q1

October 28, 2019

0.1 Question 1 (8 marks): implement gradient descent with Armijo line-search for the Total-Variation denoising problem. Use the pseudo-Huber function to smooth the problem.

[ ]:
```
def line_search_arm(lambda_, mu, x, f_x, grad_f_x, norm_grad_f_x, gamma):
    # 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
    # norm_grad_f_x: the norm of grad_f_x
    # gamma: parameter of Armijo line-search as was defined in the lectures.

    alpha = 1.0
    loss = gamma * (norm_grad_f_x ** 2.0)
    while f_tv(lamb=lambda_, mu=mu, x=x-alpha*grad_f_x, z_n=noisy_image_vec) >␣
 ↪f_x - alpha * loss:
        alpha /= 2.0
    return alpha

def gradient_descent_arm(x0, epsilon, lambda_, mu, max_iterations, gamma):
    # 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.
    # gamma: parameter of Armijo line-search as was defined in the lectures.

    x_updated = x0.copy().toarray()
    f_vals = []
    norm_vals = []
    t1 = time.time()
    for i in range(1, max_iterations+1):
        current_grad = grad_f_tv2(lamb=lambda_, mu=mu, x=x_updated,␣
 ↪z_n=noisy_image_vec)
        tx = time.time()
        current_grad_norm = np.linalg.norm(current_grad)
        if current_grad_norm <= epsilon:
```

```
            break
        norm_vals.append(current_grad_norm)
        f_vals.append(f_tv(lamb=lambda_, mu=mu, x=x_updated,␣
↪z_n=noisy_image_vec))
        alpha = line_search_arm(lambda_=lambda_, mu=mu, x=x_updated,␣
↪f_x=f_vals[-1], grad_f_x=current_grad, norm_grad_f_x=current_grad_norm,␣
↪gamma=gamma)
        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. Norm. 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 Armijo line-search to denoise the image. Parameter tunning is not given for this assignment. You will have to tune all parameters yourself. Regarding the quality of the output image, pick the λ parameter that makes the error

$$\frac{1}{n^2}\|z_{output} - z_{clean}\|_2$$

as small as possible, where z_{output} is the output of the algorithm. Find λ by trial and error. Note that the smoothing parameter μ affects the quality of the output as well. Pick μ small enough such that the above error does not improve much for smaller values of μ. I will measure the running time only for your chosen parameters λ and μ, therefore, make sure to seperate any code that does trial and error and the code that reports the result for the chosen parameters.

[ ]:
```
# Initialize parameters of gradient descent (search was done separately)
lambda_ = 45
mu = 0.1
epsilon = 1.0e-2
gamma = 0.3
max_iterations = 200

# Set x0 equal to the vectorized noisy image.
# Write your code here.
optimized_gd_arm, f_vals_arm = gradient_descent_arm(x0=noisy_image_vec,
                                                    lambda_=lambda_,
                                                    mu=mu,
                                                    epsilon=epsilon,
                                                    gamma=gamma,
```

```
                                                    ␣
↪max_iterations=max_iterations)
```