MATH 761 Final Project Codes

khohmeier

November 2023

```
library(reticulate)
```

```
## Warning: package 'reticulate' was built under R version 4.0.5
```

```
# FROM THE LECTURE NOTES, LECT07-09: For an over-sampled DCT matrix, the larger F is, the more coherent
# We compute coherence using the mutual coherence formula defined in LECT07-09.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize

np.random.seed(1)


def gen_A(M, N, F):
  """Generates a random DCT matrix of size M x N with F coefficients.

  Args:
    M (int): Number of rows in the matrix.
    N (int): Number of columns in the matrix.
    F (int): Number of coefficients in the matrix.

  Returns:
    A (np.ndarray): DCT matrix of size M x N with F coefficients."""
  w = np.random.randn(M, 1)
  A = np.zeros((M, N))

  # generate the over complete DCT matrix
  for j in range(1, N):
    for i in range(1, M):
      A[i, j] = (1 / np.sqrt(N)) * np.cos(2 * np.pi * j * w[i, 0] / F)
  return A


def soft_thresholding(v, kappa):
  """Performs soft thresholding on a vector as a helper function to l1_admm.

  Args:
    v (np.ndarray): A vector.
    kappa (float): Thresholding parameter.

  Returns:
```

```
    v_thresh (np.ndarray): Thresholded vector.""" ""
  return np.maximum(0, v - kappa) - np.maximum(0, -v - kappa)


def shrinkL1L2(y, lamb, alpha):
  """Performs L1-L2 shrinkage on a vector.

  Args:
    y (np.ndarray): A vector.
    lamb (float): Regularization parameter.
    alpha (float): Thresholding parameter.

  Returns:
    x (np.ndarray): Thresholded vector.""" ""
  x = np.zeros(np.size(y))
  if max(np.abs(y)) > lamb:
    x = np.maximum(0, np.abs(y) - lamb) * np.sign(y)
    x = x * (np.linalg.norm(x) + alpha * lamb) / (np.linalg.norm(x))
  elif max(np.abs(y)) >= (1 - alpha) * lamb:
    idx = np.abs(y).argmax()
    x[idx] = (np.abs(y[idx]) + (alpha - 1) * lamb) * np.sign(y[idx])
  return x


def l1_admm(A,b,lam,rho,max_iter,tol=None,x_true=None,x=None,y=None,u=None):
  """Solves L1 minimization using Alternating Direction of Multipliers (ADMM).

  Args:
    A (np.ndarray): DCT matrix.
    b (np.ndarray): Right-hand side vector.
    lam (float): Regularization parameter.
    rho (float): ADMM parameter.
    max_iter (int): Maximum number of iterations.
    tol (float): Tolerance.
    x_true (np.ndarray): Ground truth solution.
    x (np.ndarray): Current estimate.
    z (np.ndarray): Current dual variable.
    u (np.ndarray): Current primal variable.

  Returns:
    x (np.ndarray): Solution.
    k + 1 (int): Number of iterations used to reach solution.
    history (dict): Dictionary containing information collected at each iteration of ADMM."""
  # Initialize variables if not provided
  if x is None:
    x = np.zeros(A.shape[1])
  if y is None:
    y = np.zeros(A.shape[1])
  if u is None:
    u = np.zeros(A.shape[1])
  history = {'residual': [], 'rel_error': [], 'error_gt': [], 'objval': []}
  for k in range(max_iter):
    v = y - (u / rho)
```

```
    x_old = np.copy(x)
    x = soft_thresholding(v, lam / rho)  # divide lam by rho for correct thresholding
    y = np.linalg.inv(A.T @ A + rho * np.eye(N)) @ (A.T @ b + rho * x + u)  # matrix inversion term was
    u = u + rho * (x - y)
    # residual & error computations
    residual = np.linalg.norm(A @ x - b)
    if np.linalg.norm(x) > 0:
      rel_error = np.linalg.norm(x - x_old) / np.linalg.norm(x)
    else:
      rel_error = None
    if x_true is not None:
      error_gt = np.linalg.norm(x - x_true) / np.linalg.norm(x_true)
    else:
      error_gt = None
    history['objval'].append(np.linalg.norm(A @ x - b,1))
    history['residual'].append(residual)
    history['error_gt'].append(error_gt)
    history['rel_error'].append(rel_error)
    if tol is not None and x_true is not None and (np.linalg.norm(x - x_true) < tol):
      break
  return x, k + 1, history


def admm_l1_l2(A, b, lamb, alpha, rho, num_iters, tol=None, x_true=None):
  """Solves L1 - L2 minimization using Alternating Direction of Multipliers (ADMM).

  Args:
    A (np.ndarray): DCT matrix.
    b (np.ndarray): Right-hand side vector.
    lam (float): Regularization parameter.
    alpha (float): ADMM parameter.
    rho (float): ADMM parameter.
    num_iters (int): Maximum number of iterations.
    tol (float): Tolerance.
    x_true (np.ndarray): Ground truth solution.

  Returns:
    x (np.ndarray): Solution.
    k + 1 (int): Number of iterations used to reach solution.
    history (dict): Dictionary containing information collected at each iteration of ADMM."""
  x = np.zeros(A.shape[1])
  y = np.zeros(A.shape[1])
  u = np.zeros(A.shape[1])
  # Precompute to save computational effort
  Atb = A.T @ b
  AAt = A.T @ A
  # Cholesky decomposition of (A^TA + rho*I) for faster inversion
  L = np.linalg.cholesky(AAt + rho * np.eye(A.shape[1]))
  history = {'residual': [], 'rel_error': [], 'error_gt': [], 'objval': []}
  for k in range(num_iters):
    # x-update (using the previously computed Cholesky decomposition)
    xold = np.copy(x)
    x = shrinkL1L2(y - u, lamb / rho, alpha)
```

```
    # y-update
    rhs = Atb + rho * (x + u)
    y = np.linalg.solve(L.T, np.linalg.solve(L, rhs))
    # u-update
    u = u + x - y
    # stop conditions and outputs
    residual = np.linalg.norm(A @ x - b) / np.linalg.norm(b)
    if np.linalg.norm(x) > 0:
      rel_error = np.linalg.norm(x - xold) / np.linalg.norm(x)
    else:
      rel_error = None
    if x_true is not None:
      error_gt = np.linalg.norm(x - x_true) / np.linalg.norm(x_true)
    else:
      error_gt = None
    if x_true is not None:
      error_gt = np.linalg.norm(x_true - x) / np.linalg.norm(x_true)
    if tol is not None and x_true is not None and (np.linalg.norm(x - x_true) < tol):
      break
    history['objval'].append(np.linalg.norm(x,1) - np.linalg.norm(x,2))
    history['rel_error'].append(rel_error)
    history['error_gt'].append(error_gt)
    history['residual'].append(residual)
  return x, k + 1, history


def admml1ratiol2(A, b, rho_one, rho_two, num_iters, tol=None, x_true=None):
  x = np.random.normal(0, 1, A.shape[1])
  y = np.random.normal(0, 1, A.shape[1])
  v = np.random.normal(0, 1, A.shape[1])
  w = np.random.normal(0, 1, A.shape[1])
  u = np.random.normal(0, 1, A.shape[1])
  e = np.random.normal(0, 1, A.shape[1])
  #f = np.zeros(A.shape[1])
  #eta = 0
  history = {'residual': [], 'rel_error': [], 'error_gt': [], 'objval': []}
  for k in range(num_iters):
    f = (rho_one / (rho_one + rho_two)) * (y - (1 / rho_one) * v) + (rho_two / (rho_one + rho_two)) * (
    #small_id = 1e-5 * np.eye(A.shape[0])
    x = (np.eye(A.shape[1]) - A.T @ np.linalg.pinv(A @ A.T) @ A) @ f + A.T @ np.linalg.pinv(A @ A.T) @ 
    d = x + (v / rho_one)
    eta = np.linalg.norm(d, 2)
    c = np.linalg.norm(u, 1)
    D = c / (rho_one * eta**3)
    C = ((27 * D + 2 + np.sqrt(27 * D + 2)**2 - 2) / 2)**(1 / 3)
    tau = (1 / 3) + (1 / 3) * (C + (1 / C))
    if d.all() == 0:
      y = e
    else:
      y = tau * d
    u = np.max(np.abs(x + (w / rho_two)) - 1 / (rho_two * np.linalg.norm(y, 2)), 0) * np.sign(x + (w / 
    v = v + rho_one * (x - y)
    w = w + rho_two * (x - u)
```

```python
    # stop conditions and outputs
    residual = np.linalg.norm(A @ x - b) / np.linalg.norm(b)
    if np.linalg.norm(x) > 0:
      rel_error = np.linalg.norm(x - x_true) / np.linalg.norm(x)
      if x_true is not None:
        error_gt = np.linalg.norm(x - x_true) / np.linalg.norm(x_true)
      #error_gt = np.linalg.norm(x_true - x) / np.linalg.norm(x_true)
    else:
      error_gt = None
      rel_error = None
    if tol is not None and x_true is not None and (np.linalg.norm(x - x_true) < tol):
      break
    #history['objval'].append(soft_thresholding(A @ x - b, lamb / rho).sum())
    history['objval'].append(np.linalg.norm(x,1)/np.linalg.norm(x,2))
    history['rel_error'].append(rel_error)
    history['error_gt'].append(error_gt)
    history['residual'].append(residual)
  return x, k + 1, history
```

```python
# Parameters
M = 250
N = 500
rho = 10  # Penalty parameter for ADMM
lam = 1e-6  # Weight of the L1 norm term in the objective
max_iter = N  # Maximum number of iterations
abstol = 1e-3  # Absolute tolerance
alpha = 1e-3  # Weight of the L2 norm term in the objective
rho_one = 1.5
rho_two = 0.5
F_low = 1
F_med = 10
F_high = 50
A_lowF = gen_A(M, N, F_low)
A_medF = gen_A(M, N, F_med)
A_highF = gen_A(M, N, F_high)

### PROBLEM 1: VARY THE COHERENCE ###
# Problem setup - generate ground truth xg and b (with noise)
xg = np.zeros(N)
s = 25
indices = np.random.choice(np.arange(N), replace=False,size=s)  # select 's' random indices
xg[indices] = np.random.uniform(low=0.01, high=0.05,size=s)  # assign small, random non-zero values
b_lowF = np.matmul(A_lowF, xg)  # Example measurement vector b, using low F A
b_lowF = b_lowF + 0.01 * np.random.normal(0, 1, b_lowF.shape)
b_medF = np.matmul(A_medF, xg)
b_medF = b_medF + 0.01 * np.random.normal(0, 1, b_medF.shape)
b_highF = np.matmul(A_highF,xg)  # Example measurement vector b, using high F A
b_highF = b_highF + 0.01 * np.random.normal(0, 1, b_highF.shape)

# Call the L1 ADMM solver
# Low F/low coherence first
x_admm_lowF, iters_admm_lowF, hist_admm_lowF = l1_admm(A_lowF,b_lowF,lam,rho,max_iter,abstol,x_true=xg)
#print(x_admm_lowF)
```

```
# Med F/med coherence
x_admm_medF, iters_admm_medF, hist_admm_medF = l1_admm(A_medF,b_medF,lam,rho,max_iter,abstol,x_true=xg)
#print(x_admm_medF)
# High F/high coherence
x_admm_highF, iters_admm_highF, hist_admm_highF = l1_admm(A_highF,b_highF,lam,rho,max_iter,abstol,x_true
#print(x_admm_highF)

# implement L1 - L2 ADMM for low F A
x_l1l2_lowF, iters_l1l2_lowF, hist_l1l2_lowF = admm_l1_l2(A_lowF,b_lowF,lam,alpha,rho,max_iter,abstol,x_
#print(x_l1l2_lowF)
# implement L1-L2 ADMM for med F A
x_l1l2_medF, iters_l1l2_medF, hist_l1l2_medF = admm_l1_l2(A_medF,b_medF,lam,alpha,rho,max_iter,abstol,x_
#print(x_l1l2_medF)
# implement L1-L2 ADMM for high F A
x_l1l2_highF, iters_l1l2_highF, hist_l1l2_highF = admm_l1_l2(A_highF,b_highF,lam,alpha,rho,max_iter,abst
#print(x_l1l2_highF)

# L1/L2 ADMM
# Low F/low coherence
x_admml1dl2_lowF, iters_admml1dl2_lowF, hist_admml1dl2_lowF = admml1ratiol2(A_lowF, b_lowF, rho_one, rho
#print(x_admml1dl2_lowF)
# Med F/med coherence
x_admml1dl2_medF, iters_admml1dl2_medF, hist_admml1dl2_medF = admml1ratiol2(A_medF, b_medF, rho_one, rho
#print(x_admml1dl2_medF)
# High F/high coherence
x_admml1dl2_highF, iters_admml1dl2_highF, hist_admml1dl2_highF = admml1ratiol2(A_highF, b_highF, rho_one
#print(x_admml1dl2_highF)

# PLOT ALL MODELS TOGETHER #
# Create a single figure with subplots arranged in 1 row and 3 columns
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# RESIDUALS #
# Low Coherence plots
axs[0].plot(hist_admm_lowF['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C79F438>]
```

```
axs[0].plot(hist_l1l2_lowF['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C79F710>]
```

```
axs[0].plot(hist_admml1dl2_lowF['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C79FA90>]
```

```
axs[0].set_title('Low Coherence, F = 1')
```

```
## Text(0.5, 1.0, 'Low Coherence, F = 1')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Residual')
```

```
## Text(0, 0.5, 'Residual')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med Coherence plots
```

```
## <matplotlib.legend.Legend object at 0x000000000C6D3358>
```

```
axs[1].plot(hist_admm_medF['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7B9EF0>]
```

```
axs[1].plot(hist_l1l2_medF['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7C8208>]
```

```
axs[1].plot(hist_admml1dl2_medF['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7C8518>]
```

```
axs[1].set_title('Medium Coherence, F = 10')
```

```
## Text(0.5, 1.0, 'Medium Coherence, F = 10')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Residual')
```

```
## Text(0, 0.5, 'Residual')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# High Coherence plots
```

```
## <matplotlib.legend.Legend object at 0x000000000C746D68>
```

```python
axs[2].plot(hist_admm_highF['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7D5978>]
```

```python
axs[2].plot(hist_l1l2_highF['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7D5C50>]
```

```python
axs[2].plot(hist_admml1dl2_highF['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7D5F60>]
```

```python
axs[2].set_title('High Coherence, F = 50')
```

```
## Text(0.5, 1.0, 'High Coherence, F = 50')
```

```python
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```python
axs[2].set_ylabel('Residual')
```

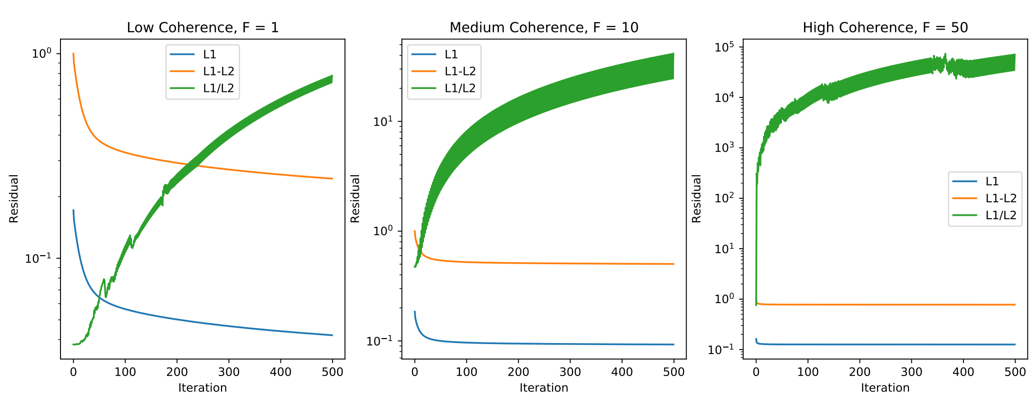
```
## Text(0, 0.5, 'Residual')
```

```python
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000C786208>
```

```python
plt.show()

# RELATIVE ERROR #
# Create a single figure with subplots arranged in 1 row and 3 columns
```



```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# Low Coherence plots
axs[0].plot(hist_admm_lowF['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB63518>]
```

```
axs[0].plot(hist_l1l2_lowF['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB63780>]
```

```
axs[0].plot(hist_admml1dl2_lowF['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB63B38>]
```

```
axs[0].set_title('Low Coherence, F = 1')
```

```
## Text(0.5, 1.0, 'Low Coherence, F = 1')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Relative Error')
```

```
## Text(0, 0.5, 'Relative Error')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med Coherence plots
```

```
## <matplotlib.legend.Legend object at 0x000000002E9919B0>
```

```
axs[1].plot(hist_admm_medF['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB6CA58>]
```

```
axs[1].plot(hist_l1l2_medF['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB6CD30>]
```

```
axs[1].plot(hist_admml1dl2_medF['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB75048>]
```

```
axs[1].set_title('Medium Coherence, F = 10')
```

```
## Text(0.5, 1.0, 'Medium Coherence, F = 10')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Relative Error')
```

```
## Text(0, 0.5, 'Relative Error')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# High Coherence plots
```

```
## <matplotlib.legend.Legend object at 0x000000000FAA4C88>
```

```
axs[2].plot(hist_admm_highF['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB7A400>]
```

```
axs[2].plot(hist_l1l2_highF['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB7A6D8>]
```

```
axs[2].plot(hist_admml1dl2_highF['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB7A9E8>]
```

```
axs[2].set_title('High Coherence, F = 50')
```

```
## Text(0.5, 1.0, 'High Coherence, F = 50')
```

```
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[2].set_ylabel('Relative Error')
```

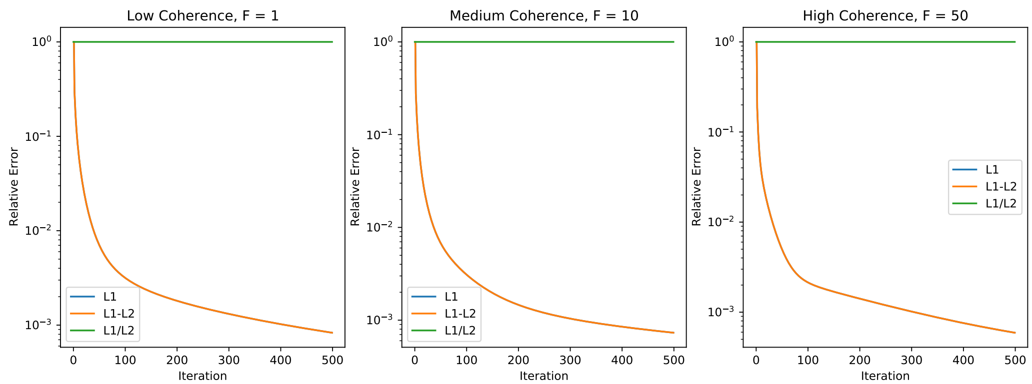
```
## Text(0, 0.5, 'Relative Error')
```

```
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000DB512E8>
```

```
plt.show()

# OBJECTIVE FUNCTIONS #
# Create a single figure with subplots arranged in 1 row and 3 columns
```




```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# Low Coherence plots
axs[0].plot(hist_admm_lowF['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E00C160>]
```

```
axs[0].plot(hist_l1l2_lowF['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E0E1240>]
```

```
axs[0].plot(hist_admml1dl2_lowF['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E0E1550>]
```

```
axs[0].set_title('Low Coherence, F = 1')
```

```
## Text(0.5, 1.0, 'Low Coherence, F = 1')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med Coherence plots
```

```
## <matplotlib.legend.Legend object at 0x000000000C7C82B0>
```

```
axs[1].plot(hist_admm_medF['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF90C50>]
```

```
axs[1].plot(hist_l1l2_medF['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF90CF8>]
```

```
axs[1].plot(hist_admml1dl2_medF['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F9BA5C0>]
```

```
axs[1].set_title('Medium Coherence, F = 10')
```

```
## Text(0.5, 1.0, 'Medium Coherence, F = 10')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# High Coherence plots
```

```
## <matplotlib.legend.Legend object at 0x000000000FAA3128>
```

```
axs[2].plot(hist_admm_highF['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF9D4E0>]
```

```
axs[2].plot(hist_l1l2_highF['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF9D7F0>]
```

```
axs[2].plot(hist_admml1dl2_highF['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF9D860>]
```

```
axs[2].set_title('High Coherence, F = 50')
```

```
## Text(0.5, 1.0, 'High Coherence, F = 50')
```

```
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

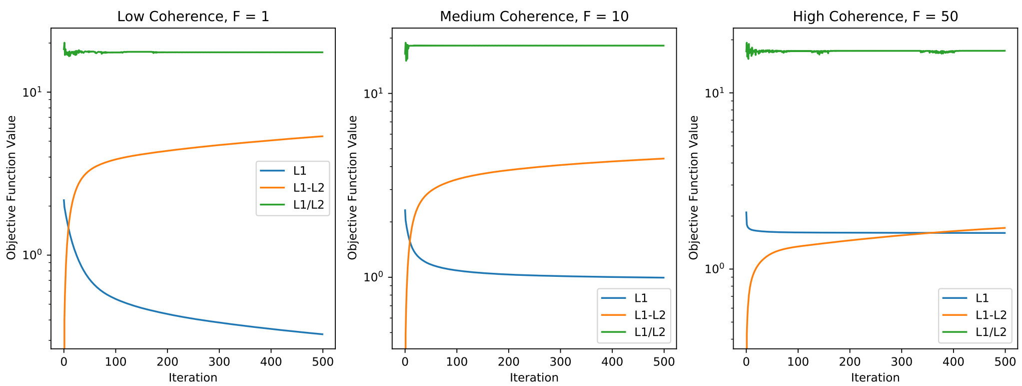
```
axs[2].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000DFF9400>
```

```
plt.show()
```




```
### PROBLEM 2: VARY THE SPARSITY ###
A = np.random.normal(0, 0.1, (M, N))
s_high = 25
s_med = 100
s_low = 250
# Problem setup - generate ground truth xg and b (with noise)
xg = np.zeros(N)
# high sparsity
indices_high = np.random.choice(np.arange(N), replace=False,size=s_high)  # select 's' random indices
xg[indices_high] = np.random.uniform(low=0.01, high=0.05,size=s_high)  # assign small, random non-zero v
b_s_high = np.matmul(A, xg)
b_s_high = b_s_high + 0.01 * np.random.normal(0, 1, b_s_high.shape)
# medium sparsity
indices_med = np.random.choice(np.arange(N), replace=False,size=s_med)  # select 's' random indices
xg[indices_med] = np.random.uniform(low=0.01, high=0.05,size=s_med)  # assign small, random non-zero va
b_s_med = np.matmul(A, xg)
b_s_med = b_s_med + 0.01 * np.random.normal(0, 1, b_s_med.shape)
# low sparsity
indices_low = np.random.choice(np.arange(N), replace=False,size=s_med)  # select 's' random indices
xg[indices_low] = np.random.uniform(low=0.01, high=0.05,size=s_med)  # assign small, random non-zero va
```

```
b_s_low = np.matmul(A, xg)
b_s_low = b_s_low + 0.01 * np.random.normal(0, 1, b_s_low.shape)

# Sparsity - L1 #
x_admm_high, iters_admm_high, hist_admm_high = l1_admm(A,b_s_high,lam,rho,max_iter,abstol,x_true=xg)
x_admm_med, iters_admm_med, hist_admm_med = l1_admm(A,b_s_med,lam,rho,max_iter,abstol,x_true=xg)
x_admm_low, iters_admm_low, hist_admm_low = l1_admm(A,b_s_low,lam,rho,max_iter,abstol,x_true=xg)

# Sparsity - L1-L2 #
x_l1l2_high, iters_l1l2_high, hist_l1l2_high = admm_l1_l2(A, b_s_high, lam, alpha, rho, max_iter, abstol
x_l1l2_med, iters_l1l2_med, hist_l1l2_med = admm_l1_l2(A, b_s_med, lam, alpha, rho, max_iter, abstol, x
x_l1l2_low, iters_l1l2_low, hist_l1l2_low = admm_l1_l2(A, b_s_low, lam, alpha, rho, max_iter, abstol, x

# Sparsity - L1/L2 #
x_admml1dl2_high, iters_admml1dl2_high, hist_admml1dl2_high = admml1ratiol2(A, b_s_high, rho_one, rho_tw
x_admml1dl2_med, iters_admml1dl2_med, hist_admml1dl2_med = admml1ratiol2(A, b_s_med, rho_one, rho_two, r
x_admml1dl2_low, iters_admml1dl2_low, hist_admml1dl2_low = admml1ratiol2(A, b_s_low, rho_one, rho_two, r

# PLOT ALL MODELS TOGETHER #
# Create a single figure with subplots arranged in 1 row and 3 columns
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# RESIDUALS #
# High sparsity plots
axs[0].plot(hist_admm_high['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FACBC50>]
```

```
axs[0].plot(hist_l1l2_high['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FACB4E0>]
```

```
axs[0].plot(hist_admml1dl2_high['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FACB400>]
```

```
axs[0].set_title('High Sparsity (s = 25)')
```

```
## Text(0.5, 1.0, 'High Sparsity (s = 25)')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Residual')
```

```
## Text(0, 0.5, 'Residual')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med sparsity plots
```

```
## <matplotlib.legend.Legend object at 0x000000000DFF2BA8>
```

```
axs[1].plot(hist_admm_med['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DFB5048>]
```

```
axs[1].plot(hist_l1l2_med['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DFB5EF0>]
```

```
axs[1].plot(hist_admml1dl2_med['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF1E240>]
```

```
axs[1].set_title('Medium Sparsity (s=100)')
```

```
## Text(0.5, 1.0, 'Medium Sparsity (s=100)')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Residual')
```

```
## Text(0, 0.5, 'Residual')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# Low sparsity plots
```

```
## <matplotlib.legend.Legend object at 0x000000000E0135C0>
```

```
axs[2].plot(hist_admm_low['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB6CE48>]
```

```
axs[2].plot(hist_l1l2_low['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DFBC438>]
```

```
axs[2].plot(hist_admml1dl2_low['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DFBC1D0>]
```

```
axs[2].set_title('Low Sparsity (s=250)')
```

```
## Text(0.5, 1.0, 'Low Sparsity (s=250)')
```

```
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[2].set_ylabel('Residual')
```

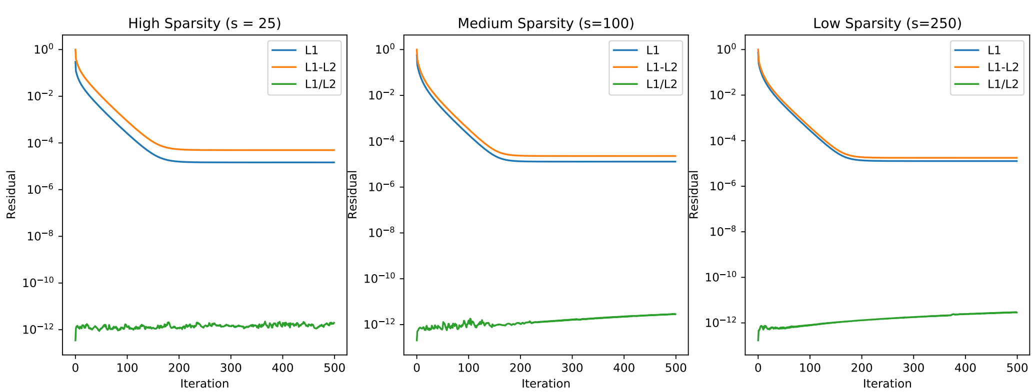
```
## Text(0, 0.5, 'Residual')
```

```
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000E022748>
```

```
plt.show()

# RELATIVE ERROR #
# Create a single figure with subplots arranged in 1 row and 3 columns
```




```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# High sparsity plots
axs[0].plot(hist_admm_high['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E00C2B0>]
```

```
axs[0].plot(hist_l1l2_high['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F9A9DD8>]
```

```
axs[0].plot(hist_admml1dl2_high['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F9A9710>]
```

```
axs[0].set_title('High Sparsity (s=25)')
```

```
## Text(0.5, 1.0, 'High Sparsity (s=25)')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Relative Error')
```

```
## Text(0, 0.5, 'Relative Error')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med sparsity plots
```

```
## <matplotlib.legend.Legend object at 0x000000000DEB2048>
```

```
axs[1].plot(hist_admm_med['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7D5588>]
```

```
axs[1].plot(hist_l1l2_med['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7D5828>]
```

```
axs[1].plot(hist_admml1dl2_med['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FACB278>]
```

```
axs[1].set_title('Medium Sparsity (s=100)')
```

```
## Text(0.5, 1.0, 'Medium Sparsity (s=100)')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Relative Error')
```

```
## Text(0, 0.5, 'Relative Error')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# Low sparsity plots
```

```
## <matplotlib.legend.Legend object at 0x000000000E022710>
```

```
axs[2].plot(hist_admm_low['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F9D3438>]
```

```
axs[2].plot(hist_l1l2_low['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F908518>]
```

```
axs[2].plot(hist_admml1dl2_low['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F9D3EB8>]
```

```
axs[2].set_title('Low sparsity (s=250)')
```

```
## Text(0.5, 1.0, 'Low sparsity (s=250)')
```

```
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[2].set_ylabel('Relative Error')
```

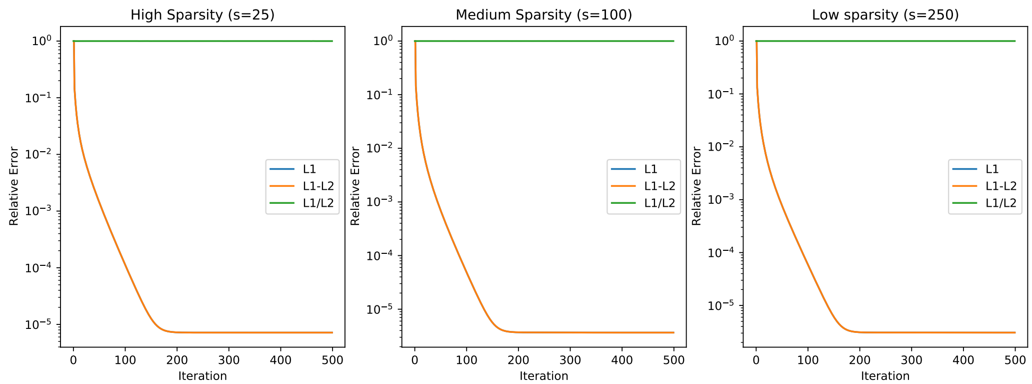
```
## Text(0, 0.5, 'Relative Error')
```

```
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000F7ACA20>
```

```
plt.show()

# OBJECTIVE FUNCTIONS #
# Create a single figure with subplots arranged in 1 row and 3 columns
```




```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# High sparsity plots
axs[0].plot(hist_admm_high['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7D7438>]
```

```
axs[0].plot(hist_l1l2_high['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB6BA90>]
```

```
axs[0].plot(hist_admml1dl2_high['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB6BC50>]
```

```
axs[0].set_title('High Sparsity (s=25)')
```

```
## Text(0.5, 1.0, 'High Sparsity (s=25)')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med sparsity plots
```

```
## <matplotlib.legend.Legend object at 0x000000000DEDCC50>
```

```
axs[1].plot(hist_admm_med['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F9174E0>]
```

```
axs[1].plot(hist_l1l2_med['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E022E80>]
```

```
axs[1].plot(hist_admml1dl2_med['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB6B860>]
```

```
axs[1].set_title('Medium Sparsity (s=100)')
```

```
## Text(0.5, 1.0, 'Medium Sparsity (s=100)')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# Low sparsity plots
```

```
## <matplotlib.legend.Legend object at 0x000000000FB01F28>
```

```
axs[2].plot(hist_admm_low['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F90B3C8>]
```

```
axs[2].plot(hist_l1l2_low['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F90B630>]
```

```
axs[2].plot(hist_admml1dl2_low['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F90BBE0>]
```

```
axs[2].set_title('Low sparsity (s=250)')
```

```
## Text(0.5, 1.0, 'Low sparsity (s=250)')
```

```
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

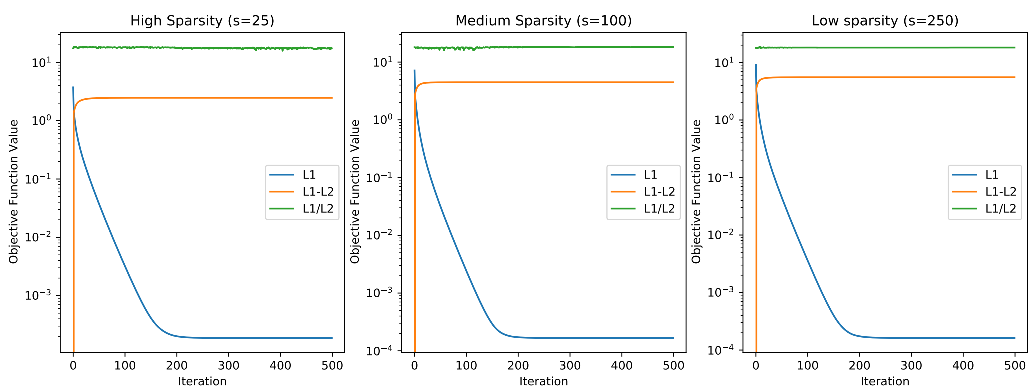
```
axs[2].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000DB4C630>
```

```
plt.show()
```




```
### PROBLEM 3: VARY THE RANGES OF THE NON-ZERO ELEMENTS ###
# Problem setup - generate ground truth xg and b (with noise)
# sparsity fixed at s = 25
xg = np.zeros(N)
# high range
indices_high = np.random.choice(np.arange(N), replace=False,size=s)  # select 's' random indices
xg[indices_high] = np.random.uniform(low=1, high=100,size=s)
b_high = np.matmul(A, xg)
b_high = b_high + 0.01 * np.random.normal(0, 1, b_high.shape)
# medium range
indices_med = np.random.choice(np.arange(N), replace=False,size=s)  # select 's' random indices
xg[indices_med] = np.random.uniform(low=1, high=10,size=s)
b_med = np.matmul(A, xg)
b_med = b_med + 0.01 * np.random.normal(0, 1, b_med.shape)
# low range
indices_low = np.random.choice(np.arange(N), replace=False,size=s)  # select 's' random indices
xg[indices_low] = np.random.uniform(low=1, high=1.5,size=s)
b_low = np.matmul(A, xg)
b_low = b_low + 0.01 * np.random.normal(0, 1, b_low.shape)
```

```
# Range - L1 #
x_admm_hr, iters_admm_hr, hist_admm_hr = l1_admm(A,b_high,lam,rho,max_iter,abstol,x_true=xg)
x_admm_mr, iters_admm_mr, hist_admm_mr = l1_admm(A,b_med,lam,rho,max_iter,abstol,x_true=xg)
x_admm_lr, iters_admm_lr, hist_admm_lr = l1_admm(A,b_low,lam,rho,max_iter,abstol,x_true=xg)

# Range - L1-L2 #
x_l1l2_hr, iters_l1l2_hr, hist_l1l2_hr = admm_l1_l2(A, b_high, lam, alpha, rho, max_iter, abstol, x_tru
x_l1l2_mr, iters_l1l2_mr, hist_l1l2_mr = admm_l1_l2(A, b_med, lam, alpha, rho, max_iter, abstol, x_true
x_l1l2_lr, iters_l1l2_lr, hist_l1l2_lr = admm_l1_l2(A, b_low, lam, alpha, rho, max_iter, abstol, x_true

# Range - L1/L2 #
x_admml1dl2_hr, iters_admml1dl2_hr, hist_admml1dl2_hr = admml1ratiol2(A, b_high, rho_one, rho_two, max_
x_admml1dl2_mr, iters_admml1dl2_mr, hist_admml1dl2_mr = admml1ratiol2(A, b_med, rho_one, rho_two, max_i
x_admml1dl2_lr, iters_admml1dl2_lr, hist_admml1dl2_lr = admml1ratiol2(A, b_low, rho_one, rho_two, max_i

# PLOT ALL MODELS TOGETHER #
# Create a single figure with subplots arranged in 1 row and 3 columns
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# RESIDUALS #
# Wide range plots
axs[0].plot(hist_admm_hr['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DFE13C8>]
```

```
axs[0].plot(hist_l1l2_hr['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E0E1AC8>]
```

```
axs[0].plot(hist_admml1dl2_hr['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E0E15F8>]
```

```
axs[0].set_title('Wide range Residuals, [1,100]')
```

```
## Text(0.5, 1.0, 'Wide range Residuals, [1,100]')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Residual')
```

```
## Text(0, 0.5, 'Residual')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med range plots
```

```
## <matplotlib.legend.Legend object at 0x000000000E0E1198>
```

```
axs[1].plot(hist_admm_mr['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FB0EFD0>]
```

```
axs[1].plot(hist_l1l2_mr['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E0FD630>]
```

```
axs[1].plot(hist_admml1dl2_mr['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E15D978>]
```

```
axs[1].set_title('Medium Range Residuals, [1,10]')
```

```
## Text(0.5, 1.0, 'Medium Range Residuals, [1,10]')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Residual')
```

```
## Text(0, 0.5, 'Residual')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# Low range plots
```

```
## <matplotlib.legend.Legend object at 0x000000000DB7A940>
```

```
axs[2].plot(hist_admm_lr['residual'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E13F390>]
```

```
axs[2].plot(hist_l1l2_lr['residual'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E154240>]
```

```
axs[2].plot(hist_admml1dl2_lr['residual'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E1802B0>]
```

```
axs[2].set_title('Narrow Range Residuals, [1,1.5]')
```

```
## Text(0.5, 1.0, 'Narrow Range Residuals, [1,1.5]')
```

```
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[2].set_ylabel('Residual')
```

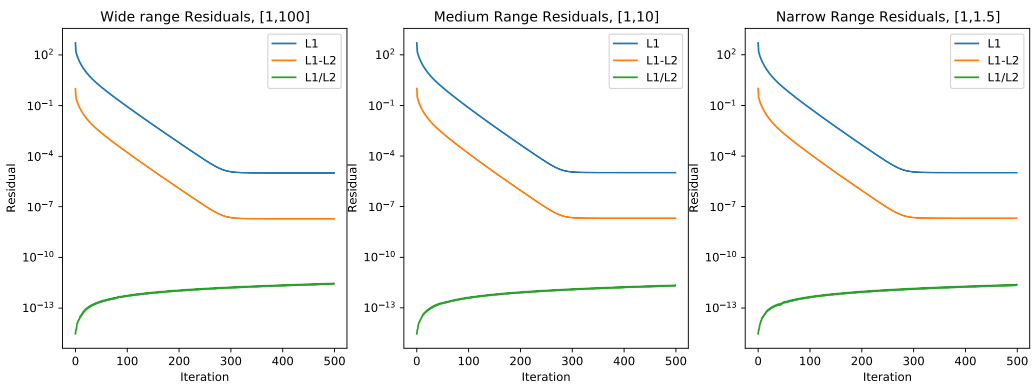
```
## Text(0, 0.5, 'Residual')
```

```
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000FAA2358>
```

```
plt.show()

# RELATIVE ERROR #
```




```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# Wide range plots
axs[0].plot(hist_admm_hr['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FA55EF0>]
```

```
axs[0].plot(hist_l1l2_hr['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FA55550>]
```

```
axs[0].plot(hist_admml1dl2_hr['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DEDC550>]
```

```
axs[0].set_title('Wide range Relative Error, [1,100]')
```

```
## Text(0.5, 1.0, 'Wide range Relative Error, [1,100]')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Relative Error')
```

```
## Text(0, 0.5, 'Relative Error')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med range plots
```

```
## <matplotlib.legend.Legend object at 0x000000000E1C73C8>
```

```
axs[1].plot(hist_admm_mr['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DEDC780>]
```

```
axs[1].plot(hist_l1l2_mr['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DEE3978>]
```

```
axs[1].plot(hist_admml1dl2_mr['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DEE3630>]
```

```
axs[1].set_title('Medium Range Relative Error, [1,10]')
```

```
## Text(0.5, 1.0, 'Medium Range Relative Error, [1,10]')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Relative Error')
```

```
## Text(0, 0.5, 'Relative Error')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# Low range plots
```

```
## <matplotlib.legend.Legend object at 0x000000000E0AF978>
```

```
axs[2].plot(hist_admm_lr['rel_error'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FB01470>]
```

```
axs[2].plot(hist_l1l2_lr['rel_error'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7275F8>]
```

```
axs[2].plot(hist_admml1dl2_lr['rel_error'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000C7270F0>]
```

```
axs[2].set_title('Narrow Range Relative Error, [1,1.5]')
```

```
## Text(0.5, 1.0, 'Narrow Range Relative Error, [1,1.5]')
```

```
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[2].set_ylabel('Relative Error')
```

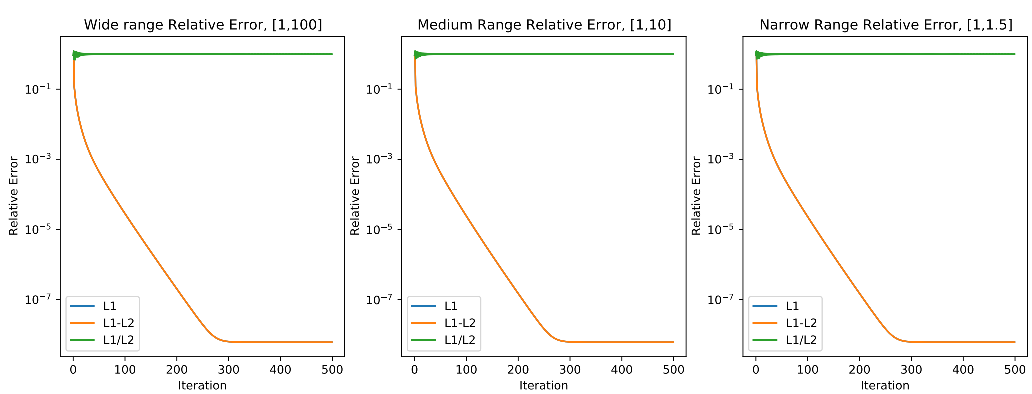
```
## Text(0, 0.5, 'Relative Error')
```

```
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000FA55748>
```

```
plt.show()

# OBJECTIVE FUNCTIONS #
# Create a single figure with subplots arranged in 1 row and 3 columns
```



```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
# Wide range plots
axs[0].plot(hist_admm_hr['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DB7AB70>]
```

```
axs[0].plot(hist_l1l2_hr['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E054D68>]
```

```
axs[0].plot(hist_admml1dl2_hr['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000F9B2C18>]
```

```
axs[0].set_title('Wide Range Objective Functions, [1,100]')
```

```
## Text(0.5, 1.0, 'Wide Range Objective Functions, [1,100]')
```

```
axs[0].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[0].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[0].set_yscale("log")
axs[0].legend()
# Med range plots
```

```
## <matplotlib.legend.Legend object at 0x000000000DB7A630>
```

```
axs[1].plot(hist_admm_mr['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF0B240>]
```

```
axs[1].plot(hist_l1l2_mr['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000E16EA20>]
```

```
axs[1].plot(hist_admml1dl2_mr['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000FA0EC50>]
```

```
axs[1].set_title('Medium Range Objective Functions, [1,10]')
```

```
## Text(0.5, 1.0, 'Medium Range Objective Functions, [1,10]')
```

```
axs[1].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

```
axs[1].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[1].set_yscale("log")
axs[1].legend()
# Narrow range plots
```

```
## <matplotlib.legend.Legend object at 0x000000000FA7D080>
```

```
axs[2].plot(hist_admm_lr['objval'], label="L1")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF3F1D0>]
```

```
axs[2].plot(hist_l1l2_lr['objval'], label="L1-L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF3FCC0>]
```

```
axs[2].plot(hist_admml1dl2_lr['objval'], label="L1/L2")
```

```
## [<matplotlib.lines.Line2D object at 0x000000000DF0B208>]
```

```
axs[2].set_title('Low Range Objective Functions, [1,1.5]')
```

```
## Text(0.5, 1.0, 'Low Range Objective Functions, [1,1.5]')
```

```
axs[2].set_xlabel('Iteration')
```

```
## Text(0.5, 0, 'Iteration')
```

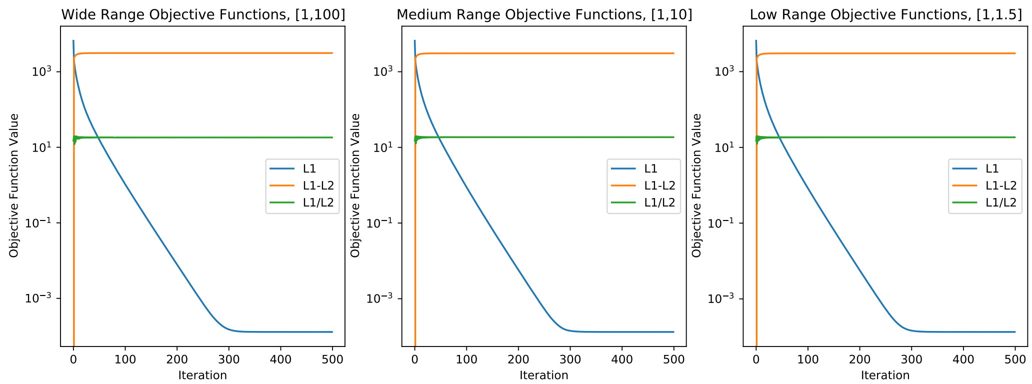
```
axs[2].set_ylabel('Objective Function Value')
```

```
## Text(0, 0.5, 'Objective Function Value')
```

```
axs[2].set_yscale("log")
axs[2].legend()
# Show the figure
```

```
## <matplotlib.legend.Legend object at 0x000000000C741C18>
```

```
plt.show()
```



Wide Range Objective Functions, [1,100]
Medium Range Objective Functions, [1,10]
Low Range Objective Functions, [1,1.5]
Objective Function Value
Iteration
L1
L1-L2
L1/L2