q3

October 11, 2019

0.1 Question 3: implement gradient descent with line-search for the denoising problem. Marks: 15

[ ]:
```
# Write a line-search-ls function here.
def line_search_ls(lambda_, x, f_x, grad_f_x):
    # lambda_: the regularization parameter
    # x: the current estimate of t=he variable
    # f_x: the value of the objective function at x
    # grad_f_x: the gradient of the objective function at x

    # Write your code here.
    alpha = 1.0
    while f(lamb=lambda_, x=x-alpha*grad_f_x, z_n=noisy_image_vec) >= f_x:
        alpha /= 2.0
    return alpha

# Write gradient descent + line-search here.
def gradient_descent_ls(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.
    x_updated = x0.copy()
    f_vals = []
    norm_vals = []
    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))
```

```
        alpha = line_search_ls(lambda_=lambda_, x=x_updated, f_x=f_vals[-1],␣
↪grad_f_x=current_grad)
        x_updated = x_updated - alpha * 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}: alpha = {alpha}, Function = {f_vals[-1]}, Function␣
↪Diff. =  {f_diff}, Grad. Norm = {norm_vals[-1]}, Grad. Diff. = {grad_diff}")
    t2 = time.time()
    print(f"Iterations (Total) time = {t2-t1}")
    return x_updated, np.array(f_vals)
```

0.2 Call Gradient Descent with line-search

[ ]:
```
# 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_ls, f_vals_ls = gradient_descent_ls(x0=noisy_image_vec,
                                                  lambda_=lambda_,
                                                  epsilon=epsilon,
                                                  max_iterations=max_iterations)
```

0.3 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.

[ ]:
```
# Write your code here.

# Plot the rellative objective function vs number of iterations.
rate_ls = (f_vals_ls - f_star) / (f_vals_ls[0] - f_star)

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

print(f"Max. Abs. Diff. between GD-LS and Linear Solver = {np.abs(diff_ls).
↪max()}")
```

```python
fig = plt.figure(figsize=(8, 6))
plt.plot(rate_ls, label=("Gradient descent + LS"), 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_ls, label=("Gradient descent + LS"), 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()
```