Pr. 1. (sol/hsj62)
If $\boldsymbol{A}$ or $\boldsymbol{B}$ is $\boldsymbol{0}$ then the problem is trivial, so assume they are both nonzero.
Denote the "thin" SVDs of $\boldsymbol{A}$ and $\boldsymbol{B}$ by $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$ and $\boldsymbol{B} = \boldsymbol{X}\boldsymbol{\Omega}\boldsymbol{Y}'$, where we omit the usual thin SVD subscripts for simplicity. Here $\boldsymbol{\Sigma}$ and $\boldsymbol{\Omega}$ are square and symmetric and invertible (but possibly different sizes if $\boldsymbol{A}$ and $\boldsymbol{B}$ have different ranks).
Now $\boldsymbol{A}\boldsymbol{B}' = \boldsymbol{0}$ means $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'\boldsymbol{Y}\boldsymbol{\Omega}\boldsymbol{X}' = \boldsymbol{0}$. Multiplying on the left by $\boldsymbol{\Sigma}^{-1}\boldsymbol{U}'$ and on the right by $\boldsymbol{X}\boldsymbol{\Omega}^{-1}$ shows that $\boldsymbol{V}'\boldsymbol{Y} = \boldsymbol{0}$. So $\begin{bmatrix}\boldsymbol{V} & \boldsymbol{Y}\end{bmatrix}$ is a matrix with orthonormal columns.
Likewise $\boldsymbol{A}'\boldsymbol{B} = \boldsymbol{0}$ leads to $\boldsymbol{U}'\boldsymbol{X} = \boldsymbol{0}$, so $\begin{bmatrix}\boldsymbol{U} & \boldsymbol{X}\end{bmatrix}$ is a matrix with orthonormal columns.
Therefore, the following decomposition is a valid (thin) SVD of $\boldsymbol{A}+\boldsymbol{B}$, to within permutations for sorting the singular values:

$$\boldsymbol{A} + \boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}' + \boldsymbol{X}\boldsymbol{\Omega}\boldsymbol{Y}' = \begin{bmatrix}\boldsymbol{U} & \boldsymbol{X}\end{bmatrix} \begin{bmatrix}\boldsymbol{\Sigma} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Omega}\end{bmatrix} \begin{bmatrix}\boldsymbol{V} & \boldsymbol{U}\end{bmatrix}'.$$

(This "SVD of a sum" can be useful itself.) Thus

$$\|\boldsymbol{A} + \boldsymbol{B}\|_* = \|\boldsymbol{\Sigma}\|_* + \|\boldsymbol{\Omega}\|_* = \|\boldsymbol{A}\|_* + \|\boldsymbol{B}\|_*.$$

Now there is one more subtle point we should address here. When $\boldsymbol{A}$ and $\boldsymbol{B}$ are both $M \times N$, we need to be sure that $\text{size}(\boldsymbol{\Sigma}) + \text{size}(\boldsymbol{\Omega}) \leq \min(M, N)$. Specifically, if $\boldsymbol{U}$ is $M \times r_1$ and $\boldsymbol{X}$ is $M \times r_2$, then we need $r_1 + r_2 \leq M$ for the above thin SVD if $\boldsymbol{A} + \boldsymbol{B}$ to be valid. This inequality is assured by the condition $\boldsymbol{U}'\boldsymbol{X} = \boldsymbol{0}$ because $\boldsymbol{U}$ and $\boldsymbol{X}$ are each orthonormal bases in $\mathbb{F}^M$, so if the sum of their dimensions were to exceed M then their spans would have a nontrivial intersection which would contradict $\boldsymbol{U}'\boldsymbol{X} = \boldsymbol{0}$. Likewise for $\boldsymbol{V}$ and $\boldsymbol{Y}$.

Pr. 2. (sol/hsj75)

(a) Because $\boldsymbol{H}$ is Hermitian, it has real eigenvalues that, by the **Gershgorin disk theorem** satisfy $\lambda(\boldsymbol{H}) \geq h_{ii} - \sum_{j\neq i} |h_{ij}|$, and that latter quantity is nonnegative by the assumed diagonal dominance. Thus, because all of the eigenvalues of $\boldsymbol{H}$ are nonnegative, $\boldsymbol{H}$ is positive semidefinite.

(b) Let $\boldsymbol{H} \triangleq \boldsymbol{D} - \boldsymbol{B} = \text{diag}|\boldsymbol{B}|\,\boldsymbol{1} - \boldsymbol{B}$. Then $h_{ii} = \sum_j |b_{ij}| - b_{ii} = \left(\sum_{j\neq i} |b_{ij}|\right) + (|b_{ii}| - b_{ii}) \geq \sum_{j\neq i} |b_{ij}|$, because $|b| - b \geq 0$. Also for $j \neq i$: $h_{ij} = -b_{ij}$ so $\sum_{j\neq i} |h_{ij}| = \sum_{j\neq i} |b_{ij}| \leq h_{ii}$.
Thus $\boldsymbol{H}$ is diagonally dominant so $\boldsymbol{D} - \boldsymbol{B} \succeq \boldsymbol{0}$, *i.e.*, $\boldsymbol{D} \succeq \boldsymbol{B}$.

Pr. 3. (sol/hsj77)

(a) The gradient is $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) + \delta^2\boldsymbol{x}$

(b)

$$\begin{aligned}\|\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{z})\|_2 &= \left\|\left(\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) + \delta^2\boldsymbol{x}\right) - \left(\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y}) + \delta^2\boldsymbol{z}\right)\right\|_2 \\ &= \left\|(\boldsymbol{A}'\boldsymbol{A} + \delta^2\boldsymbol{I})(\boldsymbol{x} - \boldsymbol{z})\right\|_2 \leq \left\|\boldsymbol{A}'\boldsymbol{A} + \delta^2\boldsymbol{I}\right\|_2 \|\boldsymbol{x} - \boldsymbol{z}\|_2,\end{aligned}$$

where

$$\left\|\boldsymbol{A}'\boldsymbol{A} + \delta^2\boldsymbol{I}\right\|_2 \leq \|\boldsymbol{A}'\boldsymbol{A}\|_2 + \left\|\delta^2\boldsymbol{I}\right\|_2 = \|\boldsymbol{A}'\boldsymbol{A}\|_2 + \delta^2 \leq \|\boldsymbol{A}'\boldsymbol{A}\|_1 + \delta^2$$

(c) When $\boldsymbol{A} = \begin{bmatrix}\boldsymbol{I}_5 \\ \boldsymbol{1}_5\boldsymbol{1}_5'\end{bmatrix}$ and $\delta = 2$, $\boldsymbol{A}'\boldsymbol{A} = \boldsymbol{I}_5 + 5\boldsymbol{1}_5\boldsymbol{1}_5' \implies \|\boldsymbol{A}'\boldsymbol{A}\|_1 = 1 + 25 = 26 \implies \|\boldsymbol{A}'\boldsymbol{A}\|_1 + \delta^2 = 30$

(d) In this case $\|\boldsymbol{A}'\boldsymbol{A}\|_2 = \|\boldsymbol{I}_5 + 5\boldsymbol{1}_5\boldsymbol{1}_5'\|_2 = 1 + 5\,\|\boldsymbol{1}_5\boldsymbol{1}_5'\|_2 = 1 + 5 \cdot 5 = 26 \implies \|\boldsymbol{A}'\boldsymbol{A}\|_1 + \delta^2 = 30$.
Here the "upper bound" is conveniently exactly the same as the Lipschitz constant itself, but this equality is not true in general.

Pr. 4. (sol/hsj31)

(a) $\{\boldsymbol{e}_1, \boldsymbol{e}_4\}$ is an orthonormal basis for the null space of $\boldsymbol{X}$.

(b) $\{\boldsymbol{e}_2, \boldsymbol{e}_3\}$ is an orthonormal basis for the orthogonal complement of the null space of $\boldsymbol{X}$.

(c) The projection of $\boldsymbol{x}$ onto the orthogonal complement of the null space of $\boldsymbol{X}$ is $\boldsymbol{P}_{\mathcal{N}(\boldsymbol{X})^\perp}\boldsymbol{x} = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 0 \end{bmatrix}$

(d) Using: `Y = ones(3,3); (U,s,V) = svd(Y); V0=V[:,2:end]` an orthonormal basis for the null space of $\boldsymbol{Y}$ is $\{\begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}/\sqrt{2}, \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix}/\sqrt{6}\}$

An orthonormal basis for the orthogonal complement of the null space of $\boldsymbol{Y}$ is $\{\boldsymbol{v}_1\}$, $\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}/\sqrt{3}$

The projection of $\boldsymbol{w}$ onto the orthogonal complement of the null space of $\boldsymbol{Y}$ is $\boldsymbol{P}_{\mathcal{N}(\boldsymbol{Y})^\perp}\boldsymbol{w} = \boldsymbol{v}_1(\boldsymbol{v}_1'\boldsymbol{w}) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \frac{6}{\sqrt{3}}$

(e) A possible `Julia` implementation is

```
function orthcompnull(A, x)
#
# Syntax:       z = orthcompnull(A, x)
#
# Inputs:       A : m x n matrix
#               x : vector of length n
#
# Outputs:      z : vector of length n
#
# Description:  Projects x onto the orthogonal complement of the null space
#               of the input matrix A
#
# For full credit, your solution should be computationally efficient.

# Parse inputs
x = vec(x) # make sure it is a column

(~, s, V) = svd(A)
r = rank(A) # unfortunate redundancy with svdvals, could avoid by using s

# orthonormal basis for orthogonal complement of the null space of A:
Vr = V[:,1:r]

return Vr * (Vr' * x)

end
```

The above solution uses a `svd` call. The SVD itself calls a **QR decomposition** that is related to **Gram-Schmidt orthogonalization.** It is likely that an even more efficient solution exists by using the QR decomposition directly to find the basis $\boldsymbol{V}_r$ for the orthogonal complement of the null space of $\boldsymbol{A}$. A QR approach is not required for full credit because we have not covered the QR decomposition.

Graders: accept solutions that use QR instead of SVD, as long as no `inv` or `pinv` or other expensive operations are used and as long as the basis matrix $\boldsymbol{V}_r$ is used efficiently with parentheses like this: `Vr * (Vr' * x)`

Because $\mathcal{N}^\perp(\boldsymbol{A}) = \mathcal{R}(\boldsymbol{A}')$, having a basis $\boldsymbol{V}_r$ for the range space of $\boldsymbol{A}'$ suffices. The following code also passes.

```
function orthcompnull(A,x)
(Q,~) = qr(A') # QR approach to getting basis for range(A')
return Q * (Q' * x);
end
```

Pr. 5. (sol/hsj76)

(a) A possible `Julia` implementation is

```
function lssd(A, b, x0=0, nIters=10)
#
# Syntax:       x = lssd(A, b, x0, nIters)
#
# Inputs:       A is a m x n matrix
#
#               b is a vector of length m
#
#               x0 is the initial starting vector (of length n) to use
#               (x0 is optional; the default is to initialize with 0)
#
#               nIters is the number of iterations to perform
#               (nIters is optional; the default value is 10)
#
# Outputs:      x is a vector of length n containing the approximate solution
#
# Description:  Performs steepest descent to solve the least squares problem
#               \min_x \|b - A x\|_2
#
# Notes:
#               Because this is a quadratic cost function, there is a
#               closed-form solution for the step size each iteration,
#               so no "line search" procedure is needed.
#
#               A full-credit solution uses only *one* multiply by A
#               and one by A' per iteration.

# Parse inputs
b = vec(b) # make sure it is a column

if x0 == 0
    x0 = zeros(size(A,2))
else
    x0 = vec(x0) # ensure it is a column
end

# steepest descent
x = x0
Ax = A * x # initialize A * x

for _ in 1:nIters
    g = A' * (Ax - b) # gradient
    d = - g # search direction: negative gradient
    Ad = A * d
    normAd = norm(Ad)
    if normAd == 0 # done!
        return x
    else
        step = -d' * g / normAd.^2
    end
    x += step * d
    Ax += step * Ad
end

return x
end
```

(b) Figure 1 shows a plot of $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|$ versus iteration k for one realization. Clearly the $\boldsymbol{x}_k$ iterates are converging to $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{b}$.

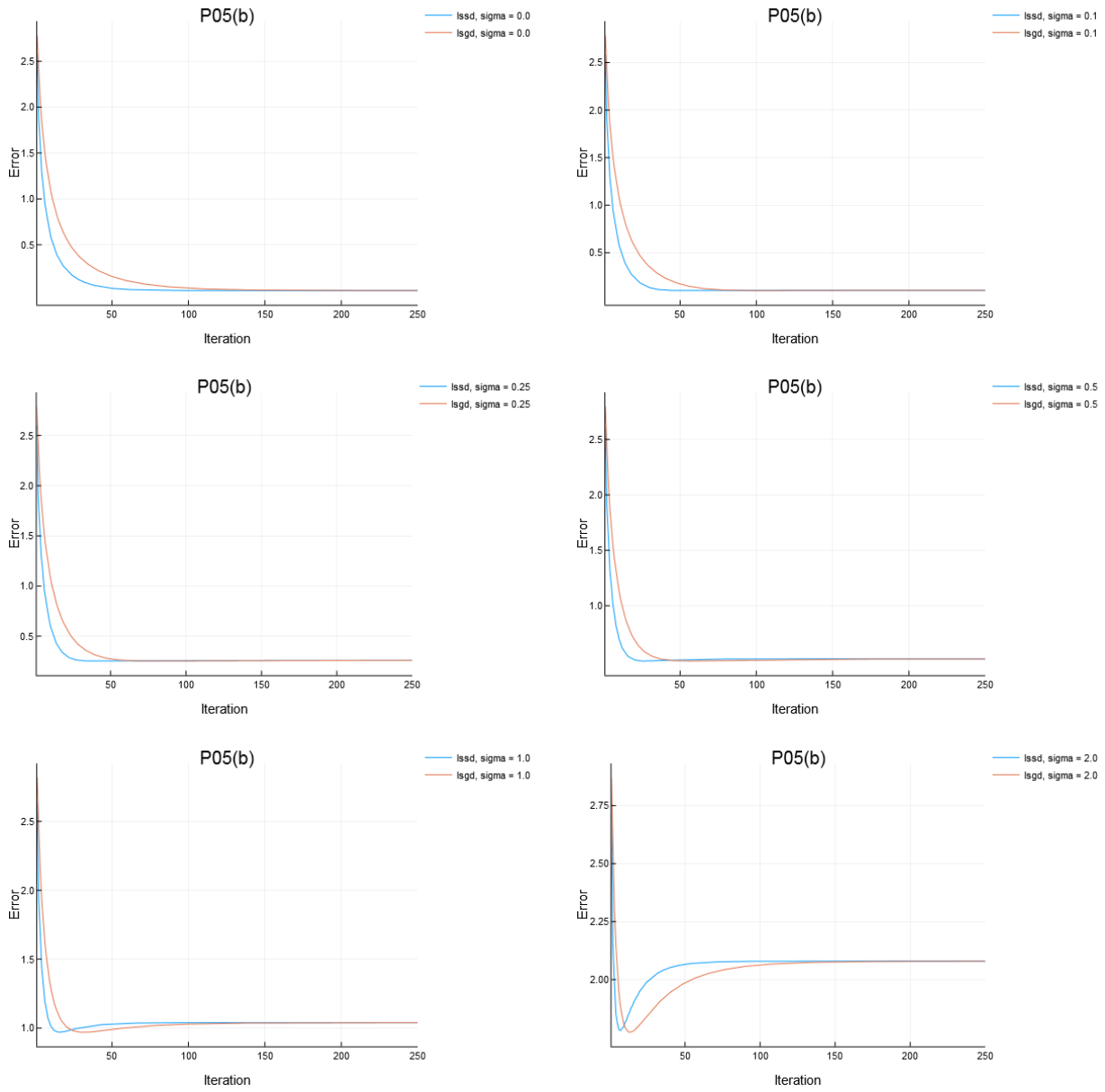


FIGURE 1. Convergence of gradient descent and steepest descent for least squares problems with different noise levels. Steepest descent converges faster than GD for these quadratic problems.

Pr. 6. (sol/hs101)

(a) Recall that $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I}_n \otimes \boldsymbol{D}_m \\ \boldsymbol{D}_n \otimes \boldsymbol{I}_m \end{bmatrix}$ so $\boldsymbol{A}'\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I}_n \otimes \boldsymbol{D}_m' & \boldsymbol{D}_n' \otimes \boldsymbol{I}_m \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_n \otimes \boldsymbol{D}_m \\ \boldsymbol{D}_n \otimes \boldsymbol{I}_m \end{bmatrix} = \boldsymbol{I}_n \otimes (\boldsymbol{D}_m'\boldsymbol{D}_m) + (\boldsymbol{D}_n'\boldsymbol{D}_n) \otimes \boldsymbol{I}_m$ and using the triangle inequality: $\|\boldsymbol{A}'\boldsymbol{A}\|_\infty = \|\boldsymbol{I}_n \otimes (\boldsymbol{D}_m'\boldsymbol{D}_m) + (\boldsymbol{D}_n'\boldsymbol{D}_n) \otimes \boldsymbol{I}_m\|_\infty \le \|\boldsymbol{I}_n \otimes (\boldsymbol{D}_m'\boldsymbol{D}_m)\|_\infty + \|(\boldsymbol{D}_n'\boldsymbol{D}_n) \otimes \boldsymbol{I}_m\|_\infty = \|\boldsymbol{D}_m'\boldsymbol{D}_m\|_\infty + \|\boldsymbol{D}_n'\boldsymbol{D}_n\|_\infty = 4 + 4 = 8$, because a typical absolute row sum of $\boldsymbol{D}_m'\boldsymbol{D}_m$ is $|-1| + 2 + |-1| = 4$.
So $b = 8$ is an upper bound on $\sigma_1^2(\boldsymbol{A})$.
In fact, one can show that for periodic boundary conditions, where $\boldsymbol{A}$ is circulant, $\sigma_1^2(\boldsymbol{A}) = 8$ exactly.
You can verify that numerically using some small values for m and n.

(b) Running OGM produced the following results:

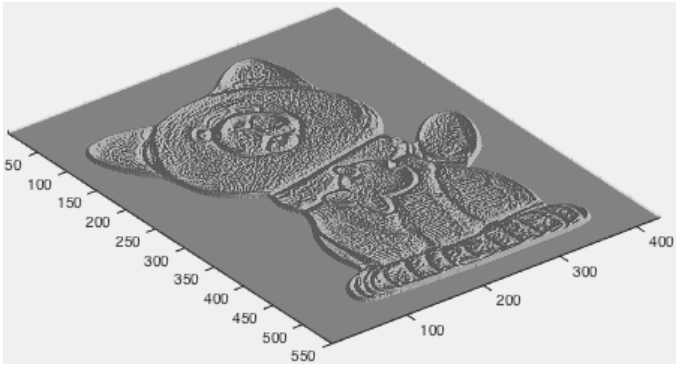


FIGURE 2. Surface Produced by OGM after 5 Iterations

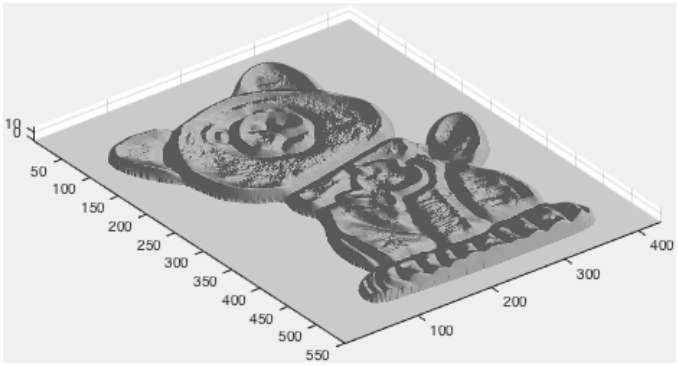


FIGURE 3. Surface Produced by OGM after 25 Iterations

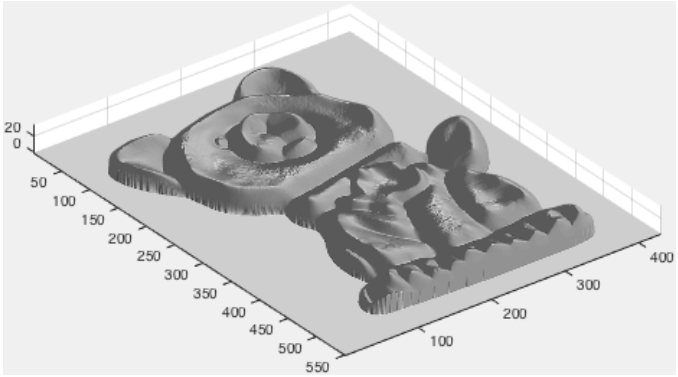


FIGURE 4. Surface Produced by OGM after 50 Iterations

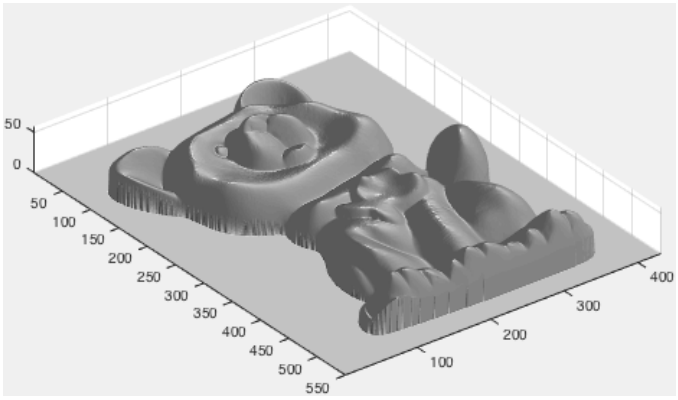


FIGURE 5. Surface Produced by OGM after 100 Iterations

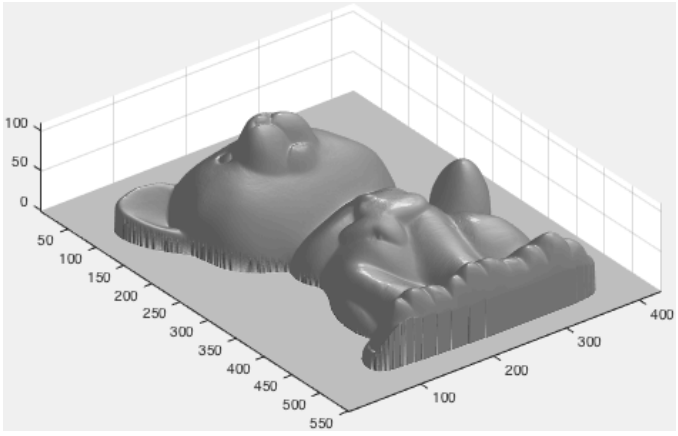


FIGURE 6. Surface Produced by OGM after 250 Iterations

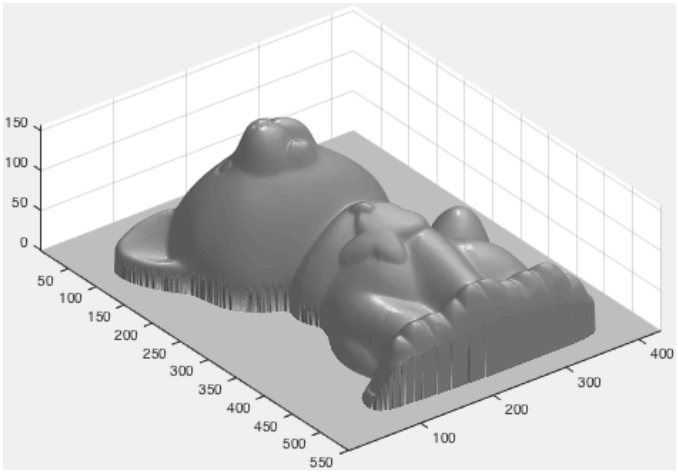


FIGURE 7. Surface Produced by OGM after 500 Iterations

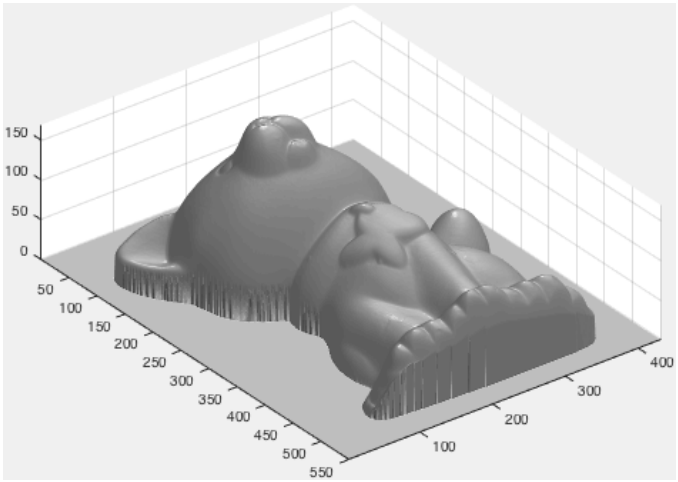


FIGURE 8. Surface Produced by OGM after 1000 Iterations

Comparing these results to the performance of accelerated gradient, we see that OGM performs slightly better, converging to the final surface in fewer iterations than accelerated gradient descent. After 500 iterations, the surface produced by OGM is extremely close to the surface produced by `lsqr`. When running accelerated gradient descent, after 500 iterations the surface produced is still clearly different from the final surface produced and appears less filled out. We observe a similar difference when comparing the results after 100 and 250 iterations. We previously observed that accelerated gradient descent significantly outperforms standard gradient and, as such, OGM also outperforms standard gradient descent.

Pr. 7. (sol/hs052)

(a) A possible `Julia` implementation is

```
function dist2locs(D, d)
#
# Syntax:       Xr = dist2locs(D, d)
#
# Inputs:       D is an n x n matrix such that D[i, j] is the distance from
#               object i to object j
#
#               d is the desired embedding dimension. The actual d value used
#               should be capped at the number of positive eigenvalues of
#               C * C'
#
# Outputs:      Xr is an n x d matrix whose rows contains the relative
#               coordinates of the n objects
#
# Note:         MDS is only unique up to rotation and translation, so we
#               enforce the following conventions on Xr:
#
#       [ORDER] Xr[:, i] corresponds to ith largest eigenpair of C * C'
#         [EIG] Actual d used is min(d, # positive eigenvalues of C * C')
#      [CENTER] The rows of Xr have zero centroid
#        [SIGN] The largest magnitude element of Xr[:, i] is positive
#

    # Parse inputs
    n = size(D, 1)

    # Compute correlation matrix
    S = D .* D
    P = eye(n) - ones(n, n) / n
    CCt = -0.5 * (P * S * P)
    CCt = 0.5 * (CCt + CCt') # Force symmetry

    # Compute relative coordinates
    e, V = eig(CCt)
    d = min(d, countnz(e .> 0)) # Apply [EIG]
    idx = sortperm(e, rev=true)
    Xr = V[:, idx[1:d]] * diagm(sqrt(e[idx[1:d]])) # Apply [ORDER]

    # Apply [CENTER]
    Xr .-= mean(Xr, 1)

    # Apply [SIGN]
    Xr .*= sign(Xr[findmax(abs(Xr), 1)[2]])

    return Xr
end
```

(b) Running the notebook produces Figures 9 and 10. Clearly the MDS coordinates derived from the `Dgeo` matrix are closer to the true geographic coordinates than those derived from the `Dgoogle` matrix. From Figure 10, the `Dgoogle` coordinates for Detroit, MI and Clevland, OH are the least accurate. The `Dgoogle` coordinates are less accurate because they are based on driving distances between cities, which often differ significantly from "as the crow flies" (straight line) distances between the geographic coordinates of the cities due to mountains, rivers, winding roads, etc. In contrast, the `Dgeo` coordinates are quite accurate because they are derived from the Haversine (spherical) distances between the true geographic coordinates of the cities.
Figure 11 plots the eigenvalues of the $\boldsymbol{C}\boldsymbol{C}^T$ matrices for the `Dgoogle` and `Dgeo` distances. In the `Dgeo` case, $\boldsymbol{C}\boldsymbol{C}^T$ has exactly two positive eigenvalues, so embedding it in a two-dimensional coordinate system is a good fit. This makes sense because the distances were derived from latitude and longitude coordinates, which are two-dimensional.

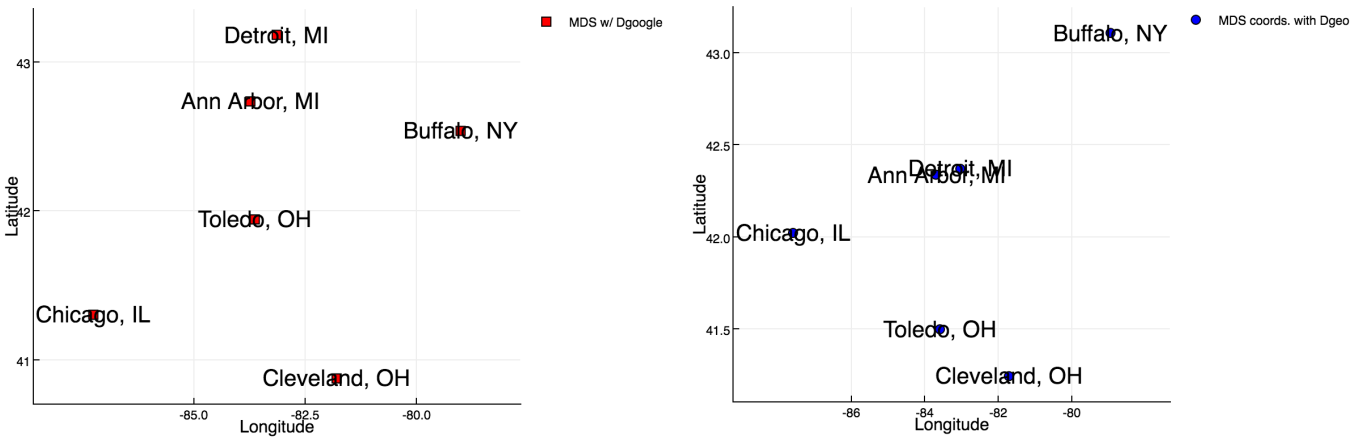


FIGURE 9. MDS coordinates recovered from Google Maps driving distances (left) and geo-distances (right).

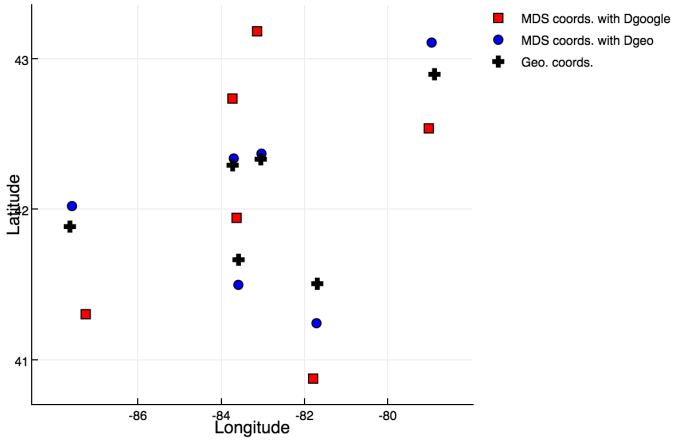
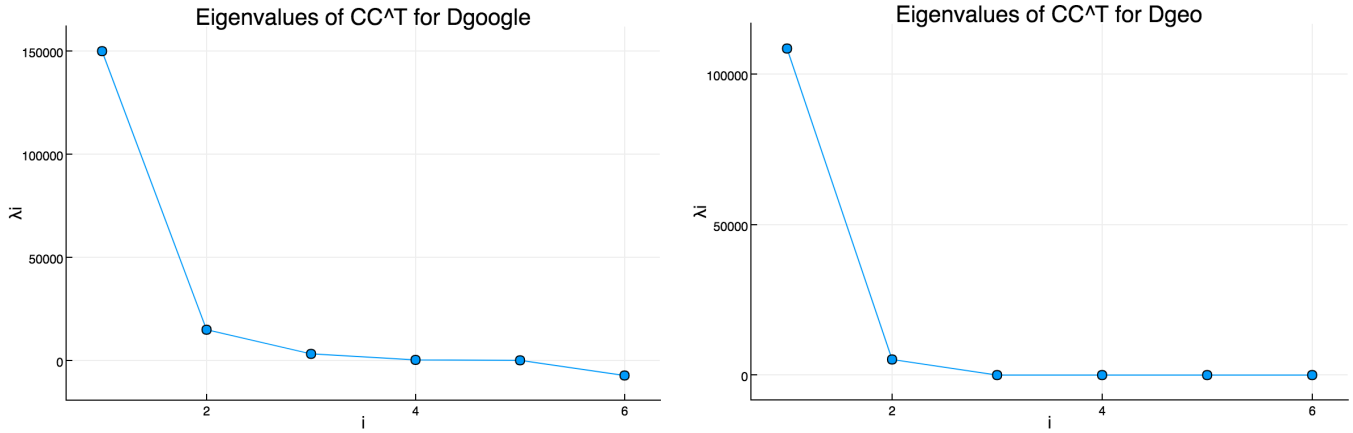


FIGURE 10. Comparison of MDS coordinates and ground truth GPS coordinates.



(A) The top 4 eigenvalues are positive, the fifth is zero, and the sixth is negative.

(B) The top 2 eigenvalues are positive, and the rest are zero.

FIGURE 11. Eigenvalues of the $\boldsymbol{CC}^T$ matrices.