q1

October 11, 2019

Question 1: implement gradient descent using the Lipschitz constant as the step-size for the denoising problem. Use eigsh method from scipy.sparse.linalg to compute the Lipschitz constant. Marks: 10

[ ]:
```
def gradient_descent_lip(x0, epsilon, lambda_, max_iterations):
    # x0: is the initial guess for the x variables
    # epsilon: is the termination tolerance parameter
    # lambda_: is the regularization parameter of the denoising problem.
    # max_iterations: is the maximum number of iterations that you allow the␣
 ↪algorithm to run.

    # Write your code here.
    l_t1 = time.time()
    L = matrix_p2_norm(G(lamb=lambda_), tol=0)
    l_t2 = time.time()
    print(f"Lipschitz Constant = {L}")
    x_updated = x0.copy()
    f_vals = []
    norm_vals = []
    it_t1 = time.time()
    for i in range(1, max_iterations+1):
        current_grad = grad_f(lamb=lambda_, x=x_updated, z_n=noisy_image_vec)
        current_grad_norm = vector_norm(current_grad)
        if current_grad_norm <= epsilon:
            break
        norm_vals.append(current_grad_norm)
        f_vals.append(f(lamb=lambda_, x=x_updated, z_n=noisy_image_vec))
        x_updated = x_updated - (1.0 / L) * current_grad
        f_diff = (f_vals[-1] - f_vals[-2]) if len(f_vals) > 1 else None
        grad_diff = (norm_vals[-1] - norm_vals[-2]) if len(norm_vals) > 1 else␣
 ↪None
        print(f"Step = {i}: Function = {f_vals[-1]}, Function Diff. = ␣
 ↪{f_diff}, Grad. Norm = {norm_vals[-1]}, Grad. Diff. = {grad_diff}")
    it_t2 = time.time()
    print(f"Iterations (Total) time = {it_t2-it_t1}, Lip-Constant time =␣
 ↪{l_t2-l_t1}")
    return x_updated, np.array(f_vals)
```

0.1 Call Gradient Descent

[ ]:
```
# Initialize parameters of gradient descent.
lambda_ = 4
epsilon = 1.0e-2
max_iterations = 2000

# Set x0 equal to the vectorized noisy image.
# Write your code here.
optimized_gd_lip, f_vals_lip = gradient_descent_lip(x0=noisy_image_vec,
                                                     lambda_=lambda_,
                                                     epsilon=epsilon,
                                                    ␣
 ↪max_iterations=max_iterations)
```

0.2 Plot

$$(f(x_k) - f(x^*))/(f(x_0) - f(x^*))$$

vs the iteration counter k, where

$$x^*$$

is the minimizer of the denoising problem, which you can compute by using spsolve, similarly to Assignment 1.

[ ]:
```
# Finding the optimal solution
from scipy.sparse.linalg import spsolve
x_star = spsolve(A=G(lamb=lambda_), b=noisy_image_vec)
f_star = f(lamb=lambda_, x=csr_matrix(x_star).T, z_n=noisy_image_vec)
denoised_image_star = x_star.reshape((m, n), order='F')
plt.figure(1, figsize=(10, 10))
plt.imshow(denoised_image_star, cmap='gray', vmin=0, vmax=255)
plt.show()
```

[ ]:
```
# Plot the relative objective function vs number of iterations.
rate_lip = (f_vals_lip - f_star) / (f_vals_lip[0] - f_star)

denoised_image_gd_lip = optimized_gd_lip.toarray().reshape((m, n), order='F')
plt.figure(1, figsize=(10, 10))
plt.imshow(denoised_image_gd_lip, cmap='gray', vmin=0, vmax=255)
plt.show()

fig = plt.figure(figsize=(8, 6))
plt.plot(rate_lip, label=("Gradient descent + Lip."), linewidth=5.0, color␣
 ↪="black")
plt.legend(prop={'size': 20},loc="upper right")
plt.xlabel("iteration $k$", fontsize=25)
plt.ylabel("Relative distance to opt.", fontsize=25)
plt.grid(linestyle='dashed')
plt.show()
```

```
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(1, 1, 1)
ax.plot(rate_lip, label=("Gradient descent + Lip."), linewidth=5.0, color␣
 ↪="black")
ax.set_yscale('log')
plt.legend(prop={'size': 20},loc="upper right")
plt.xlabel("iteration $k$", fontsize=25)
plt.ylabel("Relative distance to opt. (LOG)", fontsize=25)
plt.grid(linestyle='dashed')
plt.show()
```

[ ]:

```

```