Inverse Problems: Problem Set №2

Group C: Mateusz Brodowicz and Anton Myshak

In [1]:
```
import scipy.io
import numpy as np
import matplotlib.pyplot as plt
import core as cr
import xmpl


from prbsets import deconv2D as conv
from importlib import reload #to reload libs online, like my_funcs = relo
from matplotlib.image import imread
```

Task 1

In [2]:
```
plt.figure(figsize=(12, 4))
plt.imshow(imread('./assignment/tasks/ex1.png'))
plt.axis('off');
```

1. Suppose we form and store the full matrix **K** (similar to the one-dimensional example) for a two dimensional source **X** of size 128×128, 256×256, and 512×512. How much memory would this require, assuming that we store **K** in double precision (i.e., numeric values will occupy 64 bit = 8 byte in computer memory)?

In [3]:
```
#solution
plt.figure(figsize=(12, 8))
plt.imshow(imread('./assignment/solutions/Q1.PNG'))
plt.axis('off');
```

Given that $X \in M_{n\times n}((R)$, we have

$$vec(Y) = Kvec(X)$$

$$y = (K_1 \otimes K_2)x$$

where $x \in R^{nn}$ and $K = (K_1 \otimes K_2) \in M_{nn\times nn}(R)$. Denote the memory needed as D and consider the following values of n.

- if n = 128,

$$D = 128^2 \times 128^2 \times 8B = 2147483648B \approx 2.14GB$$

- if n = 256,

$$D = 256^2 \times 256^2 \times 8B = 34359738368 \approx 34.36GB$$

- if n = 512,

$$D = 512^2 \times 512^2 \times 8B = 549755813888 \approx 549.76GB$$

Task 2(a)

In [4]:
```python
plt.figure(figsize=(12, 4))
plt.imshow(imread('./assignment/tasks/ex2a_extra.png'))
plt.axis('off');

plt.figure(figsize=(15, 6))
plt.imshow(imread('./assignment/tasks/ex2a.png'))
plt.axis('off');
```

We can represent $\mathbf{K}$ based on the SVD of $\mathbf{K}_1$ and $\mathbf{K}_2$. In particular, if $\mathbf{K}_1 = \mathbf{U}_1\mathbf{S}_1\mathbf{V}_1$ and $\mathbf{K}_2 = \mathbf{U}_2\mathbf{S}_2\mathbf{V}_2$, then

$$\mathbf{K} = (\mathbf{U}_2 \otimes \mathbf{U}_1)(\mathbf{S}_2 \otimes \mathbf{S}_1)(\mathbf{V}_2^\mathsf{T} \otimes \mathbf{V}_1^\mathsf{T}). \tag{5}$$

(Notice that the diagonal entries of $\mathbf{S}_2 \otimes \mathbf{S}_1$ are no longer in descending order.) Consequently, the least squares estimator can be expressed as

$$\mathbf{x}_{\mathrm{LS}} = \mathbf{K}^\dagger \mathbf{y} = (\mathbf{V}_2 \otimes \mathbf{V}_1)(\mathbf{S}_2^\dagger \otimes \mathbf{S}_1^\dagger)(\mathbf{U}_2^\mathsf{T} \otimes \mathbf{U}_1^\mathsf{T})\mathbf{y}. \tag{6}$$

where $\mathbf{A}^\dagger$ denotes the pseudoinverse (or generalized inverse) of $\mathbf{A}$. Equivalently, we have

2. Let $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{B} \in \mathbb{R}^{r,s}$. Then, the Kronecker product is given by

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2n}\mathbf{B} \\ \vdots & \vdots & & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{mr,ns}.$$

a) Use the identities $(\mathbf{A} \otimes \mathbf{B})^\mathsf{T} = \mathbf{A}^\mathsf{T} \otimes \mathbf{B}^\mathsf{T}$, $(\mathbf{A} \otimes \mathbf{B})^\dagger = \mathbf{A}^\dagger \otimes \mathbf{B}^\dagger$ and $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC} \otimes \mathbf{BD})$, where $\mathbf{C} \in \mathbb{R}^{n,p}$ and $\mathbf{D} \in \mathbb{R}^{s,t}$, to proof (5) and (6), respectively.

In [5]:
```python
#solution
plt.figure(figsize=(12, 8))
plt.imshow(imread('./assignment/solutions/Q2a.PNG'))
plt.axis('off');
```

2.a

Given $(A \otimes B)^T = A^T \otimes B^T, (A \otimes B)^\dagger = A^\dagger \otimes B^\dagger, (AB \otimes CD) = (A \otimes B)(C \otimes D)$, we have,

$$\begin{aligned}(U_2 \otimes U_1)(S_2 \otimes S_1)(V_2^T \otimes V_1^T) &= (U_2 S_2 \otimes U_1 S_1)(V_2^T \otimes V_1^T) \\ &= (U_2 S_2 V_2^T \otimes U_1 S_1 V_1^T) \\ &= (U_2 S_2 V_2^T \otimes U_1 S_1 V_1^T) \\ &= K_2 \otimes K_1 = K\end{aligned}$$

Similarly, as $U_2 V_2^T = I \implies (U_2 S_2 V_2^T)^\dagger = V_2 S_2^\dagger U_2^T$

$$\begin{aligned}(V_2 \otimes V_1)(S_2^\dagger \otimes S_1^\dagger)(U_2^T \otimes U_1^T)y &= (V_2 S_2^\dagger \otimes V_1 S_1^\dagger)(U_2^T \otimes U_1^T)y \\ &= (V_2 S_2^\dagger U_2^T \otimes V_1 S_1^\dagger U_1^T)y \\ &= (V_2 S_2^\dagger U_2^T \otimes V_1 S_1^\dagger U_1^T)y \\ &= ((U_2 S_2 V_2^T)^\dagger \otimes (U_1 S_1 V_1^T)^\dagger)y \\ &= (K_2^\dagger \otimes K_1^\dagger)y \\ &= K^\dagger y = \mathbf{x}_{LS}\end{aligned}$$

Task 2(b)

In [6]:
```python
plt.figure(figsize=(12, 4))
plt.imshow(imread('./assignment/tasks/ex2b_extra.png'))
plt.axis('off');

plt.figure(figsize=(12, 5))
plt.imshow(imread('./assignment/tasks/ex2b.png'))
plt.axis('off');
```

(Notice that the diagonal entries of $\mathbf{S}_2 \otimes \mathbf{S}_1$ are no longer in descending order.) Consequently, the least squares estimator can be expressed as

$$\mathbf{x}_{\mathrm{LS}} = \mathbf{K}^\dagger \mathbf{y} = (\mathbf{V}_2 \otimes \mathbf{V}_1)(\mathbf{S}_2^\dagger \otimes \mathbf{S}_1^\dagger)(\mathbf{U}_2^\mathsf{T} \otimes \mathbf{U}_1^\mathsf{T})\mathbf{y}. \tag{6}$$

where $\mathbf{A}^\dagger$ denotes the pseudoinverse (or generalized inverse) of $\mathbf{A}$. Equivalently, we have

$$\mathbf{X}_{\mathrm{LS}} = \mathbf{K}_1^\dagger \mathbf{Y} (\mathbf{K}_2^\mathsf{T})^\dagger = \mathbf{V}_1(\mathbf{S}_1^\dagger(\mathbf{U}_1^\mathsf{T}\mathbf{Y}\mathbf{U}_2)(\mathbf{S}_2^\dagger)^\mathsf{T})\mathbf{V}_2^\mathsf{T}. \tag{7}$$

b) Let $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{B} \in \mathbb{R}^{r,s}$, $\mathbf{C} \in \mathbb{R}^{s,n}$. Use the identity $\mathrm{vec}(\mathbf{BCA}^\mathsf{T}) = \mathbf{A} \otimes \mathbf{B}\,\mathrm{vec}(\mathbf{C})$ to show that (6) and (7) are equivalent, i.e.,

$$\mathrm{vec}(\mathbf{V}_1(\mathbf{S}_1^\dagger(\mathbf{U}_1^\mathsf{T}\mathbf{Y}\mathbf{U}_2)(\mathbf{S}_2^\dagger)^\mathsf{T})\mathbf{V}_2^\mathsf{T}) = (\mathbf{V}_2 \otimes \mathbf{V}_1)(\mathbf{S}_2^\dagger \otimes \mathbf{S}_1^\dagger)(\mathbf{U}_2^\mathsf{T} \otimes \mathbf{U}_1^\mathsf{T})\mathbf{y}.$$

(Notice that this identity is a generalization of $\mathbf{Kx} = \mathrm{vec}(\mathbf{K}_1\mathbf{X}\mathbf{K}_2^\mathsf{T}) = (\mathbf{K}_2 \otimes \mathbf{K}_1)\mathbf{x}$.)

In [7]:
```python
#solution
plt.figure(figsize=(10, 8))
plt.imshow(imread('./assignment/solutions/Q2b.PNG'))
plt.axis('off');
```

2.b

Starting at 7), we have

$$
\begin{aligned}
\mathbf{X}_{LS} = V_1(S_1^\dagger(U_1^T Y U_2)((S_2^\dagger)^T V_2^T) &= (V_1 S_1^\dagger U_1^T)Y(U_2(S_2^\dagger)^T V_2^T)\\
&= (V_1 S_1^\dagger U_1^T)Y(U_2(S_2^\dagger)^T V_2^T)\\
&= (V_1 S_1^\dagger U_1^T)Y(V_2 S_2^\dagger U_2^T)^T\\
&= (V_2 S_2^\dagger U_2^T) \otimes (V_1 S_1^\dagger U_1^T) vec(Y)\\
&= (K_2^\dagger \otimes K_1^\dagger)y\\
&= K^\dagger y
\end{aligned}
$$

The definition of 6) tells us that $\mathbf{x}_{LS} = K^\dagger y = (V_2 \otimes V_1)(S_2^\dagger \otimes S_1^\dagger)(U_2^T \otimes U_1^T)y$, thus yielding the required equality

$$V_1(S_1^\dagger(U_1^T Y U_2)((S_2^\dagger)^T V_2^T) = (V_2 \otimes V_1)(S_2^\dagger \otimes S_1^\dagger)(U_2^T \otimes U_1^T)y$$

Task 2(c)

In [8]:
```python
plt.figure(figsize=(12, 4))
plt.imshow(imread('./assignment/tasks/ex2c_extra.png'))
plt.axis('off');

plt.figure(figsize=(12, 5))
plt.imshow(imread('./assignment/tasks/ex2c.png'))
plt.axis('off');
```

Moreover, it can be shown that we can represent the matrix vector product $\mathbf{Ax}$ as

$$\mathbf{Ax} = \mathrm{vec}(\mathbf{U}_1((\mathbf{s}_1\mathbf{s}_2^\top) \odot (\mathbf{V}_1^\top \mathbf{X} \mathbf{V}_2))\mathbf{U}_2^\top), \tag{8}$$

with $\mathbf{S}_1 = \mathrm{diag}(\mathbf{s}_1)$ and with $\mathbf{S}_2 = \mathrm{diag}(\mathbf{s}_2)$, respectively; $\odot$ denotes the Hadamard product (i.e., an entrywise matrix-matrix product). Using similar arguments, it can be shown that we can represent the Tikhonov

c) Let $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{B} \in \mathbb{R}^{r,s}$, $\mathbf{C} \in \mathbb{R}^{s,n}$, $\mathbf{a} \in \mathbb{R}^m$ and $\mathbf{b} \in \mathbb{R}^n$. Use the identities $\mathrm{vec}(\mathbf{BCA}^\top) = \mathbf{A} \otimes \mathbf{B}\,\mathrm{vec}(\mathbf{C})$ and $\mathrm{diag}(\mathbf{a}) \otimes \mathrm{diag}(\mathbf{b}) = \mathrm{diag}(\mathrm{vec}(\mathbf{ba}^\top))$ to show that (8) holds.

In [9]:
```python
#solution
plt.figure(figsize=(10, 8))
plt.imshow(imread('./assignment/solutions/Q2c.PNG'))
plt.axis('off');
```

2.c

We assume that $\mathrm{vec}(BCA^T) = A \otimes B\, \mathrm{vec}(C)$ and $\mathrm{diag}(a) \otimes \mathrm{diag}(b) = \mathrm{diag}(\mathrm{vec}(ba^T))$.
Let $(s_1 s_2^T) \odot (V_1 X V_2^T) = Z$.

$$
\begin{aligned}
\mathrm{vec}(U_1 Z U_2^T) &= \mathrm{vec}((U_2 \otimes U_1)\,\mathrm{vec}(Z)) \\
&= \mathrm{vec}((U_2 \otimes U_1)\,\mathrm{vec}((s_1 s_2^T) \odot (V_1^T X V_2))) \\
&= \mathrm{vec}((U_2 \otimes U_1)\,\mathrm{vec}((s_1 s_2^T) \odot (V_1 X^T V_2^T)^T)) \\
&= \mathrm{vec}((U_2 \otimes U_1)\,\mathrm{vec}((s_1 s_2^T) \odot (V_2 \otimes V_1)^T\,\mathrm{vec}(X^T)) \\
&= \mathrm{vec}((U_2 \otimes U_1)\,\mathrm{vec}((s_1 s_2^T) \odot (V_2 \otimes V_1)^T x)) \\
&= \mathrm{vec}((U_2 \otimes U_1)(S_2 \otimes S_1)(V_2 \otimes V_1)^T x) \\
&= (U_2 \otimes U_1)(S_2 \otimes S_1)(V_2 \otimes V_1)^T x \\
&= (U_2 S_2 V_2^T \otimes U_1 S_1 V_1^T) x \\
&= Ax
\end{aligned}
$$

Task 2(d)

In [10]:

```
plt.figure(figsize=(12, 4))
plt.imshow(imread('./assignment/tasks/ex2d_extra.png'))
plt.axis('off');

plt.figure(figsize=(12, 5))
plt.imshow(imread('./assignment/tasks/ex2d.png'))
plt.axis('off');
```

matrix-matrix product). Using similar arguments, it can be shown that we can represent the Tikhonov solution using an SVD of $\mathbf{K}_1$ and $\mathbf{K}_2$ as

$$
\mathbf{X}_\alpha = \mathbf{V}_1 \left(\left((\mathbf{s}_1 \mathbf{s}_2^T) \oslash \left((\mathbf{s}_1 \mathbf{s}_2^T)^{\circ 2} + \alpha \mathbf{E} \right) \right) \odot (\mathbf{U}_1^T \mathbf{Y} \mathbf{U}_2) \right) \mathbf{V}_2^T, \tag{9}
$$

where $\circ 2$ denotes the elementwise square (Hadamard power), i.e., $(\mathbf{s}_1 \mathbf{s}_2^T)^{\circ 2} = (\mathbf{s}_1 \mathbf{s}_2^T) \odot (\mathbf{s}_1 \mathbf{s}_2^T)$, $\oslash$ is the Hadamard division (i.e., elementwise matrix-matrix division), and $\mathbf{E}$ is an $n \times n$ matrix of all ones.

d) Proof that the Tikhonov regularized solution can be expressed as (9).

In [22]:

```
#solution
plt.figure(figsize=(10, 10))
plt.imshow(imread('./assignment/solutions/Q2d.PNG'))
plt.axis('off');
```

2.d

Let $X = USV^T$ be the SVD decomposition of the matrix X. To get the Tikhonov regularized solution we need to minimize the objective function

$$L(W) = ||Y - XW||_F^2 + \alpha||W||_F^2$$

We can obtain the grad of L and set it to zero

$$\text{grad}\, L(W) = \frac{d}{dW} L(W)||Y - XW||_F^2 + \alpha||W||_F^2 = 0$$

Which given the F norm, can be expressed as

$$\text{grad}\, L(W) = -2X^T(Y - XW) + 2\alpha(W) = 0$$

$$-X^TY - X^TXW + 2\alpha(W) = 0$$

$$(X^TX + \alpha I)W = X^TY$$

$$((USV^T)^T(USV^T) + \alpha I)W = (USV^T)Y$$

$$(VSU^TUSV^T + \alpha I)W = (USV^T)Y$$

Since U and V are orthogonal we get

$$(S^2 + \alpha I)W = (USV^T)Y$$

$$U^T(S^2 + \alpha I)W = U^T(USV^T)Y$$

$$U^T(S^2 + \alpha I)W = SV^TY$$

Let $U_1S_2V_2^T = Y$ be the SVD decomposition of Y, then

$$U^T(S^2 + \alpha I)W = SV^TU_1S_2V_2^T$$

Now by substituting (9) for W it can be seen that the formula above gives Tikhonov regularization scheme.

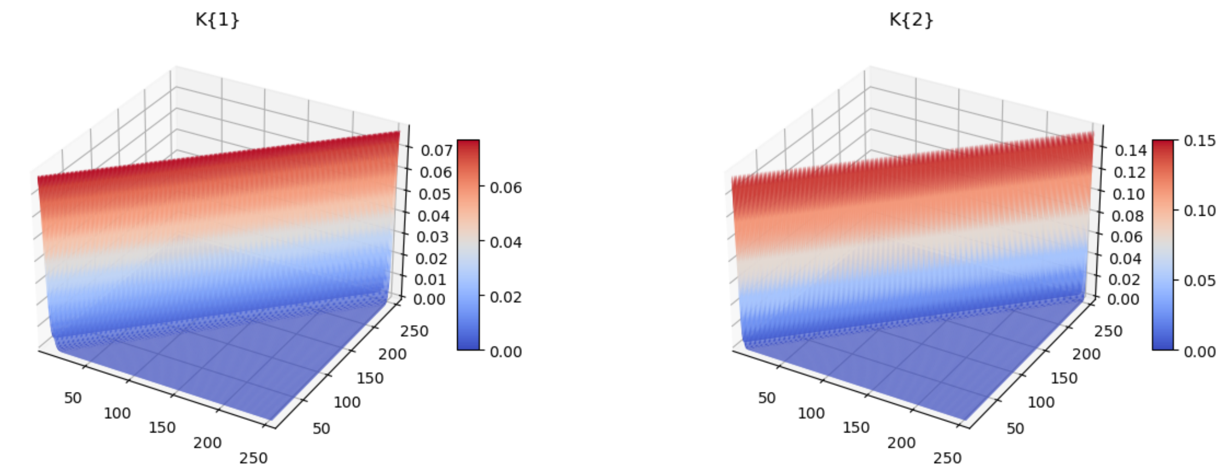
Task 3(a)

In [12]:
```
plt.figure(figsize=(12, 5))
plt.imshow(imread('./assignment/tasks/ex3a.png'))
plt.axis('off');
```

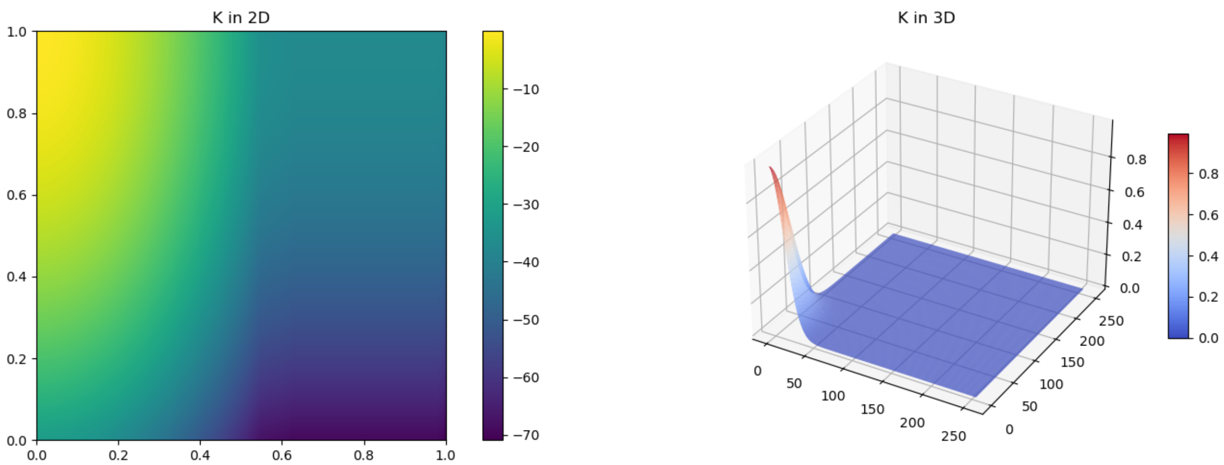
3. Next, we consider computing a solution to the two-dimensional problem of the form (1) using direct methods. We will exploit the fact that $\mathbf{K}$ is separable.

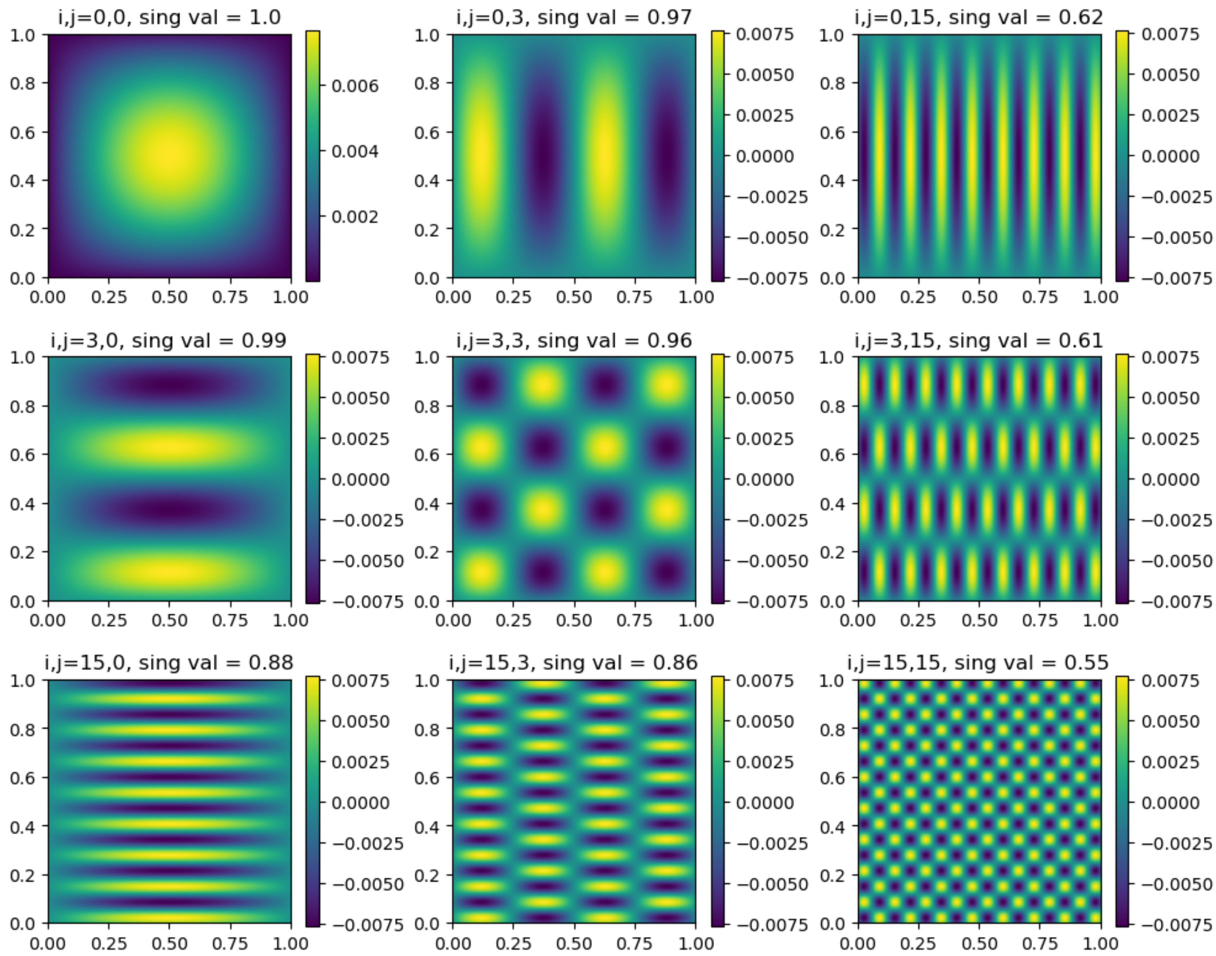
   a) According to (5) we can represent the SVD of $\mathbf{K}$ in terms of the SVDs of $\mathbf{K}_1$ and $\mathbf{K}_2$, respectively. It can be shown that the right-singular vectors of $\mathbf{K}_2 \otimes \mathbf{K}_1$ (i.e., the columns of $\mathbf{V}_2 \otimes \mathbf{V}_1$) can be represented as $\text{vec}(\mathbf{v}_{1,i}\mathbf{v}_{2,j}^T)$, where $\mathbf{v}_{1,i}$ and $\mathbf{v}_{2,j}$ are the ith and jth column of $\mathbf{V}_1$ and $\mathbf{V}_2$, respectively. Visualize the outer product $\mathbf{s}_1\mathbf{s}_2^T$ in logarithmic scale using Matlab's `imagesc` command. Moreover, visualize the right singular vectors $\mathbf{v}_{1,i}\mathbf{v}_{2,j}^T$ for all possible pairs of $i, j = 1, 4, 16$ using Matlab's `imagesc` command. What do you notice for increasing i, j about the sungalar value/vector pairs $\{(\mathbf{s}_1\mathbf{s}_2^T)_{ij}, \mathbf{v}_{1,i}\mathbf{v}_{2,j}^T\}$? **Hint:** *A template for your implementation is prbsets/deconv2D/scDeconvSVD2D.m.*

In [13]:
```python
reload(conv)
reload(cr)
K = conv.getKernel2D(256, dbg=True)
```



In [14]:
```python
reload(conv)
# reload(cr)
conv.scDeconvSVD2D()
```





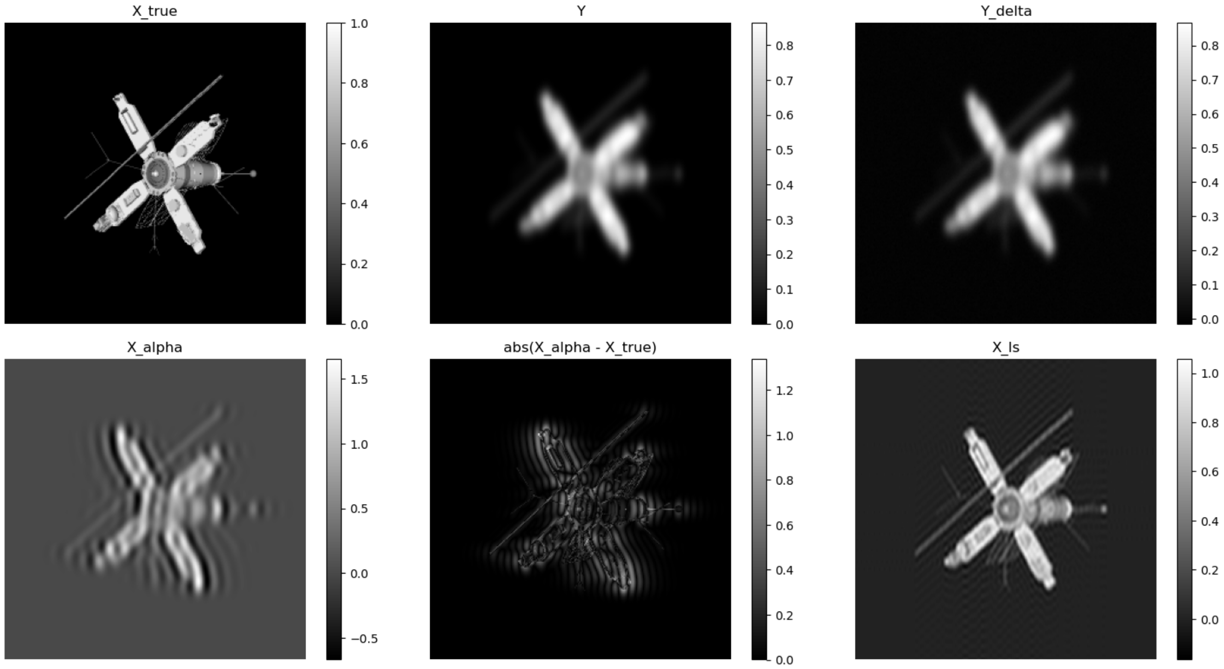
With increasing i,j the vector pair $v_{1,i} v_{2,j}^T$ provides corresponding number (i*j) of alternating areas and the singular value $(s_1s_2^T)_{ij}$ decreases

Task 3(b)

In [15]:
```
plt.figure(figsize=(12, 5))
plt.imshow(imread('./assignment/tasks/ex3b.png'))
plt.axis('off');
```

b) Solve the inverse problem using a direct method. In particular, compute the Tikhonov solution $\mathbf{X}_\alpha$ based on (9). Compare your solution to the least squares solution $\mathbf{X}_{ls}$ in (7). **Hint:** *For the least squares solution, use $\mathbf{K}_1^\dagger \mathbf{Y} (\mathbf{K}_2^T)^\dagger$. You can use Matlab's forward and backward slash operator to compute/apply the generalized inverses. A template for your implementation is prbsets/deconv2D/scDeconvTRegDirSVD2D.m.*

In [16]:
```
reload(conv)
reload(cr)
conv.scDeconvTRegDirSVD2D()
```



Task 4(a)

In [17]:
```
plt.figure(figsize=(12, 6))
plt.imshow(imread('./assignment/tasks/ex4a.png'))
plt.axis('off');
```

4. Next, we consider an iterative method to solve the optimality conditions

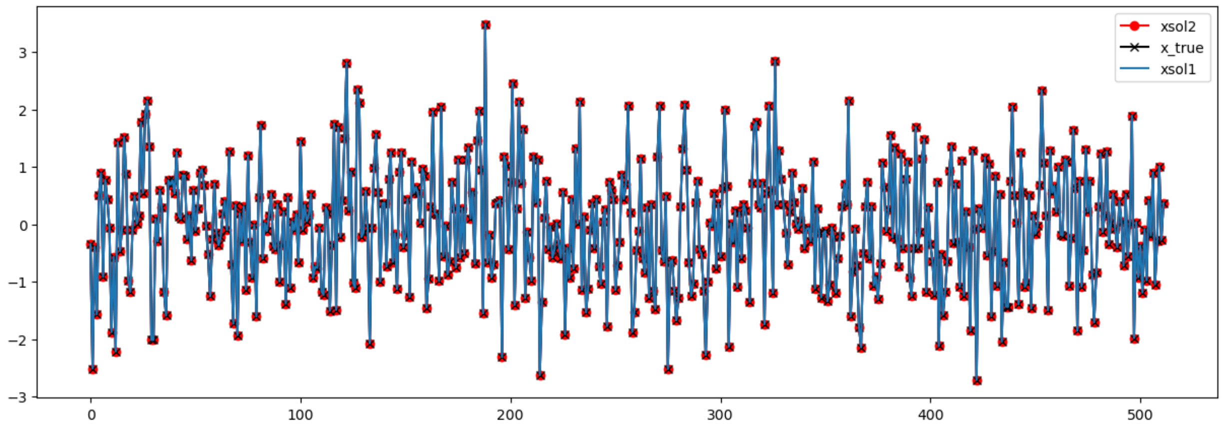
$$\mathbf{K}^{\mathsf{T}}(\mathbf{K}\mathbf{x}^{\star} - \mathbf{y}^{\delta}) + \alpha\mathbf{x}^{\star} = \mathbf{0} \tag{10}$$

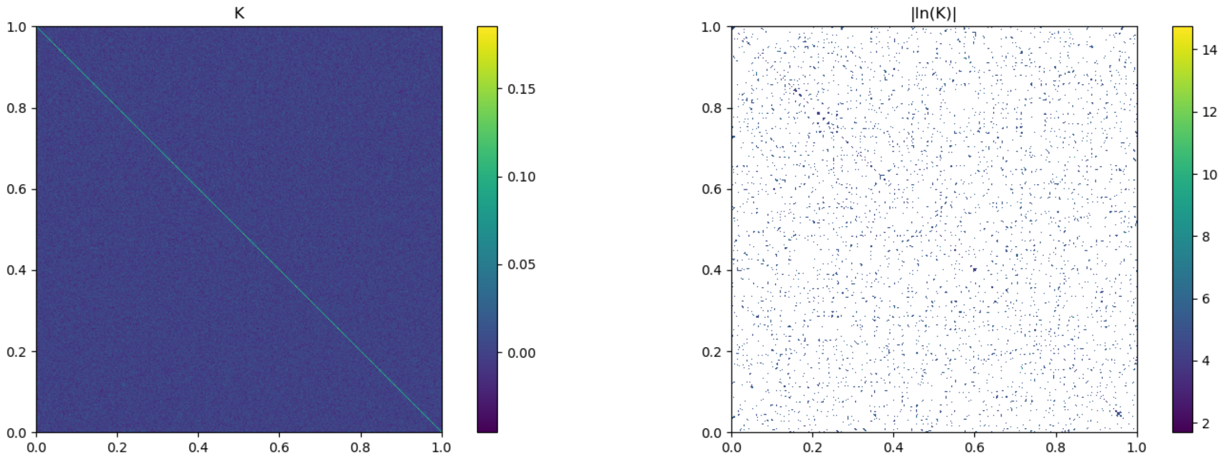
of the Tikhonov-regularized problem (as opposed to a direct method). This allows us to avoid explicitly forming and/or storing the matrix operator $\mathbf{K} \in \mathbb{R}^{n,n}$. In particular, we will consider a matrix-free (preconditioned) conjugate gradient (**CG**) method to solve the linear system (10) for $\mathbf{x}^{\star}$. This Krylov subspace method only requires an expression for the action of a matrix on a vector (i.e., an expression for the matrix-vector-product ("matvec")). In exact arithmetic it is guaranteed that the CG converges to a solution after at most r iterations, where r is the number of distinct eigenvalues of the matrix of the linear system.

a) Implement a CG algorithm. **Hint:** *A template for implementing the CG algorithm is core/runCG.m. A script to test your CG code is xmpl/exSolLSCG.m.*

In [18]:
```
reload(conv)
reload(cr)
reload(xmpl)
xmpl.exSolLSCG() #gives error cuz function runCG should be done
```

```
condition number of K: 1000.0000000000081
CG residual [[9.20261605e-07]] at iteration 180 of 512
xsol1 relative error: 31.9949685605379
xsol2 relative error: 5.879843217997166e-06
/Users/Saizt/Documents/USA/UH/UH Courses/Inverse Problems (MATH 6397)/HWS
/hw2/xmpl.py:47: RuntimeWarning: invalid value encountered in log
  im1 = ax.imshow(abs(np.log(K)), extent=[0,1,0,1])
```





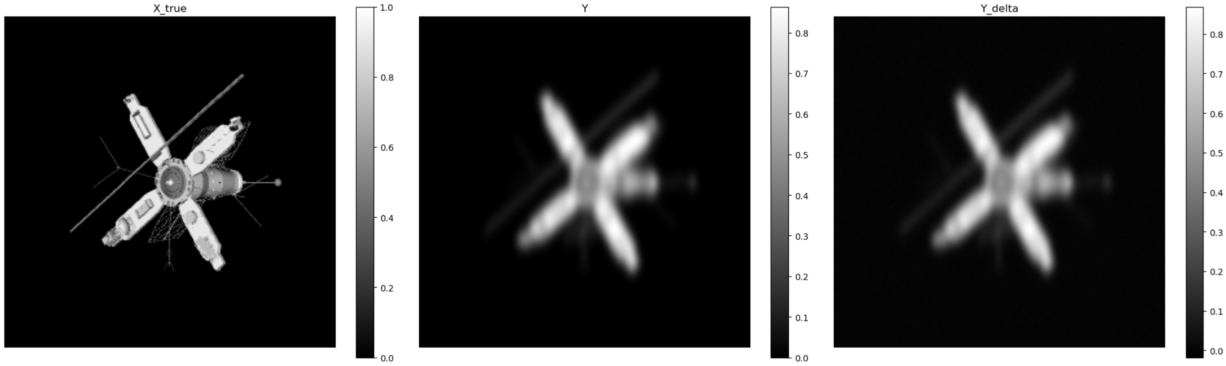
Task 4(b)

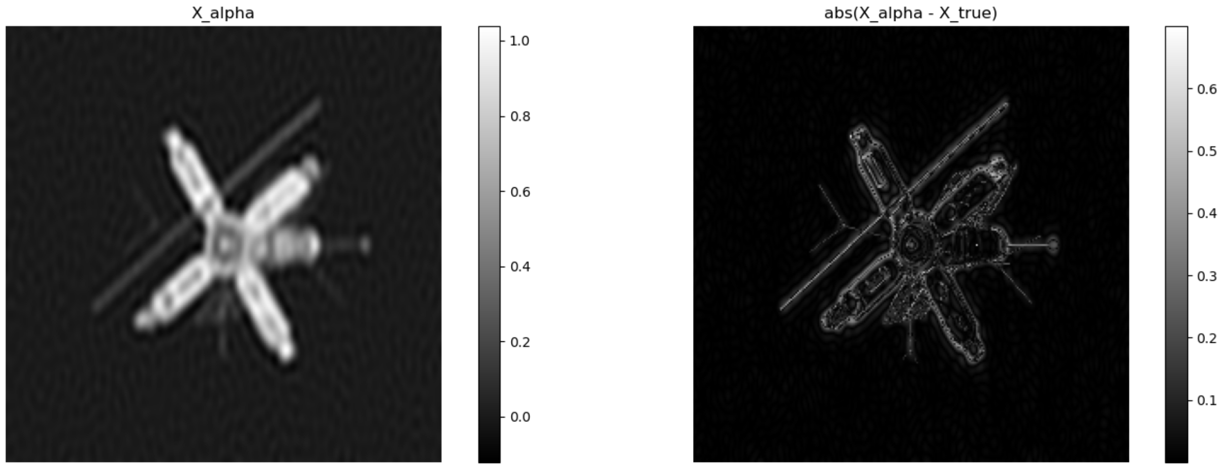
In [19]:
```
plt.figure(figsize=(12, 6))
plt.imshow(imread('./assignment/tasks/ex4b.png'))
plt.axis('off');
```

b) Use your CG algorithm to solve (10) for $\mathbf{x}^{\star}$. To evaluate this matrix vector product, we will apply one-dimensional blurring operators $\mathbf{K}_1$ and $\mathbf{K}_2$ along the individual coordinate directions. This results in significant savings in terms of memory-requirements. This expression for the matvec is given in (3). Notice that you need to apply $\mathbf{K}$ and $\mathbf{K}^{\mathsf{T}}$ (see (10)). **Hint:** *One complication is that CG expects the data in a column vector (lexicographical ordering; $nn \times 1$ column vector $\mathbf{x}$) whereas the matvec is defined for an $n \times n$ matrix $\mathbf{X}$. You can use Matlab's* `reshape` *to move between layouts. You can use Matlab's* `pcg` *implementation to set up everything before using your own implementation. A template for your implementation is prbsets/deconv2D/scDeconvTRegCGMF2D.m.*

In [20]:
```
reload(conv)
reload(cr)
conv.scDeconvTRegCGMF2D()
```

```
CG residual [[0.00090774]] at iteration 71 of 1000
```





In [ ]: