12.LP

June 5, 2025

1 STAT 207: Advanced Optimization Topics

1.1 Linear Programming

- A general linear program takes the form:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{G}\mathbf{x} \preceq \mathbf{h}. \end{aligned}$$

A linear program is a convex optimization problem, why?

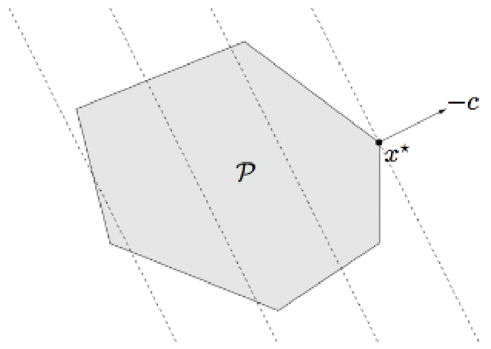


Figure 4.4 Geometric interpretation of an LP. The feasible set $\mathcal{P}$, which is a polyhedron, is shaded. The objective $c^T x$ is linear, so its level curves are hyperplanes orthogonal to c (shown as dashed lines). The point $x^\star$ is optimal; it is the point in $\mathcal{P}$ as far as possible in the direction $-c$.

- The **standard form** of a linear program (LP) is:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \succeq \mathbf{0} \end{aligned}$$

To transform a general linear program into the standard form, we introduce *slack variables* $\mathbf{s} \succeq \mathbf{0}$ such that $\mathbf{G}\mathbf{x} + \mathbf{s} = \mathbf{h}$. Then we write $\mathbf{x} = \mathbf{x}^+ - \mathbf{x}^-$, where $\mathbf{x}^+ \succeq \mathbf{0}$ and $\mathbf{x}^- \succeq \mathbf{0}$. This yields the problem:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\top(\mathbf{x}^+ - \mathbf{x}^-) \\ \text{subject to} \quad & \mathbf{A}(\mathbf{x}^+ - \mathbf{x}^-) = \mathbf{b} \\ & \mathbf{G}(\mathbf{x}^+ - \mathbf{x}^-) + \mathbf{s} = \mathbf{h} \\ & \mathbf{x}^+ \succeq \mathbf{0}, \quad \mathbf{x}^- \succeq \mathbf{0}, \quad \mathbf{s} \succeq \mathbf{0} \end{aligned}$$

The slack variables are often used to transform complicated inequality constraints into simpler non-negativity constraints.

- The **inequality form** of a linear program (LP) is:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{G}\mathbf{x} \preceq \mathbf{h} \end{aligned}$$

- Some solvers only accept the **inequality form**:
  - Simplex method (canonical form);
  - CVXOPT (Python)

```
scipy.optimize.linprog(c, A_ub=None, b_ub=None, A_eq=None, b_eq=None,
                       bounds=None, method='highs', callback=None,
                       options=None, x0=None, integrality=None)
```

1.1.1 Examples

- A piecewise-linear minimization problem can be transformed to an LP. The original problem:

$$\text{minimize} \max_{i=1,\ldots,m} (\mathbf{a}_i^T \mathbf{x} + b_i)$$

can be transformed to the following LP:

$$\begin{aligned} &\text{minimize } \mathbf{t} \\ &\text{subject to } \mathbf{a}_i^T \mathbf{x} + b_i \leq \mathbf{t}, \quad i = 1, \ldots, m, \end{aligned}$$

in $\mathbf{x}$ and $\mathbf{t}$.

Apparently, the following LP formulations:

$$\text{minimize} \max_{i=1,\ldots,m} |\mathbf{a}_i^T \mathbf{x} + b_i|$$

and

$$\text{minimize} \max_{i=1,\ldots,m} (\mathbf{a}_i^T \mathbf{x} + b_i)^+$$

are also LP.

- Any convex optimization problem, defined as:

$$\begin{aligned} \text{minimize } & f_0(\mathbf{x}) \\ \text{subject to } & f_i(\mathbf{x}) \le 0, \quad i = 1, \dots, m, \\ & \mathbf{a}_i^T \mathbf{x} = b_i, \quad i = 1, \dots, p, \end{aligned}$$

where $f_0, \dots, f_m$ are convex functions, can be transformed to the *epigraph* form:

$$\begin{aligned} \text{minimize } & t \\ \text{subject to } & f_0(\mathbf{x}) - t \le 0, \\ & f_i(\mathbf{x}) \le 0, \quad i = 1, \dots, m, \\ & \mathbf{a}_i^T \mathbf{x} = b_i, \quad i = 1, \dots, p, \end{aligned}$$

in variables $\mathbf{x}$ and t. That is why people often say linear programming is universal.

- The linear fractional programming problem, defined as:

$$\begin{aligned} \text{minimize } & \frac{\mathbf{c}^T\mathbf{x} + d}{\mathbf{e}^T\mathbf{x} + f} \\ \text{subject to } & \mathbf{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{G}\mathbf{x} \preceq \mathbf{h} \\ & \mathbf{e}^T\mathbf{x} + f > 0, \end{aligned}$$

can be transformed to an LP (linear programming) problem:

$$\begin{aligned} \text{minimize } & \mathbf{c}^T\mathbf{y} + dz \\ \text{subject to } & \mathbf{G}\mathbf{y} - z\mathbf{h} \preceq \mathbf{0}, \\ & \mathbf{A}\mathbf{y} - z\mathbf{b} = \mathbf{0}, \\ & \mathbf{e}^T\mathbf{y} + fz = 1, \\ & z \ge 0, \end{aligned}$$

in variables $\mathbf{y}$ and z, via the transformation of variables:

$$\mathbf{y} = \frac{\mathbf{x}}{\mathbf{e}^T\mathbf{x} + f}, \quad z = \frac{1}{\mathbf{e}^T\mathbf{x} + f}.$$

Refer to Section 4.3.2 of Boyd and Vandenberghe (2004) for a proof.

1.1.2 Lasso Problem

- Greedy coordinate descent: updating one coordinate (or parameter) at a time by selecting the coordinate that provides the most significant reduction in the objective function.
- Cyclic coordinate descent: updates the coordinates in a fixed cyclic order. It repeatedly cycles through the coordinates, updating each one in turn while keeping the others fixed.

Solve the Lasso ℓ-1 penalized regression problem:

$$\text{minimize } f(\beta) = \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1.$$

The coordinate direction for β_j is

$$\frac{\partial}{\partial \beta_j} f(\beta) = -X_j^\top (y - X\beta) + \lambda s_j,$$

where $s_j \in \{1, -1\}$ is the sign of β_j. Further, the directional derivates are

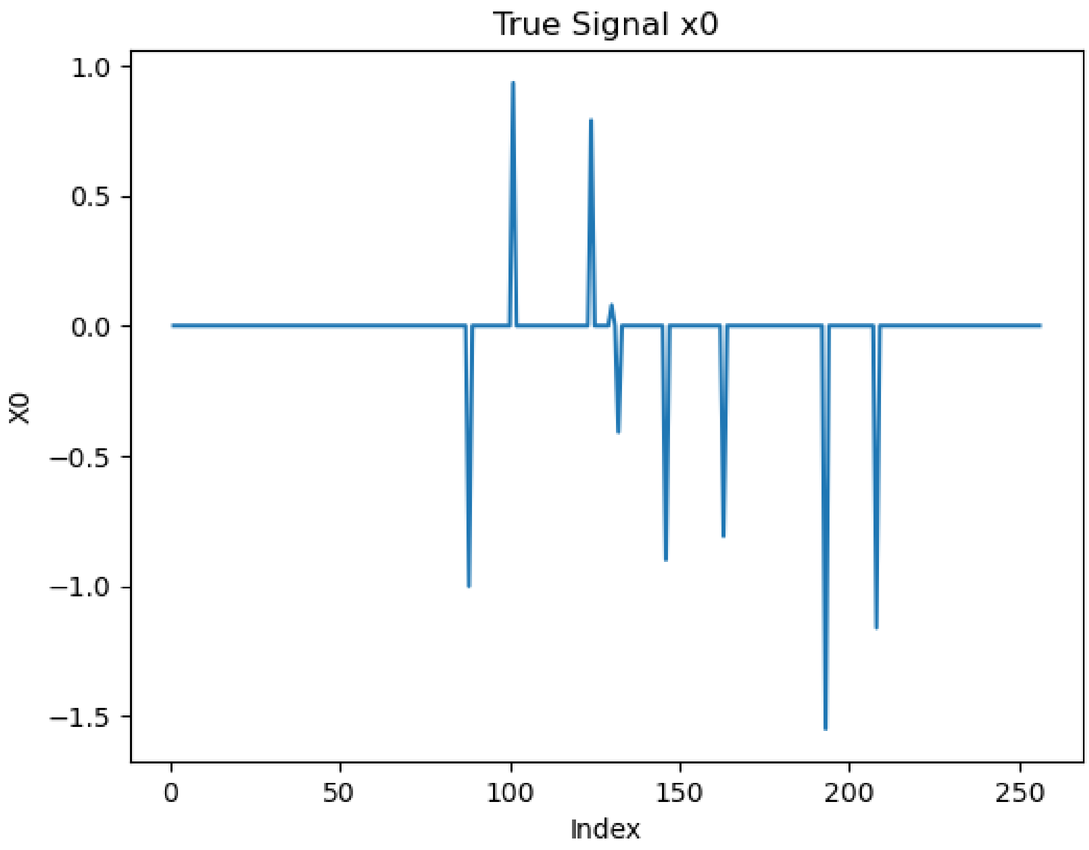
$$d_{e_j} f(\beta) = \lim_{t \downarrow 0} \frac{f(\beta + te_j) - f(\beta)}{t} = -X_j^\top (y - X\beta) + \lambda,$$

$$d_{-e_j} f(\beta) = \lim_{t \downarrow 0} \frac{f(\beta - te_j) - f(\beta)}{t} = X_j(y - X\beta) + \lambda.$$

Hence β_j moves to the right if $(y - X\beta)X_j < -\lambda$, to the left if $(y - X\beta)X_j > \lambda$, and stays fixed otherwise.

[4]:
```python
import numpy as np

def soft_thresholding(rho, lambda_):
    if rho < - lambda_:
        return (rho + lambda_)
    elif rho > lambda_:
        return (rho - lambda_)
    else:
        return 0

def lasso_coordinate_descent(X, y, lambda_,
                              num_iters=100, tol=1e-4,
                             verbose = False):
    m, n = X.shape
    beta = np.zeros(n)
    beta_prev = np.zeros(n)

    for iteration in range(num_iters):
        for j in range(n):
            X_j = X[:, j]
            residual = y - X @ beta + beta[j] * X_j  # partial residual
            rho = np.dot(X_j, residual)
            beta[j] = soft_thresholding(rho, lambda_)  # update rule

        # Check for convergence
        if np.linalg.norm(beta - beta_prev, ord=2) < tol:
            if verbose:
                print(f"Converged in {iteration + 1} iterations.")
            break
        beta_prev = beta.copy()

    return beta
```

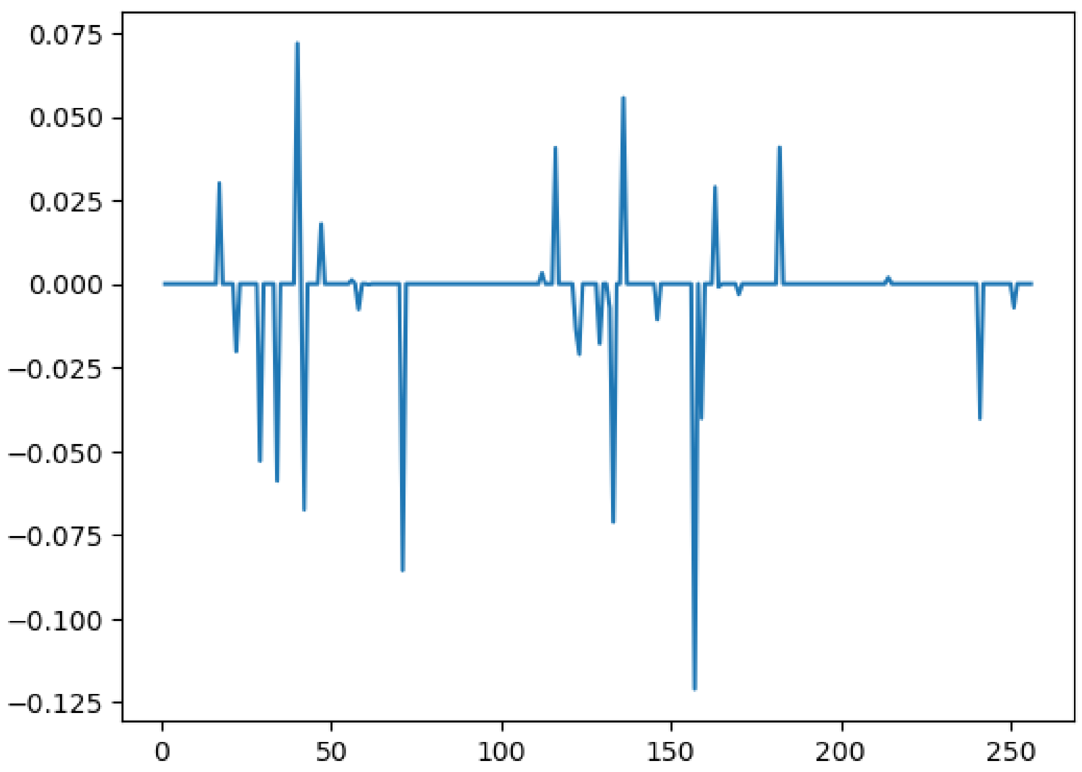
[21]:
```
import numpy as np
import matplotlib.pyplot as plt

# random seed
np.random.seed(24)

# Size of signal
n = 256

# Sparsity (# nonzeros) in the signal
s = 10

# Number of samples (undersample by a factor of 8)
m = 64

# Generate and display the signal
x0 = np.zeros(n)
nonzero_indices = np.random.choice(np.arange(n), s)
x0[nonzero_indices] = np.random.randn(s)

# Generate the random sampling matrix
A = np.random.randn(m, n) / m

# Subsample by multiplexing
y = A.dot(x0)+ 0.5 * np.random.randn(m)


# Plot the true signal
plt.figure()
plt.title("True Signal x0")
plt.xlabel("Index")
plt.ylabel("X0")
plt.plot(np.arange(1, n+1), x0)
plt.show()
```



```
[27]: # Example usage
      lambda_ = .1

      beta = lasso_coordinate_descent(A, y, lambda_,
                                      verbose = True)

      plt.plot(np.arange(1, n+1), beta)
```

```
Converged in 3 iterations.
```

```
[27]: [<matplotlib.lines.Line2D at 0x3004886d0>]
```

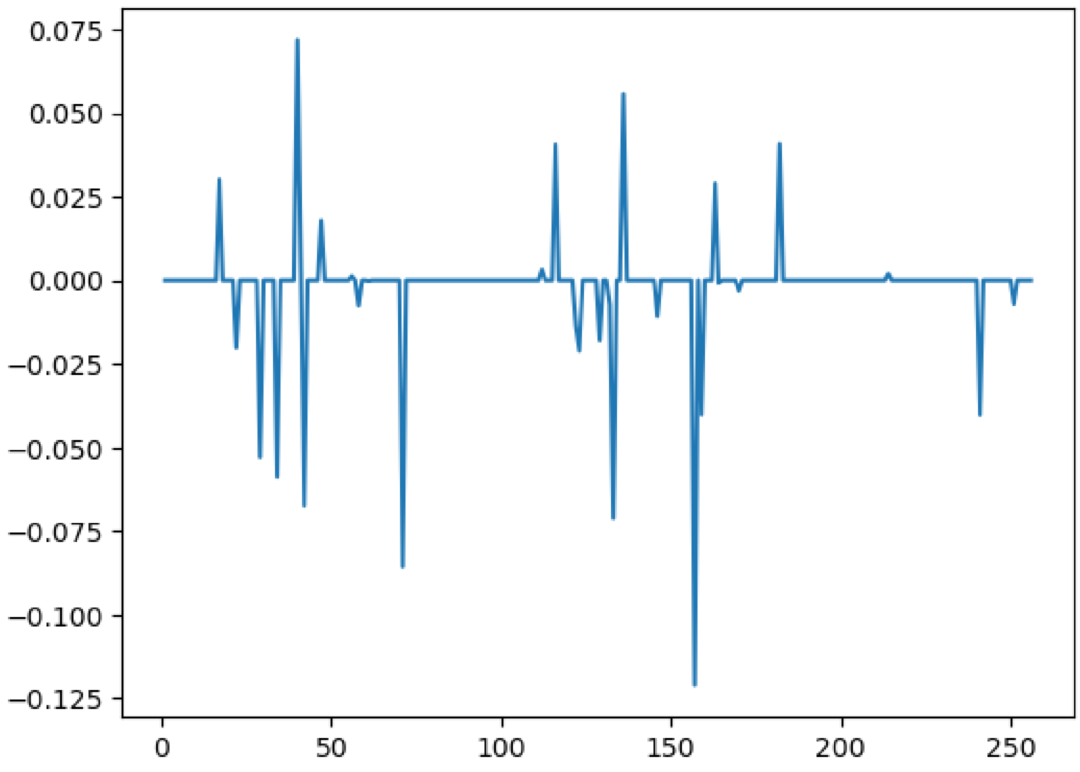


[28]:
```python
def lasso_greedy_coordinate_descent(X, y, lambda_, num_iters=100, tol=1e-4,
                                    verbose = False):
    m, n = X.shape
    beta = np.zeros(n)
    beta_prev = np.zeros(n)

    for iteration in range(num_iters):
        max_decrease = 0
        for j in range(n):
            X_j = X[:, j]
            residual = y - X @ beta + beta[j] * X_j  # partial residual
            rho = np.dot(X_j, residual)
            new_beta_j = soft_thresholding(rho, lambda_)

            # Calculate the decrease in the objective function
            decrease = abs(beta[j] - new_beta_j)
            if decrease > max_decrease:
                max_decrease = decrease
                max_index = j
                best_beta_j = new_beta_j

        # Update the coordinate with the largest decrease
        beta[max_index] = best_beta_j

        # Check for convergence
        if np.linalg.norm(beta - beta_prev, ord=2) < tol:
            if verbose:
                print(f"Converged in {iteration + 1} iterations.")
            break

        beta_prev = beta.copy()

    return beta
```

[29]:
```
# Example usage
lambda_ = 0.1

beta = lasso_greedy_coordinate_descent(A, y, lambda_,
                                       verbose = True)

plt.plot(np.arange(1, n+1), beta)
```

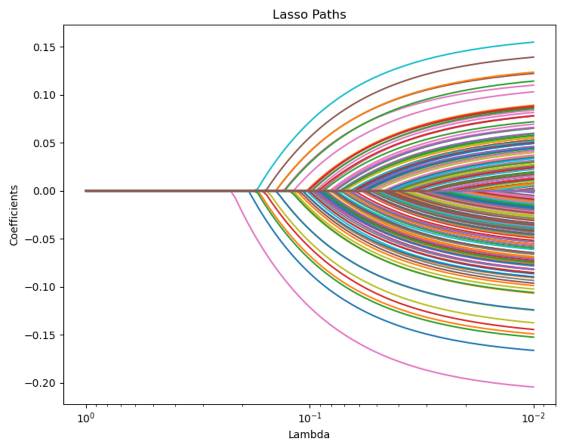
```
Converged in 44 iterations.
```

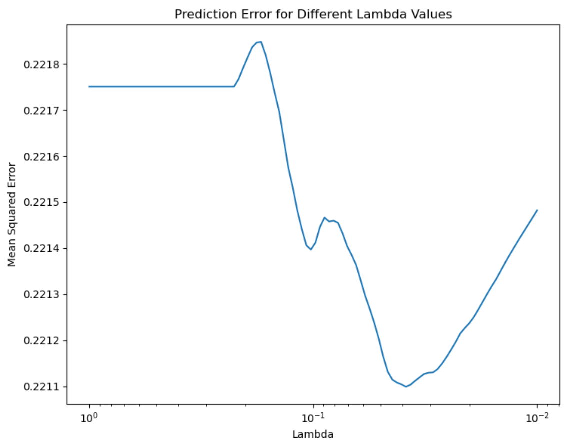
[29]:
```
[<matplotlib.lines.Line2D at 0x3004ac350>]
```



[32]:
```
# Generate the random sampling matrix
A_test = np.random.randn(m, n) / m

# Subsample by multiplexing
y_test = A_test.dot(x0)+ 0.5 * np.random.randn(m)

# Range of lambda values
lambda_values = np.logspace(0,-2, 100)

# Store the coefficients for each lambda
coefficients = []
errors = []

for lambda_ in lambda_values:
    beta = lasso_coordinate_descent(A, y, lambda_)
    coefficients.append(beta)
    predictions = A_test @ beta
    error = np.mean((y_test - predictions) ** 2)  # Mean Squared Error
    errors.append(error)

coefficients = np.array(coefficients)
errors = np.array(errors)
```

[33]:
```
# Plotting the coefficient paths and prediction errors side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Coefficient paths plot
for i in range(coefficients.shape[1]):
    ax1.plot(lambda_values, coefficients[:, i])
ax1.set_xscale('log')
ax1.invert_xaxis()  # Invert the x-axis
ax1.set_xlabel('Lambda')
ax1.set_ylabel('Coefficients')
ax1.set_title('Lasso Paths')

# Prediction errors plot
ax2.plot(lambda_values, errors)
ax2.set_xscale('log')
ax2.invert_xaxis()  # Invert the x-axis
ax2.set_xlabel('Lambda')
ax2.set_ylabel('Mean Squared Error')
ax2.set_title('Prediction Error for Different Lambda Values')

plt.tight_layout()
plt.show()
```





[34]:
```
from scipy.optimize import linprog

## Method 1

vF = np.ones(2 * n)

mAeq = np.hstack((A, -A))
vBeq = y

vLowerBound = np.zeros(2 * n)
vUpperBound = np.inf * np.ones(2 * n)

res = linprog(vF, A_eq=mAeq, b_eq=vBeq, bounds=list(zip(vLowerBound,␣
 ↪vUpperBound)))

vX = res.x[:n] - res.x[n:]

np.allclose(x0, vX)
```

[34]:
```
False
```

[35]:
```
import cvxpy as cp
import numpy as np

x = cp.Variable(n)

# Create the optimization problem
objective = cp.Minimize(cp.norm(x, 1))
constraints = [A @ x == y]
problem = cp.Problem(objective, constraints)

# Solve the problem
```

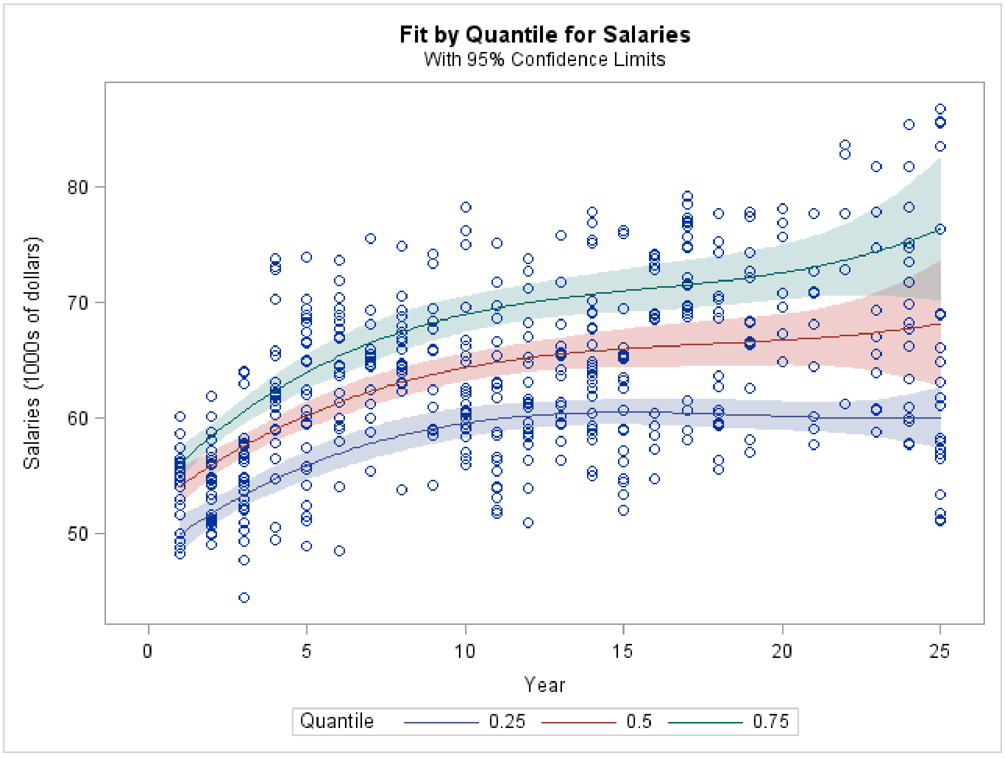
```
problem.solve()

# Retrieve the solution
x_sol = x.value

print("obj val=", problem.solve())

np.allclose(x0, x_sol)
```

```
obj val= 164.32041829954989
```

```
[35]: False
```

1.1.3 Quantile regression

- Linear regression models the mean of the response.
- However, in certain cases, the error variance may not be constant, the distribution of the response variable may exhibit asymmetry, or we may be interested in capturing specific quantiles of the response variable.
- In such situations, quantile regression provides a more suitable modeling approach.



- In a τ-quantile regression, we minimize the loss function

$$f(\beta) = \sum_{i=1}^{n} \rho_\tau(y_i - x_i^T \beta),$$

where $\rho_\tau(z) = z(\tau - 1_{z<0})$. Writing $y - X\beta = r^+ - r^-$, this is equivalent to the LP

$$\begin{aligned}\text{minimize } & \tau^T 1^T r^+ + (1-\tau)1^T r^- = y - X\beta \\ \text{subject to } & r^+ - r^- = y - X\beta \\ & r^+ \succeq 0, r^- \succeq 0\end{aligned}$$

in r^+, r^-, and β.

1.1.4 ℓ_1 Regression

A popular method in robust statistics is the median absolute deviation (MAD) regression that minimizes the ℓ_1 norm of the residual vector $||\mathbf{y} - \mathbf{X}\beta||_1$. This apparently is equivalent to the LP

$$\begin{aligned}\text{minimize} \quad & 1^T(\mathbf{r}^+ + \mathbf{r}^-) \\ \text{subject to} \quad & \mathbf{r}^+ - \mathbf{r}^- = \mathbf{y} - \mathbf{X}\beta \\ & \mathbf{r}^+ \succeq 0, \quad \mathbf{r}^- \succeq 0\end{aligned}$$

in $\mathbf{r}^+$, $\mathbf{r}^-$, and β.

ℓ_1 regression = MAD = median-quantile regression.

1.1.5 Dantzig selector

- Candes and Tao 2007 Propose a variable selection method called the Dantzig selector that solves:

  $$\text{minimize } ||X^T(y - X\beta)||_\infty \text{subject to } \sum_{j=1}^p |\beta_j| \le t,$$

  which can be transformed to an LP.

- The method is named after George Dantzig, who invented the simplex method for efficiently solving LPs in the 1950s.

[13]:
```python
from IPython.display import Image, display

# Adjust the file path as necessary
file_path = "qp.jpg"

# Display the image with 80% width
display(Image(filename=file_path, width=600))
```

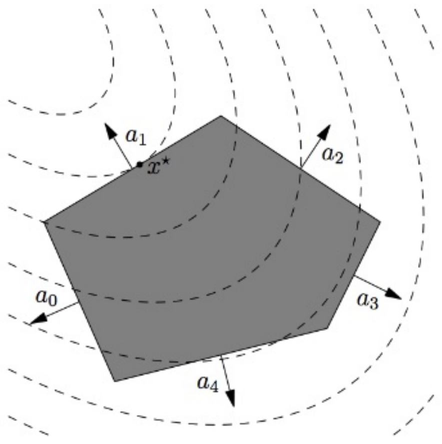


Figure 5.1: Geometric interpretation of quadratic optimization. At the optimal point $x^\star$ the hyperplane $\{x \mid a_1^T x = b\}$ is tangential to an ellipsoidal level curve.

1.2 Quadratic Programming

- A quadratic program (QP) has a quadratic objective function and affine constraint functions:

$$\begin{aligned} \text{minimize} \ & \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x} \\ \text{subject to} \ & \mathbf{G}\mathbf{x} \preceq \mathbf{h} \\ & \mathbf{A}\mathbf{x} = \mathbf{b}, \end{aligned}$$

where we require $\mathbf{P} \in \mathbb{S}_+^n$ (symmetric positive semidefinite). Apparently, linear programming (LP) is a special case of QP with $\mathbf{P} = \mathbf{0}_{n\times n}$.

1.2.1 Examples

- Least squares with linear constraints. For example, nonnegative least squares (NNLS)

$$\begin{aligned} \text{minimize} \ & \frac{1}{2}\|\mathbf{y} - \mathbf{X}\ \|_2^2 \\ \text{subject to} \ & \ \succeq \mathbf{0} \end{aligned}$$

- Lasso (Tibshirani 1996) minimizes the least squares loss with the ℓ_1 (lasso) penalty

$$\text{minimize} \ \frac{1}{2}\|\mathbf{y} - \beta_0\mathbf{1} - \mathbf{X}\ \|_2^2 + \lambda\|\ \|_1,$$

  where $\lambda > 0$ is the tuning parameter.

- Write $\beta = \beta^+ - \beta^-$, the equivalent QP is

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}(\beta^+ - \beta^-)^T X^T (I - \frac{1}{n}11^T) X (\beta^+ - \beta^-) + \\ & y^T (I - \frac{1}{n}11^T) X (\beta^+ - \beta^-) + \lambda 1^T (\beta^+ + \beta^-) \\ \text{subject to} & \beta^+ \succeq 0, \quad \beta^- \succeq 0 \end{array}$$

  in β^+, β^-.

[14]:
```
# Adjust the file path as necessary
file_path = "ridge.jpg"

# Display the image with 80% width
display(Image(filename=file_path, width=600))
```

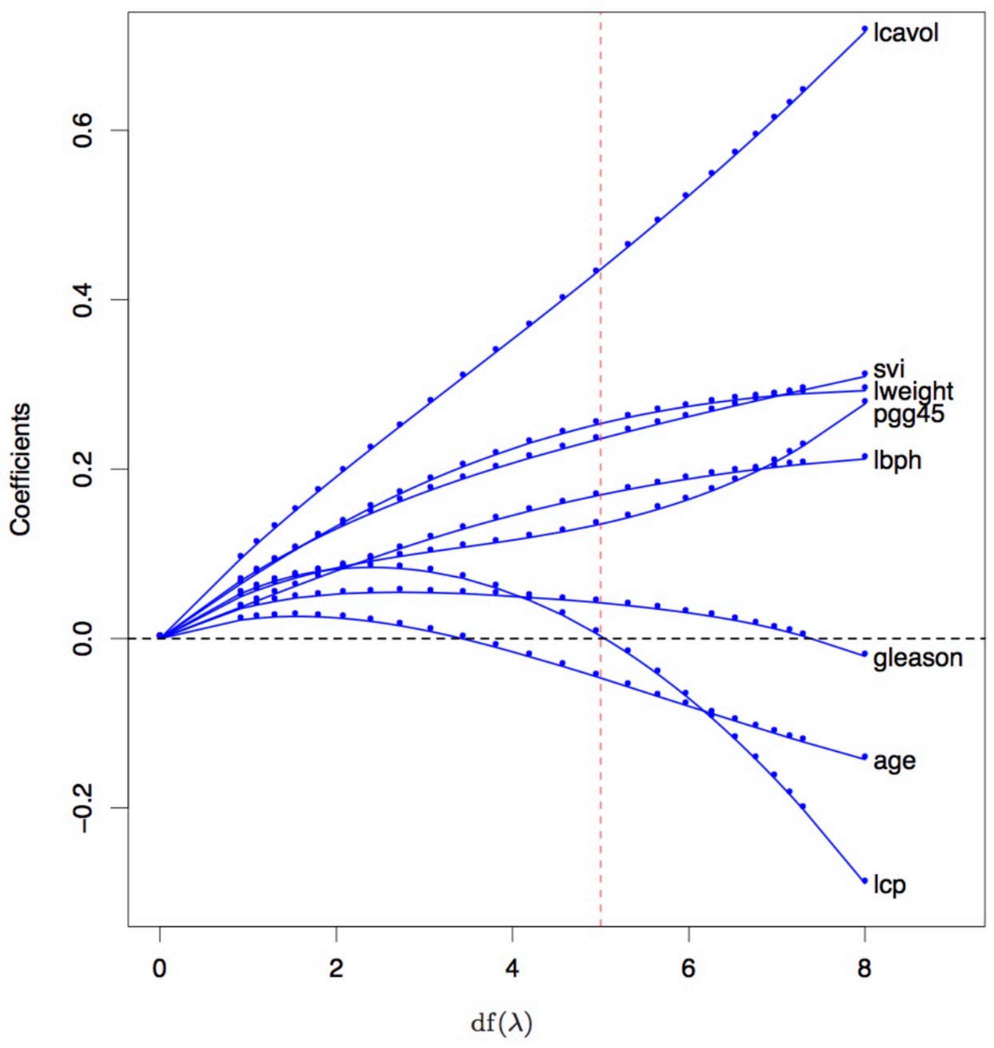


FIGURE 3.8. *Profiles of ridge coefficients for the prostate cancer example, as the tuning parameter λ is varied. Coefficients are plotted versus $\mathrm{df}(\lambda)$, the effective degrees of freedom. A vertical line is drawn at $\mathrm{df} = 5.0$, the value chosen by cross-validation.*

[15]:
```
# Adjust the file path as necessary
file_path = "lasso.jpg"

# Display the image with 80% width
display(Image(filename=file_path, width=600))
```

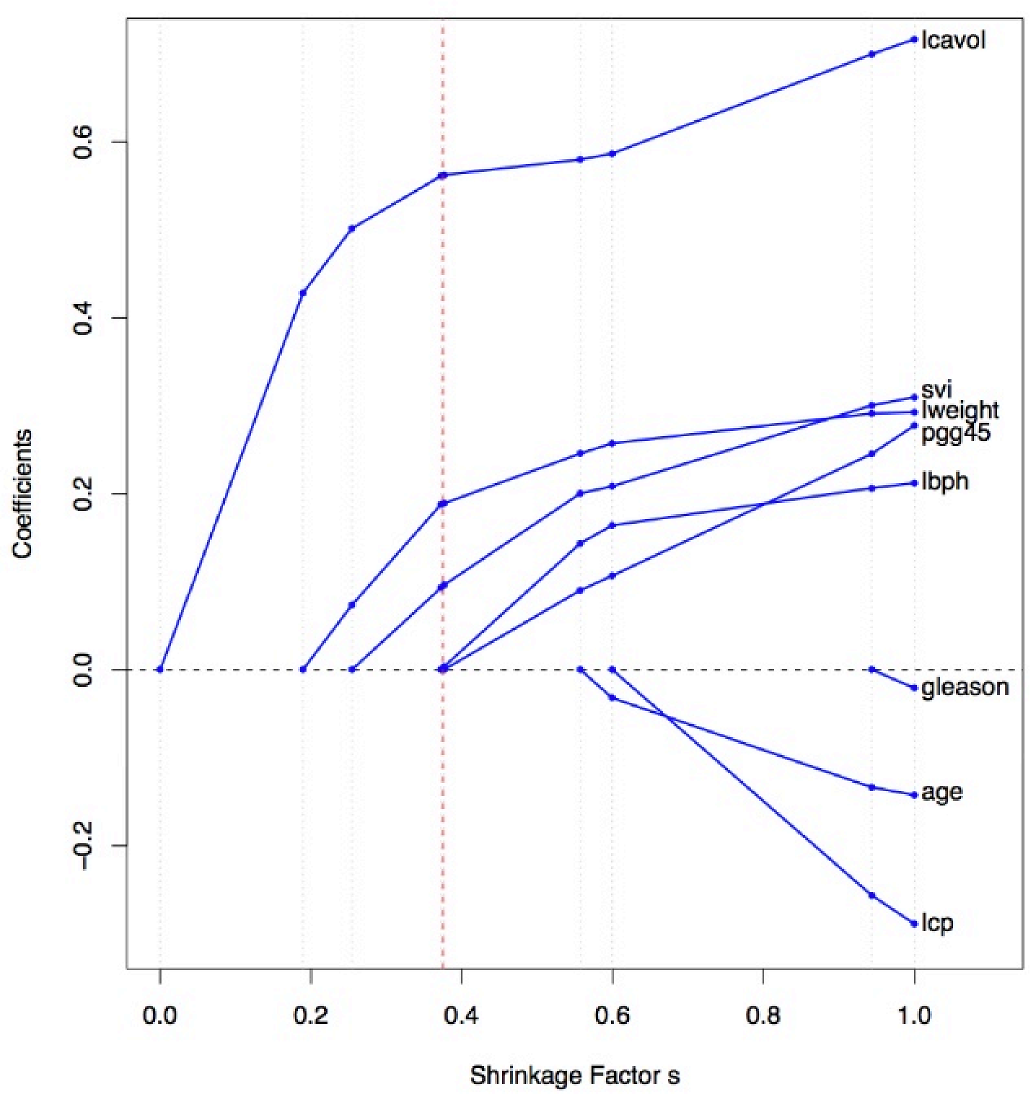


FIGURE 3.10. *Profiles of lasso coefficients, as the tuning parameter t is varied. Coefficients are plotted versus $s = t/\sum_1^p |\hat{\beta}_j|$. A vertical line is drawn at $s = 0.36$, the value chosen by cross-validation. Compare Figure 3.8 on page 65; the lasso profiles hit zero, while those for ridge do not. The profiles are piece-wise linear, and so are computed only at the points displayed; see Section 3.4.4 for details.*

- Elastic Net Zou and Hastie (2005):

$$\text{minimize } \frac{1}{2}\|\mathbf{y} - \beta_0\mathbf{1} - \mathbf{X}\ \|_2^2 + \lambda\left(\alpha\|\ \|_1 + (1-\alpha)\|\ \|_2^2\right),$$

- Image denoising by the total variation (TV) penalty or the anisotropic penalty

$$\frac{1}{2}\|y-x\|_F^2+\lambda\sum_{i,j}\sqrt{(x_{i+1,j}-x_{i,j})^2+(x_{i,j+1}-x_{i,j})^2}.$$

$$\frac{1}{2}\|y-x\|_F^2+\lambda\sum_{i,j}\left(|x_{i+1,j}-x_{i,j}|+|x_{i,j+1}-x_{i,j}|\right).$$

- The Huber loss

$$\phi(r)=\begin{cases}\frac{r^2}{M} & \text{if } |r|\le M\\ r^2-2M|r|+M^2 & \text{if } |r|>M\end{cases}$$

is commonly used in robust statistics. The robust regression problem

$$\text{minimize}\ \sum_{i=1}^{n}\phi(y_i-\beta_0-x_i^T\beta)$$

can be transformed to a QP

$$\begin{array}{ll}\text{minimize} & u^Tu+2M1^Tv-u^Tv\\ \text{subjec to} & u-v\preceq y-X\beta\preceq u+v\\ & 0\preceq u\preceq M1,\quad v\succeq 0\end{array}$$

in $u,v\in\mathbb{R}^n$ and $\beta\in\mathbb{R}^p$. Hint: write $|r_i|=(|r_i|\wedge M)+(|r_i|-M)=u_i+v_i$.

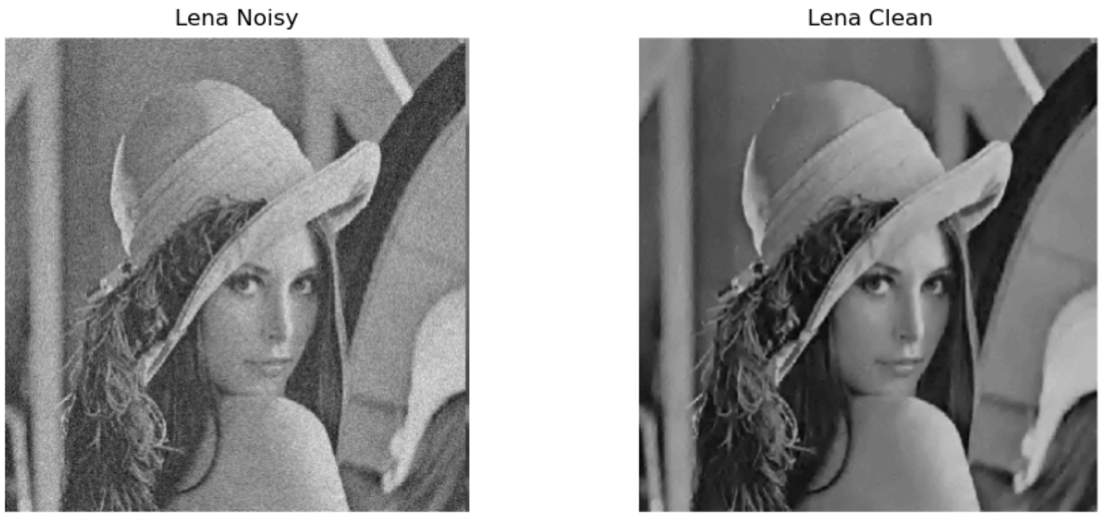
- Support Vector Machines (SVM) In two-class classification problems, we are given training data $(\mathbf{x}_i,y_i), i=1,\dots,n$, where $\mathbf{x}_i\in\mathbb{R}^n$ are feature vectors and $y_i\in\{-1,1\}$ are class labels. The SVM solves the optimization problem:

$$\text{minimize}\ \sum_{i=1}^{n}\left[1-y_i(\beta_0+\sum_{j=1}^{p}x_{ij}\beta_j)\right]_+ +\lambda\|\beta\|_2^2,$$

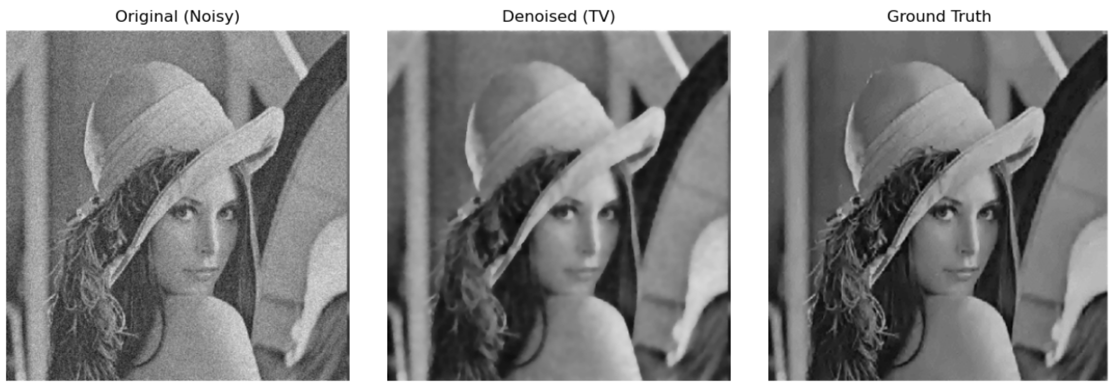
where $\lambda\ge 0$ is a tuning parameter. This is a quadratic programming problem.

[46]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io

# Load the combined image
image = io.imread('lena.jpg', as_gray=True)

# Get image dimensions
height, width = image.shape

# Split into left and right halves
midpoint = width // 2
lena_noisy = image[:, :midpoint]
lena_clean = image[:, midpoint:]

# Display both images
```

```python
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.title("Lena Noisy")
plt.imshow(lena_noisy, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title("Lena Clean")
plt.imshow(lena_clean, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```



[41]:
```python
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color
from skimage.restoration import denoise_tv_chambolle

# Apply total variation denoising (isotropic TV)
lambda_tv = 0.1  # regularization strength
img_denoised = denoise_tv_chambolle(lena_noisy, weight=lambda_tv)

# Plot
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.title("Original (Noisy)")
plt.imshow(lena_noisy, cmap='gray')
plt.axis('off')
plt.subplot(1, 3, 2)
plt.title("Denoised (TV)")
plt.imshow(img_denoised, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title("Ground Truth")
plt.imshow(lena_clean, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```



[ ]:
```python
import cvxpy as cp

Y = lena_noisy  # input image
n, m = Y.shape
print(n,m)
X = cp.Variable((n, m))

tv_penalty = cp.tv(X)
# Denoising objective
lambda_tv = 0.1
objective = cp.Minimize(0.5 * cp.sum_squares(X - Y) + lambda_tv * tv_penalty)

# Solve
problem = cp.Problem(objective)
problem.solve()

# Result
img_denoised = X.value
```

```
351 343
```

[45]:
```python
# Plot
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.title("Original (Noisy)")
plt.imshow(lena_noisy, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Denoised (TV)")
plt.imshow(img_denoised, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title("Ground Truth")
plt.imshow(lena_clean, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```