PyTikhonov documentation

Jonathan Lindbloom[*]

Last updated: December 2, 2025

Installation: `pip install pytikhonov`

The goal of this package is to provide a pure-Python, user-friendly interface to Tikhonov regularization (including regularization parameter selection) for problems of the form

$$x_\lambda \coloneqq \underset{x \in \mathbb{R}^N}{\arg\min} \; \|Ax - b\|_2^2 + \lambda \|Lx - d\|_2^2 = \underset{x \in \mathbb{R}^N}{\arg\min} \; \left\| \begin{bmatrix} A \\ \sqrt{\lambda}\, L \end{bmatrix} x - \begin{bmatrix} b \\ \sqrt{\lambda}\, d \end{bmatrix} \right\|_2, \tag{1}$$

where $A \in \mathbb{R}^{M \times N}$, $L \in \mathbb{R}^{K \times N}$, $b \in \mathbb{R}^M$, $d \in \mathbb{R}^K$, and $\lambda > 0$. Throughout, we assume that A and L satisfy the common-kernel condition

$$\operatorname{rank}\left(\begin{bmatrix} A \\ L \end{bmatrix} \right) = N, \quad \text{or, equivalently,} \quad \ker(A) \cap \ker(L) = \{0_N\}. \tag{2}$$

For classical Tikhonov regularization one commonly chooses $d = 0_K$, but the software also supports shifted problems with $d \neq 0_K$.

This package is primarily intended for small to moderate-scale problems (e.g., with N up to a few thousand) where A and L can be treated as dense matrices, with the main construct being the `TikhonovFamily`. For larger problems, the `ProjectedTikhonovFamily` interface is provided as a building block to be combined with iterative methods that generate low-dimensional bases (e.g., Krylov or model reduction techniques). The implementation relies on the generalized singular value decomposition (GSVD) of the pair (A, L), computed using `easygsvd`, as well as the notation for the GSVD described in the `easygsvd` documentation. Forming the GSVD costs $\mathcal{O}((M+K)N^2 + N^3)$ floating-point operations, but once it has been obtained, many quantities associated with the Tikhonov problem in Eq. (1) can be computed at significantly reduced cost. Much of the discussion in this documentation is based on [1, 2, 3, 4].

1 Defining a `TikhonovFamily`

The fundamental building block of this package is the `TikhonovFamily` object. Given A, L, b, and d, we represent the entire family of Tikhonov solutions $\{x_\lambda\}_{\lambda>0}$ as

```
tf = TikhonovFamily(A, L, b, d, gsvd=None, btrue=None, noise_var=None)
```

[*]Department of Mathematics, Dartmouth College, Hanover, NH, USA (jonathan@lindbloom.com).

If the GSVD of (A, L) has already been computed by `easygsvd`, it can be passed via the `gsvd` keyword argument. Otherwise, the GSVD of (A, L) is computed when the `TikhonovFamily` is instantiated.

Optionally, if the data follow the noise model

$$b = b_{\text{true}} + e, \qquad b_{\text{true}} = Ax_{\text{true}}, \qquad e \sim \mathcal{N}(0, \sigma^2 I_M),$$

and both b_{true} and the noise variance σ^2 are known, they can be supplied via `btrue` and `noise_var`, respectively. These quantities are used by various regularization parameter selection methods for λ and by the plotting and diagnostic functions.

2 Interfacing with `TikhonovFamily`

Once a `TikhonovFamily` has been defined, we can evaluate the Tikhonov solution x_λ for different values of λ. Under the GSVD of (A, L) (using the notation from the `easygsvd` documentation), the solution can be written as

$$x_\lambda = X_1 U_1^\top b + X_2 \operatorname{diag}\left(\frac{\gamma_i^2}{c_i(\gamma_i^2 + \lambda)} \right) U_2^\top b + X_2 \operatorname{diag}\left(\frac{\lambda}{s_i(\gamma_i^2 + \lambda)} \right) V_2^\top d + X_3 V_3^\top d, \quad (3)$$

where $X_1, X_2, X_3, U_1, U_2, V_2, V_3$, and the generalized singular values γ_i and cosines/sines c_i, s_i are defined as in the `easygsvd` documentation.

In code, we evaluate x_λ as follows:

```
# For a single lambda
lambdah = 1e1
x_lambdah = tf.solve(lambdah)

# For a whole range of lambdas
lambdahs = np.asarray([1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4])
X_lambdah = tf.solve(lambdahs)
```

An example of this query for the one-dimensional deblurring problem is shown in Figure 1, where A is selected as a Gaussian blurring operator and L is a discrete derivative operator.

Reciprocal parameterization

Sometimes, the Tikhonov problem is parameterized in terms of the reciprocal regularization parameter $\beta = 1/\lambda$, i.e.,

$$x_\beta := \underset{x \in \mathbb{R}^N}{\arg\min} \, \|Ax - b\|_2^2 + \beta^{-1} \|Lx - d\|_2^2 = \underset{x \in \mathbb{R}^N}{\arg\min} \left\| \begin{bmatrix} A \\ \beta^{-1/2} L \end{bmatrix} x - \begin{bmatrix} b \\ \beta^{-1/2} d \end{bmatrix} \right\|_2. \quad (4)$$

Most functions in `PyTikhonov` accept the keyword argument `reciprocate`, which can be used to switch to the parameterization in (4):

```
# For a single beta
beta = 1e1
x_beta = tf.solve(beta, reciprocate=True)

# For a whole range of betas
betas = np.asarray([1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4])
X_beta = tf.solve(betas, reciprocate=True)
```

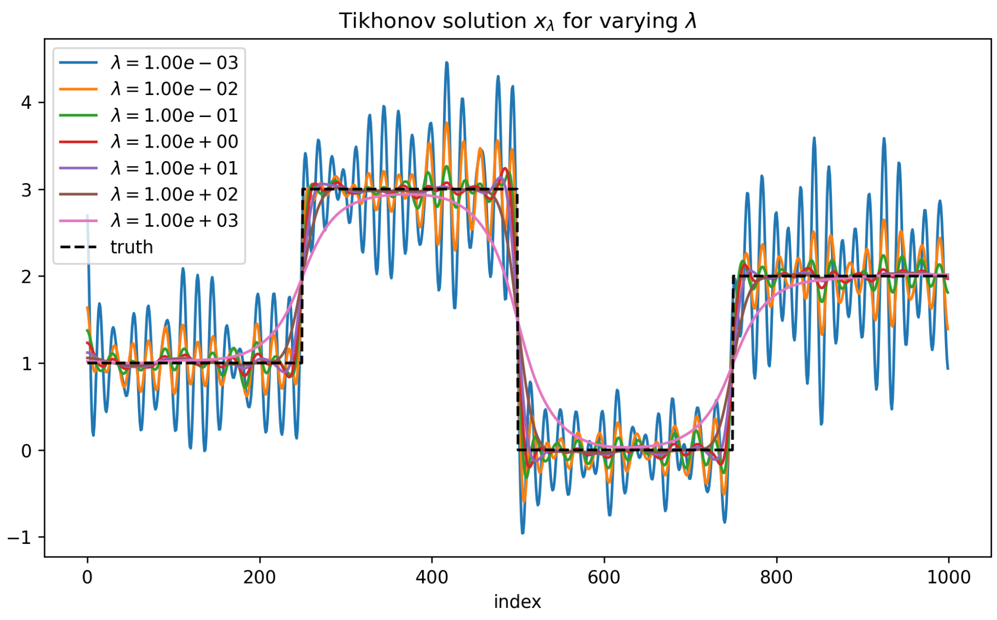

Figure 1: Family of Tikhonov solutions x_λ for the 1D deblurring problem.

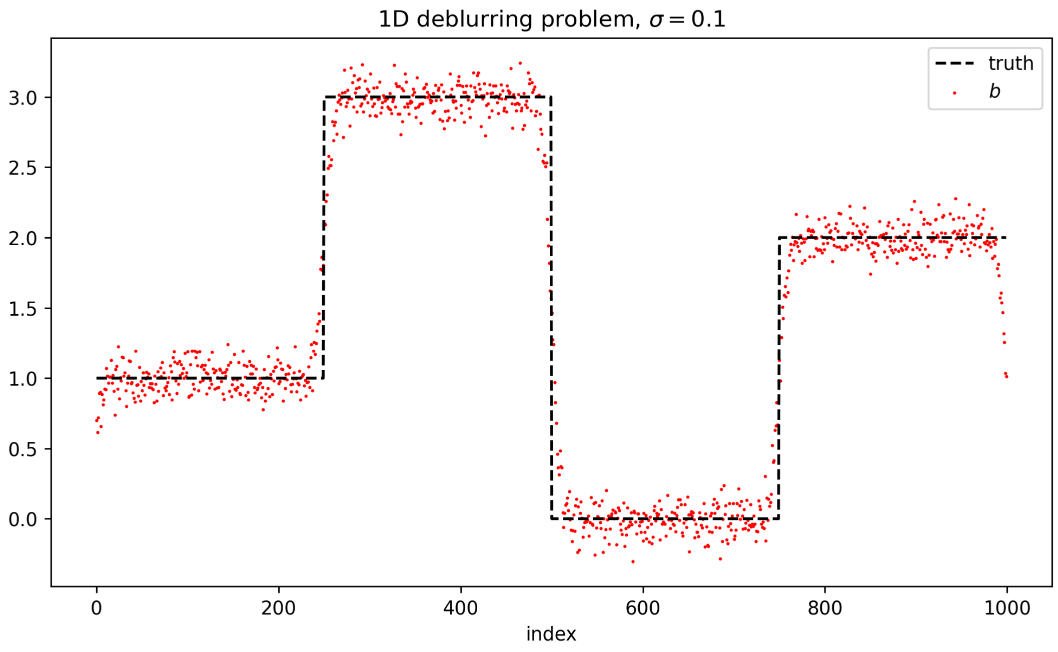

Figure 2: 1D deblurring toy problem.

Data fidelity and regularization terms

In addition to the Tikhonov solution x_λ itself, we often want the data fidelity and regularization terms

$$\left\|Ax_\lambda - b\right\|_2^2 \quad \text{and} \quad \left\|Lx_\lambda - d\right\|_2^2$$

for a range of regularization parameters. These are queried as:

```python
# Evaluate data fidelity and regularization term for varying lambdas
data_fidelities = tf.data_fidelity(lambdahs)
regularization_terms = tf.regularization_term(lambdahs)
```

Using the GSVD of (A, L), these quantities can be written as

$$\left\|Ax_\lambda - b\right\|_2^2 = \sum_{i=n_L+1}^{r_A} \left(\frac{\lambda}{\gamma_i^2 + \lambda}\right)^2 \left(u_i^\top b - \gamma_i v_i^\top d\right)^2 + \left\|(I_M - \hat{U}\hat{U}^\top)b\right\|_2^2, \tag{5}$$

$$\left\|Lx_\lambda - d\right\|_2^2 = \sum_{i=n_L+1}^{r_A} \left(\frac{\gamma_i^2}{\gamma_i^2 + \lambda}\right)^2 \frac{\left(u_i^\top b - \gamma_i v_i^\top d\right)^2}{\gamma_i^2} + \left\|(I_K - \hat{V}\hat{V}^\top)d\right\|_2^2, \tag{6}$$

where $\hat{U}$ and $\hat{V}$ collect the relevant GSVD basis vectors. A natural use-case of these formulas is to inspect the behavior of the L-curve (see Section 3.2).

Derivatives with respect to the regularization parameter

`PyTikhonov` can also evaluate derivatives of the data fidelity and regularization terms with respect to the regularization parameter. For $k \geq 1$, we have

$$\frac{d^k}{d\lambda^k}\left\|Ax_\lambda - b\right\|_2^2 = \sum_{i=n_L+1}^{r_A} \frac{(-1)^k k!\, \gamma_i^2 \left(\gamma_i^2(k-1) - 2\lambda\right)}{(\gamma_i^2 + \lambda)^{k+2}} \left(u_i^\top b - \gamma_i v_i^\top d\right)^2, \tag{7}$$

$$\frac{d^k}{d\lambda^k}\left\|Lx_\lambda - d\right\|_2^2 = \sum_{i=n_L+1}^{r_A} \frac{(-1)^k (k+1)!\, \gamma_i^2}{(\gamma_i^2 + \lambda)^{k+2}} \left(u_i^\top b - \gamma_i v_i^\top d\right)^2. \tag{8}$$

With respect to the reciprocal parameterization $\beta = 1/\lambda$, the corresponding derivatives are

$$\frac{d^k}{d\beta^k}\left\|Ax_\beta - b\right\|_2^2 = \sum_{i=n_L+1}^{r_A} \frac{(-1)^k (k+1)!\, \gamma_i^{2k}}{(1 + \beta\gamma_i^2)^{k+2}} \left(u_i^\top b - \gamma_i v_i^\top d\right)^2, \tag{9}$$

$$\frac{d^k}{d\beta^k}\left\|Lx_\beta - d\right\|_2^2 = \sum_{i=n_L+1}^{r_A} \frac{(-1)^k k!\, (\gamma_i^2)^{k-1} \left(k - 1 - 2\beta\gamma_i^2\right)}{(1 + \beta\gamma_i^2)^{k+2}} \left(u_i^\top b - \gamma_i v_i^\top d\right)^2. \tag{10}$$

In `PyTikhonov`, these derivatives are queried as follows:

```python
k = ...  # derivative order

# Derivatives in terms of lambdas
data_fidelity_derivs = tf.data_fidelity_derivative(
    lambdahs, order=k, reciprocate=False
)
```

```python
regularization_term_derivs = tf.regularization_term_derivative(
    lambdahs, order=k, reciprocate=False
)

# Derivatives in terms of betas
data_fidelity_derivs_beta = tf.data_fidelity_derivative(
    betas, order=k, reciprocate=True
)
regularization_term_derivs_beta = tf.regularization_term_derivative(
    betas, order=k, reciprocate=True
)
```

These expressions are validated against alternative expressions obtained via matrix calculus (see Appendix A).

3 Diagnostics

In addition to computing Tikhonov solutions, `PyTikhonov` provides several diagnostic tools that help assess the underlying inverse problem, the noise level, and the suitability of the chosen regularization operator.

3.1 Discrete Picard condition

The discrete Picard condition (DPC) is an assumption on the inverse problem that must be satisfied if we expect Tikhonov regularization to produce a meaningful solution. Specifically, the DPC states that, on average, the generalized Fourier coefficients $|u_i^\top b_{\text{true}}|$ decay to zero faster than the generalized singular values γ_i for indices $i = n_L + 1, \dots, r_A$. More information about the DPC can be found in [4].

When b_{true} is replaced by the noisy data b, we expect the coefficients $|u_i^\top b|$ to initially follow the same decay as $|u_i^\top b_{\text{true}}|$, and then to level off at a value determined by the noise variance σ^2. In `PyTikhonov`, we provide a visual check of the DPC via the Picard plot:

```python
# Generate the Picard plot
plot_picard(tf, plot_path=None)
```

If b_{true} and σ^2 were passed to the `TikhonovFamily` at instantiation, the Picard plot additionally shows the expected value $\mathbb{E}[|u_i^\top b|] = \sigma\sqrt{2/\pi}$ and an approximate 99% confidence interval $(0.0063\,\sigma, 2.807\,\sigma)$ for $|u_i^\top b|$, which we expect the noise-dominated coefficients to lie within for sufficiently large i. The Picard plot for the 1D deblurring problem is shown in Figure 3.

3.2 L-curve

Another useful diagnostic is the shape of the L-curve. Let

$$\rho_\lambda := \|Ax_\lambda - b\|_2^2, \quad \eta_\lambda := \|Lx_\lambda - d\|_2^2, \quad \hat{\rho}_\lambda := \log \rho_\lambda, \quad \hat{\eta}_\lambda := \log \eta_\lambda.$$

We define the L-curve as the parametric curve with points $(\hat{\rho}_\lambda/2, \hat{\eta}_\lambda/2)$ for $\lambda > 0$, which corresponds to plotting

$$\big(\log \|Ax_\lambda - b\|_2, \log \|Lx_\lambda - d\|_2\big)$$

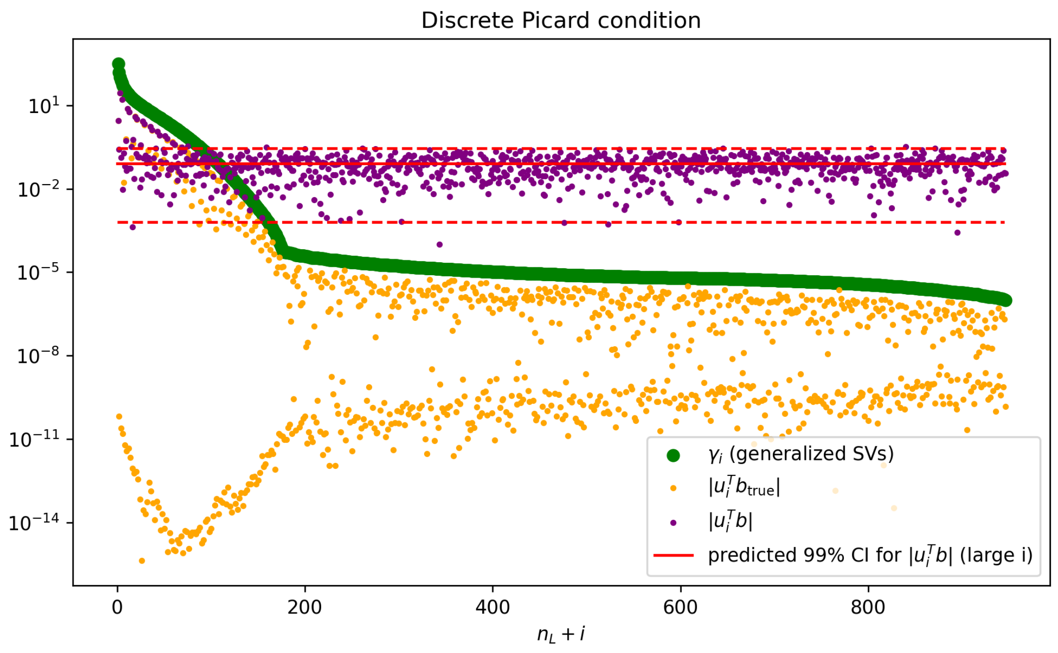

Figure 3: Picard plot for the 1D deblurring problem.

in log–log scale.

In `PyTikhonov`, these points are computed as:

```
lambdahs = np.logspace(-10, 10, num=1000, base=10)
rho_hat_half, eta_hat_half = tf.lcurve(lambdahs)
```

The L-curve for the 1D deblurring problem is shown in Figure 4. When there is a meaningful benefit from regularization, the L-curve typically exhibits a distinctive "L" shape in log–log scale. The lack of such an "L" may indicate an issue with the underlying assumptions of the problem (e.g., negligible noise, extremely high noise, or a mismatch between the choice of L and the true prior structure). For illustration, the L-curves for the 1D deblurring problem with b replaced by either b_{true} (pure signal) or e (pure noise) are shown in Figure 5.

3.3 Monitoring function

A final diagnostic tool we provide is the monitoring function $\mathcal{V}(\lambda)$ defined by

$$\mathcal{V}(\lambda) = \frac{\|Ax_\lambda - b\|_2^2}{\mathcal{T}(\lambda)}, \qquad \mathcal{T}(\lambda) := \operatorname{trace}\big(I_M - A(A^\top A + \lambda L^\top L)^{-1} A^\top\big), \tag{11}$$

where $\mathcal{T}(\lambda)$ is the *degrees of freedom* of the Tikhonov estimator for a given λ. In terms of the GSVD, $\mathcal{T}(\lambda)$ can be expressed as

$$\mathcal{T}(\lambda) = M - n_L - \sum_{i=n_L+1}^{r_A} \frac{\gamma_i^2}{\gamma_i^2 + \lambda} = M - r_A + \sum_{i=n_L+1}^{r_A} \frac{\lambda}{\gamma_i^2 + \lambda}. \tag{12}$$

We expect the graph of $(\lambda^{-1}, \mathcal{V}(\lambda))$ to exhibit an initial plateau, followed by a drop to a secondary plateau whose level is approximately the noise variance σ^2, and then

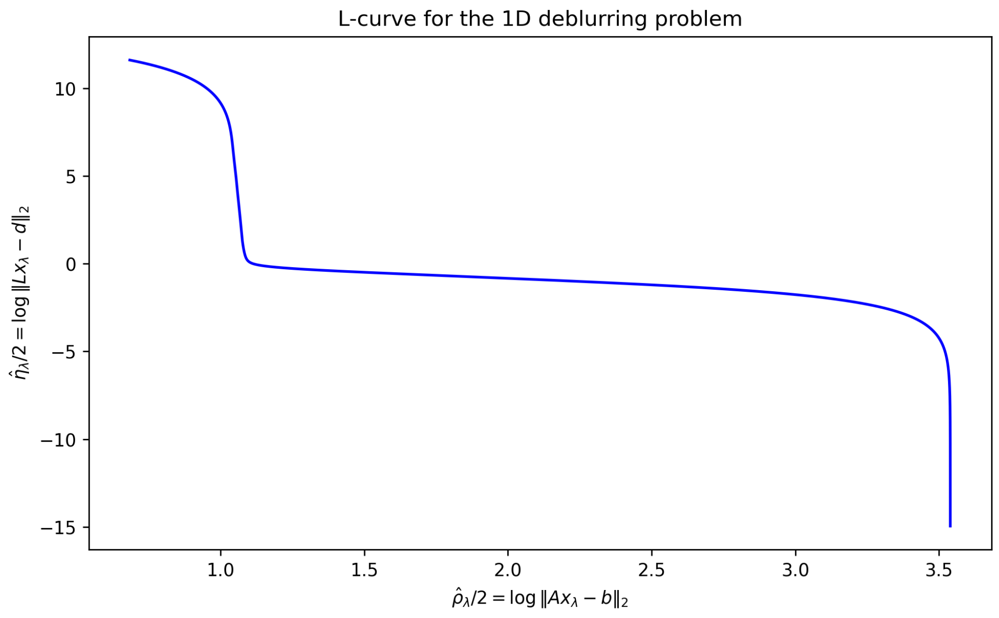

Figure 4: L-curve for the 1D deblurring problem.

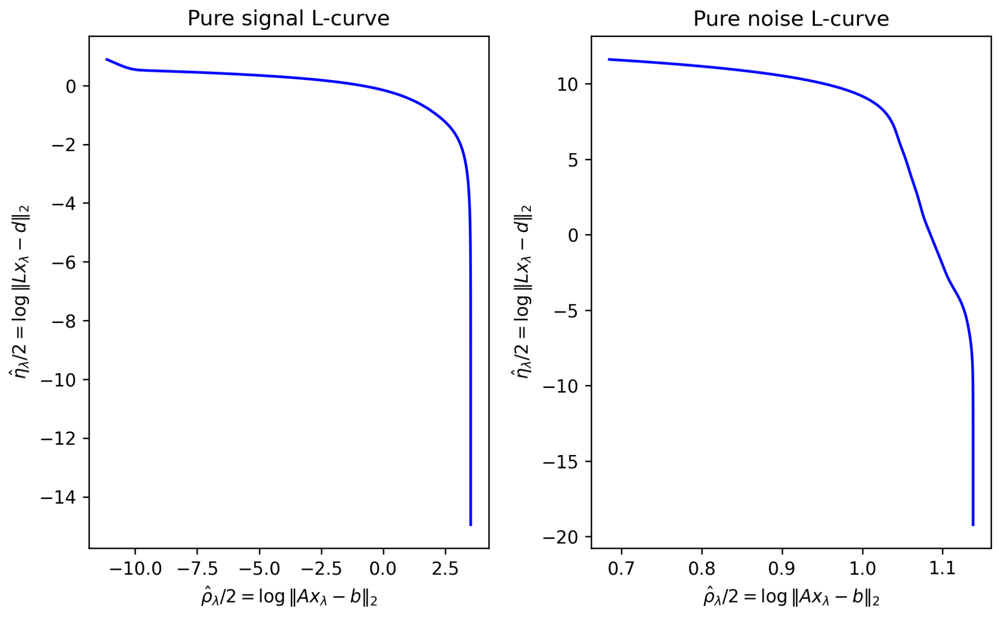

Figure 5: Pure-signal and pure-noise L-curves for the 1D deblurring problem. If the observed L-curve closely resembles either of these extremes, there may be an issue with the underlying assumptions of the problem.

additional behavior beyond this region. The value of the secondary plateau can therefore be used to estimate the noise variance when it is unknown.

`PyTikhonov` provides convenience functions to plot the monitoring function and to estimate the noise variance from it:

```python
# Plot the monitoring function; pass plot_path to save as a PNG
plot_monitoring_function(tf, plot_path=None)

# Estimate the noise variance from the monitoring function
noise_var_est = estimate_noise_variance(tf)
```

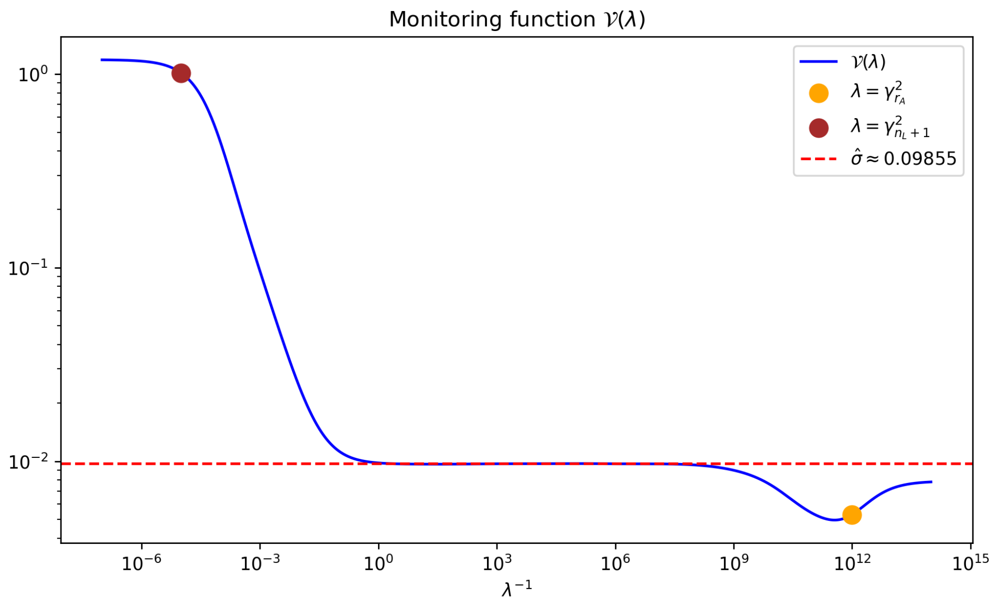

Figure 6: The monitoring function $\mathcal{V}(\lambda)$ for the 1D deblurring problem.

4 Regularization parameter selection methods

`PyTikhonov` provides access to three heuristics for selecting the regularization parameter λ and its corresponding regularized solution x_λ: the L-corner, discrepancy principle (DP), and generalized cross validation (GCV) methods. Other parameter selection strategies can be implemented directly using the `TikhonovFamily` interface; see, for example, the discussions in [1, 2].

4.1 L-corner

The L-corner heuristic selects λ as the value corresponding to the "corner" of the L-curve, which can be identified as the point of maximum curvature. See [5] for an analysis of the L-curve in terms of the GSVD. In `PyTikhonov`, the L-corner method is encapsulated in Heuristic 1.

Heuristic 1: L-corner

Let

$$\rho = \|Ax_\lambda - b\|_2^2, \quad \hat{\rho} = \log(\rho), \quad \eta = \|Lx_\lambda - d\|_2^2, \quad \hat{\eta} = \log(\eta), \tag{13}$$

where x_λ is given by Eq. (1). Define the corresponding L-curve as the curve $(\hat{\rho}/2, \hat{\eta}/2)$ parameterized by $\lambda \in \mathbb{R}_{++}$. Then select λ as the point which maximizes the curvature of the L-curve, given by

$$\kappa(\lambda) = 2\,\frac{\hat{\rho}'\hat{\eta}'' - \hat{\rho}''\hat{\eta}'}{((\hat{\rho}')^2 + (\hat{\eta}')^2)^{3/2}}. \tag{14}$$

Here we have dropped the subscripts on ρ and η that indicate the dependence on λ. The derivatives of the log-quantities obey

$$\hat{\eta}' = \eta'/\eta, \qquad \hat{\rho}' = \rho'/\rho, \qquad \hat{\eta}'' = \frac{\eta\eta'' - (\eta')^2}{\eta^2}, \qquad \hat{\rho}'' = \frac{\rho\rho'' - (\rho')^2}{\rho^2}. \tag{15}$$

All of these derivatives can be computed using the derivative interface described in Section 2.

In PyTikhonov, we provide the function lcorner, which implements an optimization routine for locating the corner of the L-curve, as well as a convenience plotting function plot_lcorner for visualizing the result:

```python
# Select the regularization parameter via the L-corner heuristic
lcorner_data = lcorner(tf)

# Visualize the result
plot_lcorner(lcorner_data, plot_path=None)
```

The result of lcorner for the 1D deblurring problem is shown in Figure 7.

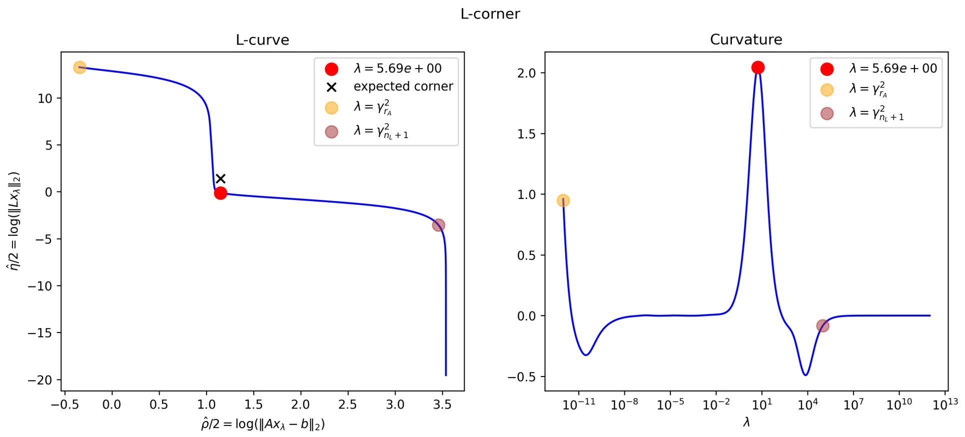

Figure 7: The result of lcorner for the 1D deblurring problem.

Some useful limiting values for the L-curve are

$$\lim_{\lambda\to 0^+} \rho = \left\|(I_M - \hat{U}\hat{U}^\top)b\right\|_2^2, \tag{16}$$

$$\lim_{\lambda\to\infty} \rho = \sum_{i=n_L+1}^{r_A} \left(u_i^\top b - \gamma_i v_i^\top d\right)^2 + \left\|(I_M - \hat{U}\hat{U}^\top)b\right\|_2^2, \tag{17}$$

$$\lim_{\lambda\to 0^+} \eta = \sum_{i=n_L+1}^{r_A} \left(\gamma_i^{-1} u_i^\top b - v_i^\top d\right)^2 + \left\|(I_K - \hat{V}\hat{V}^\top)d\right\|_2^2, \tag{18}$$

$$\lim_{\lambda\to\infty} \eta = \left\|(I_K - \hat{V}\hat{V}^\top)d\right\|_2^2, \tag{19}$$

as well as the fact that the L-corner is expected to appear near the point

$$\left(\frac{1}{2}\log\big((M-n_L)\sigma^2\big),\ \frac{1}{2}\log\left(\left\|(I_K - \hat{V}\hat{V}^\top)d\right\|_2^2 + \sum_{i=n_L+1}^{r_A} \frac{\left(u_i^\top b_{\text{true}} - \gamma_i v_i^\top d\right)^2}{\gamma_i^2}\right)\right) \tag{20}$$

on the graph of $(\hat{\rho}/2, \hat{\eta}/2)$.

4.2 Discrepancy principle (DP)

The DP heuristic is based on the observation that

$$\mathbb{E}\big[\|Ax_{\text{true}} - b\|_2^2\big] = M\sigma^2 \tag{21}$$

when the noise is distributed according to $e \sim \mathcal{N}(0, \sigma^2 I_M)$. Replacing x_{true} with x_λ in Eq. (21) leads to the following root-finding problem for λ.

Heuristic 2: Discrepancy principle

Select λ as a root of $\varphi(\lambda) : \mathbb{R}_{++} \to \mathbb{R}$ given by

$$\varphi(\lambda) = \|Ax_\lambda - b\|_2^2 - \tau^2\delta^2, \tag{22}$$

where $\delta^2 = M\sigma^2$, $\tau^2 \approx 1$ is a safeguard parameter, and x_λ is given in Eq. (1).

Unlike the L-corner heuristic, the DP heuristic requires knowledge of the noise variance σ^2. If the noise variance is unknown, one may attempt to estimate it using the monitoring function method described in Section 3.3. In terms of the L-curve $(\rho_\lambda, \eta_\lambda)$, the DP heuristic can be interpreted as selecting the λ corresponding to the intersection of the L-curve with the vertical line at $\frac{1}{2}\log(M\tau^2\sigma^2)$. Comparing this with the expected L-corner in Eq. (20), we see that Heuristic 2 should select a point slightly to the right of the expected L-corner. This is visualized for the 1D deblurring problem in Figure 8.

In `PyTikhonov`, we provide the function `discrepancy_principle`, which uses Newton's method (via `scipy.optimize.newton`) to solve the root-finding problem in Heuristic 2, and the function `plot_dp` for visualizing the result. The root-finder is applied to $\psi(\beta) \coloneqq \varphi(\beta^{-1})$ (using the reciprocal parameterization), since according to Eq. (10) we

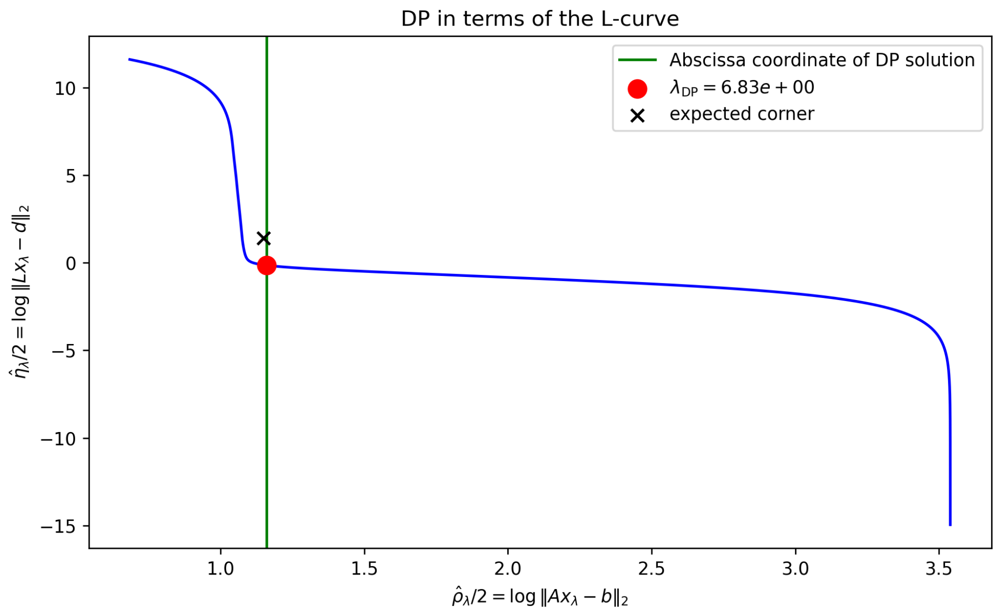

Figure 8: The DP heuristic in terms of the L-curve, for the 1D deblurring problem.

have

$$\psi'(\beta) = -\sum_{i=n_L+1}^{r_A} \frac{2\gamma_i^2}{(1+\beta\gamma_i^2)^3}\left(u_i^\top b - \gamma_i v_i^\top d\right)^2, \tag{23}$$

$$\psi''(\beta) = \sum_{i=n_L+1}^{r_A} \frac{6\gamma_i^4}{(1+\beta\gamma_i^2)^4}\left(u_i^\top b - \gamma_i v_i^\top d\right)^2, \tag{24}$$

so that $\psi(\beta)$ is strictly decreasing and convex. If $\psi > 0$ as $\beta \to 0^+$ and $\psi < 0$ as $\beta \to \infty$, then there is a unique root on $\mathbb{R}_{++}$, and Newton's method started to the left of the root is guaranteed to converge to it monotonically (e.g., see [6]). These sign conditions can be expressed compactly as

$$\left\|(I_M - \hat{U}\hat{U}^\top)b\right\|_2^2 \;\le\; \tau^2\delta^2 \;\le\; \left\|b - AX_1U_1^\top b\right\|_2^2. \tag{25}$$

If this condition is violated, then `discrepancy_principle` raises a warning and exits with a default value $\lambda = 10^{-12}$. The nonexistence of a root indicates that the regularization operator L may be ill-suited for the inversion.

Our implementation of `discrepancy_principle` initializes Newton's method at $\beta_0 = 0$ and makes use of the limiting values

$$\lim_{\beta\to 0^+} \psi(\beta) = \sum_{i=n_L+1}^{r_A} \left(u_i^\top b - \gamma_i v_i^\top d\right)^2 + \left\|(I_M - \hat{U}\hat{U}^\top)b\right\|_2^2 - \tau^2\delta^2, \tag{26}$$

$$\lim_{\beta\to\infty} \psi(\beta) = \left\|(I_M - \hat{U}\hat{U}^\top)b\right\|_2^2 - \tau^2\delta^2, \tag{27}$$

since numerical problems may occur if the case $\beta = 0$ is not treated separately. The result of `discrepancy_principle` for the 1D deblurring problem is shown in Figure 9.

```
# Select the regularization parameter via the DP heuristic
```

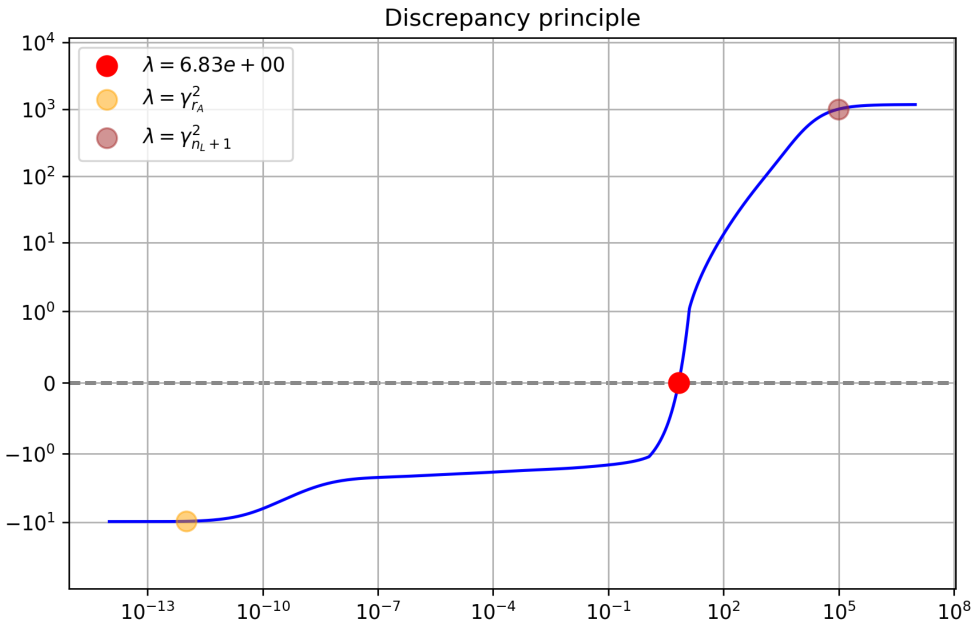

Figure 9: The result of `discrepancy_principle` for the 1D deblurring problem.

```python
dp_data = discrepancy_principle(tf)

# Visualize the result
plot_dp(dp_data, plot_path=None)
```

4.3 Generalized cross validation (GCV)

Generalized cross validation (GCV) is a parameter-choice rule that approximates leave-one-out cross validation using only a single Tikhonov solution for each λ, and, unlike the discrepancy principle, it does not require prior knowledge of the noise variance.

Heuristic 3: Generalized cross validation

Select λ as a global minimizer of

$$\mathrm{GCV}(\lambda) = \frac{\mathcal{V}(\lambda)}{\mathcal{T}(\lambda)} = \frac{\|Ax_\lambda - b\|_2^2}{\left(\mathrm{trace}(I_M - A(A^\top A + \lambda L^\top L)^{-1}A^\top)\right)^2} \tag{28}$$

on $\mathbb{R}_{++}$, where x_λ is given in Eq. (1) and $\mathcal{V}$, $\mathcal{T}$ are as in Eq. (11).

In `PyTikhonov`, we provide the function `gcvmin` for computing the minimizer, as well as `plot_gcv` for visualizing the result:

```python
# Select the regularization parameter via the GCV heuristic
gcv_data = gcvmin(tf)

# Visualize the result
```

```
plot_gcv(gcv_data, plot_path=None)
```

The result of `gcvmin` for the 1D deblurring problem is shown in Figure 10. It is well known that the GCV objective function may be very flat near its minimizer, which can make it difficult to identify the minimizing value accurately. In our implementation, we use `scipy.optimize.fminbound` to seek the minimum, using a logarithmic transformation for both λ and the objective to help rescale the problem.

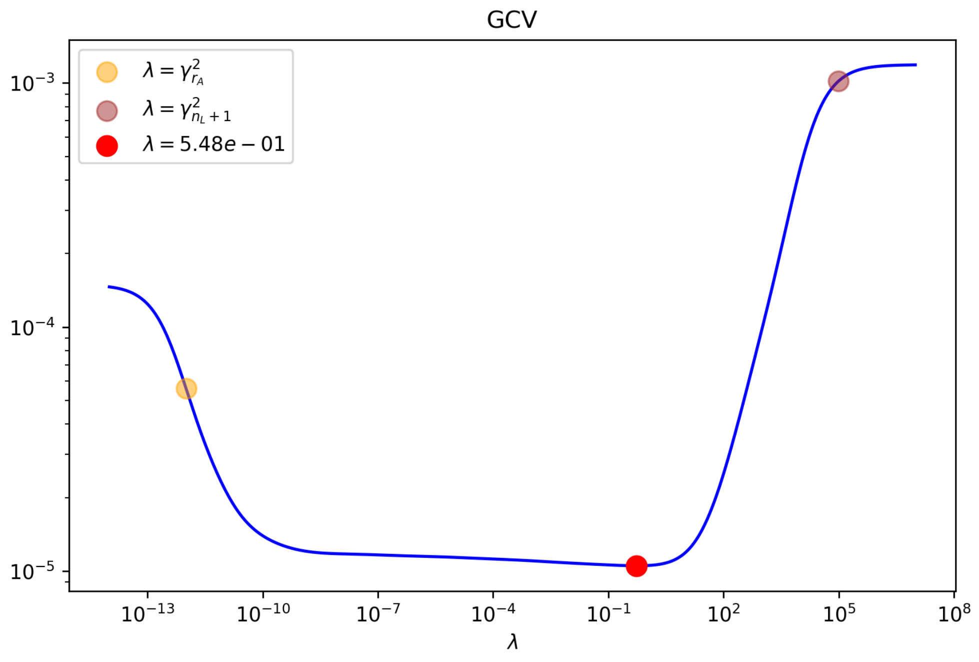

Figure 10: The result of `gcvmin` for the 1D deblurring problem.

4.4 Comparing the methods

To help compare all methods for a given problem, we provide a utility function which applies all implemented regularization parameter heuristics to the problem. The result can be visualized in terms of the L-curve using `plot_all_methods`. The result for the 1D deblurring problem is shown in Figure 11.

```
# Run all methods
all_data = all_regparam_methods(tf)

# Plot the result
plot_all_methods(all_data, plot_path=None)
```

4.5 Examining robustness of the methods

To gauge the robustness of the regularization parameter selection methods, we provide the functions `rand_dp`, `rand_lcorner`, and `rand_gcvmin`. These methods require the `TikhonovFamily` object to be instantiated with the noiseless right-hand side b_{true} and the

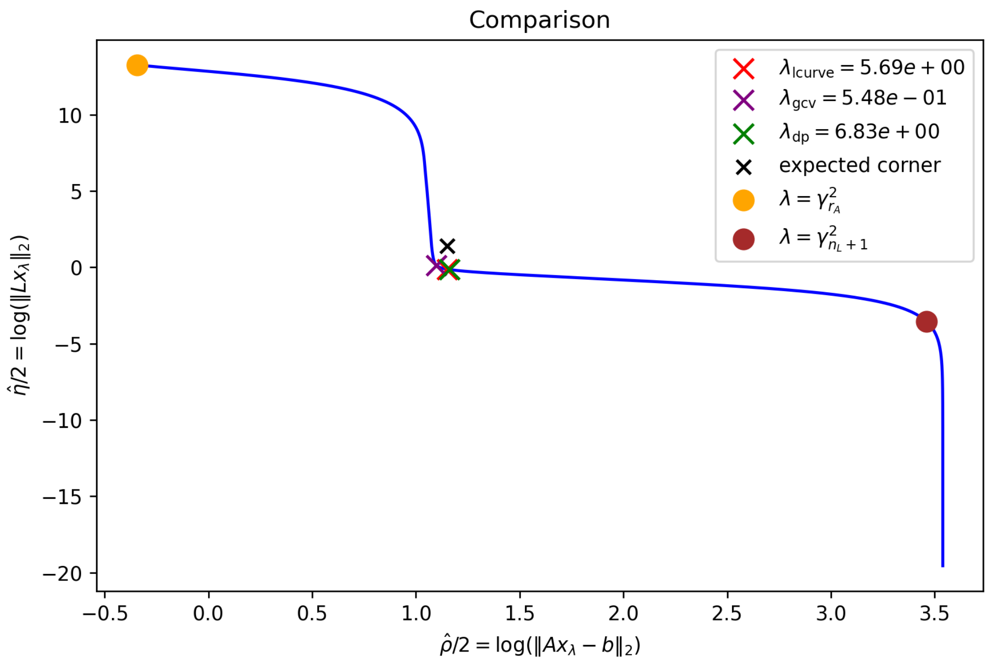

Figure 11: The result of `all_regparam_methods` for the 1D deblurring problem.

noise variance σ^2. Each function repeatedly simulates noisy right-hand sides according to

$$b = b_{\text{true}} + e, \qquad e \sim \mathcal{N}(0, \sigma^2 I_M),$$

solves the corresponding parameter selection problem for each sample, and records the resulting regularization parameters.

```
# Run randomizations and select regularization parameters
dp_lambdahs, dp_e_norms = rand_dp(tf, n_samples=10000)
lcorner_lambdahs, lcorner_e_norms = rand_lcorner(tf, n_samples=1000)
gcvmin_lambdahs, gcvmin_e_norms = rand_gcvmin(tf, n_samples=1000)
```

5 Iteratively reweighted least-squares

A simple application of `PyTikhonov` is to perform regularization parameter selection inside an iteratively reweighted least-squares (IRLS) algorithm. For example, consider the ℓ_1-regularized (total variation) problem

$$\arg\min_{x \in \mathbb{R}^N} \|Ax - b\|_2^2 + \lambda \|Lx\|_1. \tag{29}$$

This can be approximated by the sequence of weighted Tikhonov problems

$$x^{k+1} = \arg\min_{x \in \mathbb{R}^N} \|Ax - b\|_2^2 + \lambda \|W(x^k)Lx\|_2^2, \tag{30}$$

where $W(x^k) = \text{diag}(w(x^k))$ with

$$[w(x^k)]_i = \left((Lx^k)_i^2 + \epsilon^2\right)^{-1/4}.$$

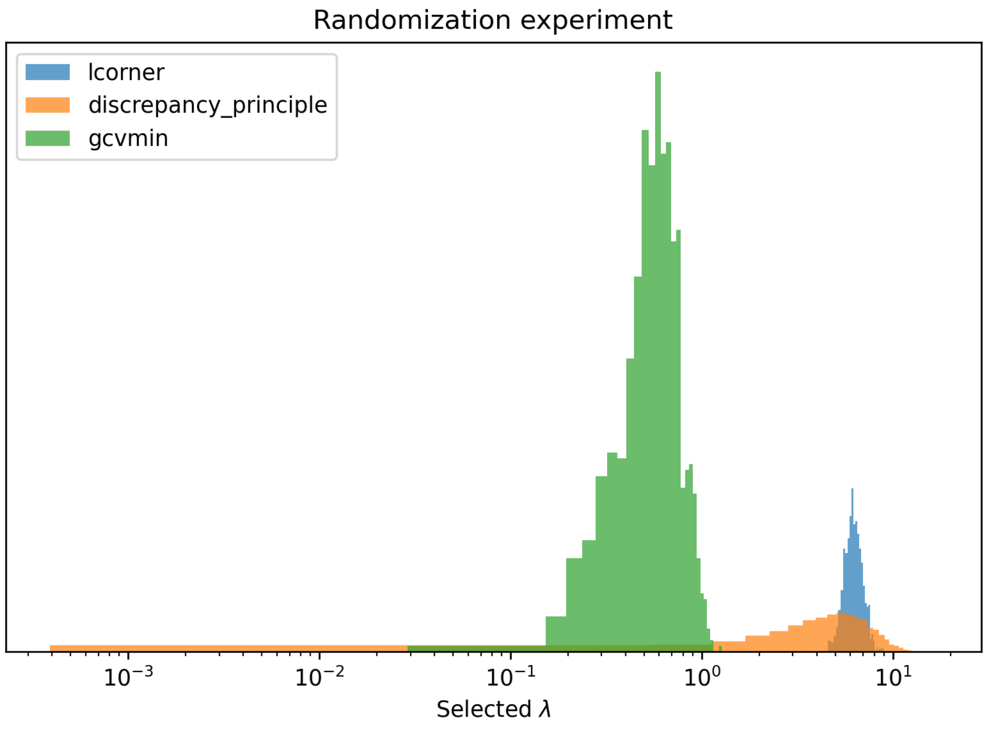

Figure 12: Randomization experiment for the 1D deblurring problem.

Note that with this choice, we have

$$\left\| W(x^k) L x \right\|_2^2 = \sum_i w_i(x^k)^2 (Lx)_i^2 \approx \sum_i \sqrt{(Lx)_i^2 + \epsilon^2},$$

which is a smooth approximation of the ℓ_1-norm.

In PyTikhonov, this can be implemented as follows:

```python
# Define weighting scheme
epsilon = 1e-3
l1_weighting_rule = lambda Lx: (Lx**2 + epsilon**2)**(-1.0 / 4.0)

# Initial guess and weights
x_irls = b.copy()
w_irls = l1_weighting_rule(L @ x_irls)

# Run IRLS iteration
n_iters = 20
lambdah_history = []

for j in range(n_iters):

    # Form weighted regularization operator
    L_weighted = np.diag(w_irls) @ L

    # Solve weighted Tikhonov problem and select lambda via L-corner
    tf_irls = TikhonovFamily(A, L_weighted, b)
    ldata = lcorner(tf_irls)

    # Store the selected lambda (optional)
    lambdah_history.append(ldata["lambdah"])

    # Update solution and weights
    x_irls = ldata["x_lambdah"]
    w_irls = l1_weighting_rule(L @ x_irls)
```

The resulting IRLS reconstruction for the 1D deblurring problem is shown in Figure 13.

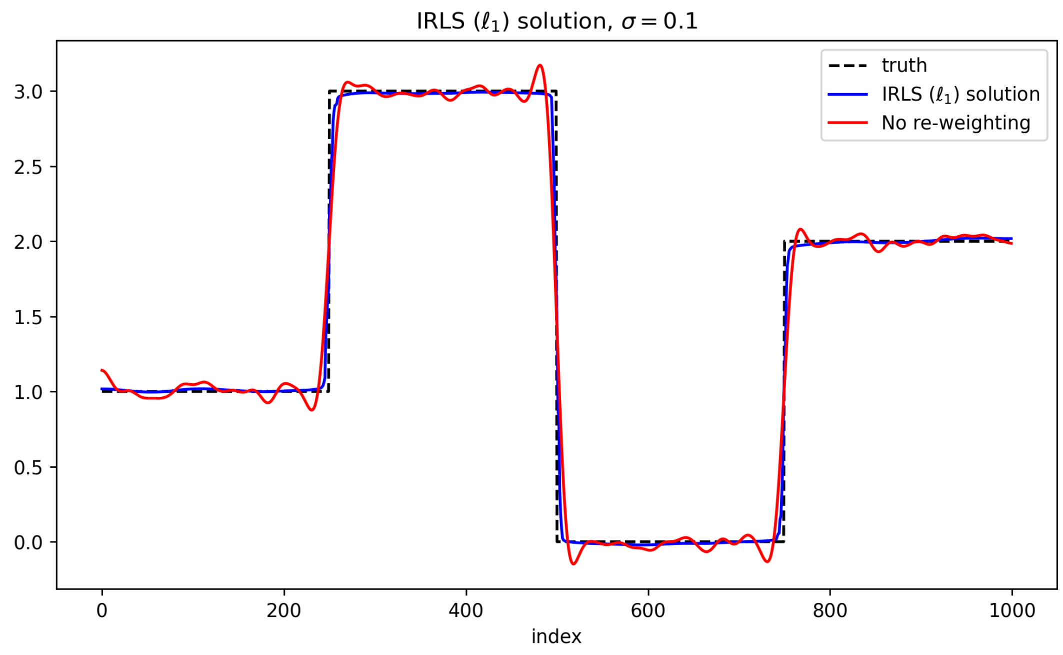

Figure 13: IRLS solution for the 1D deblurring problem.

6 Defining a `ProjectedTikhonovFamily`

The next major use of `PyTikhonov` is to define `ProjectedTikhonovFamily` objects. This interface is particularly intended for use with projection bases generated by Krylov and model reduction techniques (e.g., Golub–Kahan, Arnoldi, or POD). Given the Tikhonov problem Eq. (1), we define its projection onto the affine subspace $\underline{x} + \mathrm{col}(V)$ with $V \in \mathbb{R}^{N\times D}$ as

$$x_\lambda^{(\underline{x},V)} := \underset{x\in\underline{x}+\mathrm{col}(V)}{\arg\min} \; \|Ax - b\|_2^2 + \lambda \|Lx - d\|_2^2 \tag{31}$$

$$= \underline{x} + V z_\lambda^{(\underline{x},V)}, \tag{32}$$

where

$$z_\lambda^{(\underline{x},V)} := \underset{z\in\mathbb{R}^D}{\arg\min} \; \|AVz - \underbrace{(b - A\underline{x})}_{:=\underline{b}}\|_2^2 + \lambda \|LVz - \underbrace{(d - L\underline{x})}_{:=\underline{d}}\|_2^2. \tag{33}$$

In `PyTikhonov`, the projected family can be formed as:

```python
# Generate a projected Tikhonov family
ptf = ProjectedTikhonovFamily(
    A, L, b, d, V,
    x_under=None,
    b_under=None,    # b_under = b - A @ x_under, if already computed
    d_under=None,    # d_under = d - L @ x_under, if already computed
    AV=None,         # AV = A @ V, if already computed
    LV=None,         # LV = L @ V, if already computed
    btrue=None,
    noise_var=None
)
```

Here, A and L are typically taken to be implicit `scipy.sparse.linalg.LinearOperator` objects implementing matrix–vector products with these operators, and V is a dense array containing the projection basis (not required to have orthonormal columns). If a `TikhonovFamily` has already been formed, a `ProjectedTikhonovFamily` can be constructed from it using the `.project` method:

```python
# Project a TikhonovFamily onto x_under + col(V)
ptf = tf.project(V, x_under=None)
```

Interfacing with a `ProjectedTikhonovFamily` is very similar to `TikhonovFamily`. For example, a projected L-curve can be computed via:

```python
# Compute points along the projected L-curve
rho_hat_half, eta_hat_half = ptf.lcurve(lambdahs)
```

The projected L-curve for the 1D deblurring problem corresponding to $\underline{x} = 0_N$ and V given by the first twenty-five basis vectors generated by Golub–Kahan bidiagonalization (GKB) applied to (A, b) is shown in Figure 14, along with the L-curve for the full-scale problem. Note that the projected L-curve for any projection will lie on or above the L-curve for the full-scale problem, but may otherwise look quite different and may not exhibit an "L" shape (even when the full-scale problem does).

Regularization parameter selection can be performed exactly as with a `TikhonovFamily`:

```python
# Select regularization parameter via L-curve
lcorner_data = lcorner(ptf)

# Select regularization parameter via discrepancy principle
dp_data = discrepancy_principle(ptf)
```

The same plotting functions can also be used:

```python
# Visualize L-corner
plot_lcorner(lcorner_data)

# Visualize discrepancy principle
plot_dp(dp_data)
```

We do not implement `gcvmin` for the projected family, since there are various unresolved issues associated with how to define a projected version of $\mathcal{T}(\lambda)$ in Eq. (11).

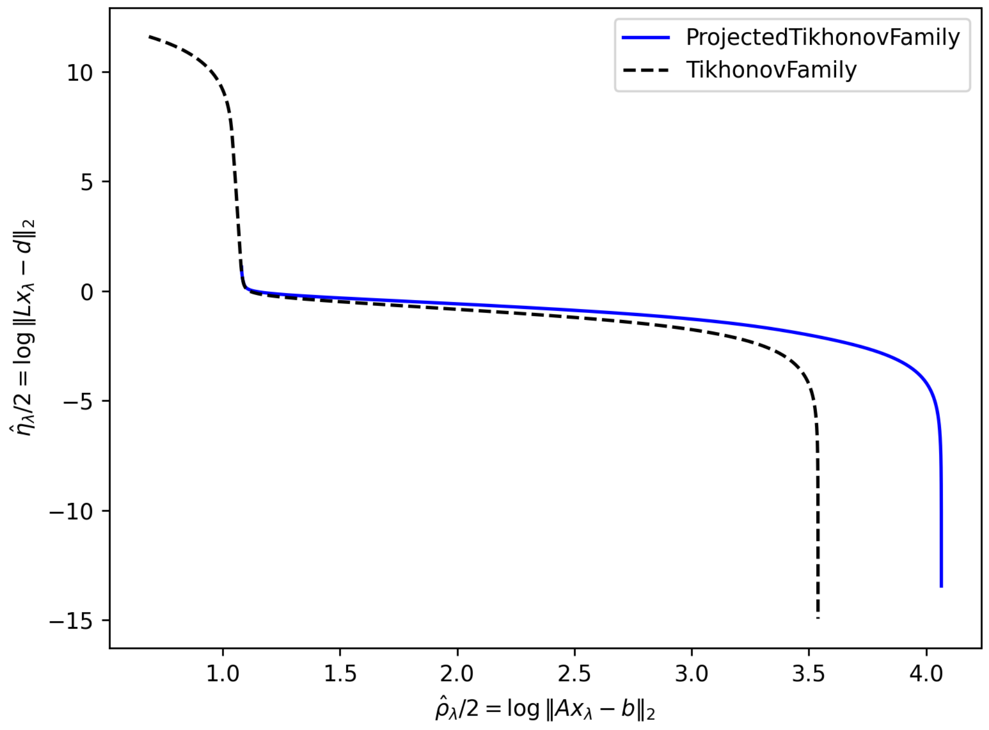

Figure 14: A projected L-curve for the 1D deblurring problem.

A Derivatives of data fidelity and regularization terms

The GSVD formulas for the derivatives of the data fidelity and regularization terms with respect to λ (or $\beta = 1/\lambda$) are validated against the following expressions obtained by direct matrix calculus calculations.

A.1 With respect to λ

Define the intermediate quantities

$$Q_\lambda = A^\top A + \lambda L^\top L, \tag{34}$$

$$x_\lambda = Q_\lambda^{-1} A^\top b, \tag{35}$$

$$r_\lambda = Ax_\lambda - b, \tag{36}$$

$$y_\lambda = Lx_\lambda - d, \tag{37}$$

and

$$x_\lambda' = -Q_\lambda^{-1} L^\top L x_\lambda, \tag{38}$$

$$x_\lambda'' = 2Q_\lambda^{-1} L^\top L Q_\lambda^{-1} L^\top L x_\lambda, \tag{39}$$

$$x_\lambda''' = -6Q_\lambda^{-1} L^\top L Q_\lambda^{-1} L^\top L Q_\lambda^{-1} L^\top L x_\lambda. \tag{40}$$

Then

$$\frac{d}{d\lambda}\,\|Ax_\lambda - b\|_2^2 = 2\,r_\lambda^\top A x_\lambda', \tag{41}$$
$$\frac{d^2}{d\lambda^2}\,\|Ax_\lambda - b\|_2^2 = 2\Big(\|Ax_\lambda'\|_2^2 + r_\lambda^\top A x_\lambda''\Big), \tag{42}$$
$$\frac{d^3}{d\lambda^3}\,\|Ax_\lambda - b\|_2^2 = 6(Ax_\lambda')^\top(Ax_\lambda'') + 2r_\lambda^\top A x_\lambda''', \tag{43}$$
$$\frac{d}{d\lambda}\,\|Lx_\lambda - d\|_2^2 = 2\,y_\lambda^\top L x_\lambda', \tag{44}$$
$$\frac{d^2}{d\lambda^2}\,\|Lx_\lambda - d\|_2^2 = 2\Big(\|Lx_\lambda'\|_2^2 + y_\lambda^\top L x_\lambda''\Big), \tag{45}$$
$$\frac{d^3}{d\lambda^3}\,\|Lx_\lambda - d\|_2^2 = 6(Lx_\lambda')^\top(Lx_\lambda'') + 2y_\lambda^\top L x_\lambda'''. \tag{46}$$

A.2 With respect to $\beta = \lambda^{-1}$

For the reciprocal parameterization $\lambda(\beta) = \beta^{-1}$, define

$$Q_\beta = A^\top A + \beta^{-1} L^\top L, \tag{47}$$
$$x_\beta = Q_\beta^{-1} A^\top b, \tag{48}$$
$$r_\beta = Ax_\beta - b, \tag{49}$$
$$y_\beta = Lx_\beta - d, \tag{50}$$

and

$$x_\beta' = \beta^{-2} Q_\beta^{-1} L^\top L\, x_\beta, \tag{51}$$
$$x_\beta'' = 2\beta^{-4} Q_\beta^{-1} L^\top L\, Q_\beta^{-1} L^\top L\, x_\beta - 2\beta^{-3} Q_\beta^{-1} L^\top L\, x_\beta, \tag{52}$$
$$x_\beta''' = 6\beta^{-6} Q_\beta^{-1} L^\top L\, Q_\beta^{-1} L^\top L\, Q_\beta^{-1} L^\top L\, x_\beta \tag{53}$$
$$- 12\beta^{-5} Q_\beta^{-1} L^\top L\, Q_\beta^{-1} L^\top L\, x_\beta + 6\beta^{-4} Q_\beta^{-1} L^\top L\, x_\beta. \tag{54}$$

Then

$$\frac{d}{d\beta}\|Ax_\beta - b\|_2^2 = 2\,r_\beta^\top A x_\beta', \tag{55}$$
$$\frac{d^2}{d\beta^2}\|Ax_\beta - b\|_2^2 = 2\big(\|Ax_\beta'\|_2^2 + r_\beta^\top A x_\beta''\big), \tag{56}$$
$$\frac{d^3}{d\beta^3}\|Ax_\beta - b\|_2^2 = 6(Ax_\beta')^\top(Ax_\beta'') + 2r_\beta^\top A x_\beta''', \tag{57}$$
$$\frac{d}{d\beta}\,\|Lx_\beta - d\|_2^2 = 2\,y_\beta^\top L x_\beta', \tag{58}$$
$$\frac{d^2}{d\beta^2}\,\|Lx_\beta - d\|_2^2 = 2\Big(\|Lx_\beta'\|_2^2 + y_\beta^\top L x_\beta''\Big), \tag{59}$$
$$\frac{d^3}{d\beta^3}\,\|Lx_\beta - d\|_2^2 = 6(Lx_\beta')^\top(Lx_\beta'') + 2y_\beta^\top L x_\beta'''. \tag{60}$$

References

[1] Per Christian Hansen. *Discrete inverse problems: insight and algorithms.* SIAM, 2010.

[2] Per Christian Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion.* SIAM, 1998.

[3] Per Christian Hansen. "Analysis of discrete ill-posed problems by means of the L-curve". In: *SIAM review* 34.4 (1992), pp. 561–580.

[4] Per Christian Hansen. "The discrete Picard condition for discrete ill-posed problems". In: *BIT Numerical Mathematics* 30.4 (1990), pp. 658–672.

[5] Per Christian Hansen. "The L-curve and its use in the numerical treatment of inverse problems". In: (1999).

[6] Lothar Reichel and Andriy Shyshkov. "A new zero-finder for Tikhonov regularization". In: *BIT Numerical Mathematics* 48.3 (2008), pp. 627–643.