Inverse Problems: Problem Set №3

Group C: Mateusz Brodowicz and Anton Myshak

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

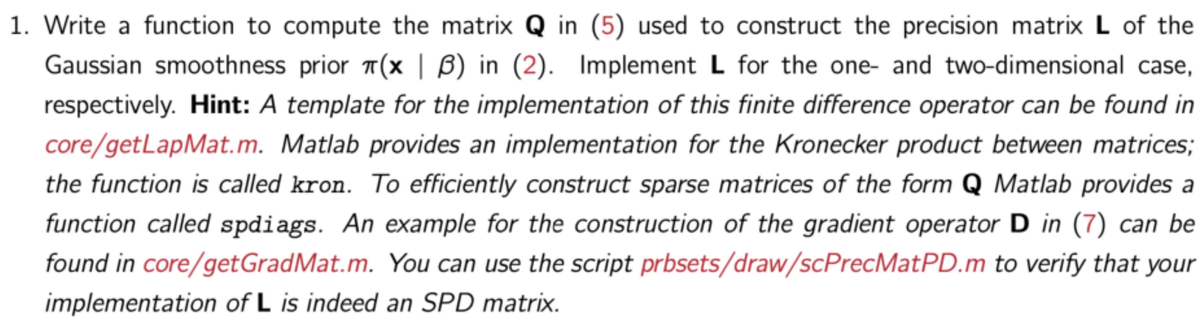
```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
%cd /content/drive/MyDrive/Colab Notebooks/MATH6397 Inverse Problems/hw3
!ls

import numpy as np
import matplotlib.pyplot as plt
import core as cr

from scipy.sparse import diags, kron, eye
from prbsets import sample1D as d1
from prbsets import draw
from importlib import reload #to reload libs online, like reload(cr)
from matplotlib.image import imread
```

```
/content/drive/MyDrive/Colab Notebooks/MATH6397 Inverse Problems/hw3
assignment  core.py  Inverse_HW3_Group_C.ipynb  prbsets  __pycache__
```

Task 1

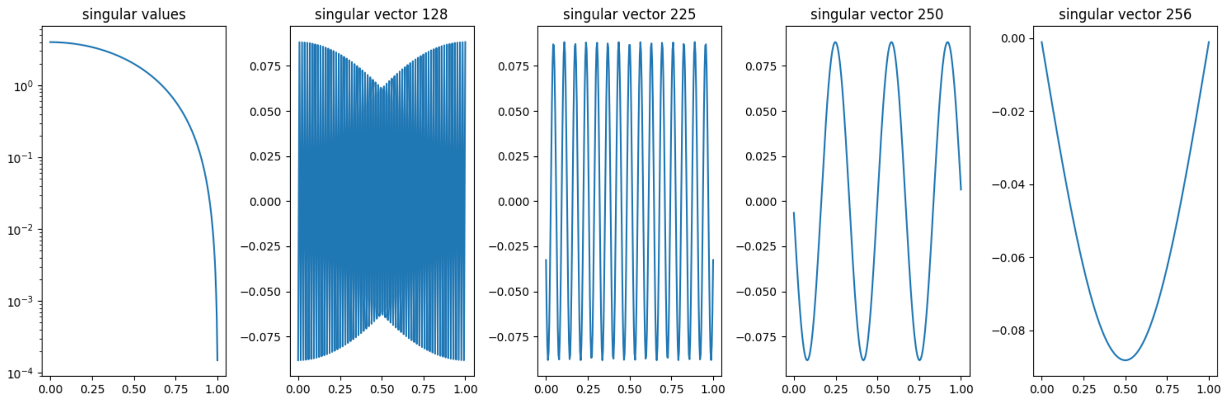
In [ ]:
```
plt.figure(figsize=(12, 4))
plt.imshow(imread('./assignment/tasks/ex1.png'))
plt.axis('off');
```



1. Write a function to compute the matrix $\mathbf{Q}$ in (5) used to construct the precision matrix $\mathbf{L}$ of the Gaussian smoothness prior $\pi(\mathbf{x} \mid \beta)$ in (2). Implement $\mathbf{L}$ for the one- and two-dimensional case, respectively. **Hint:** *A template for the implementation of this finite difference operator can be found in core/getLapMat.m. Matlab provides an implementation for the Kronecker product between matrices; the function is called* `kron`. *To efficiently construct sparse matrices of the form* $\mathbf{Q}$ *Matlab provides a function called* `spdiags`. *An example for the construction of the gradient operator* $\mathbf{D}$ *in (7) can be found in core/getGradMat.m. You can use the script prbsets/draw/scPrecMatPD.m to verify that your implementation of* $\mathbf{L}$ *is indeed an SPD matrix.*

In [ ]:
```
# reload(d1)
reload(cr)
reload(draw)
draw.scPrecMatPD()
```

```
smallest singular value 0.0001494266605350806
inner product [[225]] [78]: [[3.03576608e-17]]
inner product [78] [78]: [[1.]]
inner product [[225]] [48]: [[8.67361738e-18]]
inner product [48] [48]: [[1.]]
inner product [[225]] [62]: [[1.47451495e-17]]
inner product [62] [62]: [[1.]]
inner product [[225]] [33]: [[1.00613962e-16]]
inner product [33] [33]: [[1.]]
inner product [[225]] [138]: [[-1.20563282e-16]]
inner product [138] [138]: [[1.]]
inner product [[225]] [90]: [[6.31005664e-17]]
inner product [90] [90]: [[1.]]
inner product [[225]] [116]: [[-3.33934269e-17]]
inner product [116] [116]: [[1.]]
inner product [[225]] [121]: [[-2.42861287e-17]]
inner product [121] [121]: [[1.]]
inner product [[225]] [139]: [[-5.24753851e-17]]
inner product [139] [139]: [[1.]]
inner product [[225]] [169]: [[-6.9388939e-17]]
inner product [169] [169]: [[1.]]
error in symmetry: [[0.]]
```



Task 2

In [ ]:
```python
plt.figure(figsize=(12, 4))
plt.imshow(imread('./assignment/tasks/ex2.png'))
plt.axis('off');
```

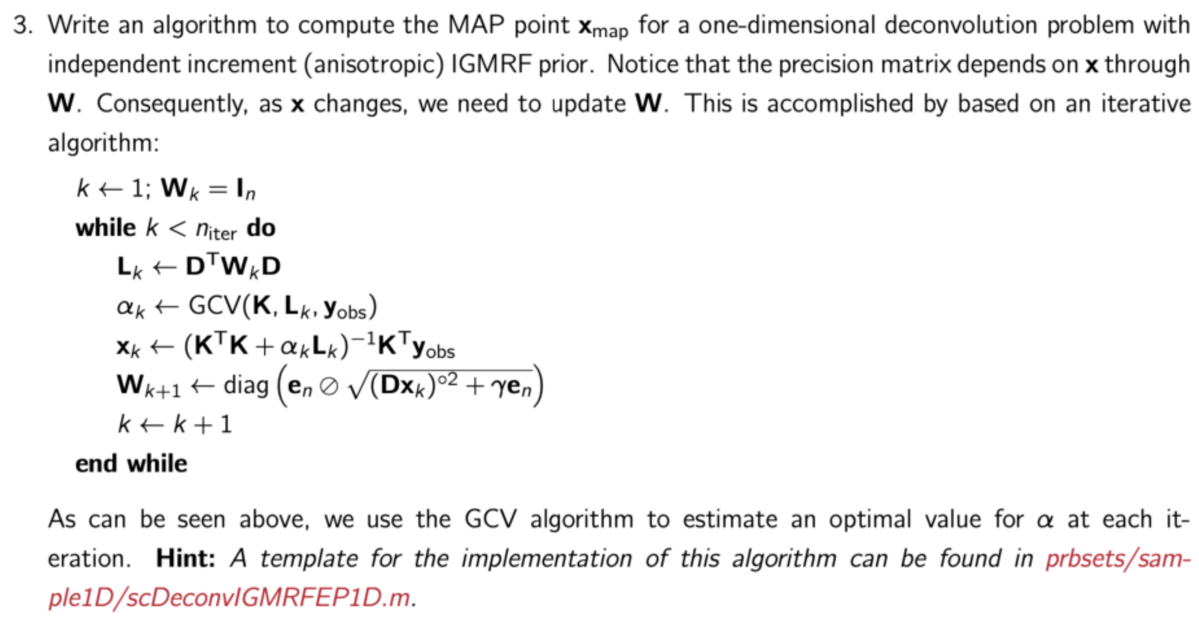
2. Implement the GCV algorithm described in §1.3 to estimate an optimal regularization parameter α. That is, implement a function that estimates α by solving the one-dimensional optimization problem (8). **Hint:** *We will use Matlab's* **`fminbnd`** *to solve* (8). *A template for the implementation of GCV is core/evalGCV.m. You can use prbsets/sample1D/scDeconvGCVLAP1D.m to test your implementation.*

In [ ]:
```python
#attention, random seed is not fixed!
reload(d1)
reload(cr)
reload(draw)
d1.scDeconvGCVLAP1D()
```

```
optimal regularization parameter alpha = 0.0023846743079548177
```

Task 3

In [ ]:
```python
plt.figure(figsize=(12, 8))
plt.imshow(imread('./assignment/tasks/ex3.png'))
plt.axis('off');
```



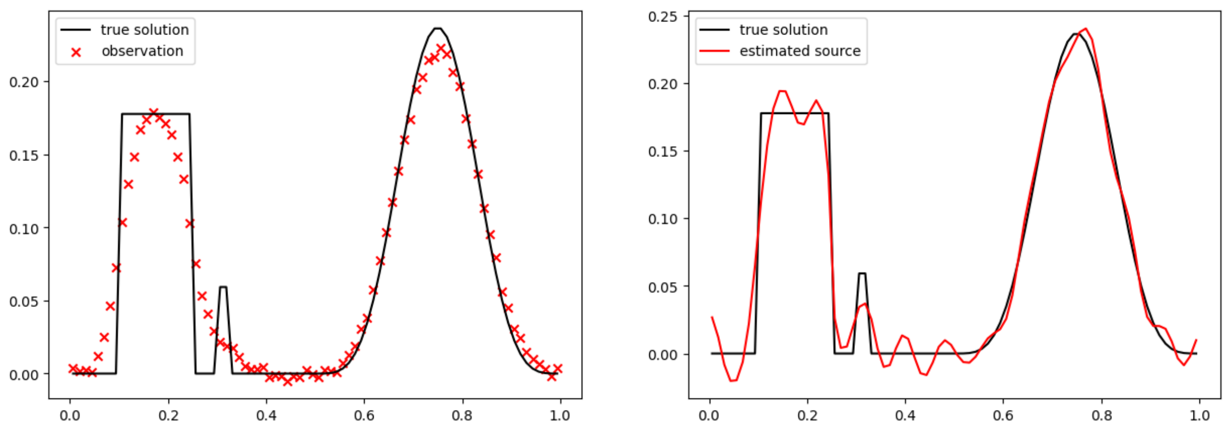
3. Write an algorithm to compute the MAP point $\mathbf{x}_{\text{map}}$ for a one-dimensional deconvolution problem with independent increment (anisotropic) IGMRF prior. Notice that the precision matrix depends on $\mathbf{x}$ through $\mathbf{W}$. Consequently, as $\mathbf{x}$ changes, we need to update $\mathbf{W}$. This is accomplished by based on an iterative algorithm:

```
$k \leftarrow 1$; $\mathbf{W}_k = \mathbf{I}_n$
**while** $k < n_{\text{iter}}$ **do**
    $\mathbf{L}_k \leftarrow \mathbf{D}^{\mathsf{T}}\mathbf{W}_k\mathbf{D}$
    $\alpha_k \leftarrow \text{GCV}(\mathbf{K}, \mathbf{L}_k, \mathbf{y}_{\text{obs}})$
    $\mathbf{x}_k \leftarrow (\mathbf{K}^{\mathsf{T}}\mathbf{K} + \alpha_k\mathbf{L}_k)^{-1}\mathbf{K}^{\mathsf{T}}\mathbf{y}_{\text{obs}}$
    $\mathbf{W}_{k+1} \leftarrow \text{diag}\left(\mathbf{e}_n \oslash \sqrt{(\mathbf{D}\mathbf{x}_k)^{\circ 2} + \gamma\mathbf{e}_n}\right)$
    $k \leftarrow k + 1$
**end while**
```

As can be seen above, we use the GCV algorithm to estimate an optimal value for α at each iteration. **Hint:** *A template for the implementation of this algorithm can be found in prbsets/sample1D/scDeconvIGMRFEP1D.m.*

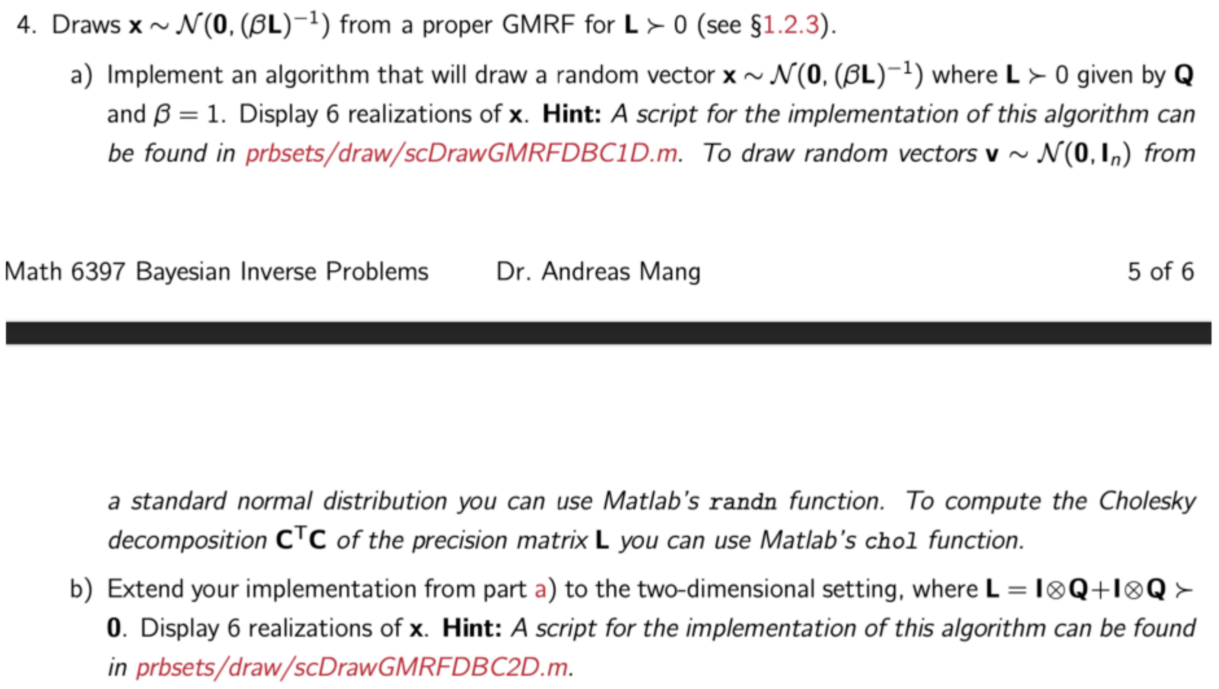
In [98]:
```python
#attention, random seed is not fixed!
reload(d1)
reload(cr)
reload(draw)
d1.scDeconvIGMRFEP1D()
```

```
iteration: 0 (alpha = 0.001639701514758859)
iteration: 1 (alpha = 7.61917207623319e-05)
iteration: 2 (alpha = 8.164593253869425e-05)
iteration: 3 (alpha = 8.252802804934801e-05)
iteration: 4 (alpha = 8.017281863038034e-05)
iteration: 5 (alpha = 8.203333383640499e-05)
iteration: 6 (alpha = 8.2820801281451e-05)
iteration: 7 (alpha = 8.340012393086003e-05)
iteration: 8 (alpha = 8.379822322181003e-05)
iteration: 9 (alpha = 8.407368544923991e-05)
```



Task 4

In [ ]:
```python
plt.figure(figsize=(12, 8))
plt.imshow(imread('./assignment/tasks/ex4.png'))
plt.axis('off');
```



4. Draws $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, (\beta\mathbf{L})^{-1})$ from a proper GMRF for $\mathbf{L} \succ 0$ (see §1.2.3).

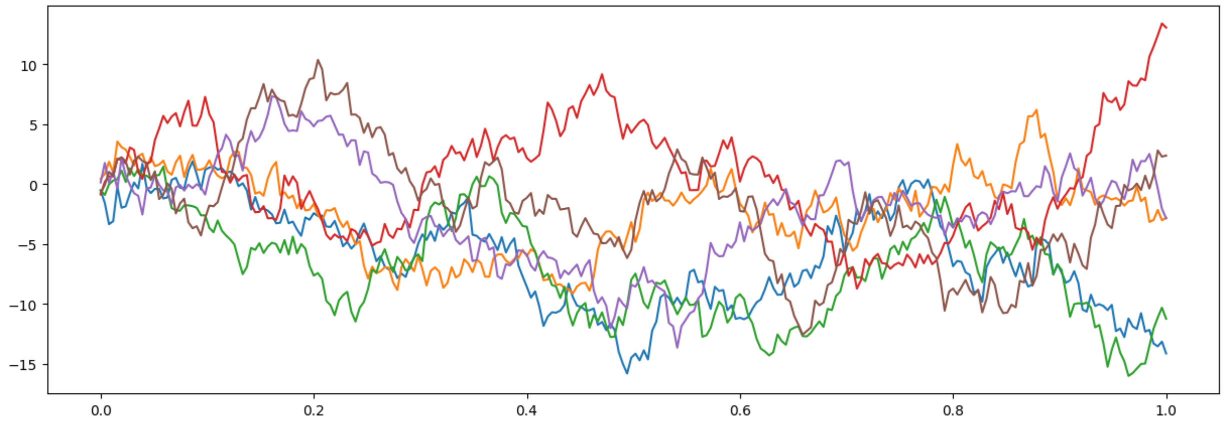
   a) Implement an algorithm that will draw a random vector $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, (\beta\mathbf{L})^{-1})$ where $\mathbf{L} \succ 0$ given by $\mathbf{Q}$ and $\beta = 1$. Display 6 realizations of $\mathbf{x}$. **Hint:** *A script for the implementation of this algorithm can be found in prbsets/draw/scDrawGMRFDBC1D.m. To draw random vectors $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$ from*

Math 6397 Bayesian Inverse Problems Dr. Andreas Mang 5 of 6

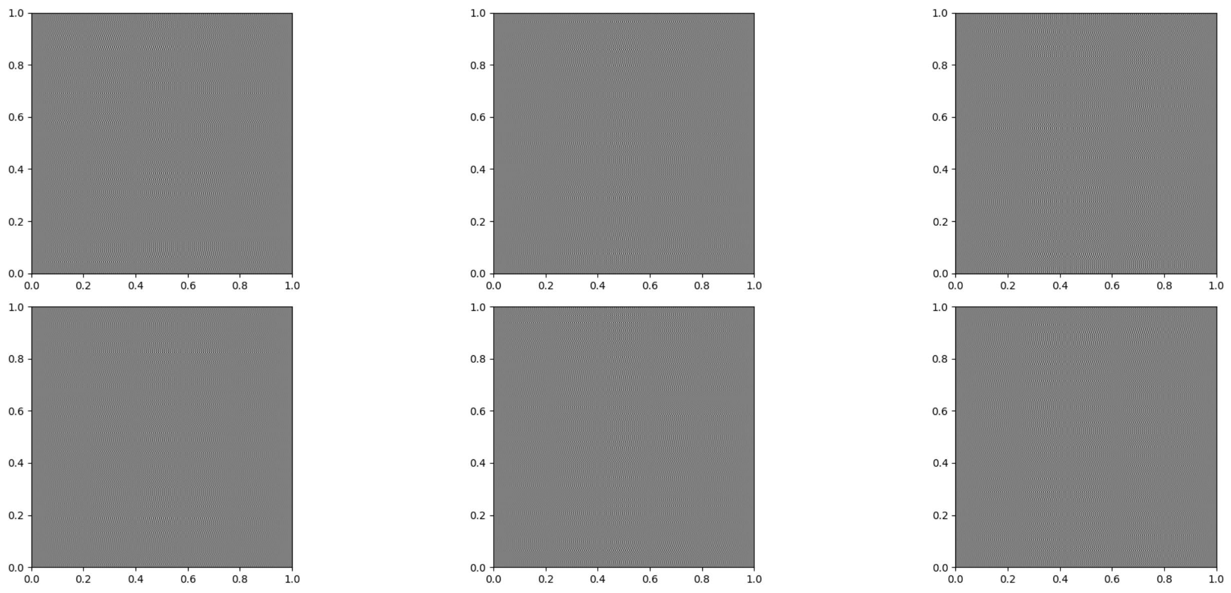
   a standard normal distribution you can use Matlab's `randn` *function. To compute the Cholesky decomposition $\mathbf{C}^\mathsf{T}\mathbf{C}$ of the precision matrix $\mathbf{L}$ you can use Matlab's* `chol` *function.*

   b) Extend your implementation from part a) to the two-dimensional setting, where $\mathbf{L} = \mathbf{I} \otimes \mathbf{Q} + \mathbf{I} \otimes \mathbf{Q} \succ \mathbf{0}$. Display 6 realizations of $\mathbf{x}$. **Hint:** *A script for the implementation of this algorithm can be found in prbsets/draw/scDrawGMRFDBC2D.m.*

In [ ]:
```python
#solution 4(a)
#attention, random seed is not fixed!
reload(d1)
reload(cr)
reload(draw)
draw.scDrawGMRFDBC1D()
```



In [ ]:
```python
#solution 4(b)
reload(d1)
reload(cr)
reload(draw)
draw.scDrawGMRFDBC2D()
```



Task 5

In [ ]:
```
plt.figure(figsize=(12, 8))
plt.imshow(imread('./assignment/tasks/ex5.png'))
plt.axis('off');
```

5. Draws $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, (\beta \mathbf{L})^{\dagger})$ from a IGMRF for which $\mathbf{L}$ is not SPD (see §1.2.4).

   a) Let $\mathbf{L}^{\dagger}$ denote the pseudoinverse of $\mathbf{L}$. Proof that $\mathbf{L}^{\dagger} = \lim_{\epsilon \searrow 0} \left(\mathbf{L}^{\mathsf{T}}\mathbf{L} + \epsilon \mathbf{I}\right)^{-1} \mathbf{L}^{\mathsf{T}}$.

   b) Implement an algorithm that will draw a random vector $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, (\beta \mathbf{L})^{\dagger})$, where $\beta = 1$, $\mathbf{L} = \mathbf{D}\mathbf{W}\mathbf{D}^{\mathsf{T}}$, $\mathbf{D}$ is as in (7), and the matrix $\mathbf{W}$ is given by $\mathbf{W} = \text{diag}(w_1, \ldots, w_n)$ with $w_i = 1$ for all $i = 1, \ldots, n$ with the exception of $w_{n/2} = 25/1000$. Use the algorithm described in §1.2.4. Use a Cholesky decomposition to efficiently apply the inverse of $(\tilde{\mathbf{D}}^{\mathsf{T}}\tilde{\mathbf{D}} + \varepsilon \mathbf{I})^{-1}$. Set ϵ to the square root of machine precision. Display 6 realizations of $\mathbf{x}$. **Hint:** *A script for the implementation of this algorithm can be found in prbsets/draw/scDrawIGMRFNBC1D.m. The gradient operator* $\mathbf{D}$ *in (7) is implemented in core/getGradMat.m.*

   c) Extend your implementation to the two-dimensional setting. The weight matrix $\mathbf{W}$ will have weights $w_{ij} = 1.0$ unless the points are located on the boundary of a circle along which we set $w_{ij} = 25/1000$. Display 6 realizations of $\mathbf{x}$. **Hint:** *A script for the implementation of this algorithm can be found in prbsets/draw/scDrawIGMRFNBC2D.m. The gradient operator for the two-dimensional setting is also implemented in core/getGradMat.m.*

In [ ]:
```
#solution of 5(a)
plt.figure(figsize=(10, 10))
plt.imshow(imread('./assignment/solutions/Q5a.jpg'))
plt.axis('off');
```

Consider

$$\lim_{\varepsilon} (L^T L + I\varepsilon)^{-1} L^T = (L^T L)^{-1} L^T$$

•) ? $L L^+ L = L$

$$L[(L^T L)^{-1} L^T] L = L L^{-1} (L^T)^{-1} L^T L = L$$

•) ? $L^+ L L^+ = L^+$

$$(L^T L)^{-1} L^T L (L^T L)^{-1} L^T = L^{-1} [(L^T)^{-1} L^T] L L^{-1} (L^T)^{-1} L^T = L^{-1} (L^T)^{-1} L^T$$

$$= (L^T L)^{-1} L^T = L^+$$

•) ? $(L^+ L)^* = L^+ L$

given $L \in \mathbb{R}_{m \times n}$

$$(L^+ L)^* = [L (L^T L)^{-1} L^T] = I^* = I$$

$$(L^+ L) = L (L^T L)^{-1} L^T = I$$

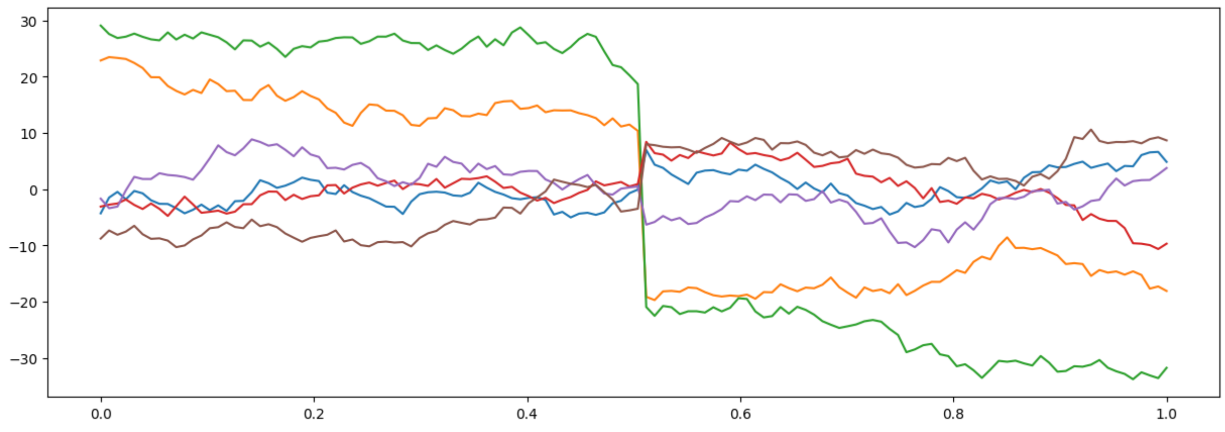
•) ? $(L L^+)^* = L L^+$

$$[L (L^T L)^{-1} L^T]^* = I^* = I$$

$$L (L^T L)^{-1} L^T = I$$

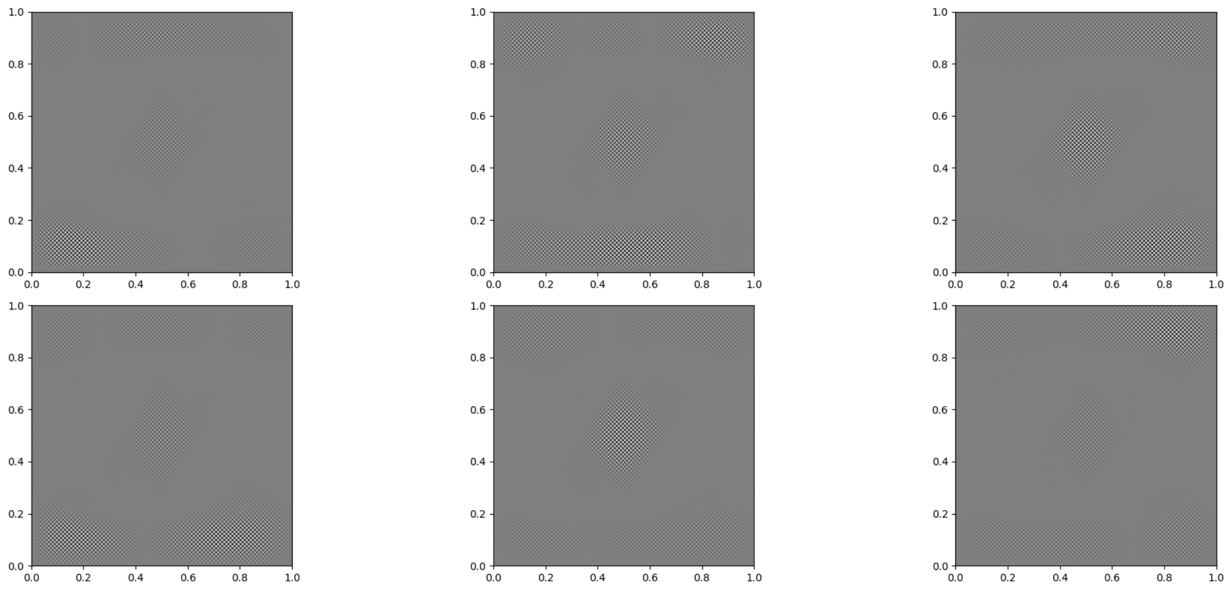
therefore $\lim_{\varepsilon \to 0} (L^T L - \varepsilon I)^{-1} L^T$ satisfies all the properties of the pseudoinverse.

In [ ]:
```
#solution 5(b)
reload(d1)
reload(cr)
reload(draw)
draw.scDrawIGMRFNBC1D()
```



In [ ]:
```
#solution 5(c)
# reload(d1)
# reload(cr)
reload(draw)
draw.scDrawIGMRFNBC2D()
```

```
(16384, 6) (6, 6) (6, 16384)
(16384, 6) (6, 6) (6, 16384)
(16384, 6) (6, 6) (6, 16384)
(16384, 6) (6, 6) (6, 16384)
(16384, 6) (6, 6) (6, 16384)
(16384, 6) (6, 6) (6, 16384)
```



Task 6

In [ ]:
```
plt.figure(figsize=(12, 8))
plt.imshow(imread('./assignment/tasks/ex6.png'))
plt.axis('off');
```

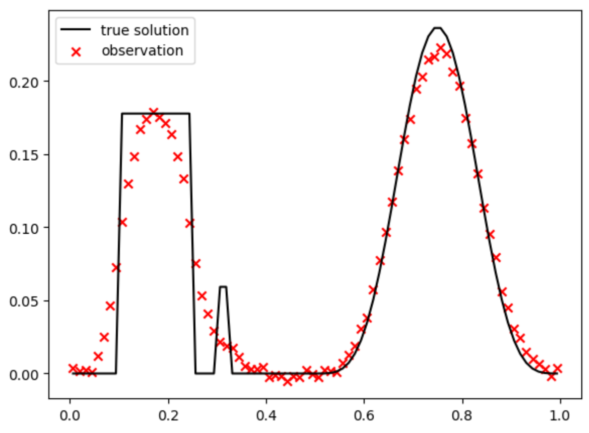
6. We consider the one-dimensional source reconstruction problem from past homework assignments. Our goal is to draw samples from the distribution defined by (3). Since we consider a one-dimensional problem, we can explicitly form the precision matrix $\tau\mathbf{K}^\mathsf{T}\mathbf{K}+\beta\mathbf{L}$. Since this matrix is SPD we can compute its Cholesky factorization. For the precision matrix $\mathbf{L}$ use $\mathbf{Q}$ in (5). Estimate τ based on the variance of $\mathbf{K}^\mathsf{T}\mathbf{K}\mathbf{x}_\alpha - \mathbf{y}_{\text{obs}}$, where $\mathbf{x}_\alpha$ is the Tikhonov solution for an adequate regularization parameter α. Select α based on GCV. Given τ and α, you can estimate β as $\beta = \alpha\tau$. The strategy to sample $\mathbf{x}$ from (3) is identical to your implementation under question 4. That is, it can be shown that samples from the distribution defined by (3) can be computed via

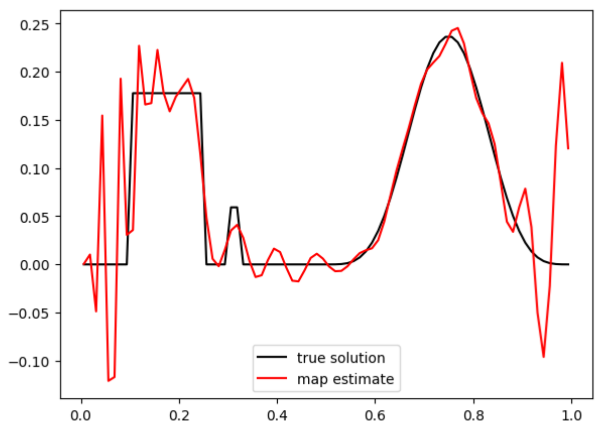
$$\mathbf{x} \mid \mathbf{y}_{\text{obs}}, \tau, \beta = \mathbf{C}^{-1}(\mathbf{C}^{-\mathsf{T}}\tau\mathbf{K}^\mathsf{T}\mathbf{y}_{\text{obs}} + \mathbf{v}), \quad \mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n).$$

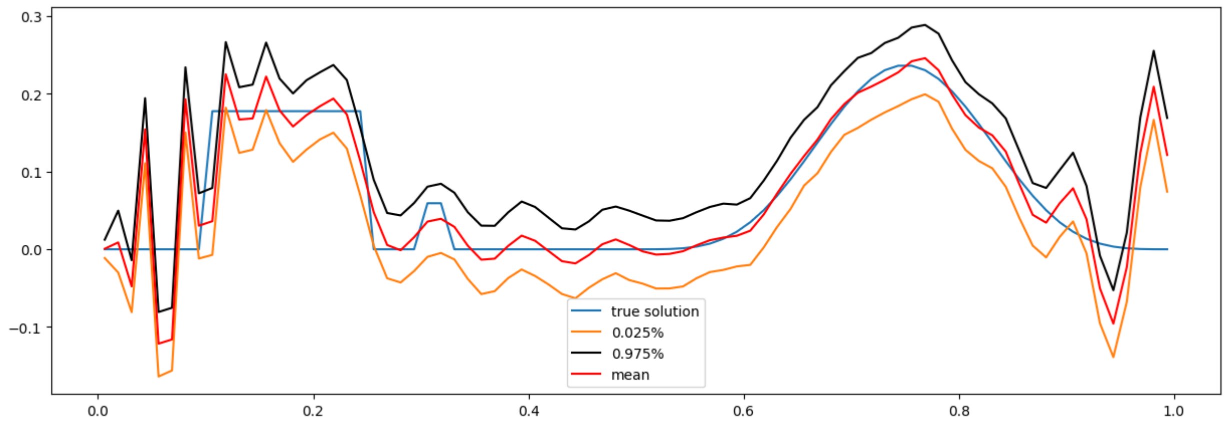
Notice that this expression involves the MAP point $\mathbf{x}_{\text{map}}$; you can use (4) to compute $\mathbf{x}_{\text{map}}$, where τ is selected as stated above. Draw 1,000 samples and visualize the mean and the 95% credibility interval. **Hint:** *A template implementation for this question is sample1D/scDeconvGMRFLAP1D.m. To compute the mean of the samples, you can use Matlab's* `mean` *function. A function to compute the 95% credibility intervals from the 1,000 samples you have drawn is implemented in core/getEmpQuant.m.*

In [94]:
```
reload(d1)
reload(cr)
reload(draw)
d1.scDeconvGMRFLAP1D()
```

```
optimal regularization parameter alpha = 0.001584182362018001
```


true solution
observation
0.00
0.05
0.10
0.15
0.20
0.0
0.2
0.4
0.6
0.8
1.0


true solution
map estimate
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
0.25
0.0
0.2
0.4
0.6
0.8
1.0


true solution
0.025%
0.975%
mean
−0.1
0.0
0.1
0.2
0.3
0.0
0.2
0.4
0.6
0.8
1.0