DeepInverse Exercise 4

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import pandas as pd
```

In [2]:

```
def initialization(shape_mtx, sparsity):
    """
    Generate the ground-truth problem setting y=Ax+e. Use zero mean and variance 1 or 0.1 as default.
    :param shape_mtx: The shape of the measurement matrix.
    :param sparsity: The sparsity of the vector x.
    :return: The measurement matrix, the sparse vector and the measurements.
    """
    # Gaussian random matrix with mean 0, variance 1
    A = np.random.normal(0, 1, size=shape_mtx)
    # Gaussian noise with mean 0, variance 0.1
    e = np.random.normal(0, 0.1, size=(shape_mtx[0], 1))
    # Sparse Gaussian signal with mean 0, variance 1
    x = np.random.normal(0, 1, size=(shape_mtx[1], 1))
    nonzero_indices = np.random.randint(0, shape_mtx[1], sparsity)
    mask = np.zeros(x.shape)
    mask[nonzero_indices] = 1
    x = x * mask
    # Measurements
    y = A @ x + e
    return A, x, y
```

In [3]:

```
def soft_thresholding(z, lambd):
    """
    Soft-thresholding operator.
    :param z: The input.
    :param lambd: The hyperparameter controls the soft-thresholding operator.
    :return: The result of soft-thresholding.
    """
    return np.multiply(np.sign(z), np.maximum(np.abs(z) - lambd, 0))
```

In [4]:

```
def g_function(A, x, y, lambd):
    """
    The objective function that to be minimized (least-squares regression problem with l1-norm regularization)
    :param A: The measurement matrix.
    :param x: The sparse vector at iteration k.
    :param y: The measurement.
    :param lambd: The hyperparameter controls the soft-thresholding operator.
    :return: The result of the objective function.
    """
    return 0.5 * np.linalg.norm(A @ x - y, ord=2) ** 2 + lambd * np.linalg.norm(x, ord=1)
```

In [5]:

```
def ISTA(A, y, lambd, tol, max_ite):
    """
    Iterated Soft Thresholding Algorithm (ISTA).
    :param A: The measurement matrix.
    :param y: The measurement.
    :param lambd: The hyperparameter controls the soft-thresholding operator.
    :param tol: The stopping criterion.
    :param max_ite: The max number of iterations.
    :return: The solution of the sparse vector, a list of iteration numbers, and a list of g_function results.
    """
    list_ite = []
    list_g = []
    dim_x = np.shape(A)[1]
    L = np.linalg.norm(A) ** 2  # Lipschitz const
    # Initialization
    step_size = 1 / L
    x_k = np.random.normal(0, 1, size=(dim_x, 1))
    g_k = g_function(A, x_k, y, lambd)
    ite_k = 1
    diff = 1000
    # Loop
    while (abs(diff) > tol) and (ite_k < max_ite):
        gradient_k = A.T @ A @ x_k - A.T @ y
        z_k = x_k - step_size * gradient_k
        x_k_plus_one = soft_thresholding(z_k, lambd / L)
        g_k_plus_one = g_function(A, x_k_plus_one, y, lambd)
        diff = g_k_plus_one - g_k
        list_ite.append(ite_k)
        list_g.append(g_k_plus_one)
        ite_k += 1
        x_k = x_k_plus_one
        g_k = g_k_plus_one
```

```python
    x_final = x_k_plus_one
    return x_final, list_ite, list_g
```

In [6]:
```python
def calculate_diff(list_g):
    """
    Calculate the g(x_k)-g(x_final) at different iterations k.
    :param list_g: The list of g_function results.
    :return: A list of g(x_k)-g(x_final) for different k.
    """
    num_ite = len(list_g)
    list_diff = np.zeros(num_ite)
    last_g = list_g[-1]
    for i in range(num_ite):
        list_diff[i] = list_g[i] - last_g
    return list_diff
```

Problem 1.1

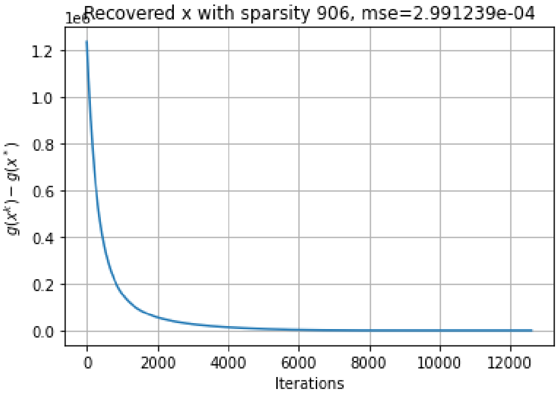
In [7]:
```python
# Parameter
shape_mtx = [2000, 1000]
sparsity = 100
lambd = 1e2
tol = 1e-4
max_ite = 15000
```

In [8]:
```python
# Create ground truth
A, x, y = initialization(shape_mtx, sparsity)

# Perform ISTA
x_final, list_ite, list_g = ISTA(A, y, lambd, tol, max_ite)

# Compute the g(x_k)-g(x_star)
list_diff = calculate_diff(list_g)
sparsity_x_final = shape_mtx[1] - np.sum(x_final == 0)
mse = mean_squared_error(x, x_final)
```

In [21]:
```python
# Plot
plt.plot(list_ite, list_diff)
plt.ylabel('$g(x^k)-g(x^*)$')
plt.xlabel('Iterations')
plt.title('x_rec with k={a}, mse={d:e}'.format(a=sparsity_x_final, d=mse))
plt.grid()
plt.show()
```



Problem 1.2

In [2]:
```python
# Parameter
shape_mtx = [500, 2000]
space_sparsity = [1, 10, 50, 100, 200, 300, 400, 500]
space_lambd = [1e-1, 1, 1e1, 1e2, 2e2, 4e2, 6e2, 8e2, 1e3, 1e4]
tol = 1e-4
max_ite = 15000
```

In [10]:
```python
# Grid search for different sparsity values
search_shape = [len(space_sparsity), len(space_lambd)]
mse = np.zeros(search_shape)
sparsity_rec = np.zeros(search_shape)
i = 0
j = 0
for sparsity in space_sparsity:
    A, x, y = initialization(shape_mtx, sparsity)
    for lambd in space_lambd:
        x_final, list_ite, list_g = ISTA(A, y, lambd, tol, max_ite)
        sparsity_rec[i, j] = shape_mtx[1] - np.sum(x_final == 0)
        mse[i, j] = mean_squared_error(x, x_final)
        j += 1
```

```python
    j = 0
    i += 1
```

In [4]:
```python
print('The MSE results')
print('Rows: The original sparsity of ground-truth x')
print('Columns: The lambda')
pd.DataFrame(mse, space_sparsity, space_lambd)
```

```
The MSE results
Rows: The original sparsity of ground-truth x
Columns: The lambda
```

Out[4]:

	0.1	1.0	10.0	100.0	200.0	400.0	600.0	800.0	1000.0	10000.0
1	0.770398	0.762696	0.546081	0.001510	0.000011	0.000011	0.000011	0.000011	0.000251	0.000251
10	0.789692	0.731377	0.601616	0.009486	0.000608	0.001879	0.002886	0.004160	0.003914	0.003914
50	0.789116	0.717103	0.538783	0.014838	0.003359	0.008419	0.014114	0.018133	0.018788	0.019163
100	0.755682	0.722655	0.576483	0.038103	0.014783	0.022539	0.032402	0.041454	0.040344	0.045654
200	0.843225	0.784812	0.614730	0.067573	0.039166	0.046448	0.060039	0.071826	0.080479	0.092701
300	0.820124	0.785606	0.653962	0.103349	0.070579	0.076079	0.086289	0.099546	0.109967	0.132234
400	0.881314	0.873162	0.724600	0.141609	0.106547	0.112895	0.126358	0.139774	0.151676	0.165090
500	0.899477	0.894862	0.746336	0.192951	0.163251	0.163245	0.180601	0.193520	0.194026	0.223000

In [5]:
```python
print('The sparsity of reconstructed signal x')
print('Rows: The original sparsity of ground-truth x')
print('Columns: The lambda')
pd.DataFrame(sparsity_rec, space_sparsity, space_lambd)
```

```
The sparsity of reconstructed signal x
Rows: The original sparsity of ground-truth x
Columns: The lambda
```

Out[5]:

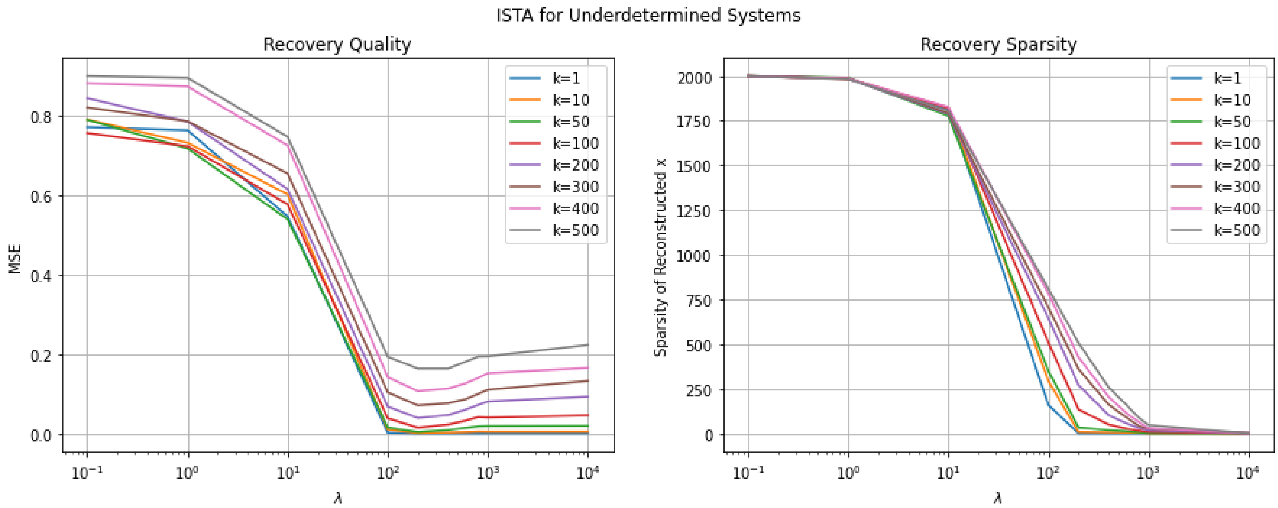
	0.1	1.0	10.0	100.0	200.0	400.0	600.0	800.0	1000.0	10000.0
1	2000.0	1985.0	1812.0	159.0	0.0	0.0	0.0	0.0	0.0	0.0
10	2000.0	1984.0	1823.0	293.0	7.0	5.0	4.0	3.0	0.0	0.0
50	2000.0	1990.0	1776.0	349.0	33.0	18.0	12.0	5.0	1.0	0.0
100	2000.0	1983.0	1821.0	509.0	134.0	50.0	27.0	15.0	8.0	0.0
200	2000.0	1985.0	1787.0	644.0	270.0	102.0	57.0	31.0	16.0	0.0
300	2000.0	1988.0	1792.0	704.0	360.0	161.0	90.0	42.0	26.0	0.0
400	1999.0	1986.0	1826.0	782.0	426.0	204.0	114.0	64.0	29.0	0.0
500	1999.0	1982.0	1805.0	809.0	506.0	258.0	160.0	88.0	47.0	0.0

In [6]:
```python
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))
fig.suptitle('ISTA for Underdetermined Systems')

for sparsity in range(len(space_sparsity)):
    ax1.plot(space_lambd, mse[sparsity, :], label="k={}".format(space_sparsity[sparsity]))
ax1.legend(loc="upper right")
ax1.set_ylabel('MSE')
ax1.set_xlabel('$\lambda$')
ax1.set_title('Recovery Quality')
ax1.grid()
ax1.set_xscale('log')

for sparsity in range(len(space_sparsity)):
    ax2.plot(space_lambd, sparsity_rec[sparsity, :], label="k={}".format(space_sparsity[sparsity]))
ax2.legend(loc="upper right")
ax2.set_ylabel('Sparsity of Reconstructed x')
ax2.set_xlabel('$\lambda$')
ax2.set_title('Recovery Sparsity')
ax2.grid()
ax2.set_xscale('log')

plt.show()
```


ISTA for Underdetermined Systems
Recovery Quality
MSE
λ
k=1
k=10
k=50
k=100
k=200
k=300
k=400
k=500
Recovery Sparsity
Sparsity of Reconstructed x
λ
k=1
k=10
k=50
k=100
k=200
k=300
k=400
k=500