Homework 4 (DRAFT)

David L. Olson

April 4, 2025

Contents

1 Problem 1

1.1 Solution

1.1.1 Part A

The `landweber()` function was implemented in MATLAB®, and a screenshot of the code is provided in figure 1.

```
function X = landweber(G, m0, d, omega, k)
% Landweber Iteration for Sttepest Decent Method
%
% Performs "k" iterations of the Landweber interation on Gm = d, with
% initial solution m0, and parameter omega.
%
% On output, X is a matrix of size n by k. The k columns of X are the
% iterates m1, m2, ..., mk.
%
% For convergence, we need 0 < omega < 2/s(1)^2. Outside of this range, you
% can expect the method to diverge.

%% Input and Output Arguments

arguments (Input)
    G (:,:) {isfloat, mustBeReal, mustBeFinite}
    m0 (:,1) {isfloat, mustBeReal, mustBeFinite}
    d (:,1) {isfloat, mustBeReal, mustBeFinite}
    omega (1,1) {isfloat, mustBeReal, mustBeFinite, mustBePositive}
    k (1,1) {mustBeInteger, mustBePositive}
end

arguments (Output)
    X (:,:) {isfloat, mustBeReal, mustBeFinite}
end


%% Error Checking

[m, n] = size(G);
assert(...
    n == length(m0), ...
    "Initial model m0 must have the same number of columns as G!");
assert(...
    m == length(d), ...
    "Data vector d must have the same number of rows as G!");

assert(...
    omega < (2/(svds(G,1)^2)), ...
    "This omega value will not let the iterations converge to a solution!")


%% Implementation

X = zeros(n, k);
X(:,1) = m0;
for ii = 2 : k
    X(:,ii) = X(:,ii-1) - omega * G.' * (G*X(:,ii-1) - d);
end


end
```

Figure 1: MATLAB® Landweber Function Screenshot

1.1.2 Part B

MATLAB ® code was modified from `ex_6_2.m` to load image data, apply noise, and plot the resulting images. The fixed step size

$$\omega = 0.95\frac{2}{s_1^2} = 6190.5$$

was computed using the `svds()` function in MATLAB ® . A total of 500 iterations were running using the `landweber()` function shown in figure 1. Iterations 10, 50, 100, 150, 250, and 500 are evaluated.

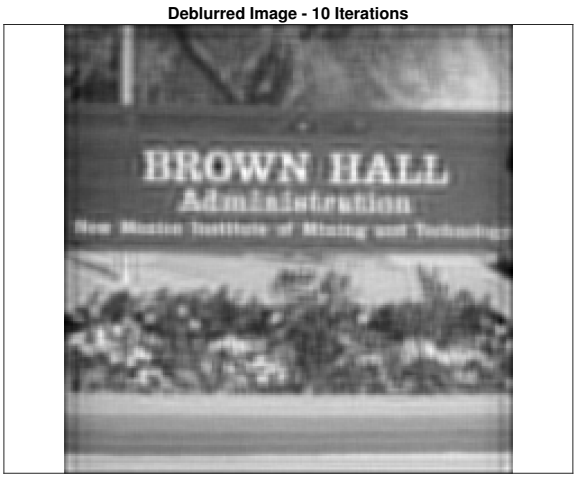

Figure 2: Deblurred Image - 10 Iterations

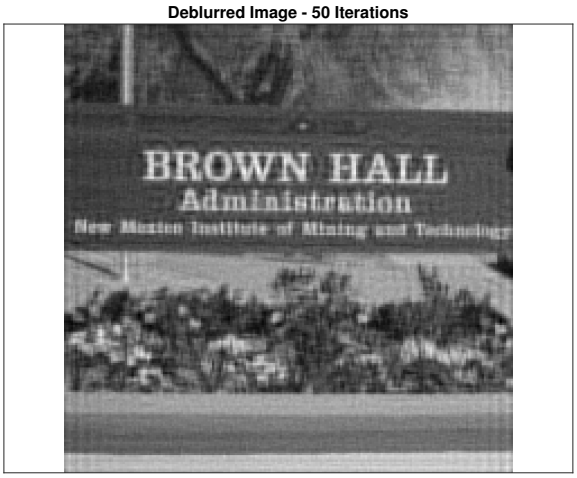

Figure 3: Deblurred Image - 50 Iterations

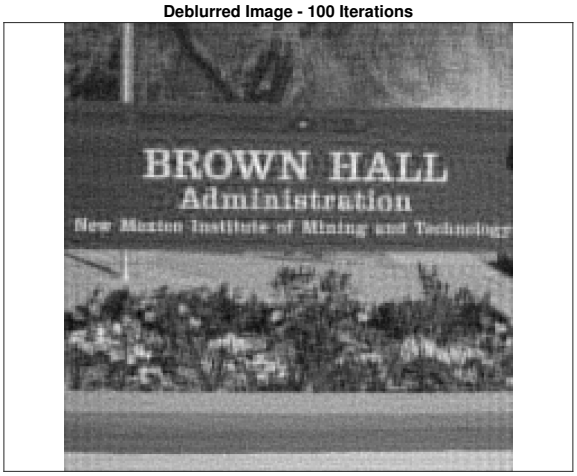

Figure 4: Deblurred Image - 100 Iterations

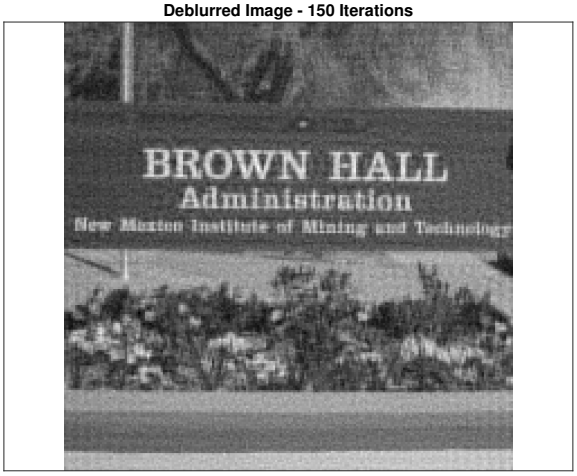

Figure 5: Deblurred Image - 150 Iterations

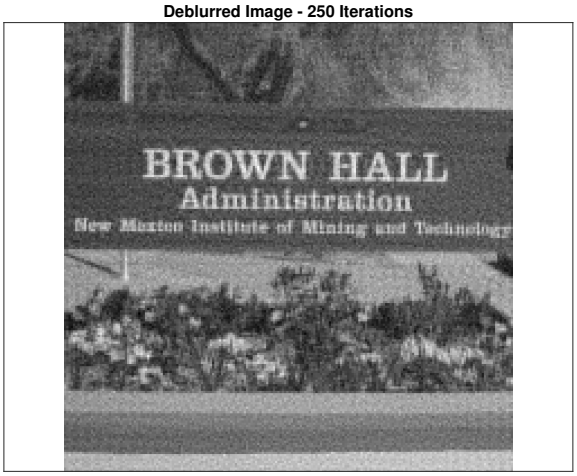

Figure 6: Deblurred Image - 250 Iterations

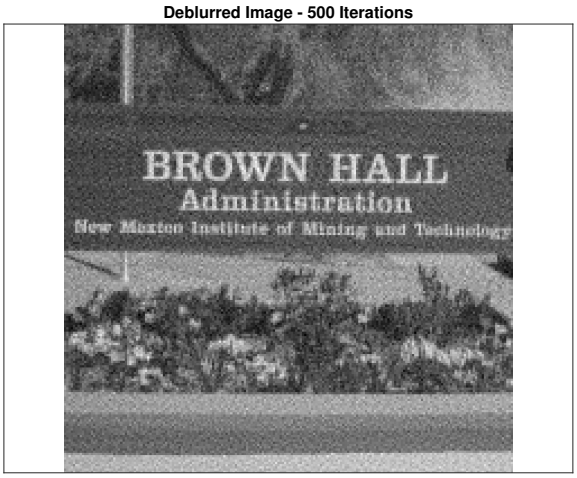

Figure 7: Deblurred Image - 500 Iterations

It becomes clear that there is a "diminishing rate of returns" on the number of iterations used to deblur the image. In my opinion, it seems that the sharpest image appears at 150 iterations. The images after this number of iterations are also as sharp, but from there I can no longer tell a difference.

It is less clear in this PDF, but on my screen in MATLAB it shows just a bit clearer. Unfortunately, you'll just have to take my word for it.