# A reduced order modeling method for improving online computation time and accuracy using mesh coarsening

Tina White

## 1. Introduction

Reduced order modeling (ROM) methods approximate a solution to a high dimensional model L(w, parameters) = 0 of order N, where L represents a partial or ordinary differential equation, and where w represents a state vector. Given a well-chosen reduced order basis (ROB), the state vector can be approximated by w ~= dot(Vk,y) where Vk is the reduced order basis and y is a vector of order n << N. The vector y is thus a representation of w in a much lower dimensional state space. Within this space, a low dimensional model L(Vy, parameters) ~= 0 can be solved as a residual minimization problem.

Construction of a representative reduced order basis Vk is crucial to the accuracy of the model. In one method, Vk is constructed based on previously computed solutions to the high dimensional model, which are then clustered. Within each of these clusters, a proper orthogonal decomposition is performed to extract a number of dimensions in the space, n, with the highest energy. An improvement is proposed to supplement this method, improving both its online speed of computation and accuracy.

The basic idea is that while running the online lower dimensional solver, the mesh that will be used can be reduced in size based on information already known from the pre-computed snapshots. This method is expected to be especially useful for ROMs of moving discontinuities like shocks. This is because a very fine mesh is required proximate the shock to resolve the discontinuity, while far from the shock the fine mesh is excessive and computationally expensive. The original mesh will need to be fine throughout the domain where a shock is expected. However, if the shock is moving, this domain will also move over time. Within individual clusters in time, this domain will be considerably smaller than it would be over the entire original mesh, as shown in the Berger's equation example test case in Figure 1.

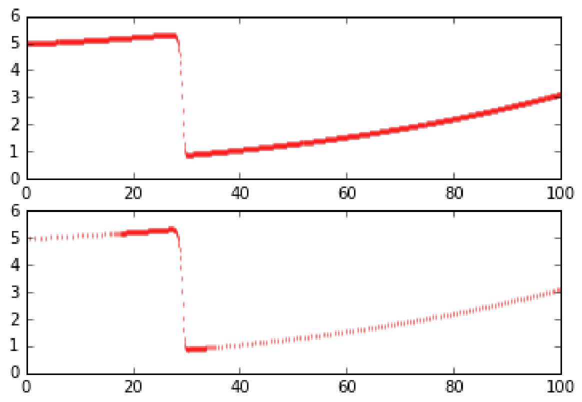


Figure 1: A Berger's equaion solution vector represented on a mesh size 1000 reduced to mesh size 289 with mesh points concentrated around shock location

## 2. Method

The mesh coarsening method requires increased offline computation time, which is mostly spent in (1) generating new coarse meshes and (2) building up the additional information required to work with the new reduced order bases (matrices, vectors, interpolations). A summary of the baseline offline procedure for calculating the reduced order basis (Vk) in each cluster is described below. A summary and detailed description of the new mesh coarsening procedure follows. The offline procedure requires four additional steps shown in bold.

OFFLINE PROCEDURE SUMMARY (BASELINE)

1. Start with a snapshot matrix and initial mesh from high dimensional model
2. Find clusters among the columns of the snapshot matrix
3. Perform POD on clusters to get Vk

OFFLINE PROCEDURE SUMMARY (MESH COARSENED)

1. Start with a snapshot matrix and initial mesh from high dimensional model
2. Find clusters among the columns of the snapshot matrix
3. **Create new, coarser mesh within each cluster**
4. **Calculate interpolation matrices for fast online access to linear interpolations 1. between initial mesh and coarsened meshes and 2. between coarsened meshes**
5. **Reconstruct original clusters and cluster centers in new mesh coordinates using interpolation matrices**
6. Perform POD on coarsened clusters to get Vk
7. **Pre-calculate indexed matrices B (and C) for faster online computation**

OFFLINE PROCEDURE DETAILS (MESH COARSENED)

1. Start with a snapshot matrix and initial mesh from high dimensional model
    a. Snapshot matrix = snaps (size Q x M)
    b. Initial mesh nodes = nodes (size M+1)
    c. Initial mesh element centers = elems (size M)
2. Find clusters among the the columns of the snapshot matrix
    a. nclusters = 7
    b. ck, Xk = k-means_cluster_me(snaps, nclusters)
    c. ck = List of k cluster centers (size M)
    d. Xk = List of k clustered matrices, total dimension same as snapshot matrix (total size Q x M, individual size N[k] x M)
3. Create new, coarser mesh within each cluster
    a. Test how far apart mesh points can be while still linear interpolating between them and compare the interpolated results to the initial mesh solutions in this cluster.
    b. If the error between all the interpolated solutions and the initial mesh solutions within this cluster is below a certain tolerance, the mesh can be coarsened.
    c. Two new parameters can be controlled 1. tolerance (tol) and 2. max mesh stretching (stretching).
    d. Outputs nodes for each mesh and element centers
        i. nodes_new, elems_new = coursen(Xk[k], nodes, tol, stretching)
        ii. nodes_new (size m[k]+1) and elems_new (size m[k])

4. Calculate interpolation matrices for fast online access to linear interpolations 1. between initial mesh and coarsened meshes and 2. between coarsened meshes
    a. Ai[k] = matrix_interpolation(elems_new[k], elems)
    b. Ar[k] = matrix_interpolation(elems, elems_new[k])
    c. Ai (size M x m[k])
    d. A[k,k] = matrix_interpolation(elems_new[k], elems_new[k])
    e. A (size m[k] x m[k])
    f. matrix_interpolation = a short algorithm to express any interpolation as dot product between a matrix A and a vector of locations loc1, given the original location vector loc1, and the new location vector loc2,
        i. A = matrix_interpolation(loc2, loc1)
        ii. If the same locations exists in both meshes, A[i,j] = 1
            1. A[i,loc1.index(loc2[i])] = 1
        iii. Otherwise, interpolate between two nearest locations on either side (j1 and j2)
            1. A[i,j1] = 1-(loc2[i]-loc1[j1])/(loc1[j2]-loc1[j1])
            2. A[i,j2] = (loc2[i]-loc1[j1])/(loc1[j2]-loc1[j1])
5. Reconstruct original clusters and cluster centers in new coordinates using interpolation matrices
    a. Xk[k] = dot(Ai[k], Xk[k])
    b. Xk (size m[k] x N[k])
    c. ck[k] = dot(Ai[k],ck[k])
    d. ck (size m[k])
6. Perform POD on coarsened clusters
    a. N = npod = 40
    b. Vk[k] = pod(Xk_coarse[k], npod)
    c. Vk (size m[k] by n)
7. Pre-calculate indexed matrices B (and C) for faster online computation
    a. B = dot(A[k][k], Vk[k])
    b. C = dot(Vk[k].T,B[k][k])

It’s also important to note that the details of creating the coarse meshes in step 3 have some impact on the results, and their current implementation could be improved, but those details are not outlined in this report.

Next, in order to solve the reduced order model at each time step, the inputs to the online solver require modifications. These modifications are improvements, which increase both the speed and accuracy of the online solver. Below is the original procedure for calculating the inputs to the ROM, followed by the updated procedure for mesh coarsened inputs.

ONLINE PROCEDURE SUMMARY (BASELINE)

1. Use ODE solver to calculate y_next where solution vector y is size n x 1, Vk is size M x n, and elems is size M
2. Determine which cluster the solution is nearest to currently
3. If the cluster changes, project solution vector y onto next basis Vk[k_next]

ONLINE PROCEDURE DETAILS (BASELINE)

4. y_next = ode_solver (y_current, Vk[k_current], elems)
5. k_next = np.argmin([np.mean(np.square(dot(Vk[k],y_next)-ck[k]))
6. y_next = dot(Vk[k_next].T, dot(Vk[k_current], y_next))

ONLINE PROCEDURE SUMMARY (MESH COARSENED)

1. Use ODE solver to calculate y_next where solution vector y is size n x 1, Vk is size m[k] x n, and elems is size m[k]
2. Determine which cluster the solution is nearest to currently
3. If the cluster changes, project solution vector y onto next basis Vk[k_next]

ONLINE PROCEDURE DETAILS (MESH COARSENED)

1. y_next = ode_solver (y_current, Vk[k_current], elems[k_current])
2. k_next = np.argmin([np.mean(np.square(dot(Bk[k_current][k],y_next)-ck[k]))
3. y_next = dot(Ck[k_current][k_next], y_next)

Once the solver has finished, the solution for the ROM, w, will be equal to approximately w = dot(Vk, y) at each time step. If using mesh coarsening, the solution can still be expressed as w = dot(Vk, y). However, the solution can also be expressed on the original mesh if required using w = dot(Ar, dot(Vk, y)) where Ar is the interpolation matrix calculated in the Offline Procedure Step 4.

One benefit of this method is that the linear interpolation can easily be expressed as a matrix multiplication, which greatly simplifies its implementation. Only the projections onto the reduced order matrices Vk need to be replaced with the matrices Bk and Ck, where Bk and Ck are pre-calculated offline. Since A is the interpolation matrix, and Bk = dot(A[k_current][k], Vk[k]), the term dot(Bk[k_current][k],y_next) represents the projection of current state vector into the real, physical solution space. Then, since Ck = dot(Vk[k].T,B[k][k]), the term dot(Ck[k_current][k_next], y_next) represents the projection of the current state vector into the next reduced order basis. So, Bk and Ck in the new method serve similar functions as Vk in the original method, only they need to be pre-computed. They are also smaller matrices, so the online computation of the projections using the new method is actually faster than in the original method.

## 3. Results

The Berger's equation test case is used to demonstrate the effectiveness of the method. First, Figure 2 shows a few solution vectors, w, selected from a pre-computed snapshot matrix of Berger's equation solutions. In his solution, the shock discontinuity moves from left to right while decreasing in strength.

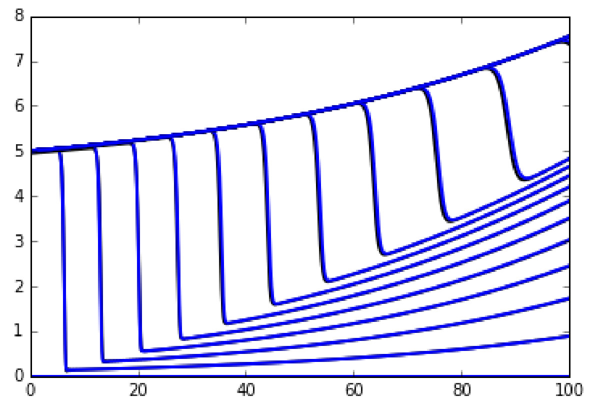
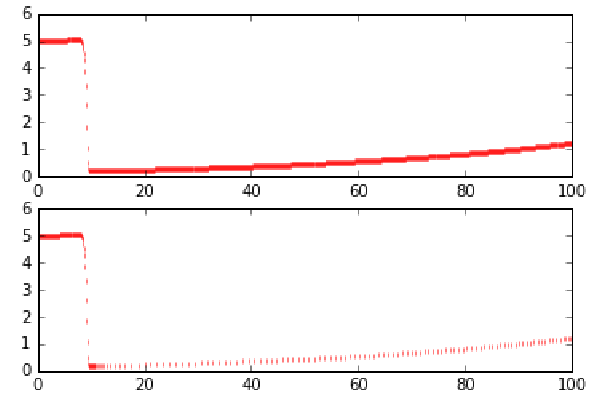
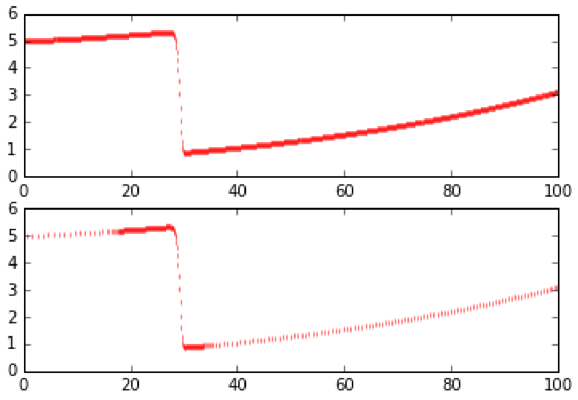


Figure 2: Berger's equation solutions

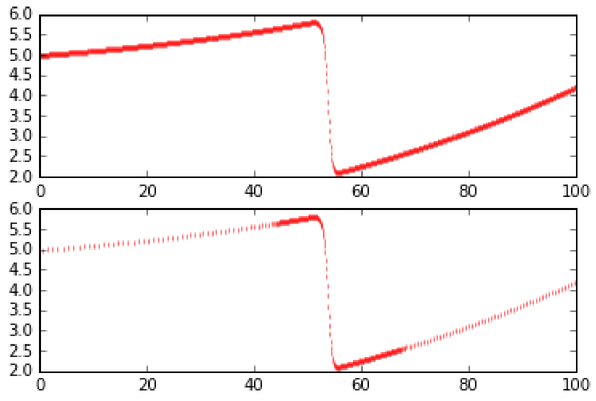
Next, mesh coarsening was applied clusters of these snapshots. The solutions are shown on both the original and coarsened mesh for four example snapshots in different clusters.



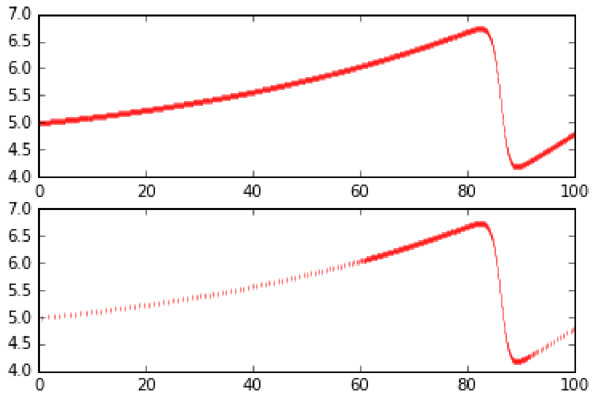
1000 -> 212 points



1000 -> 289 points



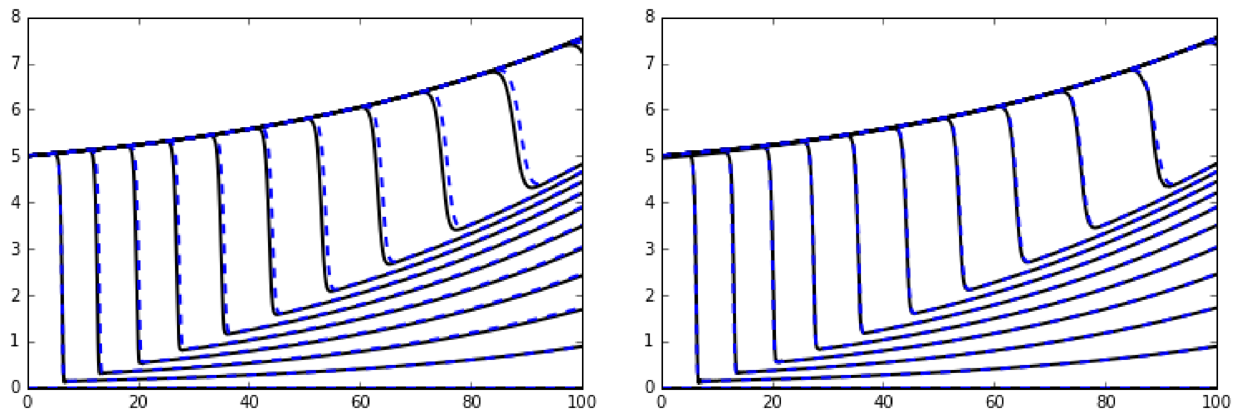
1000 -> 343 points



1000 -> 405 points

Figure 3: Original mesh points and coarsened mesh points for some example snapshots

Next, the online ROM was run using both the baseline method and mesh coarsening method in order to compare the speed and accuracy of the two methods. The solutions are comparable because the same number of clusters (k = 7) and number of singular vectors (n = 40) are used in both solvers. The absolute error was between each high dimensional snapshot point and its corresponding solution from the low dimensional solver was calculated. There errors were averaged and normalized by the average snapshot value in order to get an estimate for the % error. Figure 4 shows a comparison of the two methods.



**Baseline Method**

Mesh sizes = [1001, 1001, 1001, 1001, 1001, 1001, 1001]

Compute time = 6.4440 seconds

Average % Error = 0.02434

**Mesh Coarsening Method**

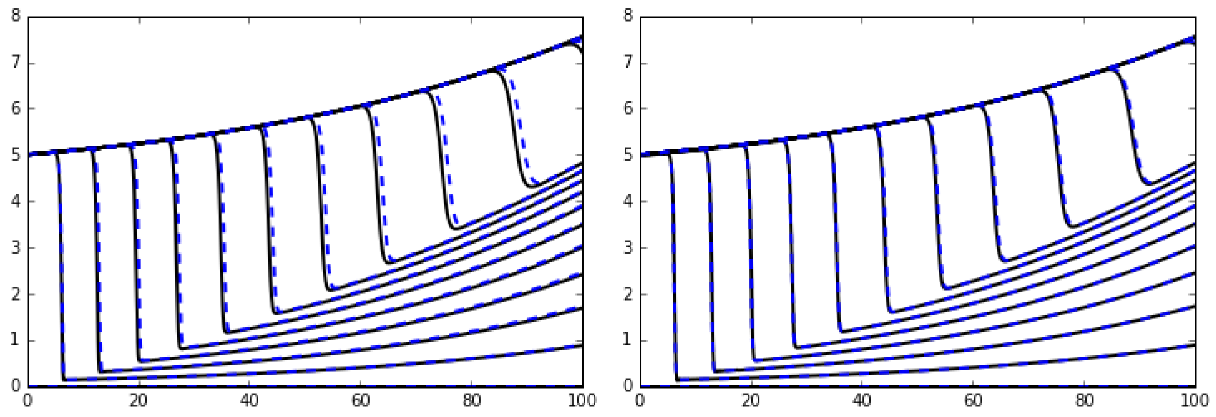
Mesh sizes = [548, 443, 440, 515, 352, 485, 473]

Compute time = 5. 5757 seconds

Average % Error = 0.00480

Figure 4: Solution vector plot and statistics for baseline and mesh coarsening methods for a given test case with nclust = 7 and n = 40

Finally, a second example is plotted in Figure 5 with different parameters of nclust = 7 and n = 25 to show that this example is not an anomaly, but is representative of the difference between the behavior of the two methods in general.



**Baseline Method**

Mesh sizes = [1001, 1001, 1001, 1001, 1001, 1001, 1001, 1001, 1001, 1001, 1001, 1001]

Compute time = 4.9924 seconds

Average % Error = 0.02614

**Mesh Coarsening Method**

Mesh sizes = [438, 401, 349, 474, 485, 396, 444, 434, 258, 423, 456, 491]

Compute time = 4.4092 seconds

Average % Error = 0.00691

Figure 4: Solution vector plot and statistics for baseline and mesh coarsening methods for a given test case with nclust = 12 and n = 25

## 4. Conclusions

The online computation of the ROM is faster using the mesh coarsening method, as expected, but the speedup was expected to be higher. One potential cause of this issue is that the mesh coarsening procedure is very simple and non-optimal. It’s likely that when the mesh coarsening procedure is optimized, there will be further improvement in computation time. Also, the accuracy of the ROM was unexpectedly improved by this method. It’s likely that this is due to the fact that the proper orthogonal decomposition extracts the same number of singular vectors, but those vectors only needs to represent the variation in far fewer spatial dimensions. Therefore, not only is the online computation faster, it is also more accurate.