

HDSP Research Group
School of Systems and Computer Engineering, Universidad Industrial de Santander

---

Laboratory Workshop No. 2
Basic Optimization and Introduction to Inverse Problems

Workshop overview

Objective: introduce basic optimization notation and calculus, explain least squares and the pseudoinverse in $\mathbb{R}^n$, and formulate **linear inverse problems** of the form $\mathbf{y} = A\mathbf{x} + \mathbf{e}$, including deconvolution and iterative regularized reconstruction.

Flow: optimization in 1D and 2D $\rightarrow$ norms and least squares in $\mathbb{R}^n$ $\rightarrow$ pseudoinverse $\rightarrow$ linear inverse problems $\rightarrow$ deconvolution instability $\rightarrow$ proximal gradient / ISTA.

Checklist

- Plain-language explanation of optimization notation, plus practical examples that identify the variable and the objective.
- Analytical solutions for the 1D and 2D exercises, including derivative tests, Hessian criteria, and minimum verification.
- Explanation of norms and the gradient of $\|A\mathbf{x}\|_2^2$, including why $\nabla(\mathbf{x}^T A^T A\mathbf{x}) = 2A^T A\mathbf{x}$, plus the exercise on $\|A\mathbf{x} - \mathbf{y}\|_2$.
- Numerical least-squares experiment comparing the pseudoinverse and `lstsq`, with relative errors and residuals.
- Linear inverse-problem experiment with Gaussian kernel construction, explicit matrix construction, blurred/noisy reconstructions, and a discussion of instability.

- ISTA implementation for noisy deblurring, including the role of the prior and the effect of λ and the number of iterations using PSNR/SSIM and visual quality.

Table 1: Deliverables

Deliverable	Minimum content
Written solution (PDF)	Calculations, explanations, and answers.
Python notebook or scripts	Reproducible results, figures, and quality metrics (PSNR/SSIM).
Short discussion	Observations about conditioning, priors, and reconstruction quality.

Suggested tools

`numpy`, `scipy`, `matplotlib`, `scikit-image` (or your own images), `pywt` (wavelets).

1 Basic optimization: notation and calculus

1.1 Notation

An optimization problem asks a very simple question: *among all the choices we can make, which one is the best?* In mathematics, "best" usually means the choice that makes a quantity as small as possible, for example cost, error, travel time, energy, or risk. Optimization is important because in science, engineering, economics, imaging, and machine learning we constantly face competing possibilities and need a principled way to choose one instead of guessing.

In this workshop we start with **unconstrained minimization**, written as

$$\min_{x} \; f(x). \tag{1}$$

This notation can be read in plain English as: *find the value of x that makes $f(x)$ as small as possible.* More precisely:

- $\min_x$ means "search over all allowed values of x and keep the one that gives the smallest value."
- x is the **decision variable** or **unknown**. In 1D it can be one number; in higher dimensions it can be a vector, a signal, or even an image stored as a long vector.
- $f(x)$ is the **objective function** (also called cost, loss, or energy). It assigns a single score to each candidate x. Smaller values of $f(x)$ mean better candidates.

If a problem is written as a maximization, for example "maximize profit," it can usually be rewritten as a minimization by minimizing the negative profit.

Table 2: Practical meaning of x and $f(x)$ in real problems

Problem	What is x?	What is $f(x)$?
Delivery planning	A route or distribution plan.	Total fuel cost + driver time + delay penalties.
Linear regression	The vector of model coefficients.	Prediction error on the observed data.
Image deblurring	The unknown clean image.	Data mismatch plus a regularization term that favors plausible images.
Portfolio design	The amount invested in each asset.	A cost that balances risk and return.

1.2 Optima in 1D and 2D via calculus

Calculus gives a systematic way to locate **candidate optima**. The first step is to find points where the slope (or gradient) vanishes. The second step is to determine whether the point is a minimum, a maximum, or neither.

1D. If $f : \mathbb{R} \to \mathbb{R}$ is differentiable, a local minimum candidate satisfies

$$f'(x^\star) = 0. \tag{2}$$

If $f''(x^\star) > 0$, the graph bends upward and $x^\star$ is a local minimum. If $f''(x^\star) < 0$, the graph bends downward and $x^\star$ is a local maximum. If $f''(x^\star) = 0$, the second-derivative test is inconclusive and more analysis is needed. In 1D, one usually talks about minima and maxima; the word *saddle* is mainly used in dimensions 2 and higher.

2D. If $f : \mathbb{R}^2 \to \mathbb{R}$ is differentiable, a candidate point $\mathbf{x}^\star = [x^\star \; y^\star]^T$ satisfies

$$\nabla f(\mathbf{x}^\star) = \mathbf{0}. \tag{3}$$

To classify it, use the **Hessian matrix**

$$\nabla^2 f(x, y) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial y \partial x} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix}. \tag{4}$$

The Hessian tells us how the surface bends in different directions around the candidate point:

- if the Hessian is **positive definite**, the surface curves upward in every direction, so we have a strict local minimum;
- if it is **negative definite**, the surface curves downward in every direction, so we have a strict local maximum;

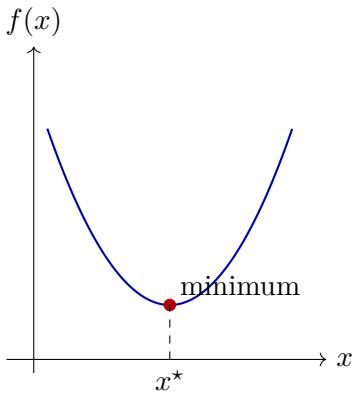
- if it is **indefinite**, the surface curves upward in some directions and downward in others, so the point is a saddle;
- if the Hessian is only semidefinite, the test is inconclusive.

For a 2×2 Hessian, a convenient test uses

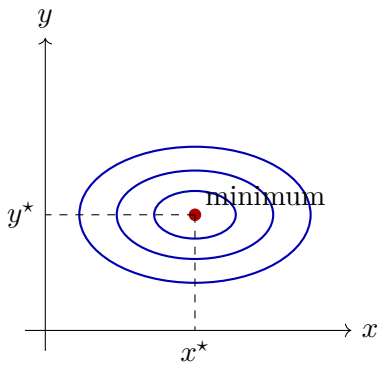
$$D = f_{xx} f_{yy} - f_{xy}^2. \tag{5}$$

Table 3: Hessian criteria in 2D

Condition at $\nabla f(\mathbf{x}^\star) = 0$	Classification	Interpretation
$D > 0$ and $f_{xx} > 0$	Local minimum	Curves upward in both principal directions.
$D > 0$ and $f_{xx} < 0$	Local maximum	Curves downward in both principal directions.
$D < 0$	Saddle point	Curves up in one direction and down in another.
$D = 0$	Inconclusive	The test does not decide the classification.



(a) 1D: a curve with optimal value at $x^\star$.



(b) 2D: contour view of a surface with minimum at $(x^\star, y^\star)$.

Figure 1: Simple geometric picture of optimal values in 1D and 2D.

1.3 Worked examples

Example 1 (1D): $f(x) = (x-2)^2 + 3$. Then

$$f'(x) = 2(x-2), \tag{6}$$

so the critical point is $x^\star = 2$. Since $f''(x) = 2 > 0$, this point is a strict local minimum; because the function is a convex parabola, it is also the global minimum.

Example 2 (2D quadratic):

$$f(x, y) = (x - 1)^2 + 2(y + 2)^2. \tag{7}$$

We obtain

$$\frac{\partial f}{\partial x} = 2(x - 1), \qquad \frac{\partial f}{\partial y} = 4(y + 2), \tag{8}$$

therefore $(x^\star, y^\star) = (1, -2)$. Its Hessian is

$$\nabla^2 f(x, y) = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}, \tag{9}$$

which is positive definite, so $(1, -2)$ is a strict local minimum. Since this is a convex quadratic, it is also the global minimum.

1.4 Exercises (solve analytically)

1. **(1D)** Minimize $f(x) = x^4 - 4x^2 + x$.

   Submit: (i) critical points, (ii) classification (min/max/saddle), (iii) global minimum over $\mathbb{R}$.

2. **(2D)** Minimize

   $$f(x, y) = x^2 + xy + y^2 - 4x - 6y. \tag{10}$$

   Submit: (i) solve $\nabla f = 0$, (ii) Hessian, (iii) verification of global minimum.

2 Optimization in $\mathbb{R}^n$: least squares and the pseudoinverse

Most real optimization problems are not one-dimensional or two-dimensional. In data fitting, inverse problems, control, imaging, and machine learning, the unknown may have tens, thousands, or even millions of components. High-dimensional optimization matters because each component of $\mathbf{x}$ can represent a model parameter, a design choice, a pixel, or a signal sample. It is also more challenging: we cannot visualize the geometry directly, the variables interact with one another, the matrices may be rank-deficient or ill-conditioned, and naive algorithms may become unstable or expensive.

Norms

A **norm** measures the size or length of a vector. The most common example is the Euclidean norm,

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}. \tag{11}$$

A norm is always nonnegative, it is zero only for the zero vector, it scales linearly with the magnitude of the vector, and it satisfies the triangle inequality. In optimization, norms are used to measure errors, distances, residuals, and regularization penalties.

Gradient of $\|A\mathbf{x}\|_2^2$

For matrix calculus, the squared Euclidean norm is the friendlier creature. Let

$$h(\mathbf{x}) = \|A\mathbf{x}\|_2^2. \tag{12}$$

Rewrite it as a quadratic form:

$$h(\mathbf{x}) = (A\mathbf{x})^T(A\mathbf{x}) = \mathbf{x}^T A^T A\mathbf{x}. \tag{13}$$

Now set

$$B = A^T A. \tag{14}$$

Then

$$h(\mathbf{x}) = \mathbf{x}^T B\mathbf{x}. \tag{15}$$

A useful general identity is

$$\nabla_{\mathbf{x}} \left(\mathbf{x}^T B\mathbf{x}\right) = (B + B^T)\mathbf{x}. \tag{16}$$

Here is why. Write the scalar explicitly:

$$\mathbf{x}^T B\mathbf{x} = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i B_{ij} x_j. \tag{17}$$

Differentiate with respect to one component x_k:

$$\frac{\partial}{\partial x_k}\left(\mathbf{x}^T B\mathbf{x}\right) = \sum_{j=1}^{n} B_{kj}x_j + \sum_{i=1}^{n} x_i B_{ik}. \tag{18}$$

The first sum is the kth entry of $B\mathbf{x}$ and the second sum is the kth entry of $B^T\mathbf{x}$. Therefore,

$$\nabla_{\mathbf{x}} \left(\mathbf{x}^T B\mathbf{x}\right) = B\mathbf{x} + B^T\mathbf{x} = (B + B^T)\mathbf{x}. \tag{19}$$

Now return to $B = A^T A$. Since

$$(A^T A)^T = A^T A, \tag{20}$$

the matrix $A^T A$ is symmetric. Therefore,

$$\nabla_{\mathbf{x}} \left(\mathbf{x}^T A^T A\mathbf{x}\right) = (A^T A + A^T A)\mathbf{x} = 2A^T A\mathbf{x}. \tag{21}$$

So the gradient of the squared norm is

$$\nabla\|A\mathbf{x}\|_2^2 = 2A^T A\mathbf{x}. \tag{22}$$

In least-squares problems we often insert the factor $\frac{1}{2}$ to simplify the derivative:

$$q(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x}\|_2^2 = \frac{1}{2}\mathbf{x}^T A^T A\mathbf{x}, \tag{23}$$

so that

$$\nabla q(\mathbf{x}) = A^T A\mathbf{x}. \tag{24}$$

2.1 Least-squares problem

Exercise: gradient of $\|A\mathbf{x} - \mathbf{y}\|_2^2$

Using the same matrix-calculus ideas, compute the gradient of

$$r(\mathbf{x}) = \|A\mathbf{x} - \mathbf{y}\|_2^2. \tag{25}$$

Hints:

- Write the norm as $(A\mathbf{x} - \mathbf{y})^T(A\mathbf{x} - \mathbf{y})$.
- Treat $A\mathbf{x} - \mathbf{y}$ as the inner variable and reuse the previous chain-rule idea.
- The matrix multiplying the residual in the gradient comes from the Jacobian of $A\mathbf{x} - \mathbf{y}$.

Solution:

$$\nabla r(\mathbf{x}) = A^T(A\mathbf{x} - \mathbf{y}) \qquad \text{for } A\mathbf{x} \neq \mathbf{y}. \tag{26}$$

Let $\mathbf{y} \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. The standard least-squares problem is

$$\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{y} - A\mathbf{x}\|_2^2. \tag{27}$$

We usually square the norm because the objective becomes smooth everywhere and the algebra is simpler. Defining

$$L(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2, \tag{28}$$

its gradient is

$$\nabla L(\mathbf{x}) = A^T(A\mathbf{x} - \mathbf{y}). \tag{29}$$

Setting the gradient to zero yields the **normal equations**:

$$A^T A\,\mathbf{x}^\star = A^T\mathbf{y}. \tag{30}$$

If A has full column rank, $A^T A$ is invertible and

$$\mathbf{x}^\star = (A^T A)^{-1} A^T\mathbf{y} \;=\; A^\dagger\mathbf{y}, \tag{31}$$

which matches the **pseudoinverse** solution.

2.2 Python exercise: pseudoinverse vs `lstsq`

Goal: numerically verify the least-squares solution.

1. Generate $m = 80,\ n = 20,\ A \sim \mathcal{N}(0, 1)$, a ground-truth vector $\mathbf{x}_{\text{true}}$, and measurements $\mathbf{y} = A\mathbf{x}_{\text{true}} + \mathbf{e}$ with Gaussian noise.
2. Compute:
   - $\hat{\mathbf{x}}_{\text{pinv}} = \text{pinv}(A)\,\mathbf{y}$

- $\hat{\mathbf{x}}_{\text{lstsq}}$ using `numpy.linalg.lstsq`

3. Compare: relative error $\|\hat{\mathbf{x}} - \mathbf{x}_{\text{true}}\|/\|\mathbf{x}_{\text{true}}\|$ and residual $\|A\hat{\mathbf{x}} - \mathbf{y}\|$.
4. Repeat for the underdetermined case ($m < n$) and comment on what solution the pseudoinverse returns.

```python
import numpy as np

m, n = 80, 20
rng = np.random.default_rng(0)
A = rng.standard_normal((m, n))
x_true = rng.standard_normal(n)
sigma = 0.05
y = A @ x_true + sigma * rng.standard_normal(m)

x_pinv = np.linalg.pinv(A) @ y
x_lstsq, *_ = np.linalg.lstsq(A, y, rcond=None)

rel_err_pinv = np.linalg.norm(x_pinv - x_true) / np.linalg.norm(x_true)
rel_err_lstsq = np.linalg.norm(x_lstsq - x_true) / np.linalg.norm(x_true)

res_pinv = np.linalg.norm(A @ x_pinv - y)
res_lstsq = np.linalg.norm(A @ x_lstsq - y)

print(rel_err_pinv, rel_err_lstsq)
print(res_pinv, res_lstsq)
```

3 Introduction to inverse problems

In this workshop we focus on **linear inverse problems**, written as

$$\mathbf{y} = A\mathbf{x} + \mathbf{e}, \tag{32}$$

where $\mathbf{x}$ is the unknown signal or image, A is the **forward operator**, $\mathbf{e}$ is measurement noise, and $\mathbf{y}$ is the observed data.

The **forward problem** is: given $\mathbf{x}$ and A, compute $\mathbf{y}$. The **inverse problem** is the opposite: given $\mathbf{y}$ and A, estimate $\mathbf{x}$. The hard part is that A may be non-invertible, rectangular, or ill-conditioned, so small noise in the measurements can cause large errors in the reconstruction.

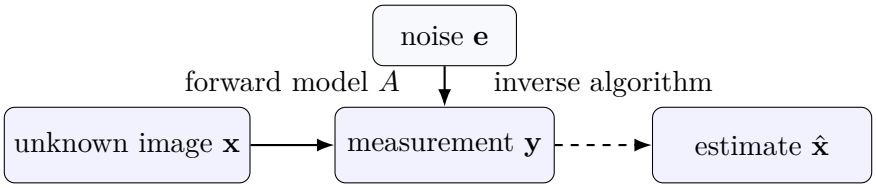


Figure 2: Forward and inverse viewpoints for the linear model $\mathbf{y} = A\mathbf{x} + \mathbf{e}$.

3.1 Exercise: deconvolution and instability

Consider a grayscale image $X \in \mathbb{R}^{h\times w}$ and its vectorized form $\mathbf{x} = \mathrm{vec}(X) \in \mathbb{R}^N$ with $N = hw$. Let $A \in \mathbb{R}^{N\times N}$ be the matrix that applies Gaussian blur.

Implementation guide:

1. **Load and preprocess an image.** Choose a grayscale image, or convert an RGB image to grayscale. Resize it to a modest size such as 32×32 or 64×64 if you plan to build the blur matrix explicitly. Normalize pixel values to $[0, 1]$.

2. **Construct the Gaussian kernel.** Choose an odd size $(2r + 1) \times (2r + 1)$ and a standard deviation σ. Define

$$k[i, j] = \frac{1}{Z} \exp\left(-\frac{i^2 + j^2}{2\sigma^2}\right), \qquad -r \le i, j \le r, \tag{33}$$

   where Z normalizes the kernel so that the entries sum to 1.

3. **Construct the matrix A.** Build A column by column. For each pixel index j, create a basis image E_j that is zero everywhere except at pixel j, where it equals 1. Blur E_j with the same Gaussian kernel and boundary rule you will use for all images. Vectorize the blurred result and place it as column j of A. In symbols,

$$A_{:,j} = \mathrm{vec}(\mathrm{blur}(E_j)). \tag{34}$$

   This makes the matrix representation fully explicit. For large images this is expensive, so keep the image small when building A directly.

4. **Vectorize the image and generate data.** Use the same ordering everywhere (for example, row-major flattening). Compute

$$\mathbf{y} = A\mathbf{x} \tag{35}$$

   for the noiseless blurred data, and then create noisy measurements

$$\mathbf{y}_\eta = A\mathbf{x} + \mathbf{e}, \tag{36}$$

   with Gaussian noise $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma_\eta^2 I)$.

5. **Compute naive reconstructions.** Reconstruct first from the noiseless data and then from the noisy data using a direct inverse when possible, or the pseudoinverse / numerical inverse approximation when the direct inverse is unstable or unavailable. Compare

$$\hat{\mathbf{x}}_{\text{clean}} \approx A^{-1}\mathbf{y}, \qquad \hat{\mathbf{x}}_{\text{noisy}} \approx A^{-1}\mathbf{y}_{\eta}. \tag{37}$$

   Reshape the recovered vectors back to images for visualization.

6. **Compare with noise added before the forward model.** Add noise of the same variance directly to the original image vector, blur it afterward, and compare the result with the previous experiment. This highlights the difference between noise amplification by inversion and ordinary noise propagation through the forward model.

```python
import numpy as np
from scipy.signal import convolve2d
from skimage import io, color, transform

X = io.imread("your_image.png")
if X.ndim == 3:
    X = color.rgb2gray(X)
X = transform.resize(X, (32, 32), anti_aliasing=True)
X = X.astype(float)
X = X / X.max()

h, w = X.shape
N = h * w

def gaussian_kernel(size=9, sigma=1.5):
    r = size // 2
    ax = np.arange(-r, r + 1)
    xx, yy = np.meshgrid(ax, ax)
    K = np.exp(-(xx**2 + yy**2) / (2 * sigma**2))
    return K / K.sum()

K = gaussian_kernel(size=9, sigma=1.5)

def blur_image(Z):
    return convolve2d(Z, K, mode="same", boundary="symm")

A = np.zeros((N, N))
for j in range(N):
    E = np.zeros((h, w))
    E.flat[j] = 1.0
    A[:, j] = blur_image(E).ravel()

x = X.ravel()
y = A @ x
```

Quality metrics: PSNR and SSIM

To evaluate reconstruction quality, compare the original image X with each reconstructed image $\hat{X}$. A basic error measure is the mean squared error (MSE):

$$\mathrm{MSE}(X, \hat{X}) = \frac{1}{N}\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2. \tag{38}$$

For images normalized to $[0, 1]$, the **peak signal-to-noise ratio** is

$$\mathrm{PSNR}(X, \hat{X}) = 10 \log_{10}\left(\frac{1}{\mathrm{MSE}(X, \hat{X})}\right) \text{ dB}. \tag{39}$$

Higher PSNR means smaller pixelwise error.

A perceptually more meaningful measure is the **structural similarity index** (SSIM), which compares local luminance, contrast, and structure. Its standard local form is

$$\mathrm{SSIM}(X, \hat{X}) = \frac{(2\mu_X\mu_{\hat{X}} + c_1)(2\sigma_{X\hat{X}} + c_2)}{(\mu_X^2 + \mu_{\hat{X}}^2 + c_1)(\sigma_X^2 + \sigma_{\hat{X}}^2 + c_2)}, \tag{40}$$

computed over local windows and then averaged across the image. SSIM values closer to 1 indicate better structural similarity.

In Python, you can report both metrics with:

```python
from skimage.metrics import peak_signal_noise_ratio, structural_similarity

psnr = peak_signal_noise_ratio(X, Xhat, data_range=1.0)
ssim = structural_similarity(X, Xhat, data_range=1.0)
print(psnr, ssim)
```

Submit: at least 3 figures (no-noise case, noise in the measurement, noise in the original image), the corresponding PSNR/SSIM values, and a short explanation of why direct inversion is unstable.

4 Iterative algorithms for inverse problems: Proximal Gradient Descent

4.1 Why do we need a *prior*?

In a linear inverse problem,

$$\mathbf{y} = A\mathbf{x} + \mathbf{e}, \tag{41}$$

there may be many candidates $\mathbf{x}$ that explain the measurements almost equally well, especially when A is ill-conditioned or when the data are noisy. The data term only says *fit the measurements*; it does not say what kind of solution is physically meaningful. A **prior** adds that extra information.

A standard regularized formulation is

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \underbrace{\tfrac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2}_{\text{data fidelity}} + \lambda \underbrace{R(\mathbf{x})}_{\text{prior / regularizer}}. \tag{42}$$

Here, the first term enforces consistency with the measurements, while the second term favors solutions that look plausible according to prior knowledge.

A prior can encode many kinds of structure, not only sparsity:

- **Sparsity:** the signal has only a few significant coefficients in a basis or transform domain (wavelets, Fourier, learned dictionaries).
- **Smoothness:** neighboring values should not change too abruptly; this appears in Tikhonov regularization.
- **Piecewise smoothness / edges:** images may be mostly smooth but still contain sharp boundaries; total variation is a common example.
- **Non-negativity or box constraints:** many physical quantities such as intensity, concentration, or density cannot be negative.
- **Low-rank or self-similarity:** useful in dynamic imaging, recommendation systems, and multi-frame reconstruction.
- **Learned priors:** denoisers, generative models, or neural networks trained on representative data can encode more complex image statistics.

In this workshop we will use a **sparse wavelet prior** because it is conceptually clean and its proximal step is easy to implement, but it is only one member of a much larger family of priors.

4.2 Proximal gradient descent and ISTA

For objectives of the form $f(\mathbf{x}) + \lambda\|\mathbf{x}\|_1$, a classic approach is the proximal method, which leads to ISTA (Iterative Shrinkage-Thresholding Algorithm). If

$$f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2, \tag{43}$$

then

$$\nabla f(\mathbf{x}) = A^T(A\mathbf{x} - \mathbf{y}). \tag{44}$$

ISTA alternates a gradient step for the smooth data term and a proximal step for the nonsmooth regularizer:

$$\mathbf{z}_{k+1} = \mathbf{x}_k - \eta\,\nabla f(\mathbf{x}_k), \tag{45}$$

$$\mathbf{x}_{k+1} = \mathrm{soft}(\mathbf{z}_{k+1};\, \eta\lambda), \tag{46}$$

with **soft-thresholding** (applied element-wise)

$$\mathrm{soft}(z;\alpha) = \begin{cases} z + \alpha, & z < -\alpha, \\ 0, & |z| \le \alpha, \\ z - \alpha, & z > \alpha. \end{cases} \tag{47}$$

4.3 Exercise: noisy deblurring with a sparse prior

Problem: given a Gaussian blur operator A and a noisy measurement

$$\mathbf{y} = A\mathbf{x} + \mathbf{e}, \tag{48}$$

recover $\mathbf{x}$ by solving

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \ \tfrac{1}{2}\|A\mathbf{x} - \mathbf{y}\|_2^2 + \lambda \, \|W\mathbf{x}\|_1, \tag{49}$$

where W is an orthonormal wavelet transform (or any basis where the image is approximately sparse).

Tasks:

1. Choose an image $\mathbf{x}$ (or use a standard one). Generate $\mathbf{y}$ by applying Gaussian blur and additive noise.
2. Implement ISTA for the problem above. *Hint:* apply soft-thresholding in the wavelet domain: $\mathbf{u} = W\mathbf{z}, \ \mathbf{u} \leftarrow \text{soft}(\mathbf{u}; \eta\lambda), \ \mathbf{x} = W^{-1}\mathbf{u}$.
3. Choose $\eta \leq 1/L$, where L approximates $\|A^T A\|$ (it can be estimated via power iteration).
4. Compare against (i) direct inversion / pseudoinverse reconstruction and/or (ii) least squares without regularization.
5. Explore the effect of λ and the number of iterations on PSNR, SSIM, and visual quality.

Guiding questions:

- What artifacts appear without a prior? How do they change when enforcing sparsity?
- Why does soft-thresholding reduce noise while preserving important coefficients?
- What happens if λ is too large or too small?
- How might the reconstruction change if you used a smoothness or total-variation prior instead of sparsity?

Base reference (reading): R. Heckel, *Deep Learning for Computational Imaging*, 2024.