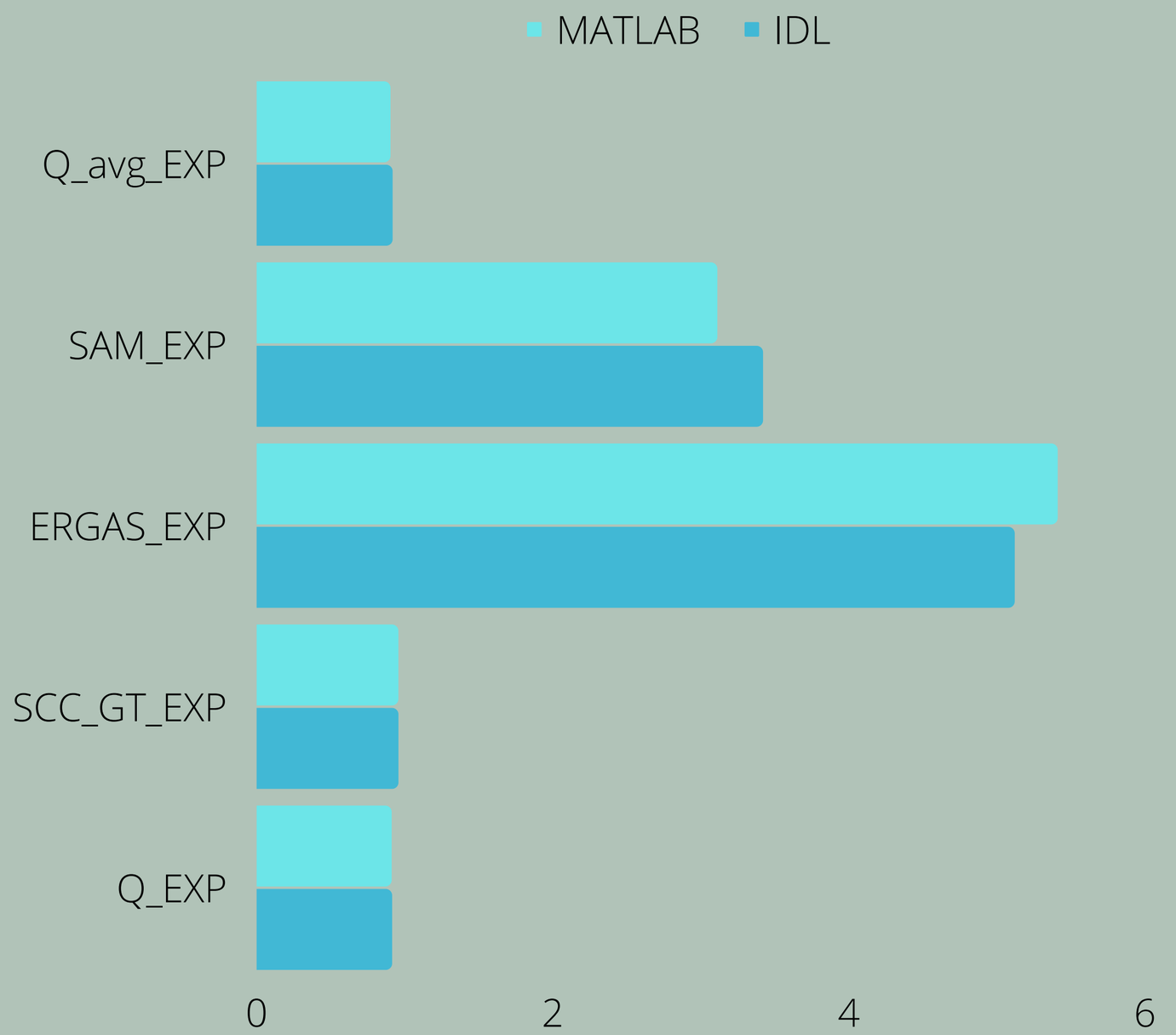


IDL



MATLAB

BT - H in IDL and MATLAB



Overview

The algorithm

Extraction

$$I_{L,GS} = \sum_{i=1}^{N} \frac{1}{N} \tilde{MS}_i$$

Injection

$$G_k = \frac{cov(\tilde{MS}_k, I_{L,GS})}{var(I_{L,GS})}$$

IDL Implementation : Prototype

```
pro GS, PAN, MS, I_GS
```

- PAN : The pachromatic image
- MS: The multispectral image
- I_GS: The output Pansharpened image with Gram Schmidt Transform

IDL Implementation : Details Extraction

$$D = P^i - I_{L,GS}$$

```
; ---------  P_i Calculation ------------------------------------
  P_i = histogram_matching(PAN, I_L_no_mean)
```

$$I_{L,GS} = \sum_{i=1}^{N} \frac{1}{N} \tilde{MS}_i$$

```
; ---------  I_L Calculation ------------------------------------
  I_L = mean(MS, Dimension=1)
  I_L_no_mean = remove_mean(I_L)
```

IDL Implementation : Calculation Injection coefficients

$$G_k = \frac{cov(\tilde{MS}_k, I_{L,GS})}{var(I_{L,GS})}$$

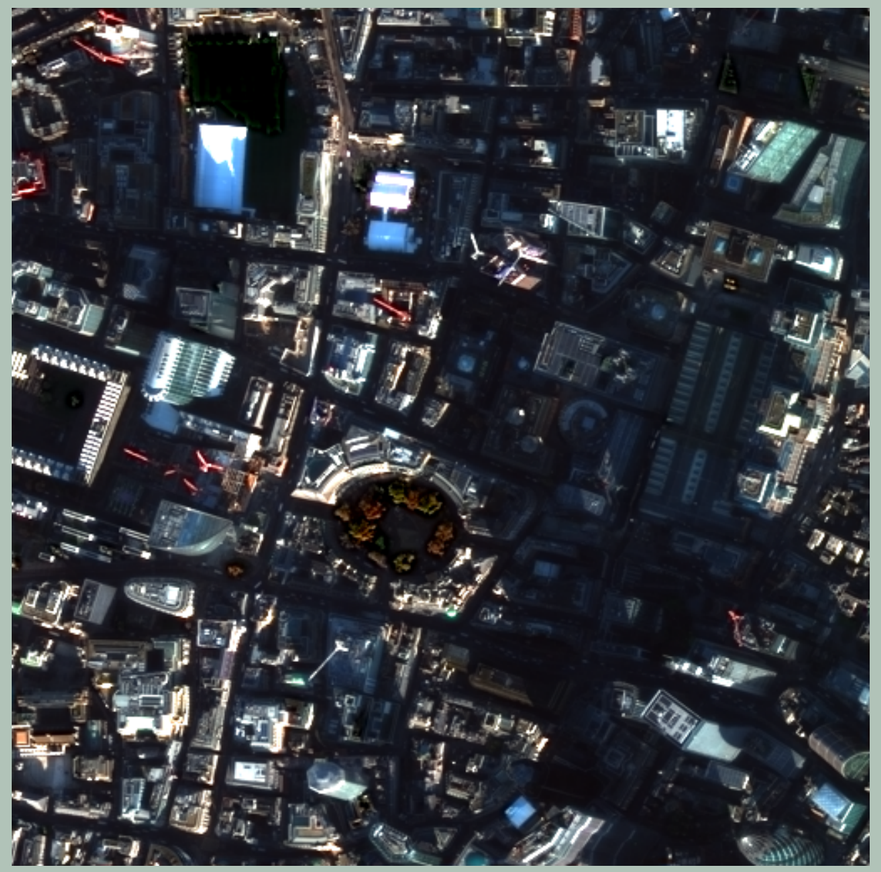
```
; --------  Coefficients Gk  ---------------------------------
  coeff  = FLTARR(channels)
  for i=0,channels-1 do coeff[i,*,*] =$
     correlate(I_L_no_mean, MS_no_mean[i,*,*], /COVARIANCE)/ $
     variance(I_L_no_mean)
```

IDL Implementation : Fusion

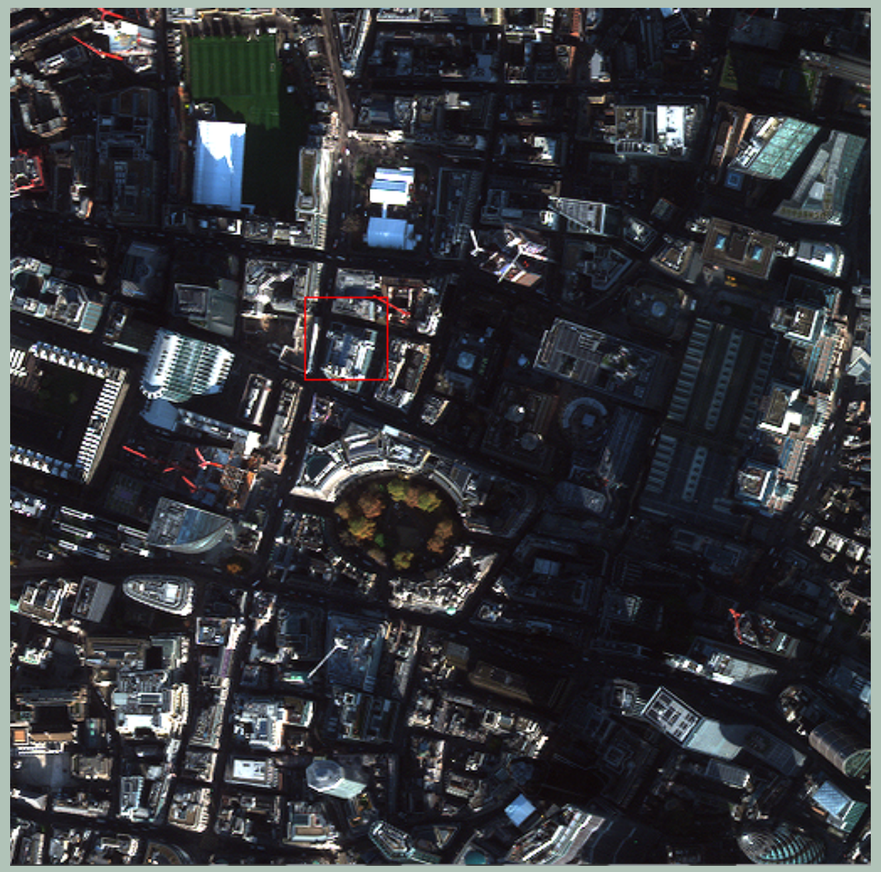
$$\hat{MS}_k = \tilde{MS}_k + G_k * (P^I - I_{L,GS})$$

```
; --------  Fusion  ---------------------------------------
  delta = P_i - I_L_no_mean

  details = fltarr(sizes)
  for i=0, channels-1 do details[i,*,*]= coeff[i]*delta

  I_GS = MS + details

  ; Normalization Fused Data
  for i=0, channels-1 do begin
    I_GS[i,*,*] =I_GS[i,*,*] - mean(I_GS[i,*,*])+mean(MS[i,*,*])
  endfor
```

IDL implementation: the results



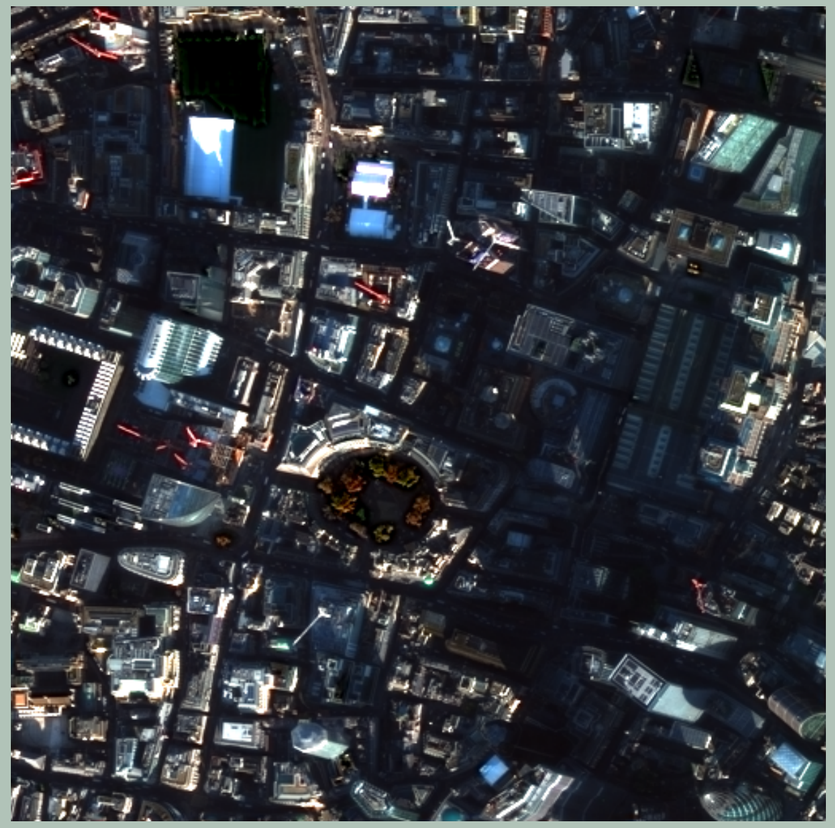
GS



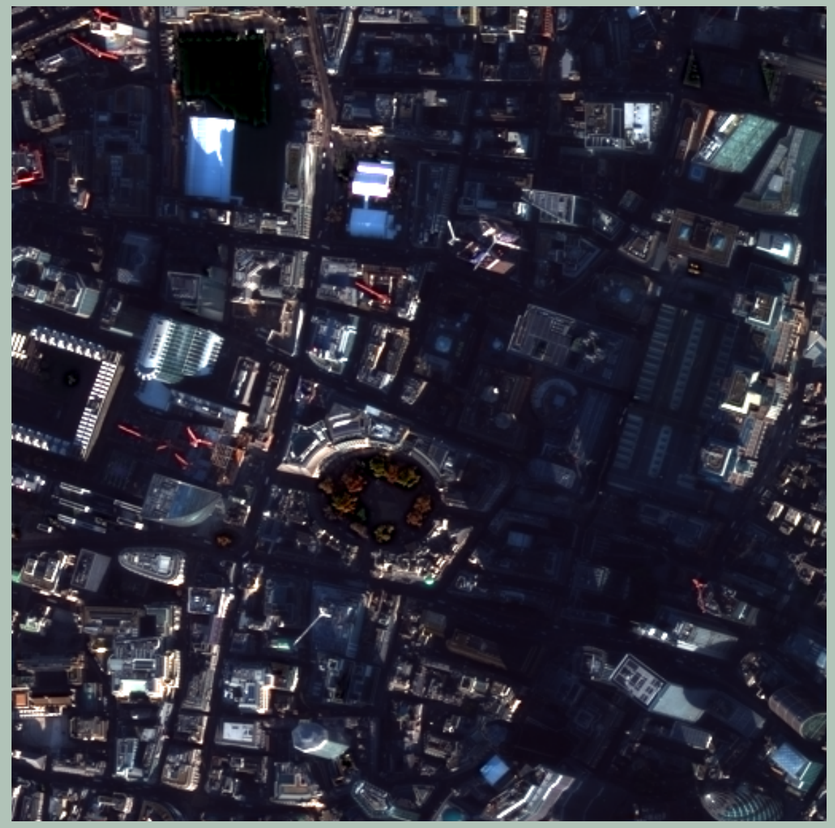
GT

GS in IDL and MATLAB

We compute the **fusion** through the **available code**

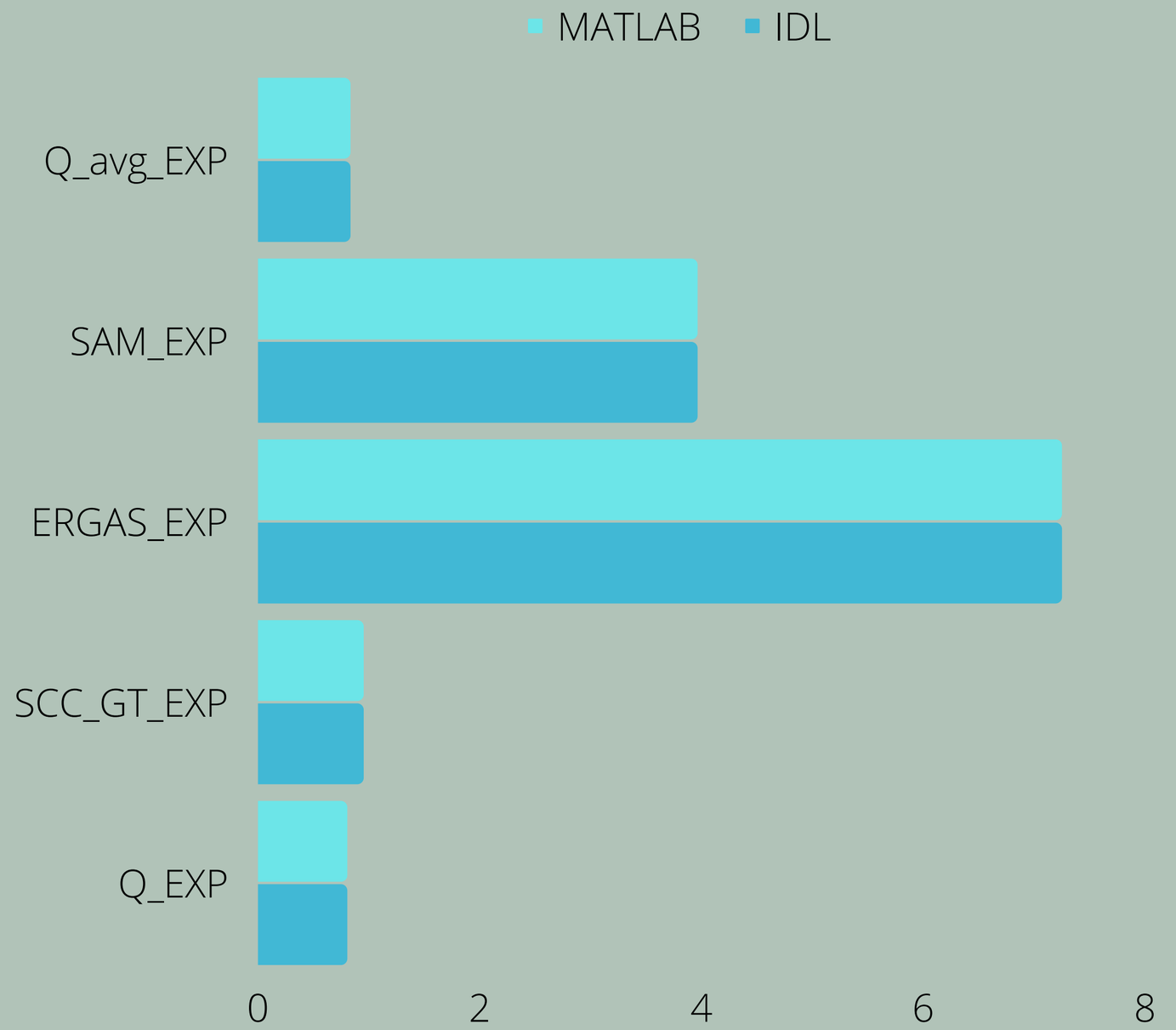


IDL



MATLAB

GS in IDL and MATLAB



Overview

The algorithm

Extraction

$$\boldsymbol{I}_{L,GSA} = \sum_{i=1}^{N} w_i \widetilde{\boldsymbol{MS}}_i$$

$$\widehat{w}_i = argmin_{w_i} \left| \boldsymbol{P}^d - \sum_{i=1}^{N} w_i \widetilde{\boldsymbol{MS}}_i \right|^2$$

Injection

$$\boldsymbol{G}_k = \frac{cov(\widetilde{\boldsymbol{MS}}_k, \boldsymbol{I}_{L,GSA})}{var(\boldsymbol{I}_{L,GSA})}$$

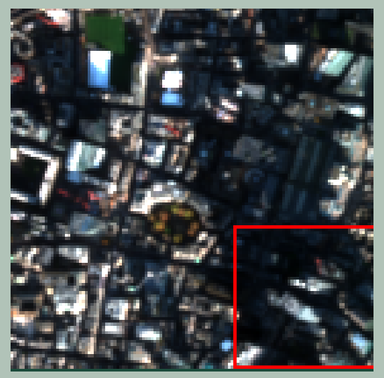
IDL implementation: the images

```
PAN = read_TIFF('.\PAirMax\GE_Lond_Urb\RR\PAN.tif')
MS = read_TIFF('.\PAirMax\GE_Lond_Urb\RR\MS.tif')
MS_LR = read_TIFF('.\PAirMax\GE_Lond_Urb\RR\MS_LR.tif')
```

```
imageLR = float(image_MS)
imageLR_LP = float(image_MS_LR)
imageHR = float(image_PAN)

size_MS = size(imageLR, /DIMENSIONS)
size_MS_LR = size(imageLR_LP, /DIMENSIONS)
size_PAN = size(imageHR, /DIMENSIONS)

channels = size_MS[0]
```

IDL implementation: the images

MS

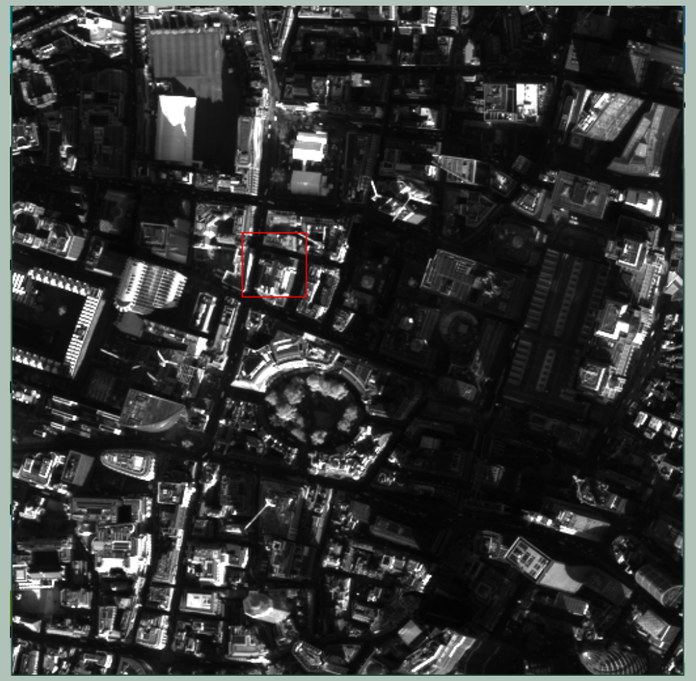


4 128 128

MS at PAN size

4 512 512

PAN



512 512

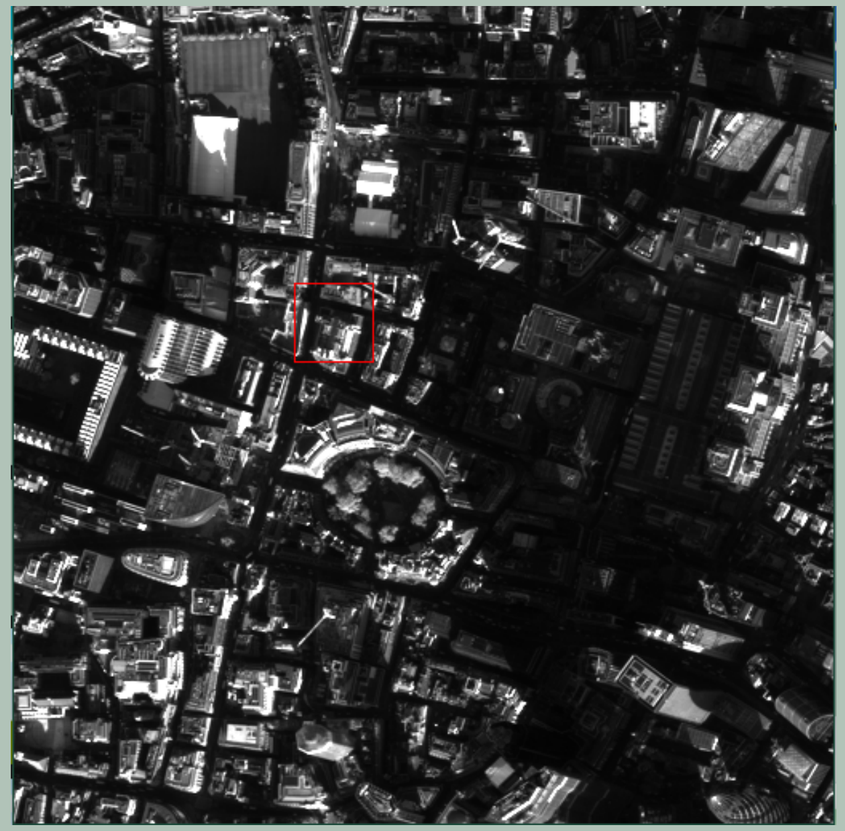
IDL implementation: the images

```
imageLR0 = remove_mean(imageLR)
imageLR_LP0 = remove_mean(imageLR_LP)
imageHR0 = remove_mean(imageHR)
```

```
ratio = 4
WT = wtn(imageHR0, ratio, /OVERWRITE)
smoothed = fltarr(size_PAN)
smoothed[0:256, 0:256] = WT[0:256, 0:256]
imageHR0 = wtn(smoothed, ratio, /INVERSE, /OVERWRITE)
imageHR0 = congrid(imageHR0, size_PAN[0]/ratio, $
size_PAN[1]/ratio, /INTERP)
size_HR0 = size(imageHR0, /DIMENSIONS)
save_image, "./output/wtn.tif", imageHR0
```

IDL implementation: the images

Original **PAN**



Smoothed and resized **PAN**

IDL implementation: the extraction

$$\hat{w}_i = argmin_{w_i} \left| P^d - \sum_{i=1}^{N} w_i \tilde{MS}_i^d \right|^2$$

```
one_matrix = make_array(size_MS_LR[1], size_MS_LR[2], $
/FLOAT, VALUE = 1)
concatenation = make_array(channels+1, size_MS_LR[1], $
size_MS_LR[2], /FLOAT, VALUE = 0)
for i=0,channels-1 do concatenation[i,*,*] = imageLR_LP0[i,*,*]
concatenation[channels,*,*] = one_matrix[*,*]
alpha = make_array(1, 1, channels+1, /FLOAT, VALUE = 0)
IHc = imageHR0[*]
size_conc = size(concatenation, /DIMENSIONS)
ILRc = reform(concatenation, [size_conc[0], size_conc[1] $
*size_conc[2]])
alpha[0,0,*] = la_least_squares(ILRc, IHc)
```

$$I_{L,GSA} = \sum_{i=1}^{N} w_i \tilde{MS}_i$$

```
one_matrix = make_array(size_MS[1], size_MS[2], /FLOAT, $
VALUE=1)
concatenation = make_array(size_MS[1], size_MS[2], $
channels+1, /FLOAT, VALUE = 0)
for i=0,channels-1 do concatenation[*,*,i] = imageLR0[i,*,*]
concatenation[*,*,channels] = one_matrix[*,*]
new_alpha = make_array(size_MS[1], size_MS[2], channels+1, $
/FLOAT, VALUE=0)
for i=0,channels do new_alpha[*,*,i]=alpha[0,0,i]
I = total(concatenation * new_alpha, 3)
I0 = I - mean(I)
```

$$D = P - I$$

```
imageHR = imageHR - mean(imageHR)
delta = imageHR - I0
size_delta = size(delta)
new_delta = make_array(channels+1, size_delta[1]*size_delta[2], $
/FLOAT, VALUE=0)
for i=0, channels do new_delta[i,*] = delta[*]
```

IDL implementation: the injection

$$G_k = \frac{cov(\tilde{MS}_k, I_{L,\mathbf{GSA}})}{var(I_{L,\mathbf{GSA}}))}$$

$$\hat{MS} = \tilde{MS} + G \circ D$$

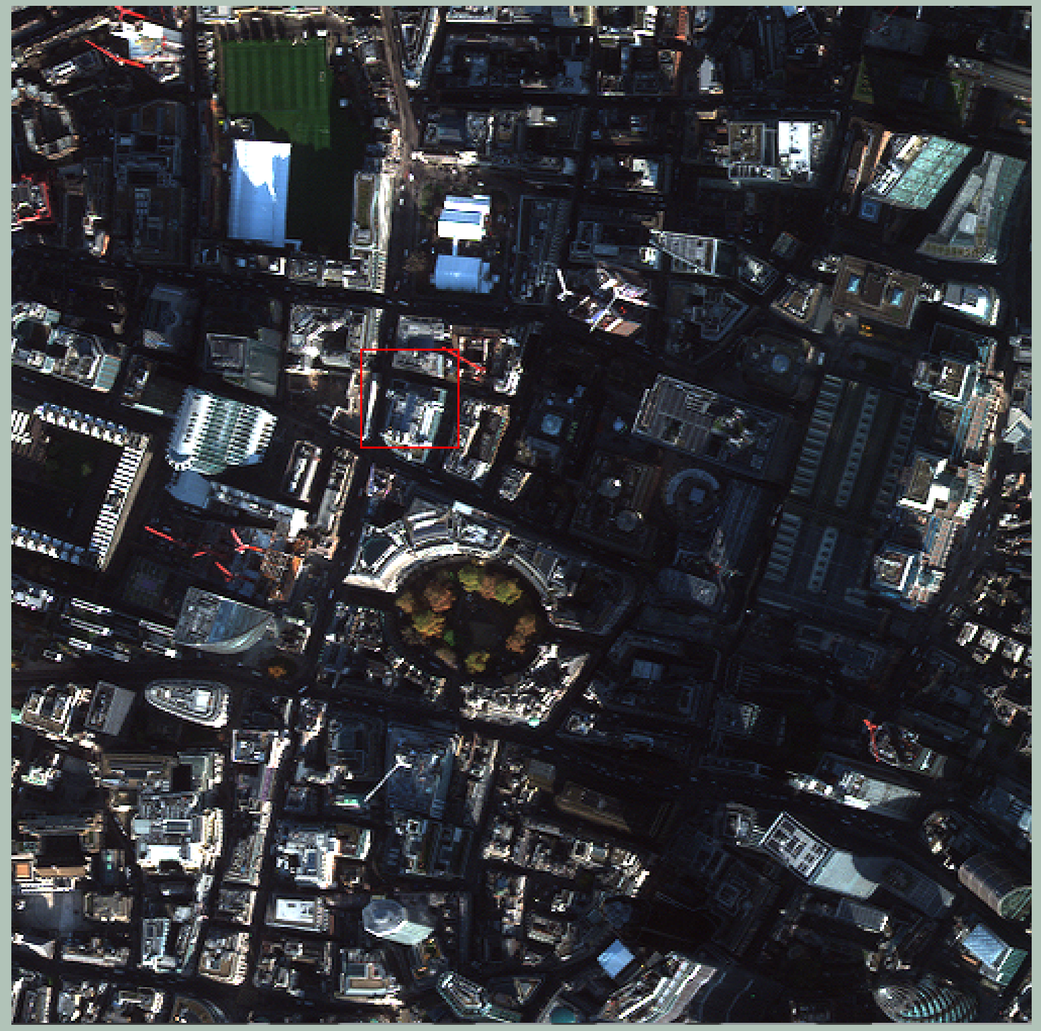
We **reshape** the result of **fusion** algorithm to adjust it at the PAN size.

```
coeff = make_array(channels+1, /FLOAT, VALUE=1)
for i=0,channels-1 do coeff[i+1] = correlate(I0, $
imageLR0[i,*,*], /COVARIANCE)/variance(I0)
```

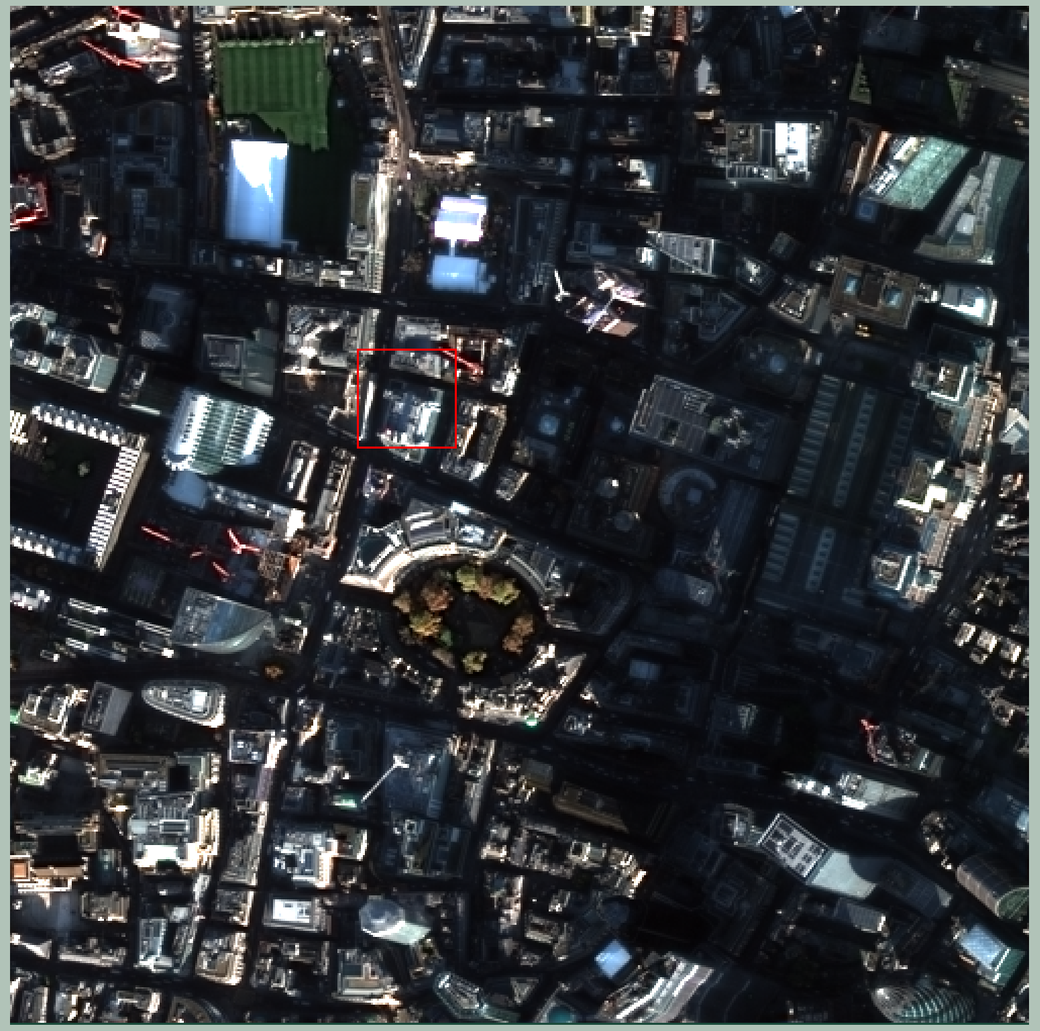
```
size_I0 = size(I0, /DIMENSIONS)
V = make_array(channels+1, size_I0[0]*size_I0[1], /FLOAT, VALUE=0)
V[0,*]=I0[*]
for i=0,channels-1 do V[i+1,*]=(imageLR0[i,*,*])[*]
gm = make_array(size(V, /DIMENSIONS), /FLOAT, VALUE=0)
for i=0,channels do gm[i,*] = coeff[i] * make_array(size_MS[1] $
*size_MS[2], /FLOAT, VALUE=1)
V_hat = V + (new_delta*gm)
size_vhat = size(V_hat)
```

```
I_fus_GSA = make_array(size_MS[0], size_MS[1], size_MS[2], $
/FLOAT, VALUE = 0)
for i=0,channels-1 do I_fus_GSA[i,*,*]=reform(V_hat[i+1,*],$
[size_MS[1], size_MS[2]])
h = make_array(size_MS[0], size_MS[1], size_MS[2], /FLOAT, $
VALUE = 0)
for i=0,channels-1 do h[i,*,*] = I_Fus_GSA[i,*,*]
for i=0,channels-1 do I_Fus_GSA[i,*,*] = h[i,*,*] - $
mean(h[i,*,*]) + mean(imageLR[i,*,*])
```

IDL implementation: the results

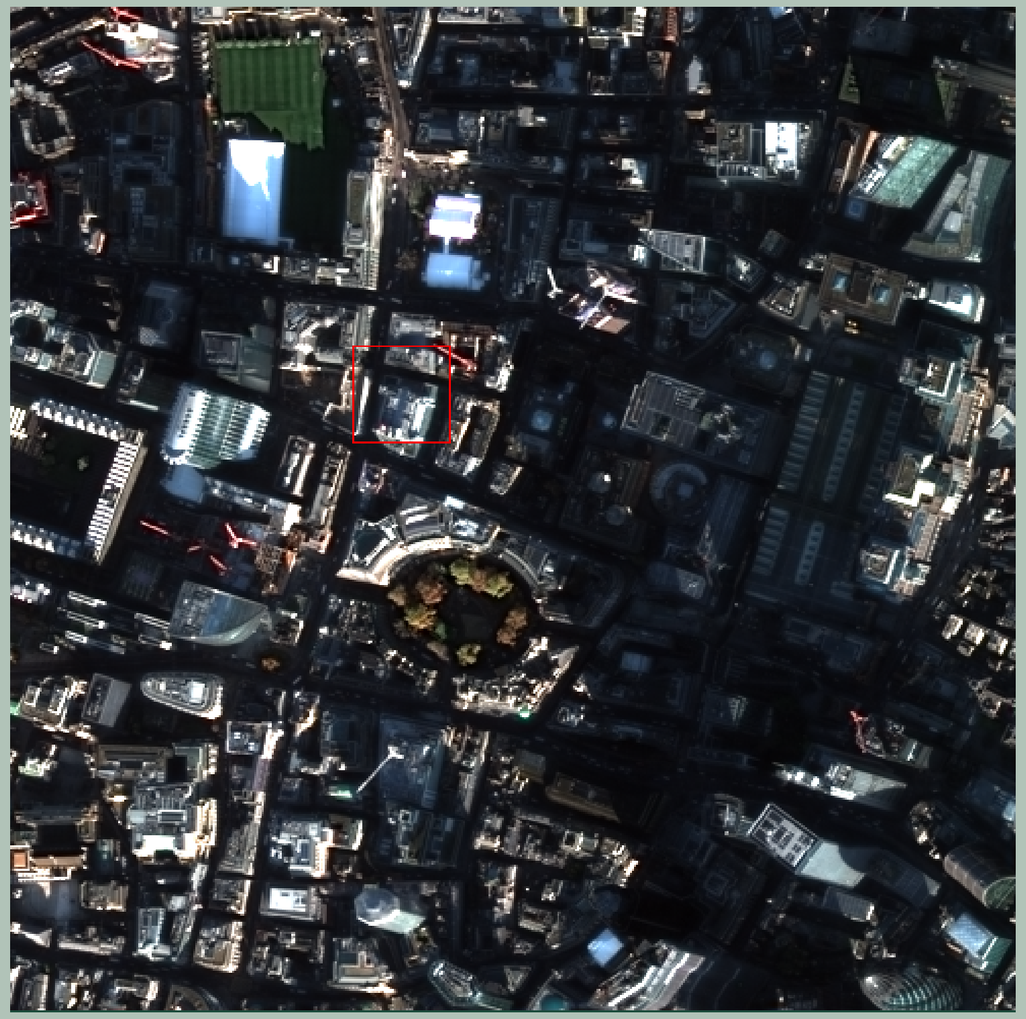


GT

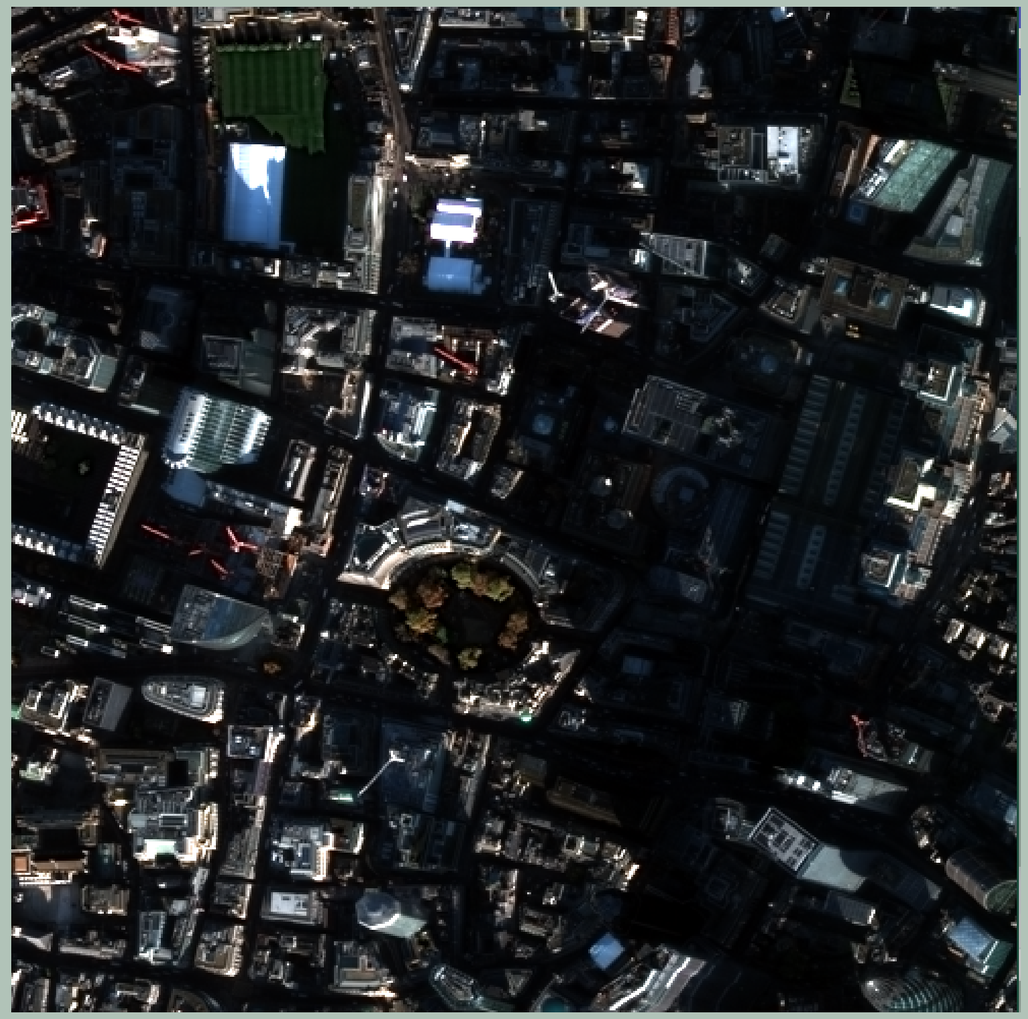


GSA

GSA in IDL and GSA in MATLAB



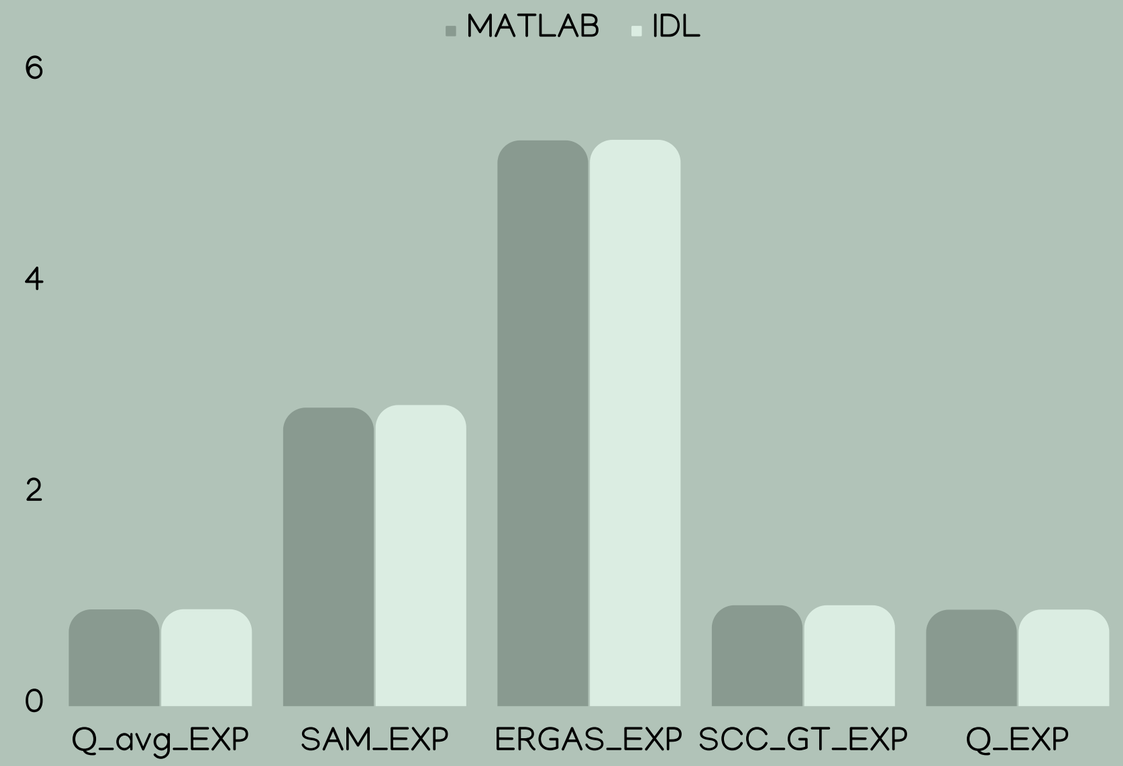
GSA_IDL



I_GSA

GSA in IDL and GSA in MATLAB

	Q2n index	Q index	SAM	ERGAS	SCC
I_GSA	0.9128	0.9099	2.8249	5.3578	0.9521
GSA_IDL	0.9147	0.9112	2.8495	5.3633	0.9521



Overview

1. Introduction
2. Brovey Transform (BT)
3. Gram-Schmidt (GS)
4. Gram-Schmidt Adaptive (GSA)
5. **Segmentation**
6. Gram-Schmidt with Segmentation (GS_Segm)

What is segmentation?

Segmentation algorithms are used **to partition images in K groups of homogenous pixels.** There are multiple algorithms that achieve this.

One such algorithm is **k-means clustering.** It aims to partition n observations into k clusters. Each observation is **assigned to the cluster with the nearest mean** (cluster center).

The k-means clustering algorithm

Initialize the algorithm: specify an initial set of means

→

Assign each sample to the cluster whose mean is closest

→

Update the cluster centers

$$m_i = \frac{1}{|\mathrm{C}_i|} \sum_{z \in \mathrm{C}_i} z$$

→

Test for completion: Stop if assignments no longer (or only marginally) change

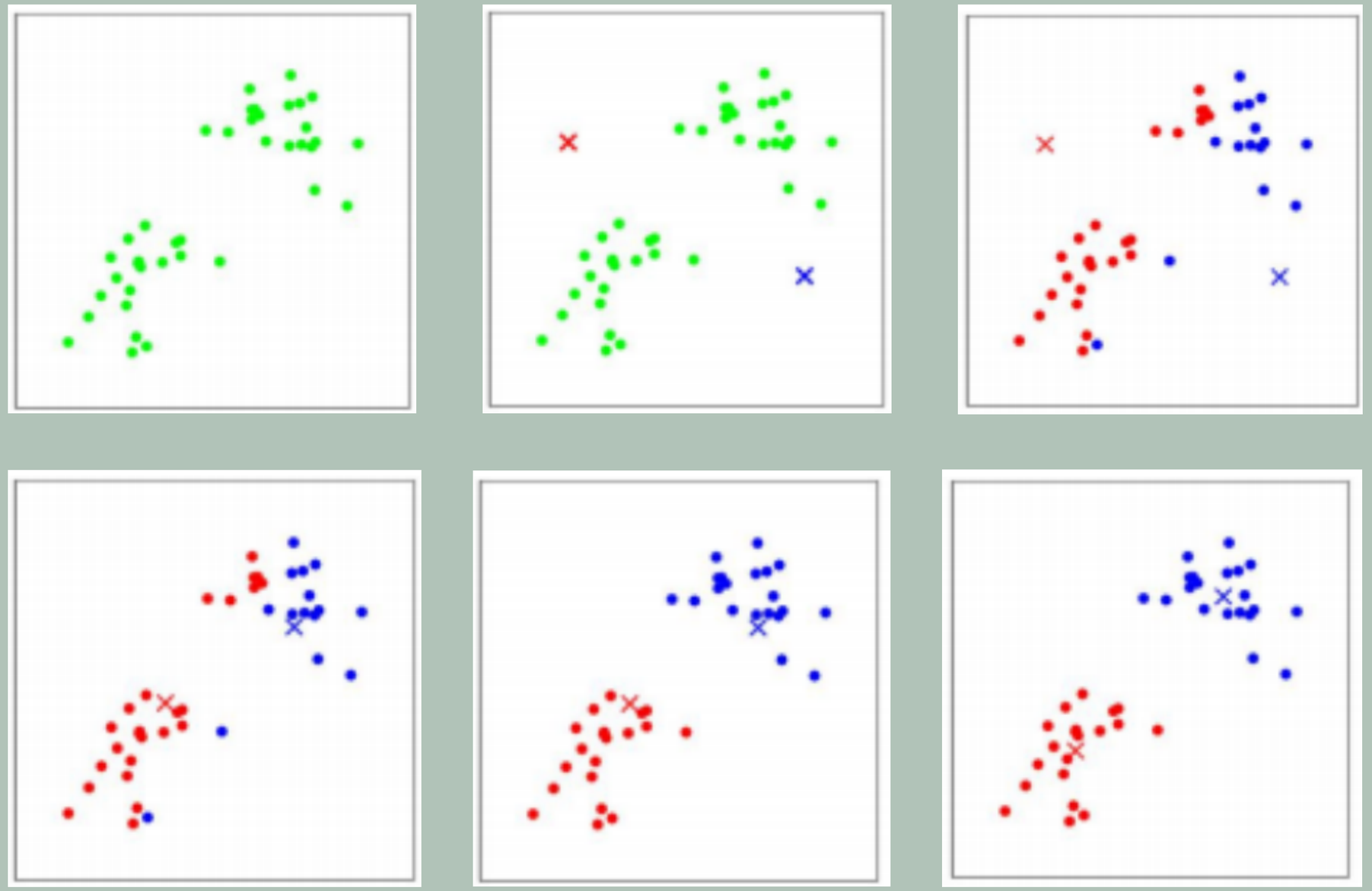
1

2

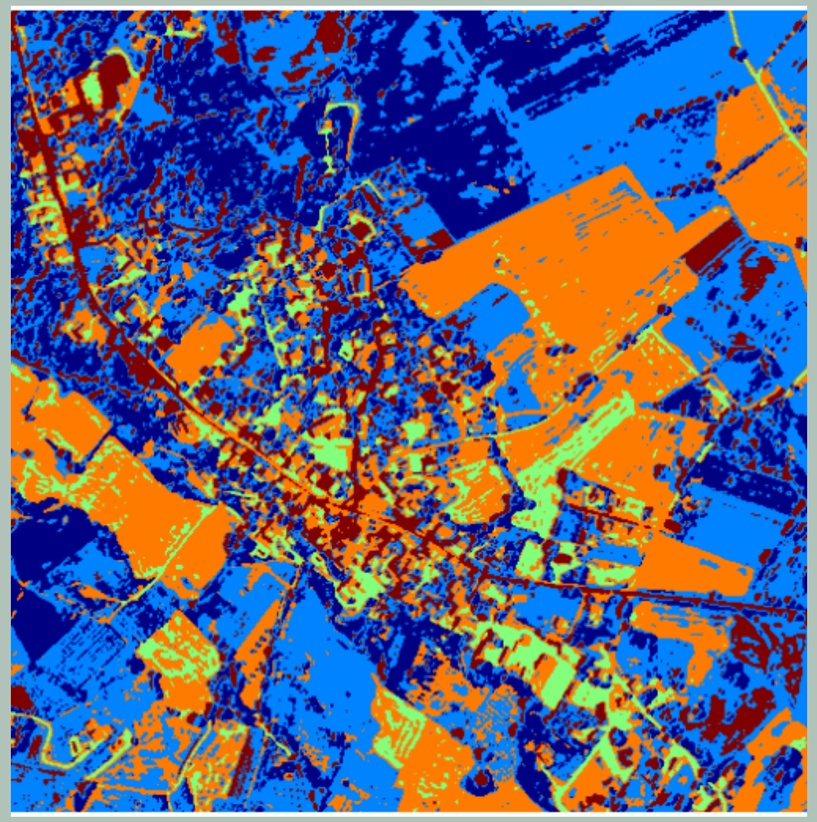
3

4

The k-means clustering algorithm



The k-means clustering algorithm



IDL implementation

IDL, like MATLAB, has **built-in functionality for the k-means algorithm**. We therefore just have to prepare the data accordingly and use the IDL functions correctly.

```
; Compute cluster weights, using three clusters:
weights = CLUST_WTS(array, N_CLUSTERS = 3)
; Compute the classification of each sample:
result = CLUSTER(array, weights, N_CLUSTERS = 3)
```

IDL implementation: preparing the data

The function k_means_clustering takes as **arguments the I_MS image and the number of clusters** the image is supposed to be partitioned into.

```
pro k_means, I_MS, segm, N_SEGM=n_segm
```

Here, we generate a **0-value array of the same dimensions** as the I_MS image. This array is then filled (see next slide) and later **resized**, because the IDL k_means function takes only a 2-dimensional array.

```
size_I_MS = size(I_MS, /dimensions)
F1 = fltarr(size_I_MS)
channels = size_I_MS[0]
num_elements = size_I_MS[1] * size_I_MS[2]
```

```
F2 = reform(F1, channels, num_elements)
```

IDL implementation: preparing the data

The loop **fills the F1 array with normalized values,** by looking at **each band separately** and **dividing each value by the maximum value** for the corresponding band.

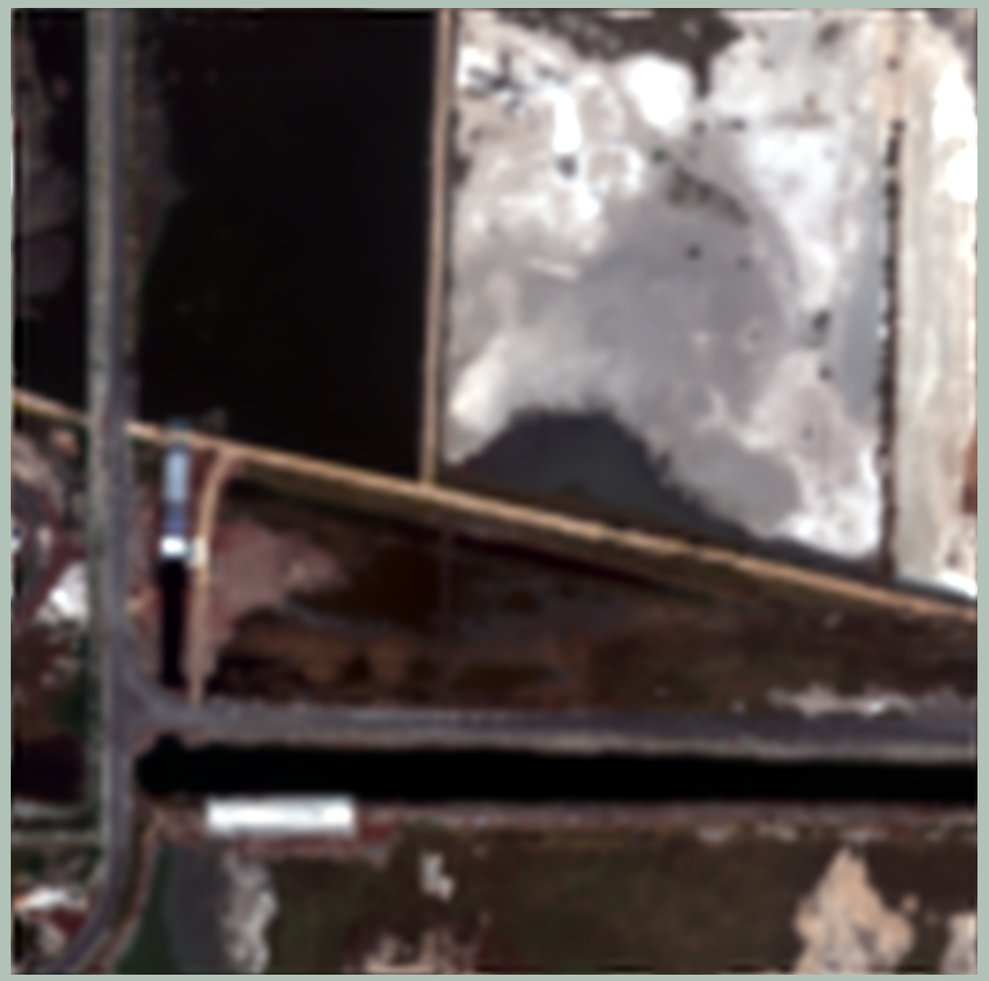
```
FOR ibands = 0, channels-1 DO BEGIN
  a = I_MS[ibands, *, *]
  max_value = max(a)
  F1[ibands, *, *] = a/max_value
ENDFOR
```

IDL implementation: using k_means

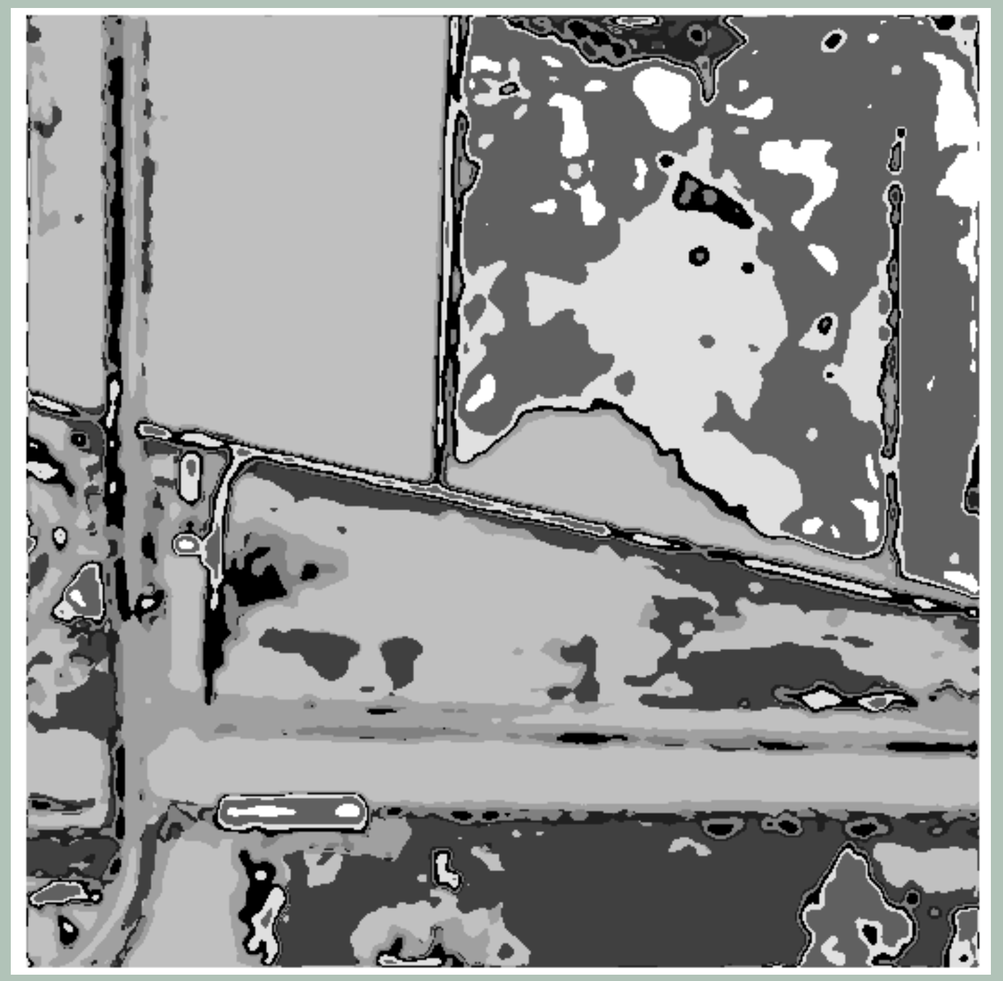
Now, we can use the **built-in k-means functionality**. We **resize the result to the image's original dimensions** and **return** the matrix, which assigns each pixel to one of the clusters.

```
weights = CLUST_WTS(F2, N_CLUSTERS = n_segm)
IDX = CLUSTER(F2, weights, N_CLUSTERS = n_segm)
segm = reform(IDX, size_I_MS[1], size_I_MS[2])
```

IDL implementation: the results



I_MS



Segmented Image

Overview

1. Introduction
2. Brovey Transform (BT)
3. Gram-Schmidt (GS)
4. Gram-Schmidt Adaptive (GSA)
5. Segmentation
6. **Gram–Schmidt with Segmentation (GS_Segm)**

IDL implementation: input

The input for the GS_Segm algorithm is the **panchromatic image,** the **multispectral image upscaled to PAN dimensions,** the **low resolution version of the PAN image,** and the **segmented image,** the result of the k_means function.

```
pro gs_segm, PAN, I_MS, I_LR_input, S, I_GS_Segm
```

IDL implementation: init

We create the **I_PAN** image by replicating the PAN image for each band of the I_MS image.

```
X = REFORM(PAN, 1, size_I_MS[1], size_I_MS[2])
temp = fltarr(size_I_MS)
for i=0,channels-1 do temp[i,*,*] = X
I_PAN = temp
```

Single band **I_LR** images are transformed in multiband images by replicating the band for each channel I_LR and I_PAN are supposed to have the same number of bands.

```
IF size_I_LR[0] EQ 2 THEN BEGIN
  X = REFORM(I_LR_input, 1, size_I_MS[1], size_I_MS[2])
  temp = fltarr(size_I_MS)
  for i=0,channels-1 do temp[i,*,*] = X
  I_LR_input = temp
ENDIF

size_I_PAN = size(I_PAN)
size_I_LR = size(I_LR_input)
IF size_I_LR[1] NE size_I_PAN[1] THEN BEGIN
  PRINT, 'I_LP should have the same number of bands as PAN'
ENDIF
```

IDL implementation: injection and coefficients

The injection matrix **DetailsHRPan** is calculated by deducting I_LR from I_PAN.

```
DetailsHRPan = I_PAN - I_LR_input
```

Then, we calculate the **coefficients**. For each I_MS and I_LR band we remove singleton dimensions. For each partition class of the segmented image, the coefficient is then calculated from the covariance and variance of I_LR and I_MS pixels that belong to this class.

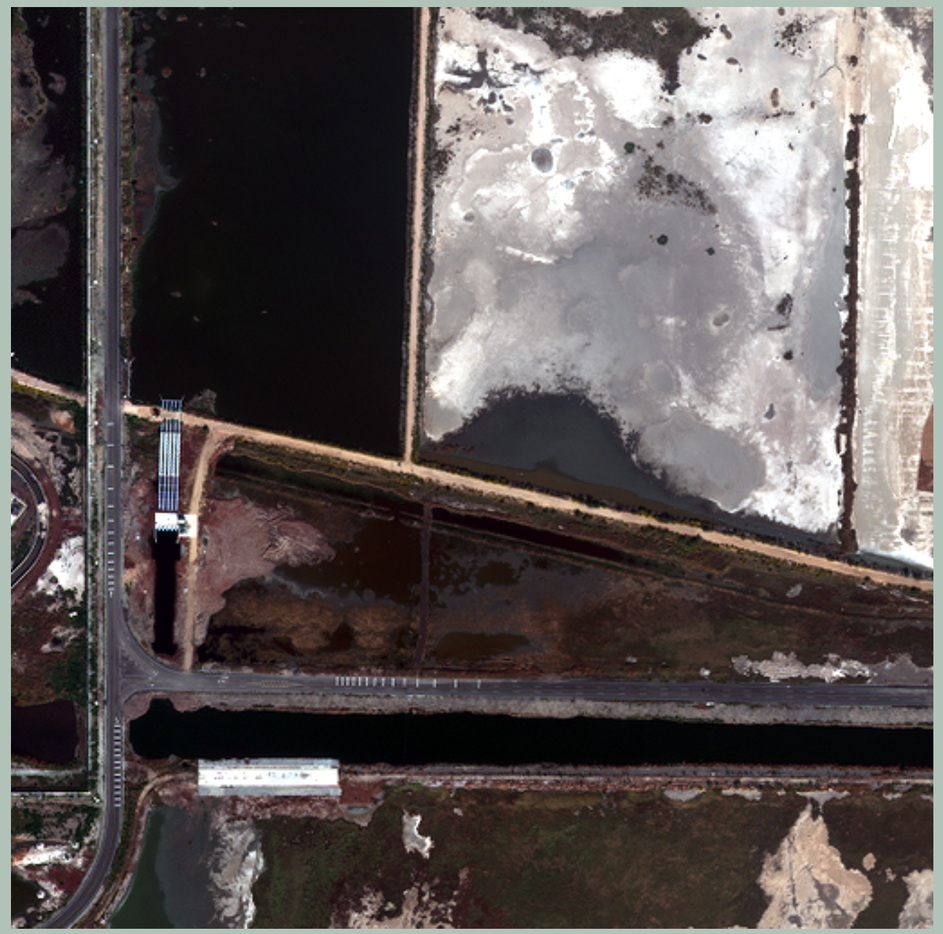
```
Coeff = fltarr(size_I_MS)
labels = S[UNIQ(S, SORT(S))]

; Calculation coefficients for each band
FOR ii = 0, size_I_MS[0]-1 DO BEGIN
  MS_Band_before = I_MS[ii, *, *]
  size_MS_Band = size(MS_Band_before, /dimensions)
  MS_Band = REFORM(MS_Band_before, size_MS_Band[1], size_MS_Band[2])

  I_LR_Band_before = I_LR_input[ii, *, *]
  size_I_LR_Band = size(I_LR_Band_before, /dimensions)
  I_LR_Band = REFORM(I_LR_Band_before, size_I_LR_Band[1], size_I_LR_Band[2])

  Coeff_Band = fltarr(size(I_LR_Band, /dimensions))

  ; Calculation Coefficient related to each partition
  FOR il=0, (size(labels, /dimensions))[0]-1 DO BEGIN
    IDX  = S EQ labels[il]
    c = CORRELATE(I_LR_Band[idx], MS_Band[idx], /covariance)
    Coeff_Band[where(idx)] = c / VARIANCE(I_LR_BAND[idx])
  ENDFOR

  Coeff[ii, *, *] = Coeff_Band
ENDFOR
```

IDL implementation: fusion

```
I_GS_Segm = Coeff * DetailsHRPan + I_MS
```

Finally, we **create the pansharpened image** by multiplying the coefficient matrix with the injection matrix and adding those details onto the I_MS image.

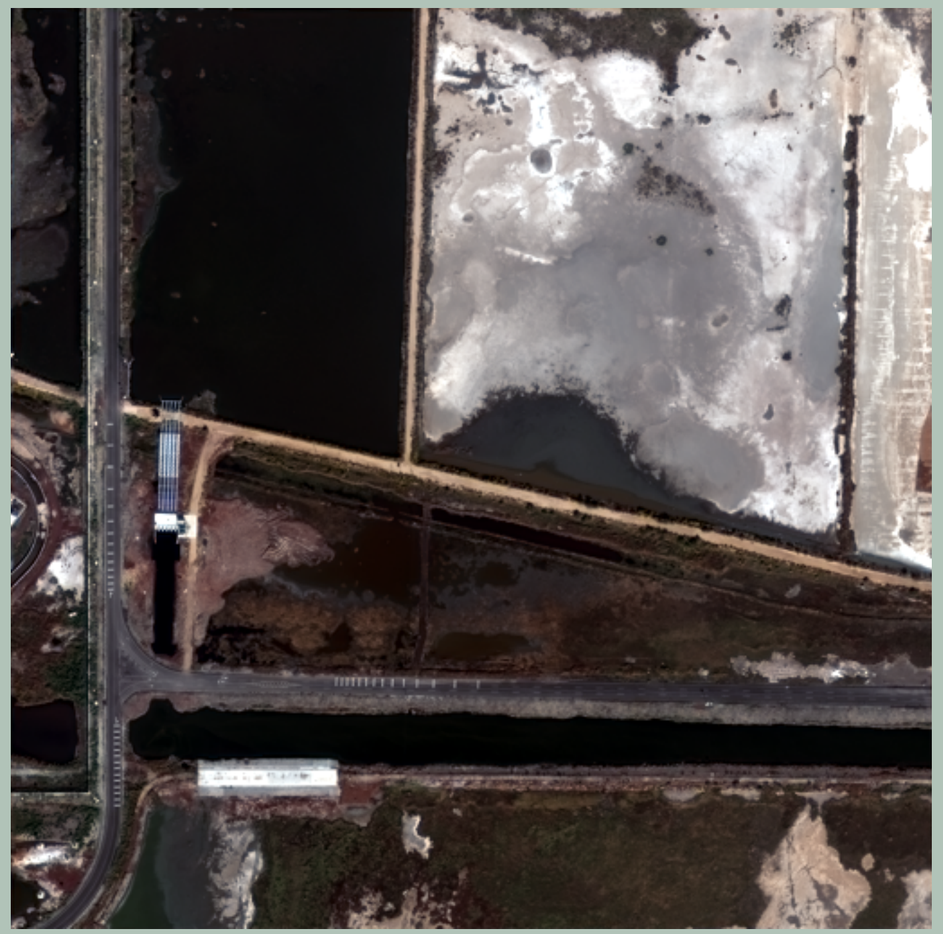
IDL implementation: the results



GT

GS_Segm (IDL)

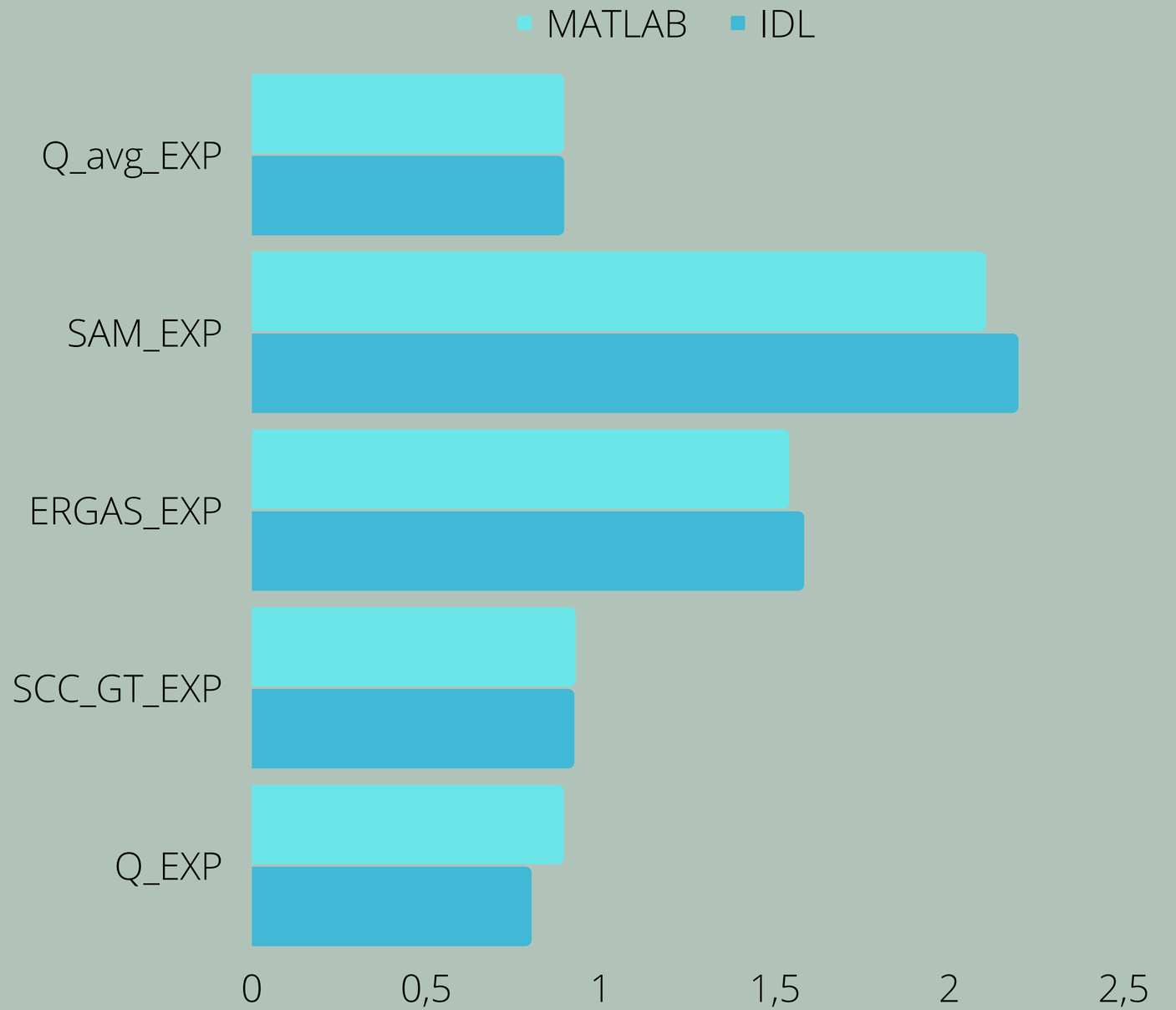
IDL implementation: the results



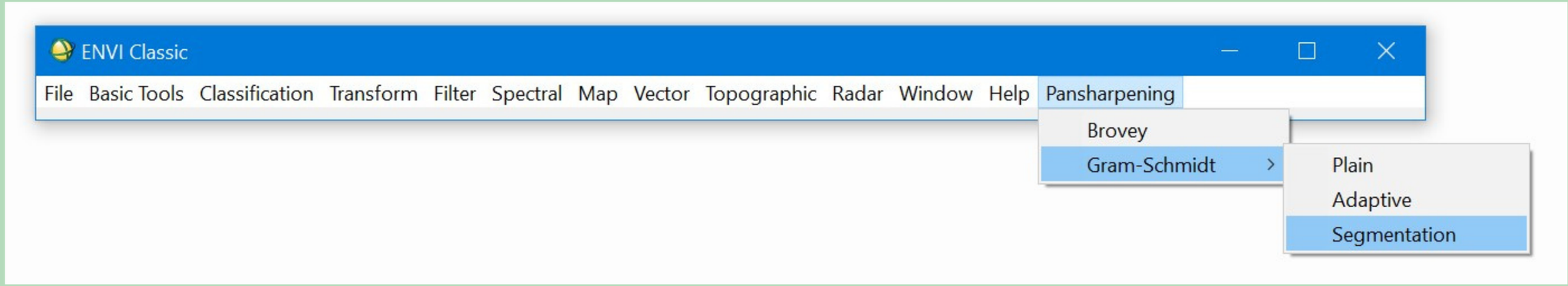
GS_Segm (Matlab)

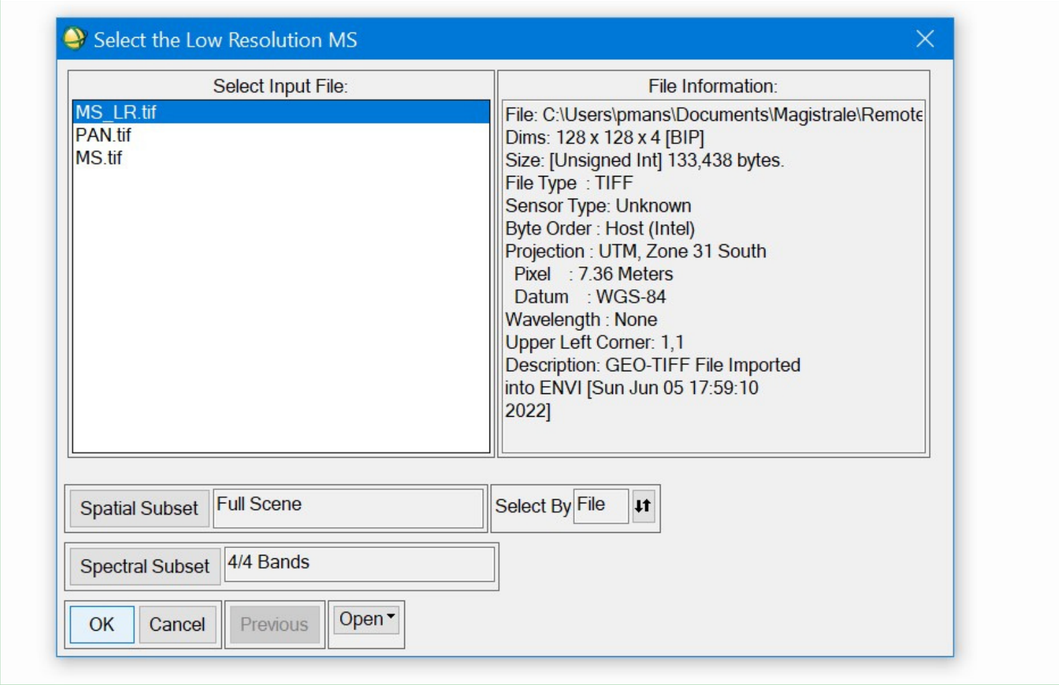
GS_Segm (IDL)

GS_Segm in IDL and MATLAB

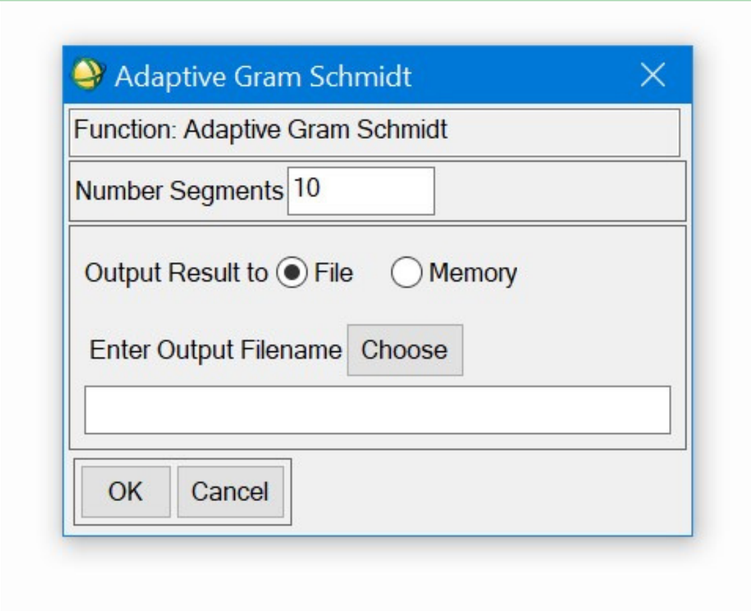


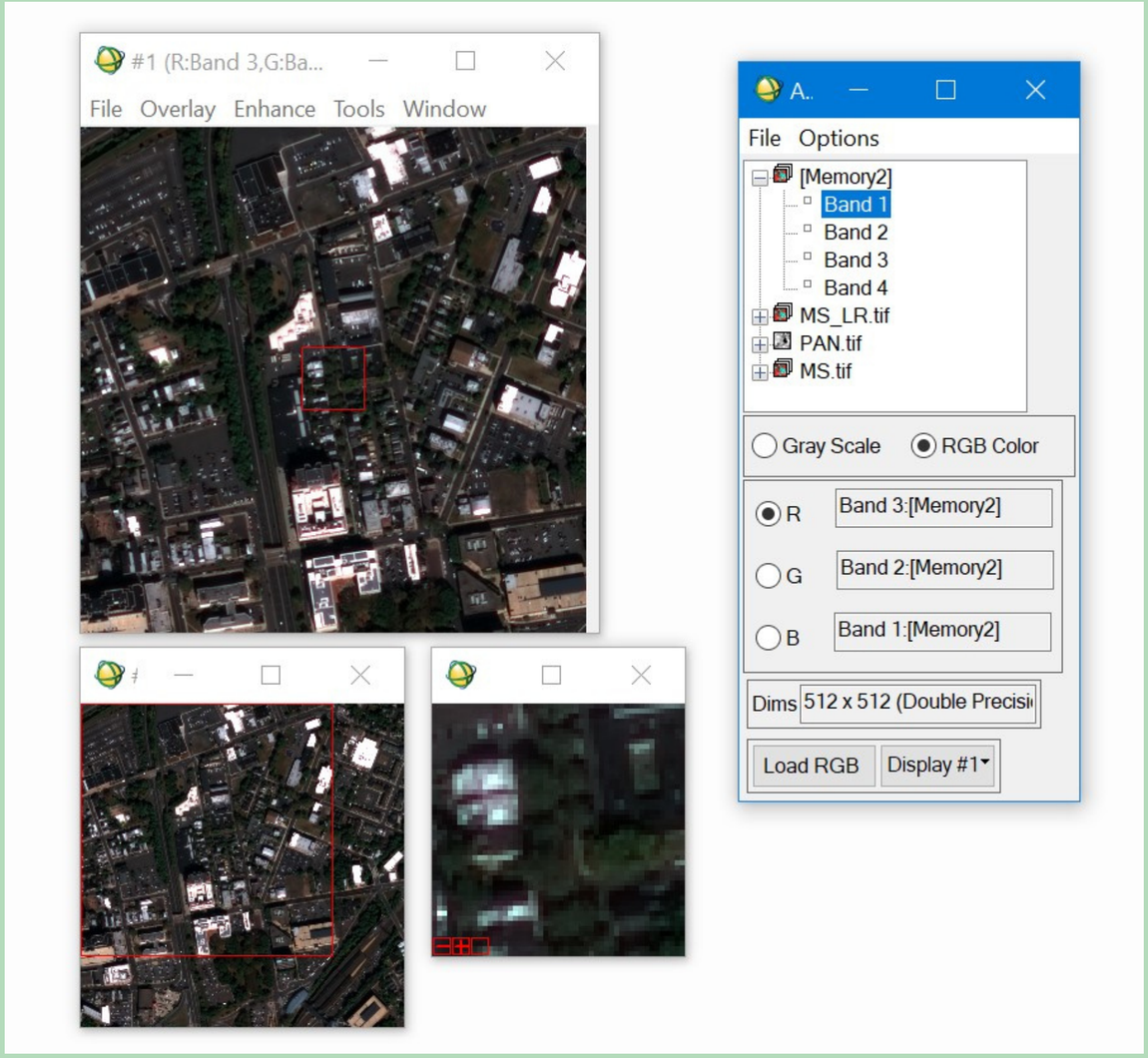
ENVI interface (1)





ENVI interface (2)





Thank you for your attention!

Brolich Nina, Carbone Alessia, Mansi Paolo