

# L13 Sparse plus Low-Rank Splitting

Ivan Slapničar

May 17, 2018

## 1 Sparse + Low-Rank Splitting

Suppose we are given a data matrix  $A$ , and know that it has a form

$$A = L + S,$$

where  $L$  is of low-rank and  $S$  is sparse, but we know neither the rank of  $L$  nor the non-zero entries of  $S$ .

*Can we recover  $L$  and  $S$ ?*

The answer is *YES, with high probability and with an efficient algorithm.*

*Sparse + Low-rank* splitting can be successfully applied to video surveillance, face recognition, latent semantic indexing, and ranking and collaborative filtering.

### 1.1 Prerequisites

The reader should be familiar with linear algebra concepts, particularly SVD and its properties and algorithms.

### 1.2 Competences

The reader should be able to apply sparse + low-rank splitting to real problems.

### 1.3 References

For more details see [E. J. Candes, X. Li, Y. Ma, and J. Wright, Robust Principal Component Analysis?](#)

**Credits:** The author thanks Dimitar Ninevski, a former IAESTE intern, for collecting and preparing some of the material.

### 1.4 Definitions

Let  $A \in \mathbb{R}^{m \times n}$  have rank  $r$ , and let  $A = U\Sigma V^T$  be its SVD.

The **nuclear norm** of  $A$  is  $\|A\|_* = \sum_{i=1}^r \sigma_i(A)$ .

Let  $\|A\|_1 = \sum_{i,j} |A_{ij}|$  denote the 1-norm of  $A$  seen as a long vector.

Let  $\|A\|_\infty = \max_{i,j} |A_{ij}|$  denote the  $\infty$ -norm of  $A$  seen as a long vector.

Given  $\tau > 0$ , the **shrinkage operator**  $\mathcal{S}_\tau : \mathbb{R} \rightarrow \mathbb{R}$  is defined by  $\mathcal{S}_\tau(x) = \text{sign}(x) \max\{|x| - \tau, 0\}$ , and is extended to matrices by applying it to each element.

Given  $\tau > 0$ , the **singular value thresholding operator** is  $\mathcal{D}_\tau(A) = U\mathcal{S}_\tau(\Sigma)V^T$ .

## 1.5 Facts

Let  $A = L + S$  be the splitting that we are looking for.

1. The problem can be formulated as  $\arg \min_{\text{rank}(L) \leq k} \|A - L\|_2$ .
2. The problem makes sense if the **incoherence conditions**

$$\max_i \|U_{:,1:r}^T e_i\|_2^2 \leq \frac{\mu r}{m}, \quad \max_i \|V_{:,1:r}^T e_i\|_2^2 \leq \frac{\mu r}{n}, \quad \|UV^T\|_\infty \leq \sqrt{\frac{\mu r}{m \cdot n}},$$

hold for some parameter  $\mu$ .

3. If the incoherence conditions are satisfied, the **Principal Component Pursuit estimate**,

$$\arg \min_{L + S = A} \|L\|_* + \lambda \|S\|_2,$$

exactly recovers  $L$  and  $S$ .

4. *Principal Component Pursuit by Alternating Directions* algorithm finds the above estimate

1. *Initialization:*  $S = 0, Y = 0, L = 0, \mu > 0, \delta = 10^{-7}$ .
2. *Iterations:* while  $\|A - L - S\|_F > \delta \|A\|_F$  repeat
  1. *SV Thresholding:*  $L = \mathcal{D}_{\mu^{-1}}(A - S - \mu^{-1}Y)$
  2. *Shrinkage:*  $S = \mathcal{S}_{\lambda\mu^{-1}}(A - L + \mu^{-1}Y)$
  3. *Updating:*  $Y = Y + \mu(A - L - S)$

### 1.5.1 Example - Random matrices

```
In [1]: # Shrinkage
function Shr{T}(x::Array{T}, τ::T)
    sign.(x).*max.(abs.(x)-τ, zero(T))
end
```

```
Out[1]: Shr (generic function with 1 method)
```

```
In [2]: A=rand(3,5)
```

```
Out[2]: 3×5 Array{Float64,2}:
0.0945503 0.76331 0.553415 0.885066 0.921344
0.95049 0.938383 0.471215 0.534323 0.443028
0.260183 0.0697949 0.523192 0.508673 0.0866353
```

```
In [3]: Shr(A,0.5)
```

```
Out[3]: 3×5 Array{Float64,2}:
0.0 0.26331 0.0534149 0.385066 0.421344
0.45049 0.438383 0.0 0.0343231 0.0
0.0 0.0 0.0231922 0.00867277 0.0
```

```
In [4]: # Singular value thresholding
function D{T}(A::Array{T},τ::T)
    #  $U, \sigma, V = svd(A)$ 
    # This can be replaced by a faster approach
    U,R=qr(A)
    u,σ,V=svd(R)
    S=Shr(σ,τ)
    k=sum(S.>zero(T))
    #  $U * (u[:, 1:k] * diagm(S[1:k])) * V[:, 1:k]'$ 
    A_mul_B!(similar(A),U,(u[:,1:k]*diagm(S[1:k]))*V[:,1:k]')
end
```

```
Out[4]: D (generic function with 1 method)
```

```
In [5]: D(A,0.5)
```

```
Out[5]: 3×5 Array{Float64,2}:
0.27 0.585206 0.459591 0.641622 0.576396
0.550542 0.644151 0.42348 0.517112 0.417607
0.163466 0.255712 0.186028 0.246405 0.212929
```

```
In [6]: function PCPAD{T}(A::Array{T})
    # Initialize
    δ=1.0e-7
    tol=δ*vecnorm(A)
    m,n=size(A)
    S=zeros(A)
    Y=zeros(A)
    L=zeros(A)
    μ=(m*n)/(4*(norm(A[:,],1)))
    μ₁=one(T)/μ
    λ=one(T)/sqrt(max(map(T,m),n))
```

```

λμ₁=λ*μ₁
ν=1e20
maxiter=1000
iterations=0
# Iterate
while (ν>tol) && iterations<maxiter
    iterations+=1
    L=D(A-S+μ₁*Y,μ₁)
    S=Shr(A-L+μ₁*Y,λμ₁)
    T₁=A-L-S
    Y+=μ*T₁
    ν=vecnorm(T₁)
end
L,S, iterations
end

```

Out[6]: PCPAD (generic function with 1 method)

In [7]: @time L,S,iter=PCPAD(A)

0.469537 seconds (232.63 k allocations: 11.565 MiB, 1.92% gc time)

Out[7]: ([0.0945504 0.685198 ... 0.627532 0.520311; 0.0805066 0.583424 ... 0.534323 0.443028; 0.

In [8]: rank(L),vecnorm(A-L-S)

Out[8]: (1, 1.9269529249929484e-7)

In [9]: L

Out[9]: 3×5 Array{Float64,2}:

0.0945504	0.685198	0.553415	0.627532	0.520311
0.0805066	0.583424	0.471215	0.534323	0.443028
0.0157433	0.11409	0.0921473	0.104488	0.0866353

In [10]: S

Out[10]: 3×5 Array{Float64,2}:

-0.0	0.0781125	0.0	0.257534	0.401033
0.869984	0.354959	0.0	0.0	0.0
0.24444	-0.0442952	0.431045	0.404184	-0.0

In [11]: # Now the real test  
# Dimensions of the matrix  
m=100

```

n=100
# Rank of the low-rank part L
k=10
# Generate L
L=rand(m,k)*rand(k,n)
rank(L)

```

Out[11]: 10

```

In [12]: # Sparsity of the sparse part S
sparsity=0.1
# Generate S
S=10*sprand(m,n,sparsity)
nnz(S)

```

Out[12]: 967

```

In [13]: # Generate the matrix A, it is a full matrix with full rank
A=L+S
rank(A)

```

Out[13]: 100

```

In [14]: # Decompose A into L1 and S1
@time L1,S1,iters=PCPAD(A);

```

1.712229 seconds (16.32 k allocations: 373.873 MiB, 4.50% gc time)

```

In [15]: iters, rank(L1), vecnorm(L), vecnorm(L-L1), vecnorm(S), vecnorm(S-S1)

```

Out[15]: (179, 10, 265.841253972897, 3.672717372801588e-5, 181.20690540475104, 2.003383819517419

Although there might be no convergence, the splitting is still good.

```
In [16]: S1
```

```

Out[16]: 100×100 Array{Float64,2}:
 0.0      -0.0      0.0      2.67588   ...     0.0      0.0      -0.0
 -0.0      5.86396   0.0      -0.0      0.0      -0.0      0.0
 -0.0      0.0      0.0      0.0      -0.0      -0.0      3.74897
 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      6.96026   -0.0      0.0      ...    -0.0      -0.0      0.0
 0.0      0.0      0.0      -0.0      0.0      5.16392   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	6.6661	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	3.86584	-0.0
-0.0	0.0	0.0	-0.0	5.26389	0.0	0.0
0.0	0.0	0.0	0.0	0.321521	-0.0	-0.0
:				..		
1.27772	-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	-0.0	-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	6.04261	-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	4.45669	0.0	-0.0	1.31467	0.0
-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
-0.0	1.18736	-0.0	-0.0	0.0	-0.0	0.0
-0.0	-0.0	0.0	0.0	-0.0	0.0	-0.0

In [17]: full(S)

Out[17]: 100×100 Array{Float64,2}:

0.0	0.0	0.0	2.67588	... 0.0	0.0	0.0	0.0
0.0	5.86396	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	3.74897
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	6.96026	0.0	0.0	... 0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	5.16392	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	6.6661	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	... 8.32626	0.0	3.86584	0.0
0.0	0.0	0.0	0.0	9.77069	5.26389	0.0	0.0
0.0	0.0	0.0	0.0	7.2489	0.321521	0.0	0.0
:				..			
1.27772	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	0.0	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	6.04261	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	4.45669	0.0	0.0	0.0	1.31467	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.18736	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

### 1.5.2 Example - Face recognition

We will try to recover missing features. The images are chosen from the [Yale Face Database](#).

In [18]: `using Images, Colors`

In [19]: *# First a single image - no help*  
`img=load("files/17.jpg")`

Out[19] :



In [20]: `show(img[1,1])`

`Gray{NOf8}(0.325)`

In [21]: *# Convert image to matrix*  
`A=map(Float64,img)`

Out[21]: `192×168 Array{Float64,2}:`

0.32549	0.305882	0.290196	...	0.0470588	0.0431373	0.0392157
0.301961	0.294118	0.290196		0.0470588	0.0431373	0.0392157
0.286275	0.294118	0.309804		0.0470588	0.0431373	0.0392157
0.290196	0.313725	0.341176		0.0470588	0.0431373	0.0431373
0.301961	0.32549	0.352941		0.054902	0.0470588	0.0431373
0.309804	0.32549	0.341176	...	0.0627451	0.0509804	0.0470588
0.321569	0.32549	0.32549		0.0745098	0.0588235	0.0470588
0.337255	0.329412	0.321569		0.0784314	0.0588235	0.0470588
0.329412	0.32549	0.313725		0.0980392	0.0705882	0.0509804

0.32549	0.329412	0.333333	0.109804	0.0862745	0.0705882
0.317647	0.337255	0.352941	...	0.117647	0.101961
0.32549	0.352941	0.376471	0.117647	0.105882	0.0980392
0.352941	0.380392	0.4	0.113725	0.101961	0.0901961
:			.. .	:	
0.027451	0.027451	0.027451	...	0.027451	0.027451
0.0313725	0.0313725	0.0313725	0.027451	0.027451	0.0313725
0.0313725	0.0313725	0.0313725	0.027451	0.027451	0.0313725
0.0313725	0.0313725	0.0313725	0.027451	0.027451	0.0313725
0.0196078	0.027451	0.0352941	0.0352941	0.0352941	0.0352941
0.0196078	0.027451	0.0352941	...	0.0352941	0.0352941
0.0156863	0.0235294	0.0313725	0.0352941	0.0352941	0.0352941
0.0117647	0.0196078	0.027451	0.0352941	0.0352941	0.0352941
0.00392157	0.0156863	0.0235294	0.0352941	0.0352941	0.0352941
0.0	0.00784314	0.0196078	0.0352941	0.0352941	0.0352941
0.0	0.00392157	0.0156863	...	0.0352941	0.0352941
0.0	0.00392157	0.0156863	0.0352941	0.0352941	0.0352941

In [22]: # Compute the splitting and show number of iterations  
@time L,S,iters=PCPAD(A)  
iters, rank(L), vecnorm(A), vecnorm(A-L-S)

23.824762 seconds (95.01 k allocations: 6.417 GiB, 3.55% gc time)

Out[22]: (1000, 83, 40.54262152276891, 0.00039768766719959264)

In [23]: colorview(Gray,L)

Out[23] :



```
In [24]: # Try S+0.5  
colorview(Gray,S)
```

Out[24] :



```
In [25]: # Another image  
img=load("files/19.jpg")
```

Out[25] :



```
In [26]: A=map(Float64,img)
L,S,iters=PCPAD(A)
@show iters, rank(L), vecnorm(A), vecnorm(A-L-S)

(iters, rank(L), vecnorm(A), vecnorm((A - L) - S)) = (1000, 84, 53.083494981456965, 0.0003518714
```

```
Out[26]: (1000, 84, 53.083494981456965, 0.0003518714694024946)
```

```
In [27]: hcat(img,colorview(Gray,L),colorview(Gray,S+0.5))
```

```
Out[27]:
```



### 1.5.3 Example - Multiple images

Each image of  $168 \times 192$  pixels is converted to a vector of length 32256. All vectors are stacked in the columns of matrix  $A$ , and the low-rank + sparse splitting of  $A$  is computed.

```
In [28]: # Load all images in the collection
dir="../files/yaleB08/"
files=readdir(dir)
```

```
Out[28]: 64-element Array{String,1}:
".directory"
"yaleB08_P00A+000E+00.pgm"
"yaleB08_P00A+000E+20.pgm"
"yaleB08_P00A+000E+45.pgm"
```

```

"yaleB08_P00A+000E-20.pgm"
"yaleB08_P00A+000E-35.pgm"
"yaleB08_P00A+005E+10.pgm"
"yaleB08_P00A+005E-10.pgm"
"yaleB08_P00A+010E+00.pgm"
"yaleB08_P00A+010E-20.pgm"
"yaleB08_P00A+015E+20.pgm"
"yaleB08_P00A+020E+10.pgm"
"yaleB08_P00A+020E-10.pgm"
:
"yaleB08_P00A-070E+00.pgm"
"yaleB08_P00A-070E+45.pgm"
"yaleB08_P00A-070E-35.pgm"
"yaleB08_P00A-085E+20.pgm"
"yaleB08_P00A-085E-20.pgm"
"yaleB08_P00A-095E+00.pgm"
"yaleB08_P00A-110E+15.pgm"
"yaleB08_P00A-110E+40.pgm"
"yaleB08_P00A-110E+65.pgm"
"yaleB08_P00A-110E-20.pgm"
"yaleB08_P00A-120E+00.pgm"
"yaleB08_P00A-130E+20.pgm"

```

```

In [31]: n=length(files)-1
         images=Array{Any}(n)
         B=Array{Any}(n)
         for i=1:n
             images[i]=load(joinpath(dir,files[i+1]))
             B[i]=map(Float64,images[i])
         end

```

```

In [34]: # See the images - last 9 images are meaningless
          # for i=1:n; display(img[i]); end

```

```

In [35]: # Form the big matrix - each image is converted to a column vector
         mi,ni=size(img[1])
         A=Array{Float64}(mi*ni,n-9)
         for i=1:n-9
             A[:,i]=vec(B[i])
         end
         size(A)

```

```

Out[35]: (32256, 54)

```

```

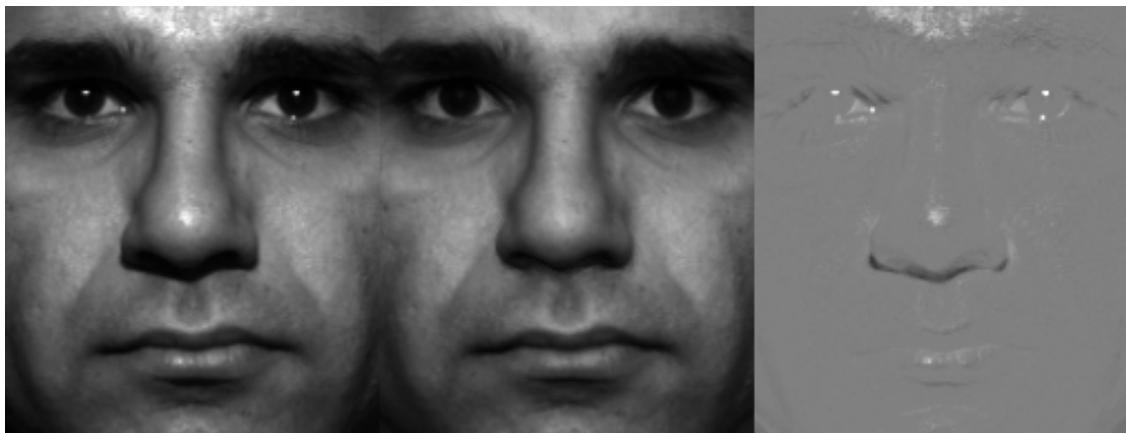
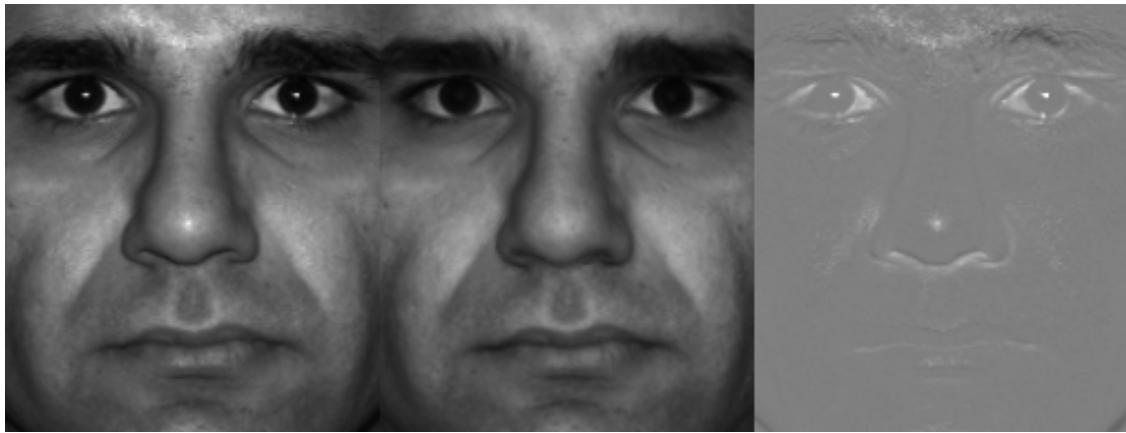
In [36]: # Now the big SVDs - 3-5 minutes
@time L,S,iters=PCPAD(A)
iters, rank(L), norm(A), norm(A-L-S)

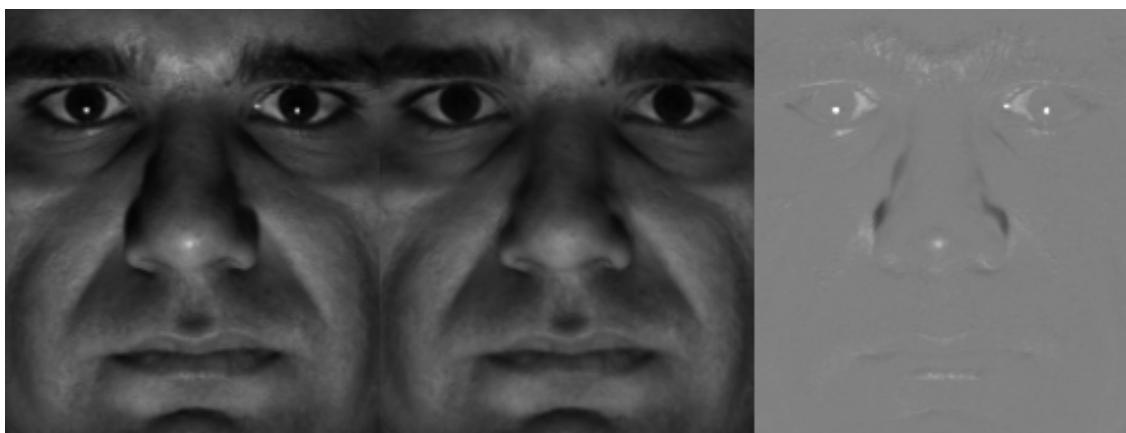
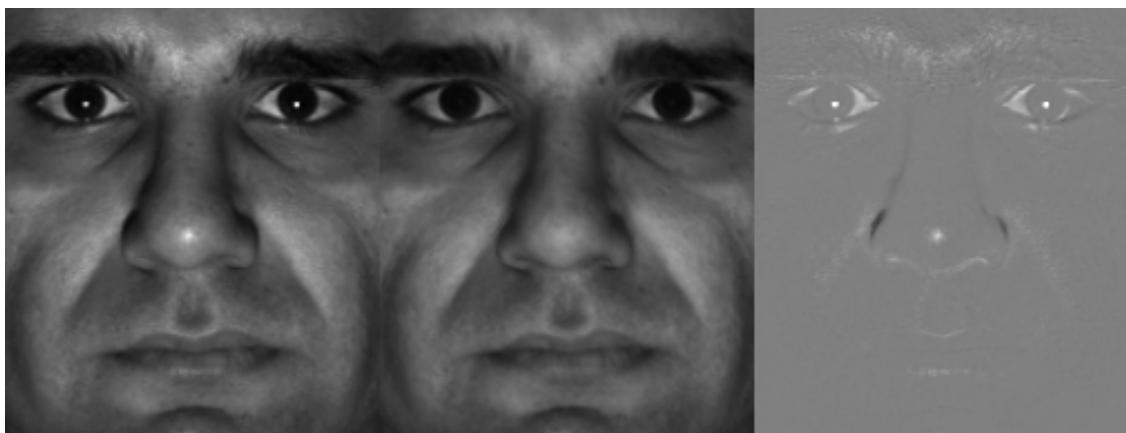
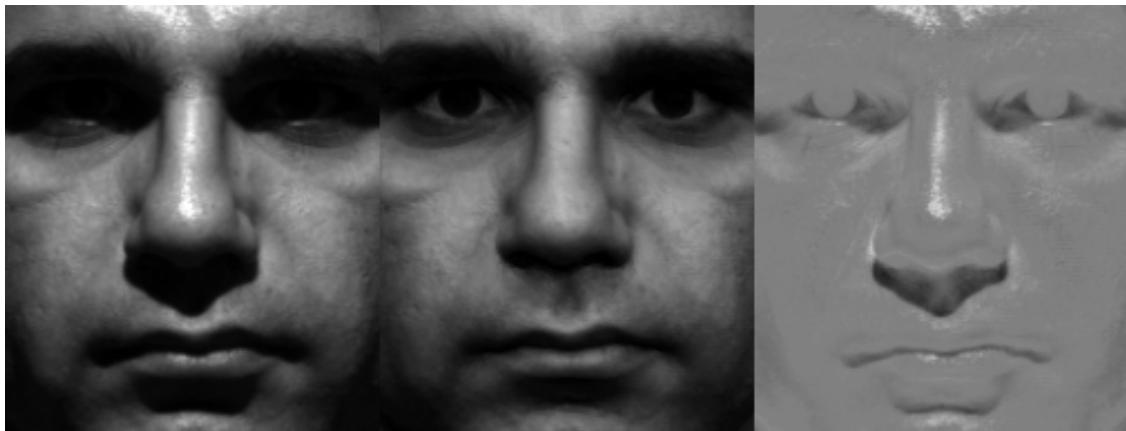
```

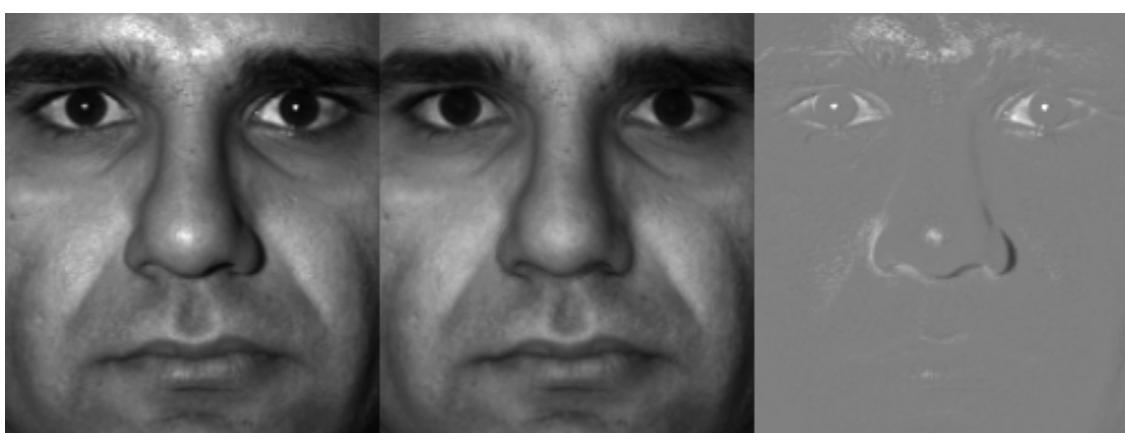
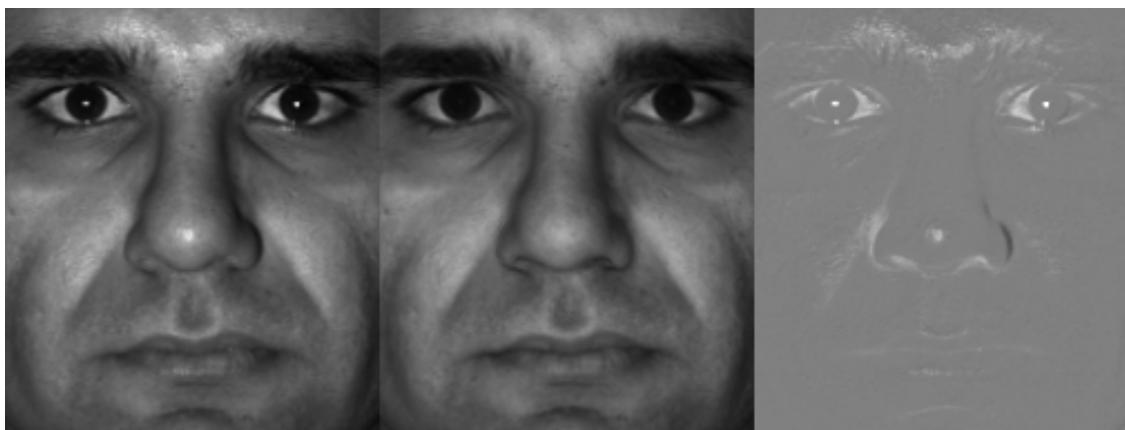
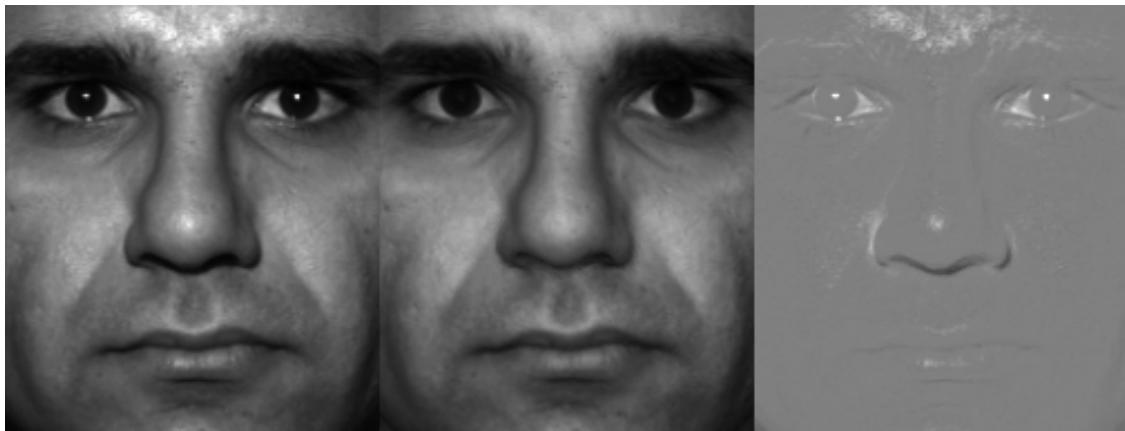
```
249.960151 seconds (51.97 k allocations: 117.332 GiB, 54.80% gc time)
```

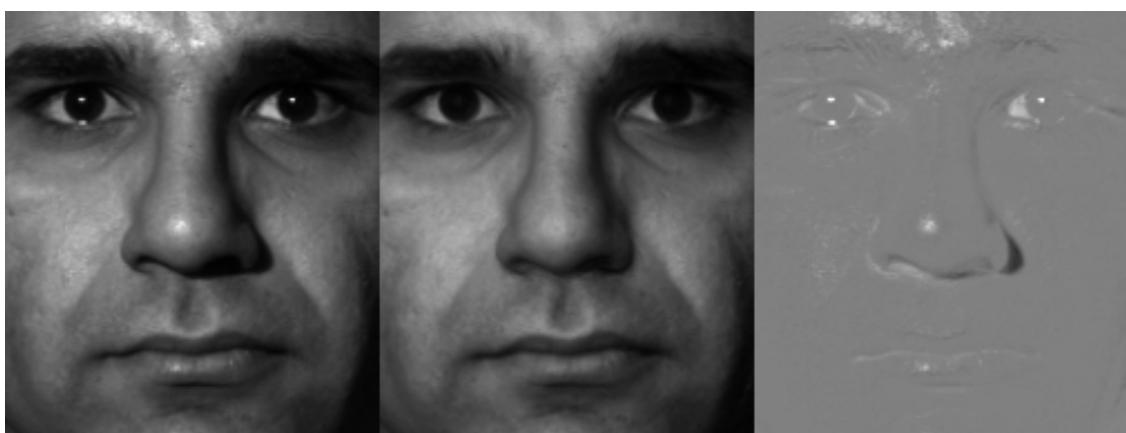
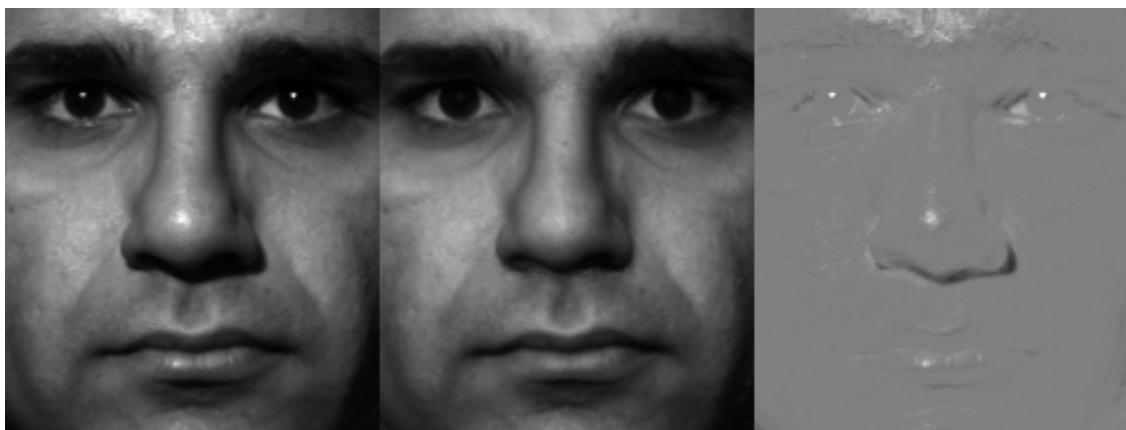
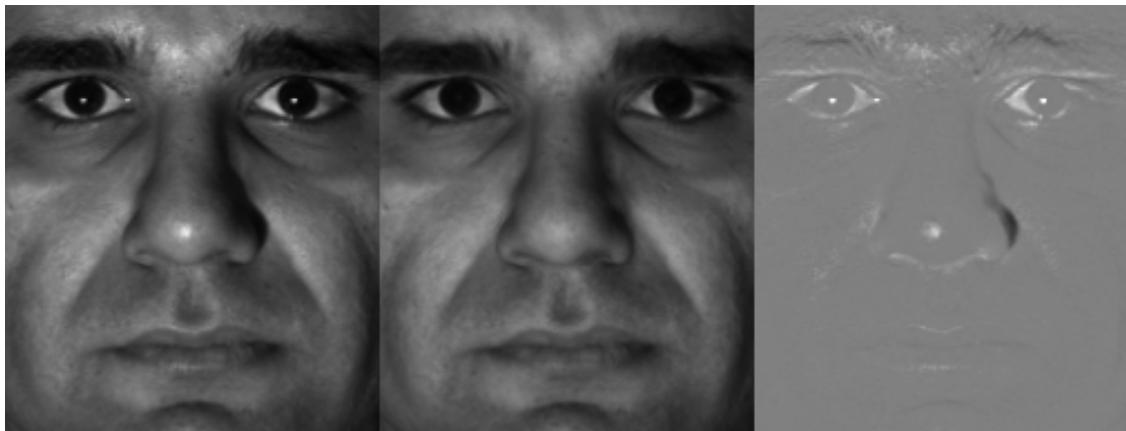
```
Out[36]: (564, 27, 363.97785343542574, 2.09511459940173e-5)
```

```
In [37]: for i=1:n-9
    Li=reshape(L[:,i],mi,ni)
    Si=reshape(S[:,i],mi,ni)+0.5
    display(hcat(img[i],colorview(Gray,Li),colorview(Gray,Si)))
end
```









#### 1.5.4 Example - Mona Lisa's smile

In [38]: `img=load("files/mona-lisa_1.jpg")`

Out[38] :



In [39]: `# Prema bilježnici S03`  
`imgsep=float(channelview(img))`

Out[39]: `3×456×620 Array{Float32,3}:`

```
[ :, :, 1 ] =  
0.396078 0.505882 0.603922 0.596078 ... 0.203922 0.290196 0.258824  
0.419608 0.529412 0.627451 0.619608 0.227451 0.313726 0.286275  
0.239216 0.34902 0.447059 0.439216 0.0862745 0.164706 0.12549  
  
[ :, :, 2 ] =  
0.411765 0.482353 0.54902 0.537255 ... 0.247059 0.329412 0.309804  
0.435294 0.505882 0.572549 0.560784 0.270588 0.356863 0.337255  
0.254902 0.32549 0.392157 0.380392 0.129412 0.196078 0.176471
```

```

[:, :, 3] =
0.498039 0.509804 0.517647 0.490196 ... 0.282353 0.337255 0.32549
0.521569 0.533333 0.529412 0.501961 ... 0.305882 0.364706 0.352941
0.341176 0.352941 0.352941 0.32549 ... 0.156863 0.203922 0.188235

...
[:, :, 618] =
0.811765 0.764706 0.788235 0.870588 ... 0.235294 0.290196 0.333333
0.764706 0.717647 0.741176 0.823529 ... 0.219608 0.270588 0.305882
0.560784 0.513726 0.537255 0.619608 ... 0.105882 0.156863 0.196078

[:, :, 619] =
0.745098 0.807843 0.882353 0.905882 ... 0.235294 0.278431 0.372549
0.698039 0.760784 0.835294 0.858824 ... 0.215686 0.25098 0.345098
0.494118 0.556863 0.631373 0.654902 ... 0.101961 0.141176 0.235294

[:, :, 620] =
0.733333 0.85098 0.929412 0.87451 ... 0.266667 0.321569 0.470588
0.686275 0.803922 0.882353 0.827451 ... 0.25098 0.301961 0.443137
0.482353 0.6 0.678431 0.623529 ... 0.137255 0.188235 0.333333

```

In [41]: `size(imgsep)`

Out[41]: (3, 456, 620)

```

In [ ]: # 7 -17 minutes
        # Red
@time RL,RS,Riter=PCPAD(imgsep[1,:,:])
        # Green
GL,GS,Giter=PCPAD(imgsep[2,:,:])
        # Blue
BL,BS,Biter=PCPAD(imgsep[1,:,:])

```

In [46]: `Giter, rank(GL), vecnorm(imgsep[2,:,:]), vecnorm(imgsep[2,:,:]-GL-GS)`

Out[46]: (1000, 243, 242.53912f0, 0.00013260565f0)

```

In [44]: # Mona Lisa's low-rank component
colorview(RGB,RL,GL,BL)

```

Out[44]:



```
In [45]: # Mona Lisa's sparse component  
colorview(RGB,RS+0.5,GS+0.5,BS+0.5)
```

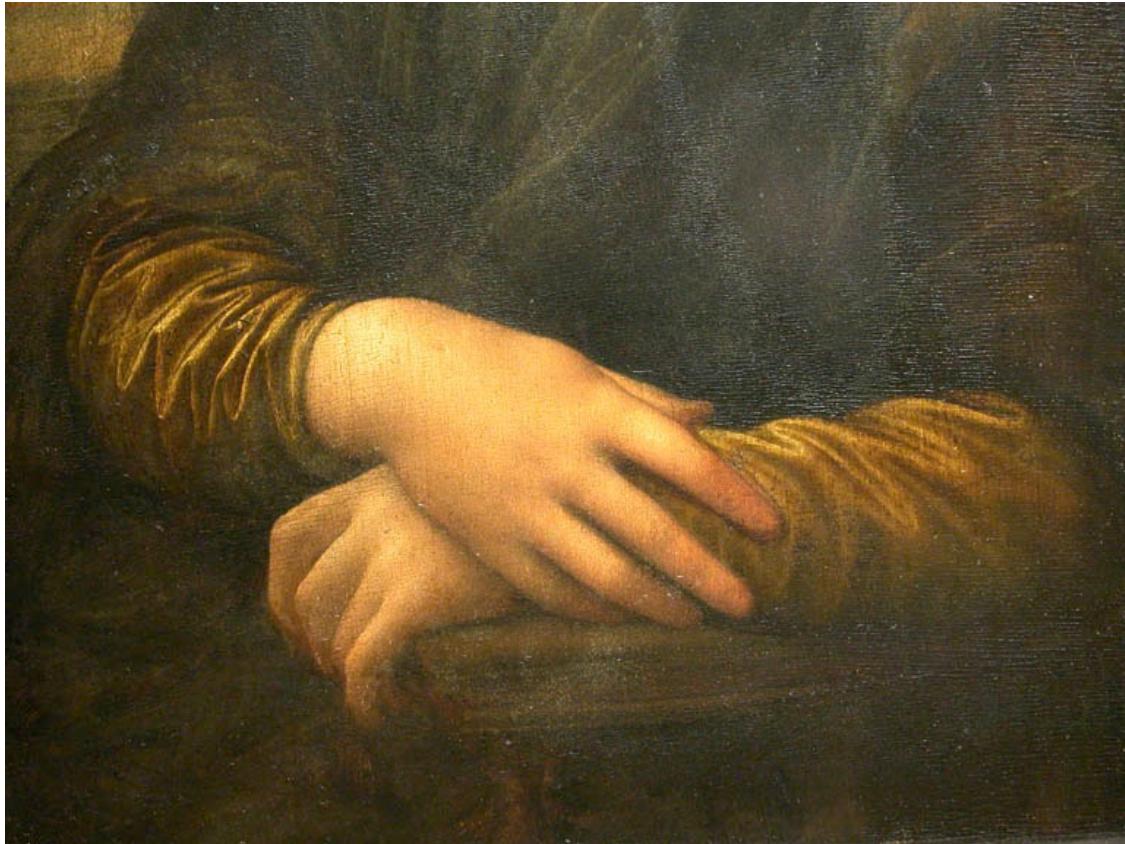
Out[45] :



### 1.5.5 Example - Mona Lisa's hands

```
In [47]: img=load("files/Mona_Lisa_detail_hands.jpg")
```

```
Out[47]:
```



```
In [48]: imgsep=float(channelview(img))
@time RL,RS,Riter=PCPAD(imgsep[1,:,:])
GL,GS,Giter=PCPAD(imgsep[2,:,:])
BL,BS,Biter=PCPAD(imgsep[1,:,:])
Giter, rank(GL), vecnorm(imgsep[2,:,:]),
vecnorm(imgsep[2,:,:]-GL-GS)
```

```
235.292231 seconds (103.02 k allocations: 43.214 GiB, 6.48% gc time)
```

```
Out[48]: (1000, 307, 215.53496f0, 0.000116073774f0)
```

```
In [49]: colorview(RGB,RL,GL,BL)
```

```
Out[49]:
```



In [50]: `colorview(RGB, RS+0.5, GS+0.5, BS+0.5)`

Out[50]:



In [ ]: