AMSC 664 PROJECT PROPOSAL

COMPUTATIONAL METHODS FOR THE STUDY OF STOCHASTIC PROCESSES

March 31, 2019

LUKE EVANS
ADVISED BY MARIA CAMERON
UNIVERSITY OF MARYLAND DEPARTMENT OF MATHEMATICS, AMSC PROGRAM

1. Introduction

The primary coding goals for the Spring semester are to:

(1) Fully implement and validate the ATLAS algorithm
(2) Implement a simulator for the genetic switch example from [**?**] and apply ATLAS
(3) Develop a local mesh method for solving PDEs using ATLAS output

The ATLAS algorithm has three stages: 1) Constructing ATLAS charts 2) Constructing local SDEs 3) Running the Reduced simulator. In the Fall, I implemented step 1) for the 1D examples (REF) in [**?**]. The reconstruction of the effective potential was verified with comparison of Maggioni and Crosskey's results, but the LMDS algorithm is trivial in one dimension. I will continue coding the ATLAS but switch to the 2D example from [**?**](example 5.3.1), in order to have a more visual validation process and to have a non-trivial case for LMDS.

The 2D example above has the advantage that it is validated in [**?**]. However, this example has intrinsic dimension of two, so the ATLAS does not reduce the dimension of the problem. Unfortunately, the only example with dimension reduction in [**?**] is a low dimensional example embedded in higher dimensions and then recovered (REWRITE). Thus, to further the goals of the project and provide a non-trivial dimension reduction for ATLAS, I will implement the ATLAS algorithm for the genetic switch system in [**?**]. Though this system is 3D, the dynamics appear to lie on a 2D manifold([**?**]). I will first implement a simulator for this system and validate the simulator against the results in [**?**].

The ATLAS algorithm assumes the user has access to a sampling of the manifold for the problem. In the 1D and 2D examples from [**?**], the manifold is simply the ambient space, so the sampling is just a subsampling of a suitably dense grid of points. However, sampling from the 2D manifold for the genetic switch will require an ensemble of simulations, and the outputs must be verified as a sufficiently dense covering of the 2D manifold. Thus, constructing the delta-net will be much more involved and will require more unit testing, outlined below. Following the delta-net construction, I will run the ATLAS algorithm on the system. Similar validation will follow this step as in the original ATLAS implementation, in particular with regard to the learned reduced simulator.

After implementing the ATLAS on the genetic switch, I will try to develop a method to create local mesh structure on the ATLAS charts, particularly a mesh structure that will be amenable to the Ordered Line Integral Method (OLIM) [**?**, **?**] for solving the quasipotential. I plan to follow a similar approach to the local mesh method developed in [**?**], whereby a 3D point cloud is locally projected to 2D triangular meshes and a PDE solver is applied locally.

Subsequently, I plan to apply the OLIM method to the resulting mesh. OLIM was generalized to radial meshes in [**?**], and I plan to follow a similar approach to local quadrilateral meshes on ATLAS charts. (Continue..)

Create landmarks is not a sub-routine, too granular? ATLAS

- `delta_net(net,init,delta,rho,is_random)`
  - sub-sample a given set of points to create a delta-net, i.e. a set of points Γ in a domain $\mathcal{M}$ such that
    - $d(y_i, y_j) > \delta$ for all $y_i, y_j \in \Gamma$
    - Given $x \in \mathcal{M}$, there exists $y \in \Gamma$ with $d(x, y) < \delta$.
  - identify neighboring points, i.e net points with $d(y_i, y_j) < 2(\delta)$.
  - inputs:
    - `init` - $D \times N$ matrix, set of N vectors in R^d
    - `delta` - coarseness of delta-net
    - `rho` - given distance function
  - outputs:
    - `net` - output $\delta-$net, columns are data points,
    - `neighbors` - struct array, each net point index has a struct with key value 'nbr' that stores the indices of close by net points
- `MDS(square_dist,d)`
  - inputs:
    - `square-dist` - m x m matrix of square-d distances from a data set X, with D rows and m columns, where each column is a data point
    - `D` - input dimension of data vectors of X
    - `d` - desired dimension for embedding data set
  - outputs:
    - `scaling` - Matrix of embedded data points in R^d, columns are embedded data vectors
    - `V` - Matrix of first d eigenvectors of centered square-dist matrix, columns are eigenvectors
    - `eigvals` - First d eigenvalues of centered square-dist matrix
  - Unit Test: (Euclidean) deterministic and random test, PCA some data and make sure LMDS gets same result
  - Unit Test: (Non-Euclidean) Check if results coincide with MATLAB built-in `cmdscale` function
- `LMDS(L,Z,rho,d)`
  - Implementation of the Landmark Multi-Dimensional Scaling Algorithm(ref the proposal) for a given set of landmarks and data
  - inputs:
    - `L` - set of landmarks for Z, array with columns in ^{D} as landmark vectors
    - `Z` - set of data points, array with columns in ^{D} as data vectors
    - `rho` - given distance function
    - `d` - intrinsic dimension, LMDS projects columns of Z, L onto R^d

  - outputs:
    * `embed_L` - MDS output for landmarks, array with columns in ^{d} as projected landmark vectors
    * `embed_Z` - projected Z data, array with columns in ^{d} as projected data vectors
  - Unit Test: (Euclidean) Check if `embed_Z` is the projection of Z onto the principal axes of L. More precisely, verify that `embed_Z` $= V^T Z$, where the columns of V are eigenvectors corresponding to the highest d eigenvalues of the covariance matrix of L.
- `construction`
  - Constructs delta-net, landmarks, ATLAS, and SDE simulator
  - inputs
    * `S` - SDE simulator, see `simulator.m` for example
    * `init` - D X N matrix, set of N vectors in R^d
    * `delta` - homogenization scale, affects density of sample net
    * `rho` - given distance function
    * `m` - desired number of landmarks to generate for each net point
    * `p` - num sample paths for each point in net
    * `t_0` - desired time of simulation for each call to `S`
    * `d`- intrinsic dimension of dynamics
  - outputs:
    * `new_Sim` - struct with key values that specify parameters for simulating a constant coefficient SDE
    * `net` - columns are data points, all distances ¿= delta apart
    * `neighbors` - struct array, each net point index has a struct with key value 'nbr' that stores the indices of close by net points
  - `compute_local_SDE`
    * Description: Computes the drift vector B_n and diffusion matrix Σ_n for the constant coefficient SDE corresponds to the n-th chart of delta net point n :

$$X_t = B_n dt + \Sigma_n dW.$$

      Parameters `B,Sigma,C` will be updated corresponding to net point n.
    * inputs:
      · `n` - integer correponding to n-th net point
      · `neighbors` - struct array, each net point index has a struct with key value 'nbr' that stores the indices of close by net points
      · `embed_L` - MDS output for landmarks, array with columns in ^{d} as projected landmark vectors
      · `embed_X` - LMDS output for X points simulated at landmarks, array with columns in ^{d} as projected data vectors
      · `p` - num sample paths for each point in net
      · `t_0` - desired time of simulation for each call to `S`

· `C` - struct array, the entry $C(n,j).C$ will be a vector in ^{d} which corresponds to the chart coordinates of the j-th neighbor of net point n.
· `B` - $d \times N$ array, the $n-$th column will be a vector in ^{d} which refers to the drift term for the local SDE of net point n.
· `Sigma` - list of arrays, `Sigma(:,:,n)` will be a matrix with columns in ^{d}, and is the coefficient matrix Σ for the local SDE of net point n.
* : outputs
· `C` - as above, updated for n-th net point
· `B` - as above, updated for n-th net point
· `Sigma` - as above, updated for n-th net point
* Unit Test: Define a delta net with points at valleys of potential, verify that diffusion coefficients and variances reflect that
– `compute_switching_maps`
* Description:
* Inputs:
· `n` - integer correponding to n-th net point
· `neighbors` - as above.
· `embed_L` - as above.
· `embed_X` - as above.
· `p` - num sample paths for each point in net
· `T`- empty struct array at first call of function, containts transition mappings, will have entries `T(i,j).T` representing the transition matrix from chart `i` to chart `j`
· `mu`- empty struct array at first call of function, local coordinates for mean of common landmarks, struct array with `mu(n,j).mu` representing the mean landmark for $y_j \cup y_n$, represented in coordinates of the chart y_n
· `Phii` - 1-d case only, this is the MDS transformation multiplier, `Phi(n).Phi` = +-1, transformation for chart corresponding to n-th delta-net point
* Outputs:
· `T` - as above, updated for all n net points
· `mu` - as above, updated for all n net points
* Unit Test: Compare switching map coordinate changes with actual chart coordinates of points in different charts
• `learned_simulator_step`
– inputs:
* `x` - initial point for the simulator, vector in ^{d}
* `i` - chart index for `x`
* `new_S` - new simulator struct, struct containing computed `T,B,C,mu,Phi` structs mentioned above as `new_S.T,,new_S.B` etc.
* `neighbors` - as above

        * `d` - intrinsic dimension of dynamics
        * `dt` - desired time-step length
        * `delta` - as above
    - outputs:
        * `x` - new x value after timestep, vector in ^{d}
        * `j` - chart index of `x` above, used for future timesteps
    - Unit Test: Histogram Validation

ATLAS with genetic Switch

- `gswitch_simulator`
    - Unit Test: (Deterministic) Check stable points, Saddle
    - Unit Test: (Stochastic) Check Metastable regions, MST
- `gswitch_sampler`
- `gswitch_learned_simulator_step`
    - Unit Test: `compute_transition_times`
-

Technique for smooth manifolds

- `augment_ATLAS`
- `2D_mesh`
- `local_mesh`

Solving PDEs on the manifold

-