Inverse Problems: Problem Set №1

Group C: Mateusz Brodowicz and Anton Myshak

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


from prbsets import deconv1D as conv
from importlib import reload #to reload libs online, like cr = reload(cr)
from matplotlib.image import imread
```

Task 1

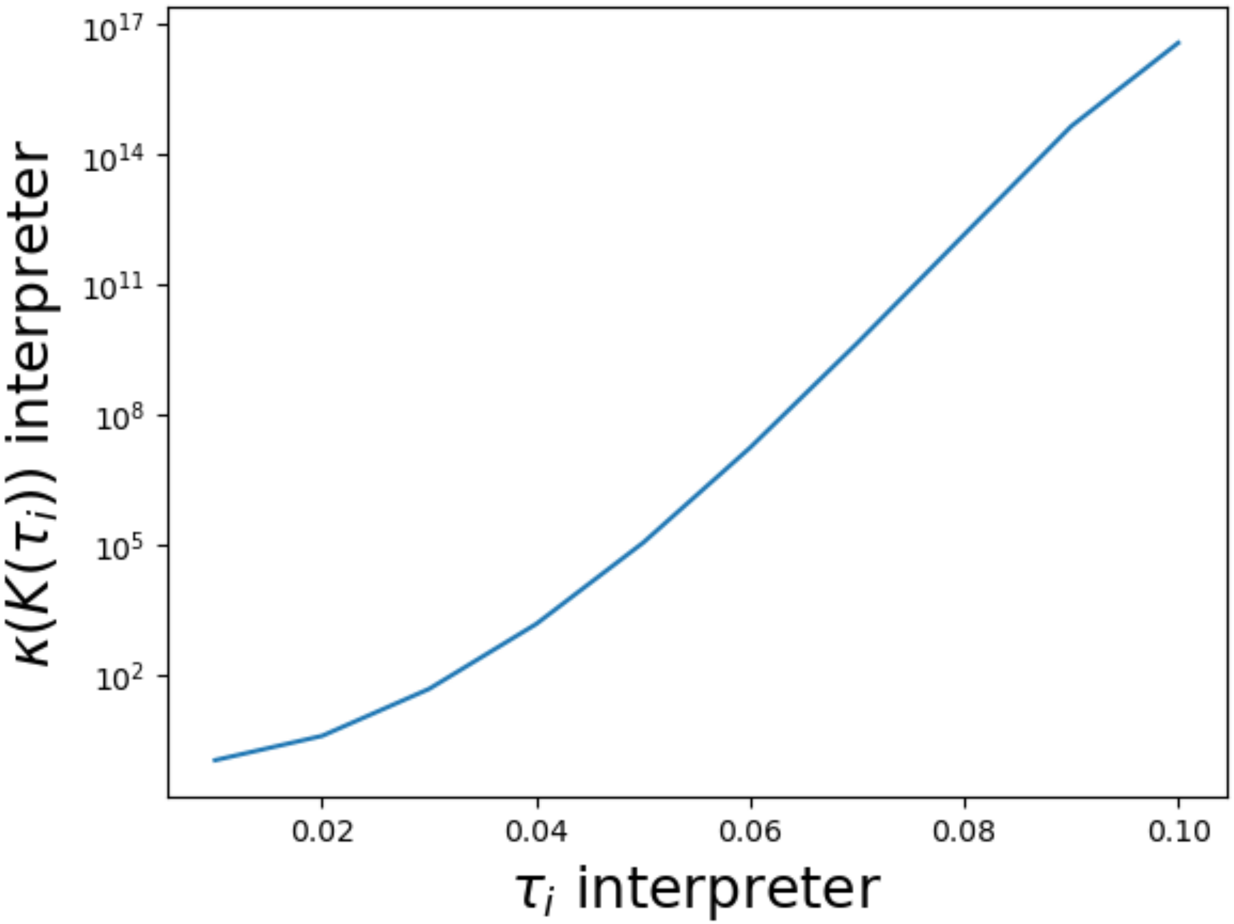
In [2]:
```
plt.figure(figsize=(11,4))
plt.imshow(imread('./assignment/tasks/ex1.png'))
plt.axis('off');
```

1. First, we explore the conditioning of the problem (5) as a function of the parameterization of the kernel as well as the discretization accuracy. **Hint:** *To compute the condition number of a matrix, you can use Matlab's* `cond` *command. The script to work on this assignment is prbsets/deconv/scCondK1D.m.*

   a) Compute the condition number of **K** as a function of τ and plot it (semi-logarithmic plot). Set n to 32. Select $\tau = \gamma(1\mathrm{e}-2)$, where γ represents integers from 1 to 10. What are your observations?

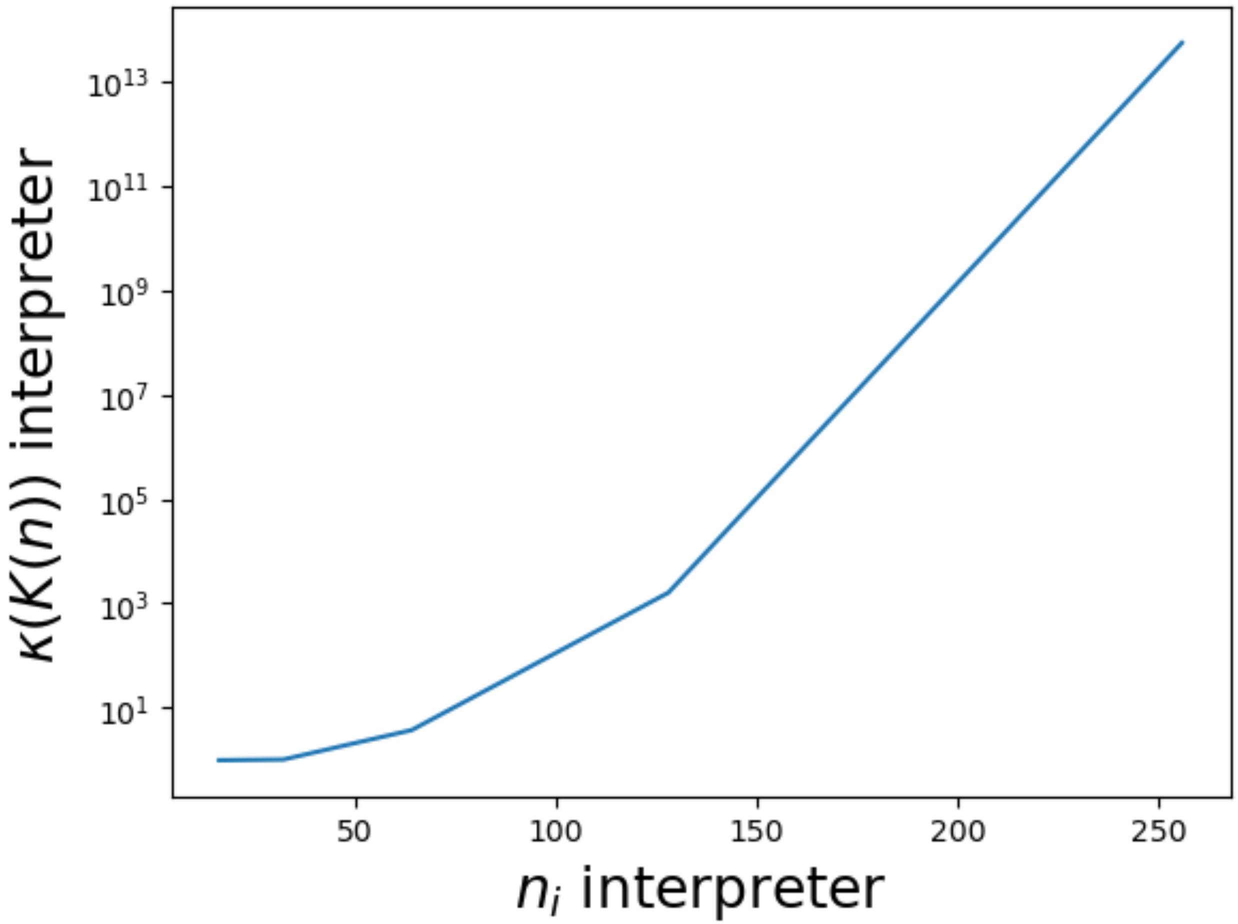
   b) Compute the condition number of **K** as a function of n and plot it (semi-logarithmic plot). Set $\tau = 1\mathrm{e}-2$. Select $n \in \{16, 32, 64, 128, 256\}$. What are your observations?

In [3]:
```python
# statement (a)
# first part of the scCondK1D file
reload(conv)
reload(cr)

n = 32 # number of points
tau = np.linspace(0.01, 0.1, 10) # kernel parameterization

# initialize memory
m = len(tau)
kappa = np.zeros((m,1))

# compute condition number estimate as a function of sigma
for i in range(m):
    K = conv.getKernel1D( n, tau[i] )
    kappa[i] = np.linalg.cond(K) # ADD YOUR CODE HERE

# visualize computed condition numbers
plt.figure()
plt.plot(tau, kappa)
plt.yscale('log')
plt.xlabel(r'$\tau_i$ interpreter', size=19)
plt.ylabel(r'$\kappa(K(\tau_i))$ interpreter', size=19);
```



We can notice exponential growth in the begginning, then the growth becomes linear, but after $\tau=0.09$ the line starts to concave down.

In [4]:
```python
# statement (b)
# second part of the scCondK1D file
reload(conv)
reload(cr)

tau = 0.01 # standard deviation
n = [16, 32, 64, 128, 256] # number of grid points

# initialize memory
m = len(n);
kappa = np.zeros((m,1));

# compute condition number estimate as a function of sigma
for i in range(m):
    K = conv.getKernel1D(n[i], tau)
    kappa[i] = np.linalg.cond(K) # ADD YOUR CODE HERE

# visualize computed condition numbers
plt.figure()
plt.plot(n, kappa)
plt.yscale('log')
plt.xlabel(r'$n_i$ interpreter', size=19)
plt.ylabel(r'$\kappa(K(n))$ interpreter', size=19);
```



We can notice exponential growth.

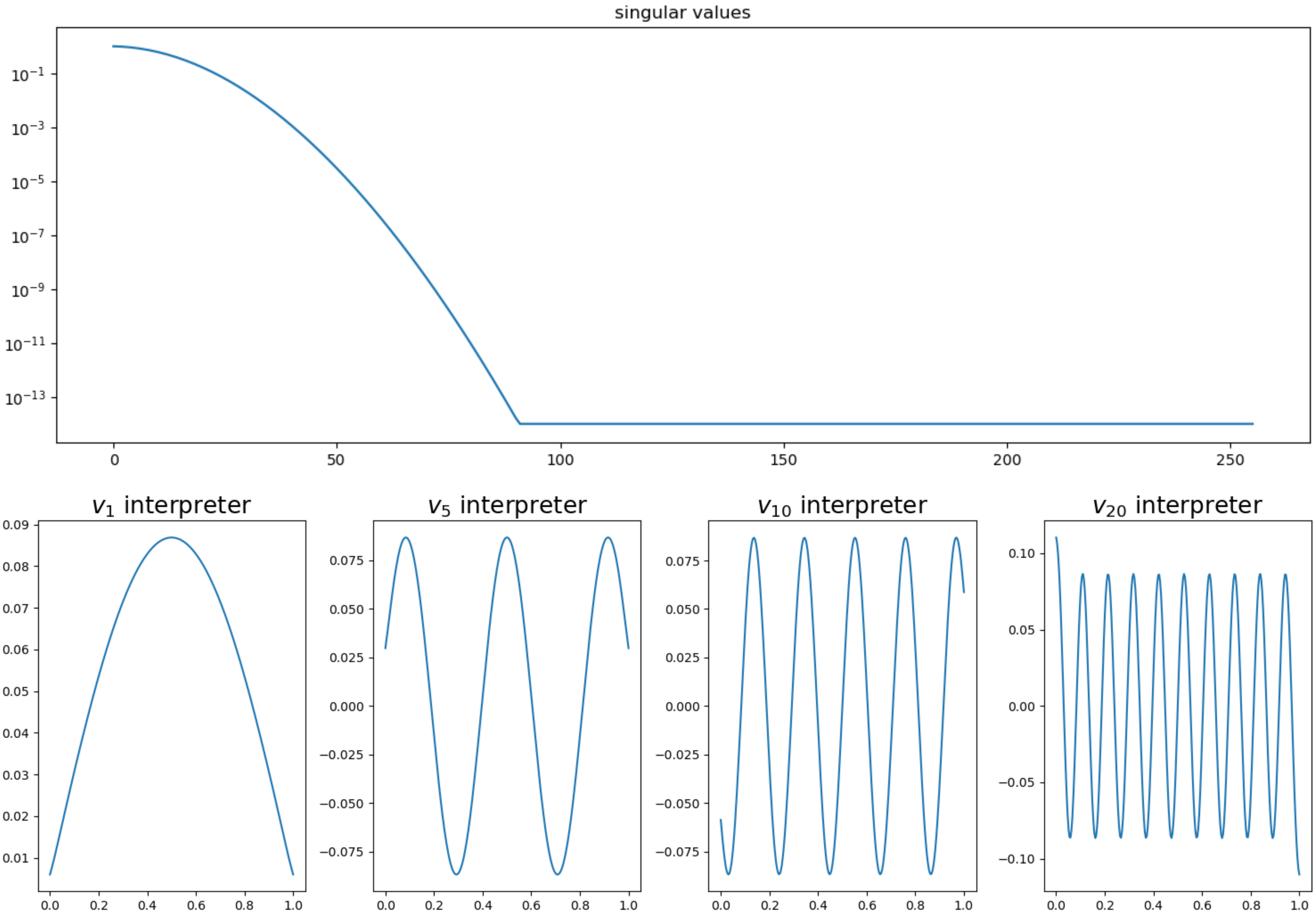
Task 2(a)

In [5]:
```
plt.figure(figsize=(15,5))
plt.imshow(imread('./assignment/tasks/ex2a.png'))
plt.axis('off');
```

2. The first approach we are going to consider to compute a solution to (**??**) is based on the truncated singular value decomposition (**TSVD**). That is, we are going to consider the decomposition $\mathbf{K} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T}$, $\mathbf{U} = \begin{bmatrix}\mathbf{u}_1, \dots, \mathbf{u}_m\end{bmatrix} \in \mathbb{R}^{m,m}$, $\mathbf{V} = \begin{bmatrix}\mathbf{v}_1, \dots, \mathbf{v}_n\end{bmatrix} \in \mathbb{R}^{n,n}$, $\mathbf{S} = \operatorname{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m,n}$, $p = \min\{n, m\}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$. Under the assumption that $\mathbf{K} \succeq 0$, we have that $\sigma_i \geq 0$, $i = 1, \dots, p$, the singular values coincide with the eigenvalues of $\mathbf{K}$, and $\mathbf{U} = \mathbf{V}$. Moreover, the column $\mathbf{u}_j \in \mathbb{R}^m$ for an orthonormal basis with $\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}$, i.e., $\mathbf{U}^\mathsf{T} = \mathbf{U}^{-1}$. The truncated SVD of $\mathbf{K}$ is given by $\mathbf{K} = \mathbf{U}_r\mathbf{S}_r\mathbf{V}_r^\mathsf{T}$, with $\mathbf{U}_r \in \mathbb{R}^{m,r}$, $\mathbf{S} \in \mathbb{R}^{r,r}$, $\mathbf{V}_r \in \mathbb{R}^{n,r}$.

   a) Compute the SVD of the matrix $\mathbf{K}$. Plot the singular vectors (semi-logarithmic plot; threshold the values at $1\mathrm{e}-12$, i.e., all values below $1\mathrm{e}-12$ are set to $1\mathrm{e}-12$). Plot the singular vectors $\mathbf{v}_j$ (i.e., the columns of the matrix $\mathbf{V}$) corresponding to the singular vectors σ_j, $j \in \{1, 2, 10, 20\}$. What are your observations; i.e., how do the singular vectors $\mathbf{v}_j$ behave as the singular values σ_j decrease? **Hint:** *To compute the SVD you can use Matlab's `svd` function. The script for this assignment is prbsets/deconv/scCompKerSVD1D.m.*

In [6]:
```
reload(conv)
reload(cr)
conv.scCompKerSVD1D()
```



singular values
v_1 interpreter
v_5 interpreter
v_{10} interpreter
v_{20} interpreter

Task 2(b)

In [7]:
```
plt.figure(figsize=(11,10))
plt.imshow(imread('./assignment/tasks/ex2b.png'))
plt.axis('off');
```

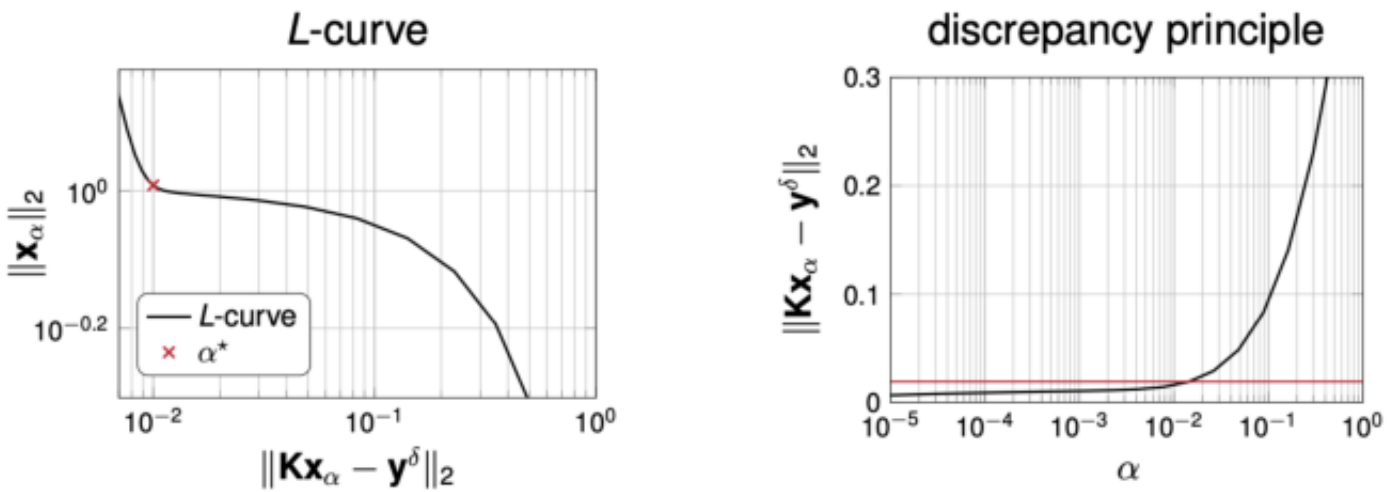


Figure 4: Choosing the regularization parameter α: The red cross on the L-curve (left plot), which corresponds to the point with largest curvature, yields the optimal regularization parameter according to the L-curve criterion. For the discrepancy criterion (right plot), the optimal parameter corresponds to the intersection of the data misfit curve with the red line indicating the noise level.

b) Compute the TSVD for the target ranks $r \in \{5, 10, 50\}$. To compute the rank-r approximations, you can simply compute the full SVD and select the first r columns of the matrices $\mathbf{U}$ and $\mathbf{V}$, and the $r \times r$ submatrix of the diagonal matrix $\mathbf{S}$. Compute $\mathbf{K}_r = \mathbf{U}_r\mathbf{S}_r\mathbf{V}_r$, $\mathbf{U}_r \in \mathbb{R}^{m,r}$, $\mathbf{S} \in \mathbb{R}^{r,r}$, $\mathbf{V}_r \in \mathbb{R}^{n,r}$. **Hint:** *To compute the SVD you can use Matlab's `svd` function. The script for* this assignment is prbsets/deconv/scTSVDK1D.m. To implemented the TSVD, you can use the template *core/tSVD.m*.

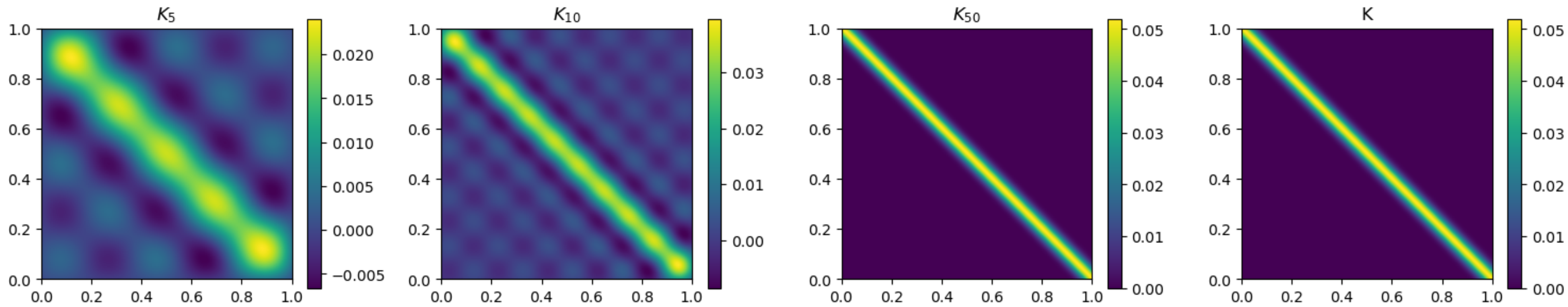
i. Visualize the matrices $\mathbf{K}_r$, $r \in \{5, 10, 50\}$ and $\mathbf{K}$. **Hint:** *You can use Matlab's* `imagesc` *to visualize these matrices.*

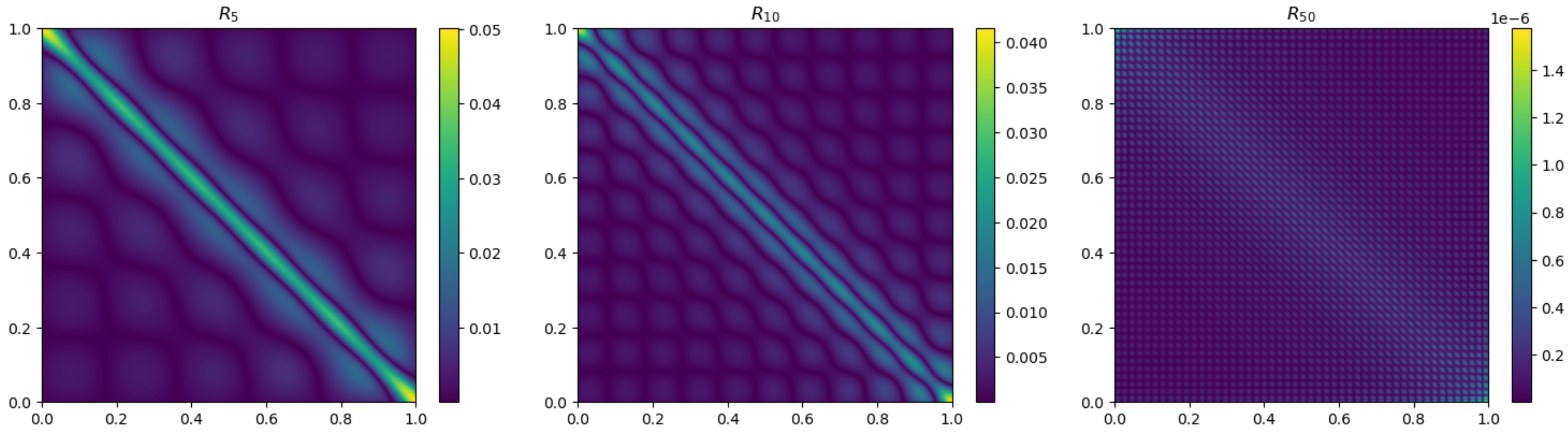
ii. Visualize the point-wise absolute value of the residual matrix $\mathbf{R}_r = |\mathbf{K}_r - \mathbf{K}|$. **Hint:** *You can use Matlab's* `imagesc` *to visualize the matrices* $\mathbf{R}_r$.

iii. Compute the relative error $e_r = \|\mathbf{K} - \mathbf{K}_r\|_2 / \|\mathbf{K}\|_2$. How does the error behave?

In [8]:
```
reload(conv)
reload(cr)
conv.scTSVDK1D()
```

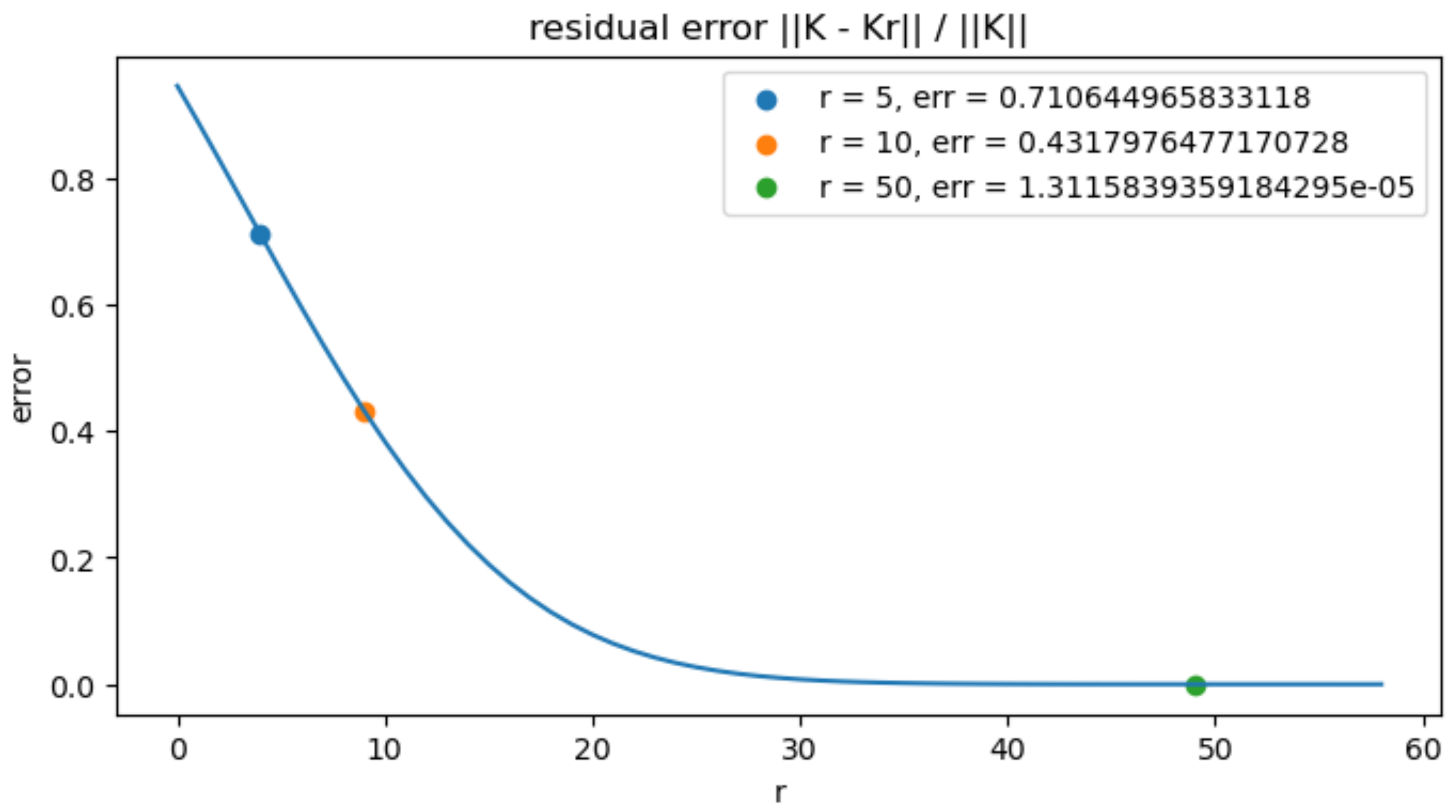
```
error (rank  5) =  0.710644965833118
error (rank 10) =  0.4317976477170728
error (rank 50) =  1.3115839359184295e-05
```





How does the error behave?

In [9]:
```
reload(conv)
reload(cr)
conv.PlotResErr()
```



Task 2(c)

In [10]:
```
plt.figure(figsize=(15,6))
plt.imshow(imread('./assignment/tasks/ex2c.png'))
plt.axis('off');
```

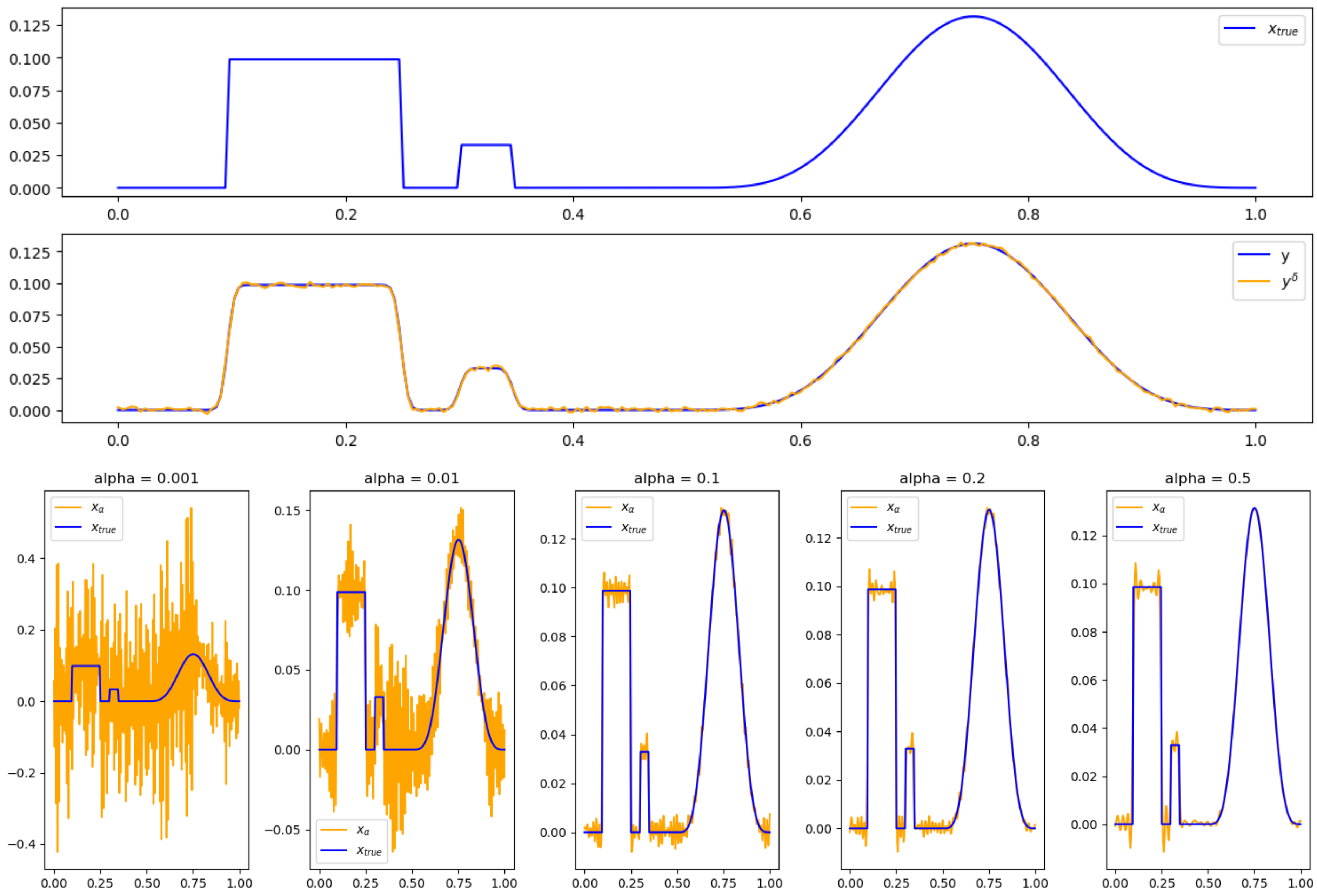
c) Consider the cases $\tau \in \{5\mathrm{e}-3, 2\mathrm{e}-2\}$. Set $n = 256$. Select the noise level δ such that the signal-to-noise ratio $\|\mathbf{K}\mathbf{x}_{\text{true}}\|/\sqrt{n\delta^2}$ is equal to a constant $\gamma = 50$. Use a TSVD to compute the regularized solution $\mathbf{x}_\alpha = \mathbf{R}_\alpha \mathbf{y}^\delta$, where

$$\mathbf{R}_\alpha \mathbf{y}^\delta = \sum_{i=1}^{n} w(\sigma_i)\langle \mathbf{u}_i, \mathbf{y}^\delta \rangle \mathbf{u}_i \quad \text{with} \quad w(\sigma_i) = \begin{cases} \sigma_i^{-1} & \text{for } \sigma_i > \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

As indicated above, we define the TSVD in terms of a threshold α instead of a target rank r. Consider the thresholds $\alpha \in \{1\mathrm{e}-3, 1\mathrm{e}-2, 1\mathrm{e}-1, 2\mathrm{e}-1, 5\mathrm{e}-1\}$ to compute the regularized solutions $\mathbf{x}_\alpha$. Plot the solutions $\mathbf{x}_\alpha$ and compare them to the true solution $\mathbf{x}_{\text{true}}$. **Hint:** *One way of computing the truncated SVD is to compute the SVD and identify the index j for which the singular value $\sigma_j < \alpha$. This can be accomplished by using Matlab's `find` function. The pseudoinverse can then be computed by inverting the truncated SVD matrix, i.e.,* $\mathbf{R}_\alpha = (\mathbf{U}_r \mathbf{S}_r \mathbf{V}_r^\top)^{-1}$, *where $r = j$. The script for this assignment is prbsets/deconv/scDeconvTSVD1D.m. To implemented the TSVD based on thresholding, you can use the template core/tSVDTH.m.*

In [11]:
```
reload(conv)
reload(cr)
conv.scDeconvTSVD1D()
```

```
condition number of K: 1619.741154844817
```



x_{true}
y
y^δ
alpha = 0.001
alpha = 0.01
alpha = 0.1
alpha = 0.2
alpha = 0.5
x_α
x_{true}

Task 3(a)

In [12]:
```
plt.figure(figsize=(15,5))
plt.imshow(imread('./assignment/tasks/ex3a.png'))
plt.axis('off');
```

3. The second approach we are going to consider for computing a solution to (5) is based on a Tikhonov regularization scheme. Set $n = 256$ and $\tau = 5\text{e}-3$. Select the noise level δ such that the signal-to-noise ratio $\|\mathbf{K}\mathbf{x}_{\text{true}}\| / \sqrt{n\delta^2}$ is equal to a constant $\gamma = 50$. Consider the variational optimization problem

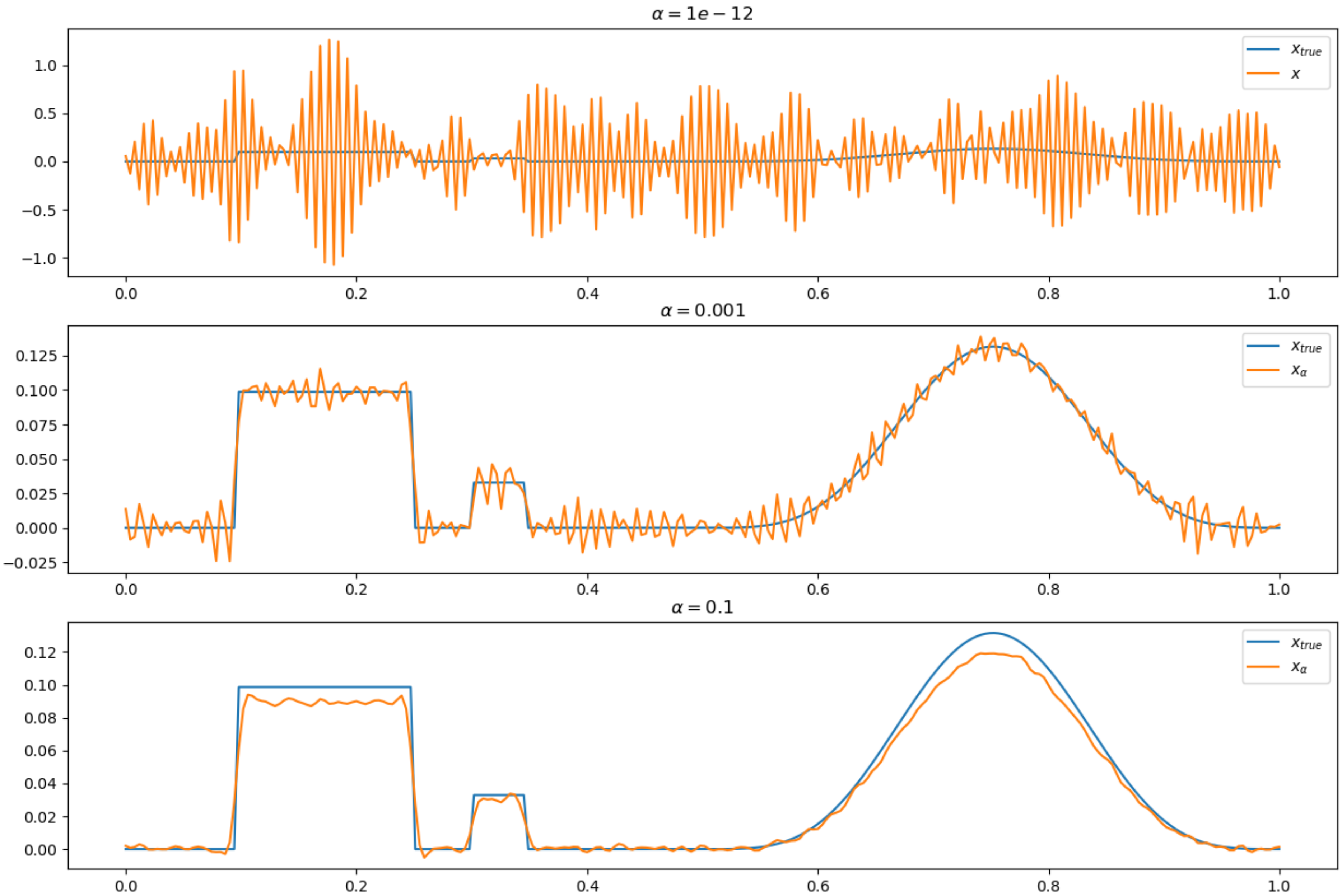
$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \frac{1}{2}\|\mathbf{K}\mathbf{x} - \mathbf{y}^\delta\|_2^2 + \frac{\alpha}{2}\|\mathbf{x}\|_2^2.$$

The associated first order optimality conditions are given by

$$\mathbf{K}^\mathsf{T}(\mathbf{K}\mathbf{x}^\star - \mathbf{y}^\delta) + \alpha\mathbf{x}^\star = \mathbf{0}. \tag{12}$$

a) Compute the solution $\mathbf{x}_\alpha$ by solving the (linear) optimality system (12) for $\alpha \in \{1\text{e}-12, 1\text{e}-3, 1\text{e}-1\}$. Plot the solution $\mathbf{x}_\alpha$ and the true solution $\mathbf{x}_{\text{true}}$. **Hint:** *To compute the solution of (12), you can use Matlab's backslash operation. A script to help with the implementation of this direct solver is prbsets/deconv/scDeconvTRegDir1D.m.*

In [13]:
```
reload(conv)
reload(cr)
conv.scDeconvTRegDir1D()
```



$\alpha = 1e-12$
x_{true}
x
$\alpha = 0.001$
x_{true}
x_α
$\alpha = 0.1$
x_{true}
x_α

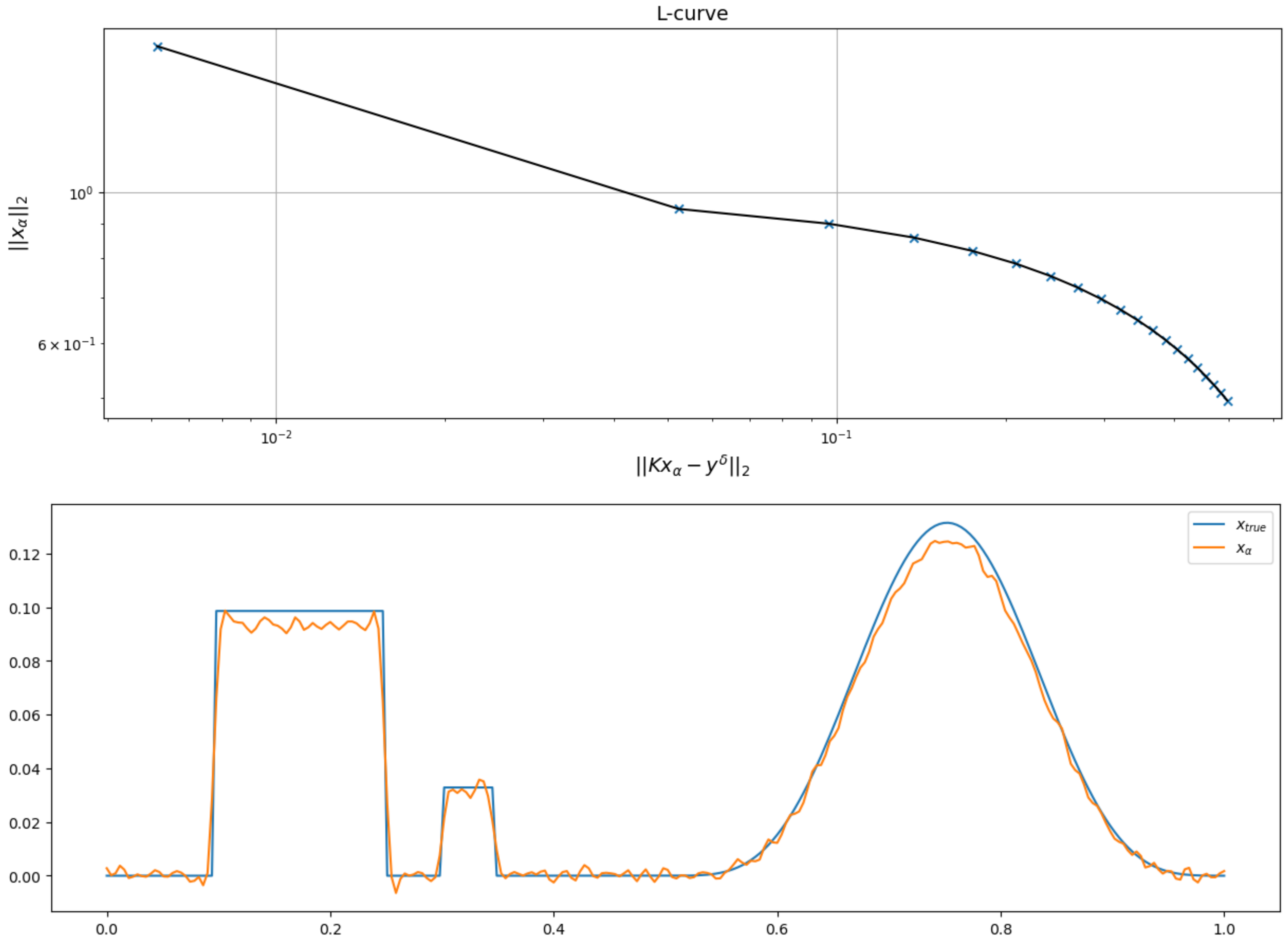
Task 3(b)

In [14]:
```
plt.figure(figsize=(15,5))
plt.imshow(imread('./assignment/tasks/ex3b.png'))
plt.axis('off');
```

b) Determine the (approximate) optimal value $\alpha_{\text{opt}} > 0$ of the regularization parameter $\alpha > 0$ for the Tikhonov regularization using the L-curve criterion. To do so, one needs to compute the solution of the inverse problem (here, this means solving the optimality system (12)) for varying regularization parameters $\alpha_i > 0$. Use the implementation from part (a) to do so. Select $\alpha_i \in [1e-5, 1]$. Select 20 different values. To determine the optimal regularization parameter, plot the norm of the computed solution $\|\mathbf{x}_\alpha\|_2$ versus the norm of the residual $\|\mathbf{K}\mathbf{x}_\alpha - \mathbf{y}^\delta\|_2$ using a "log-log plot" (i.e., using a two-dimensional graph of numerical data that uses a logarithmic scale on both the horizontal and vertical axes). The optimal regularization parameter α_{opt} is located in the corner of the resulting L-shaped curve. This is illustrated in Fig. 4. Compute the solution using the determined regularization parameter α_{opt} and compare it (visually) to the true solution $\mathbf{x}_{\text{true}}$.
Hint: *A script to help with the implementation of the search for an optimal regularization parameter $\alpha_{opt} > 0$ based on the L-curve criterion is prbsets/deconv/scDeconvTRegDir1D.m. A template for implementing a plot for the L-curve is core/evalLCurve.m.*

In [15]:
```
reload(conv)
reload(cr)

alphalist = np.linspace(1e-5, 1, 20)
K, y_delta = conv.scDeconvTRegDir1D(alphalist, plot=False)

cr.evalLCurve(K, y_delta, lambda alpha: np.linalg.lstsq(K.T@K + alpha*np.eye(256), K.T@y_delta, rcond=None)[0], a

#the best alpha should be?
best_alpha = alphalist[1]
print('\nbest alpha =', best_alpha)
x_true = conv.getDeconvSource1D(256)

plt.figure(figsize=(15, 5))
plt.plot(np.linspace(0, 1, 256), x_true, label=r'$x_{true}$')
plt.plot(np.linspace(0, 1, 256), np.linalg.lstsq(K.T@K + best_alpha*np.eye(256), K.T@y_delta, rcond=None)[0], lab
plt.legend();
```

```
alpha=1e-05:  ||r|| = 0.006141045408478906 4  ||x|| = 1.6377429071357048
alpha=0.05264105263157895:  ||r|| = 0.05228479616933959  ||x|| = 0.9454167076159778
alpha=0.10527210526315789:  ||r|| = 0.09671748777582695  ||x|| = 0.8995720063147691
alpha=0.15790315789473686:  ||r|| = 0.1374559820369508  ||x|| = 0.8581272767455912
alpha=0.2105342105263158:  ||r|| = 0.1747113326317081 8  ||x|| = 0.820393958703741 9
alpha=0.26316526315789474:  ||r|| = 0.20886961775845891  ||x|| = 0.7858710477554313
alpha=0.3157963157894737:  ||r|| = 0.24028939245610711  ||x|| = 0.7541554804593473
alpha=0.3684273684210526:  ||r|| = 0.26928290520886183  ||x|| = 0.724913058801355
alpha=0.4210584210526316:  ||r|| = 0.29611897496925293  ||x|| = 0.69786241050859
alpha=0.47368947368421055:  ||r|| = 0.3210290091814626  ||x|| = 0.6727641974101884
alpha=0.5263205263157894:  ||r|| = 0.3442128400600809  ||x|| = 0.6494132099910184
alpha=0.5789515789473684:  ||r|| = 0.36584375083505305  ||x|| = 0.6276323235415094
alpha=0.6315826315789473:  ||r|| = 0.38607266365056675  ||x|| = 0.6072677601657027
alpha=0.6842136842105263:  ||r|| = 0.40503159683696316  ||x|| = 0.5881853114520831
alpha=0.7368447368421052:  ||r|| = 0.42283651679469103  ||x|| = 0.5702672891126971
alpha=0.7894757894736841:  ||r|| = 0.43958969662656266  ||x|| = 0.5534100385385242
alpha=0.8421068421052631:  ||r|| = 0.45538167466126733  ||x|| = 0.5375218942134489
alpha=0.8947378947368421:  ||r|| = 0.47029288817139103  ||x|| = 0.522521486091543
alpha=0.947368947368421:  ||r|| = 0.4843950425955068  ||x|| = 0.5083363274886573
alpha=1.0:  ||r|| = 0.4977522644766195  ||x|| = 0.4949016306933726

best alpha = 0.05264105263157895
```



L-curve
$||x_\alpha||_2$
10^0
6×10^{-1}
10^{-2}
10^{-1}
$||Kx_\alpha - y^\delta||_2$
x_{true}
x_α
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.0
0.2
0.4
0.6
0.8
1.0

Task 3(c)

In [16]:
```
plt.figure(figsize=(15,4))
plt.imshow(imread('./assignment/tasks/ex3c.png'))
plt.axis('off');
```

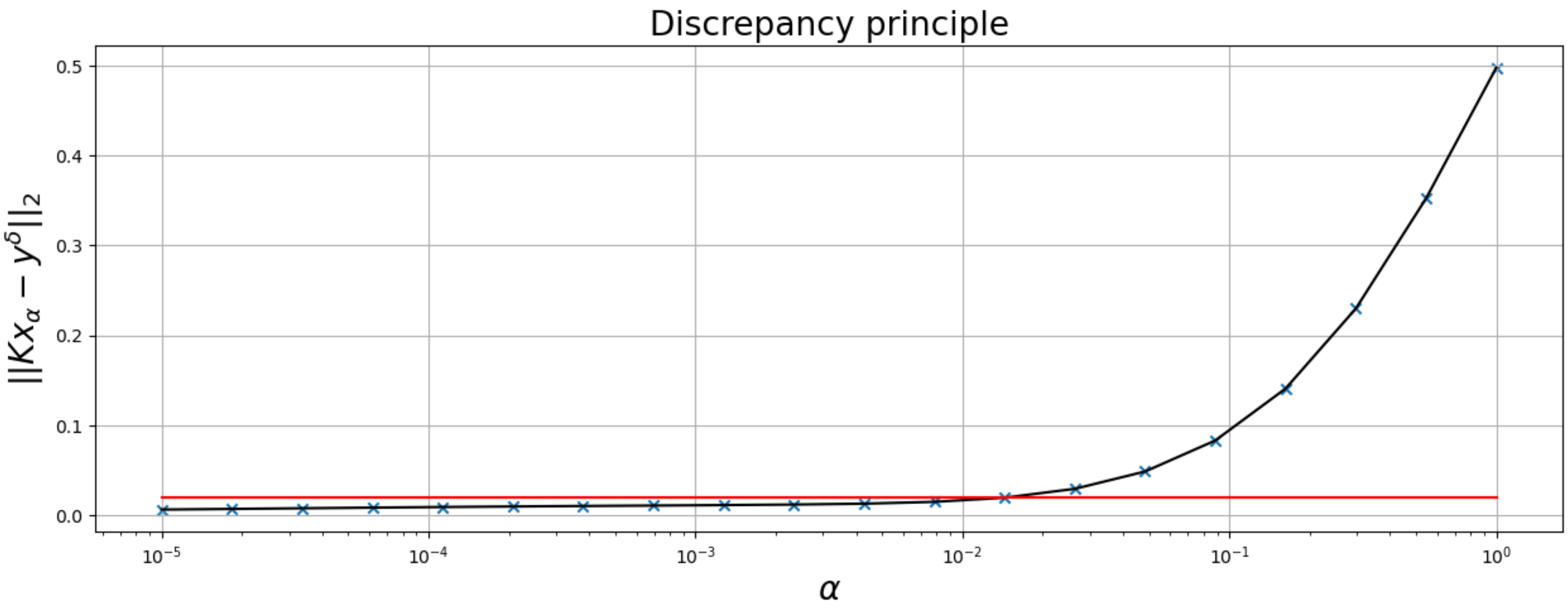
c) Determine the (approximate) optimal value $\alpha_{\text{opt}} > 0$ of the regularization parameter $\alpha > 0$ for the Tikhonov regularization using Morozov's discrepancy principle, i.e., find the largest value of α such that

$$\|\mathbf{K}\mathbf{x}_\alpha - \mathbf{y}^\delta\|_2 \leq \mu,$$

where $\mu = \|\delta\boldsymbol{\eta}\|_2$ and $\mathbf{x}_\alpha$ is the solution of the Tikhonov-regularized inverse problem with regularization parameter α. To search for an optimal α, select $\alpha_i \in [1e-5, 1]$. Select 20 different values. **Hint:** *A script to help with the implementation of the search for an optimal regularization parameter $\alpha_{\text{opt}} > 0$ based on the discrepancy principle is prbsets/deconv/scDeconvTRegMDP1D.m. A template for implementing the search for an optimal α using Morozov's discrepancy principle is core/evalDisPrinc.m.*

In [17]:
```
reload(conv)
reload(cr)
conv.scDeconvTRegMDP1P()
```

```
err = 0.01934526083266685 <= 0.01959958144993963 = delta
optimal regularization parameter: 0.01438449888287663
```



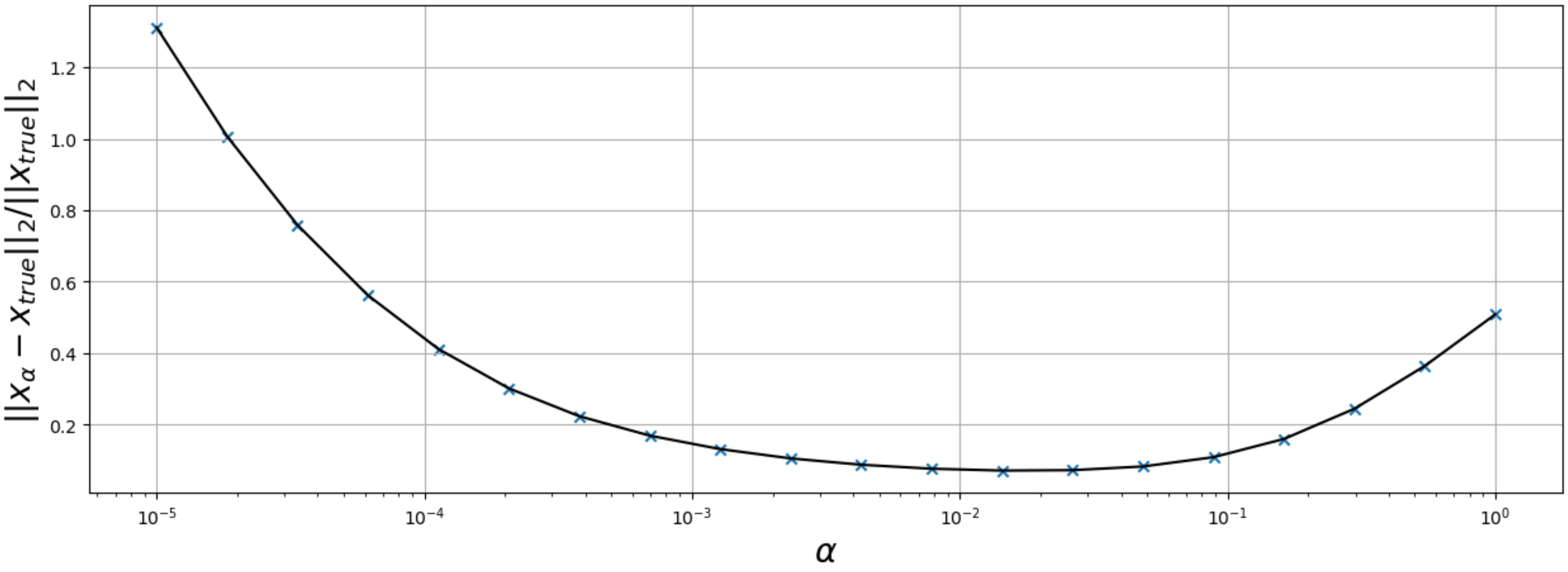
Task 3(d)

In [18]:
```
plt.figure(figsize=(12,4))
plt.imshow(imread('./assignment/tasks/ex3d.png'))
plt.axis('off');
```

d) Plot the relative error in the reconstruction, $e_\alpha = \|\mathbf{x}_{\text{true}} - \mathbf{x}_\alpha\|_2 / \|\mathbf{x}_{\text{true}}\|_2$ as a function of $\alpha > 0$, where $\mathbf{x}_\alpha \in \mathbb{R}^n$ is the Tikhonov regularized solution. Which value of α_i (approximately) minimizes this error? Compare your observations to the optimal values of α obtained for the L-curve method, Morozov's principle, and the truncated SVD. A template for the computation of this error can be found in prbsets/deconv/scDeconvTRegERR1D.m.

In [19]:
```
reload(conv)
reload(cr)
conv.scDeconvTRegERR1D()
```

```
run 0: error for alpha=1e-05: [1.31144075]
run 1: error for alpha=1.8329807108324375e-05: [1.00678683]
run 2: error for alpha=3.359818286283781e-05: [0.75823897]
run 3: error for alpha=6.158482110660267e-05: [0.56153885]
run 4: error for alpha=0.00011288378916846884: [0.41138298]
run 5: error for alpha=0.00020691380811147902: [0.30131475]
run 6: error for alpha=0.000379269019073225: [0.22335336]
run 7: error for alpha=0.0006951927961775605: [0.16906079]
run 8: error for alpha=0.0012742749857031334: [0.1313707]
run 9: error for alpha=0.002335721469090121: [0.10530221]
run 10: error for alpha=0.004281332398719391: [0.08762199]
run 11: error for alpha=0.007847599703514606: [0.07650128]
run 12: error for alpha=0.01438449888287663: [0.07128551]
run 13: error for alpha=0.026366508987303583: [0.07246206]
run 14: error for alpha=0.04832930238571752: [0.08279503]
run 15: error for alpha=0.08858667904100823: [0.10880149]
run 16: error for alpha=0.1623776739188721: [0.15983178]
run 17: error for alpha=0.2976351441631319: [0.24439586]
run 18: error for alpha=0.5455594781168515: [0.36439286]
run 19: error for alpha=1.0: [0.50852662]

best alpha = 0.01438449888287663
```



L-curve method (3b) gives us best $\alpha = 0.05264$, while Morozov's discrepancy principle (3c) and minimization of the relative error in the reconstruction (3d) provide us a different best $\alpha = 0.01438$.

Task 4

In [20]:
```
plt.figure(figsize=(15,5))
plt.imshow(imread('./assignment/tasks/ex4.png'))
plt.axis('off');
```

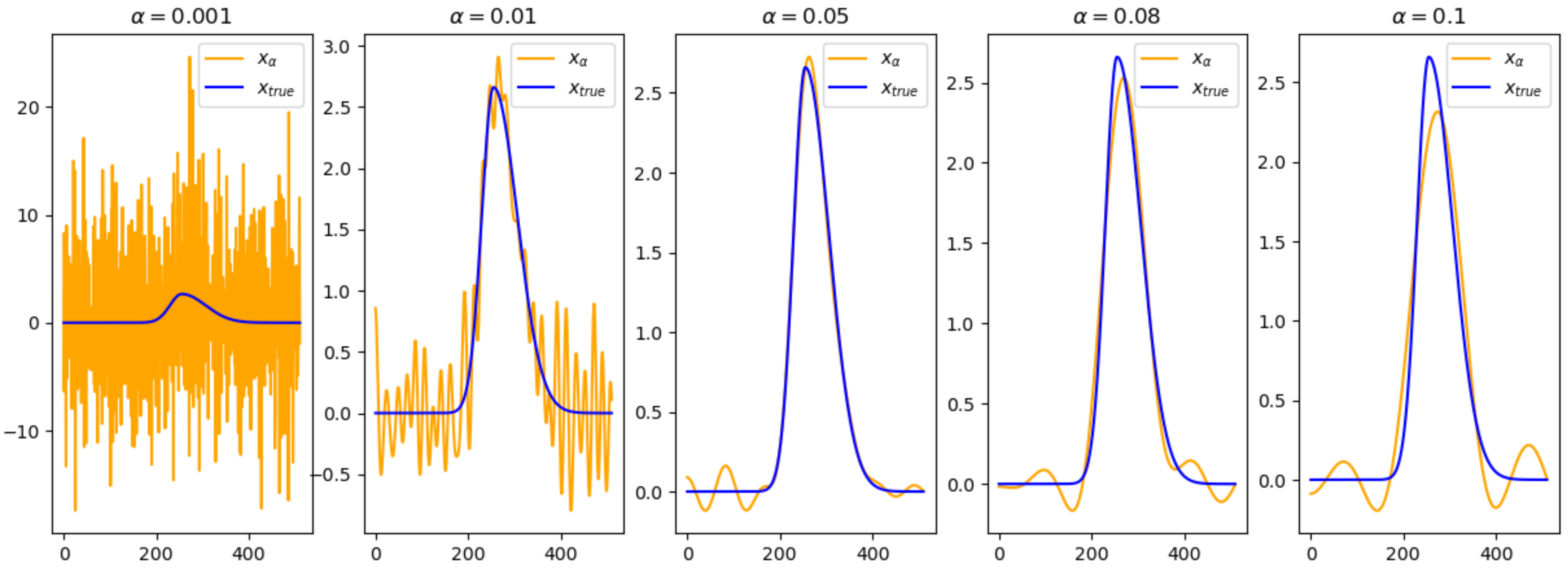
4. Next, we consider the kernel-reconstruction problem (6). Set $n = 256$. Select the noise level δ such that the signal-to-noise ratio $\|\mathbf{K}\mathbf{x}_{\text{true}}\|/\sqrt{n\delta^2}$ is equal to a constant $\gamma = 50$.

   a) Likewise to the deconvolution problem, use a TSVD to compute the regularized solution $\mathbf{x}_\alpha = \mathbf{R}_\alpha \mathbf{y}^\delta$. Consider the thresholds $\alpha \in \{1e-3, 1e-2, 5e-2, 8e-2, 1e-1\}$ to compute the regularized solutions $\mathbf{x}_\alpha$. Plot the solutions $\mathbf{x}_\alpha$ and compare them to the true solution $\mathbf{x}_{\text{true}}$. **Hint:** *A script to help you with this implementation is prbsets/deconv/scKerRecoTSVD1D.m.*

   b) Determine the (approximate) optimal value $\alpha_{\text{opt}} > 0$ of the regularization parameter $\alpha > 0$ for the Tikhonov regularization using the L-curve criterion. Compute the solution using the determined regularization parameter α_{opt} and compare it (visually) to the true solution $\mathbf{x}_{\text{true}}$. **Hint:** *A script to help with the implementation of the search for an optimal regularization parameter $\alpha_{\text{opt}} > 0$ based on the L-curve criterion is prbsets/deconv/scKerRecoTRegLC1D.m. A template for implementing a plot for the L-curve is core/evalLCurve.m.*

In [21]:

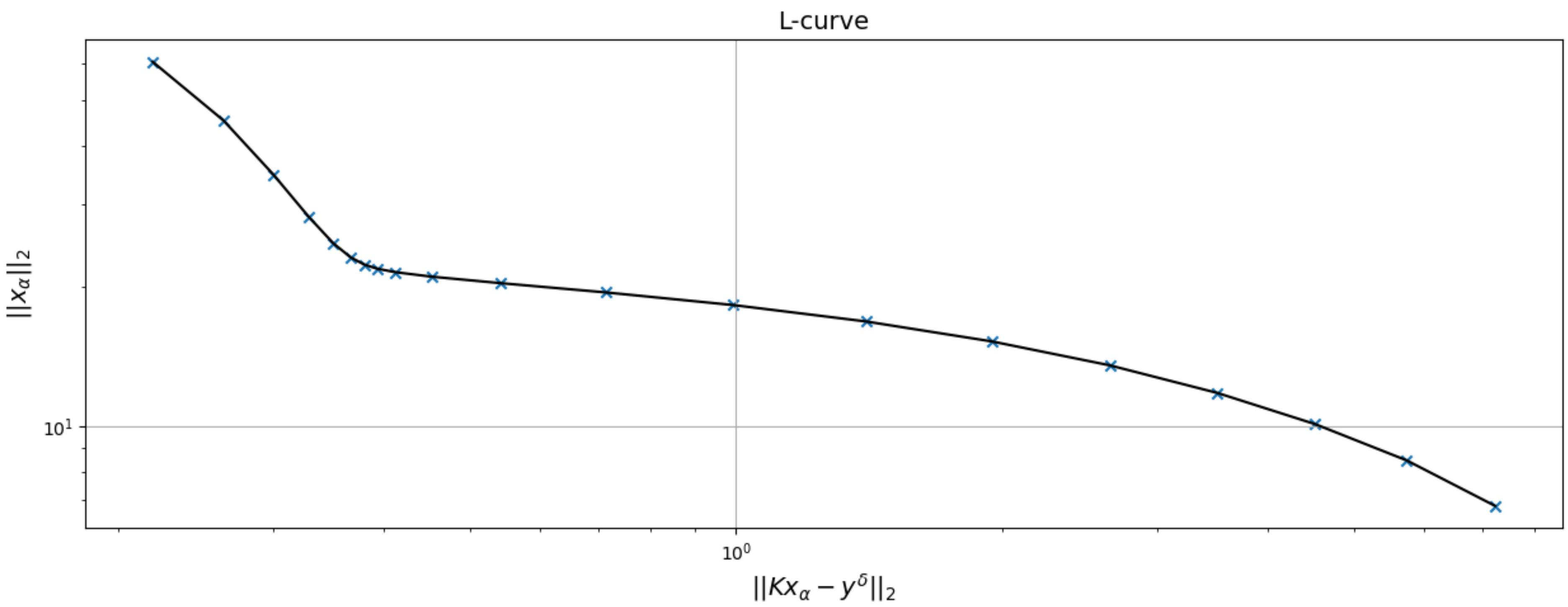
```
# statement (a)
reload(conv)
reload(cr)
conv.scKerRecoTSVD1D()
```



In [22]:

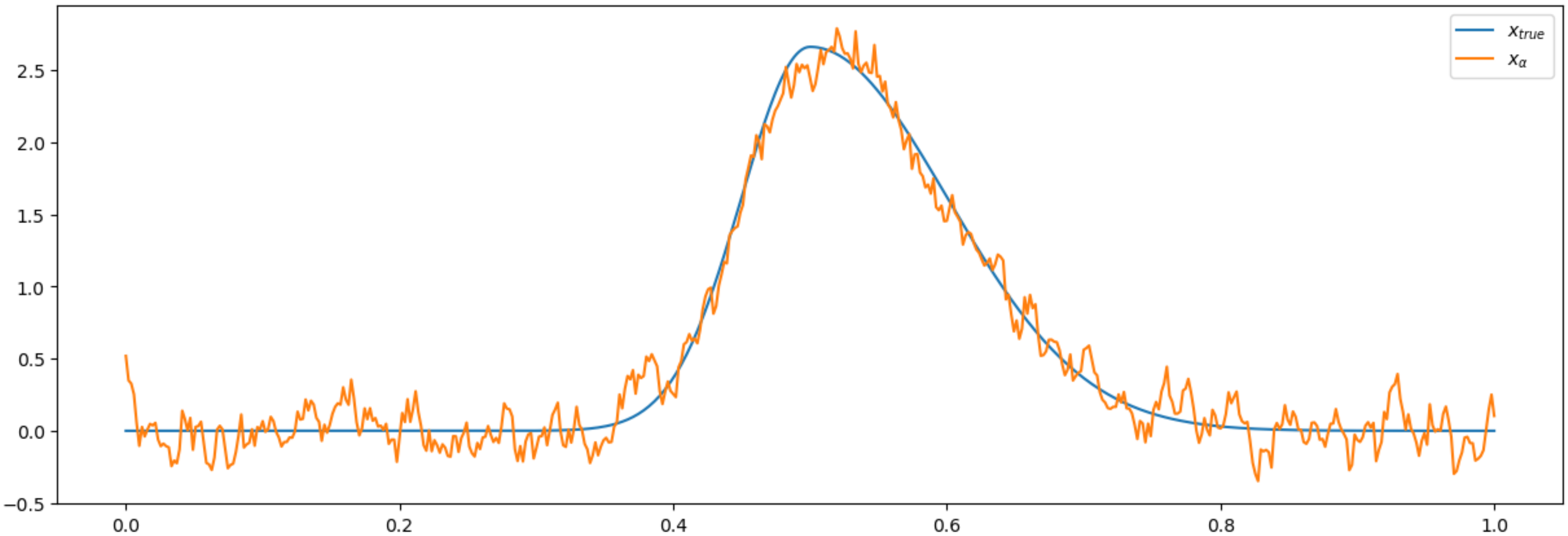
```
# statement (b)
reload(conv)
reload(cr)
conv.scKerRecoTRegLC1D()
```

```
alpha=1e-05:  ||r|| = 0.2189719660397398  ||x|| = 60.40405041797399
alpha=1.8329807108324375e-05:  ||r|| = 0.2632715091390495  ||x|| = 45.26479268222662
alpha=3.359818286283781e-05:  ||r|| = 0.29993368674657295  ||x|| = 34.644858408124136
alpha=6.158482110660267e-05:  ||r|| = 0.32872972099976555  ||x|| = 28.155735910243706
alpha=0.00011288378916846884:  ||r|| = 0.35061946392800336  ||x|| = 24.690908765273573
alpha=0.00020691380811147902:  ||r|| = 0.36707229858656426  ||x|| = 23.027696816890053
alpha=0.000379269019073225:  ||r|| = 0.38007850361406875  ||x|| = 22.24442833608155
alpha=0.0006951927961775605:  ||r|| = 0.3928934387068959  ||x|| = 21.804249336626963
alpha=0.0012742749857031334:  ||r|| = 0.4122665881354553  ||x|| = 21.427823897225466
alpha=0.002335721469090121:  ||r|| = 0.45291506917793545  ||x|| = 20.96108936303843
alpha=0.004281332398719391:  ||r|| = 0.5416662793328133  ||x|| = 20.3052772983905
alpha=0.007847599703514606:  ||r|| = 0.7126965146291593  ||x|| = 19.398602622365026
alpha=0.01438449888287663:  ||r|| = 0.9942305190949743  ||x|| = 18.223932398207065
alpha=0.026366508987303583:  ||r|| = 1.4031382574338471  ||x|| = 16.813647165678862
alpha=0.04832930238571752:  ||r|| = 1.950932008110032  ||x|| = 15.230817174683356
alpha=0.08858667904100823:  ||r|| = 2.650130484089408  ||x|| = 13.541544735262487
alpha=0.1623776739188721:  ||r|| = 3.507930647328891  ||x|| = 11.818951339289477
alpha=0.2976351441631319:  ||r|| = 4.526453719851698  ||x|| = 10.13328741132561
alpha=0.5455594781168515:  ||r|| = 5.74406783381342  ||x|| = 8.476527727386436
alpha=1.0:  ||r|| = 7.228976328601484  ||x|| = 6.776721286202993
```



In [23]:
```
#the best alpha should be?
alphalist = np.logspace(-5, 0, 20)
best_alpha = alphalist[7] # or 6 or 8
print('\nbest alpha =', best_alpha)

K = conv.getRecoMat1D(256)
x_true, s = conv.getRecoKernel1D(256)
y = K @ x_true
delta = np.linalg.norm(y) / (np.sqrt(256) * 50)
y_delta, noise = cr.addNoise(y, delta, return_noise=True)

plt.figure(figsize=(15, 5))
plt.plot(np.linspace(0, 1, 511), x_true, label=r'$x_{true}$')
plt.plot(np.linspace(0, 1, 511), np.linalg.lstsq(K.T@K + best_alpha*np.eye(511), K.T@y_delta, rcond=None)[0], lak
plt.legend();
```

```
best alpha = 0.0006951927961775605
```



In [ ]: