# ATLAS IMPLEMENTATION: USER GUIDE

## Contents


1. ATLAS Overview 1

1.1. Net Construction 2

1.2. Learning the coordinate charts 3

1.3. Learning the Simulators 3

1.4. Learning the Transition Maps and Simulating the SDE 3

2. ATLAS implementation 4

2.1. Examples 5

3. Code Summary 6

References 10


## 1. ATLAS Overview

The ATLAS algorithm was created by Miles Crosskey and Mauro Maggioni in the publication [1]. The overview below closely following sections 2 and 3 of [1], and for more details it is recommended to review those sections.

The ATLAS algorithm takes as input a simulator for a Markovian stochastic system whose dynamics are assumed to be concentrated near a low-dimensional manifold. The algorithm constructs the manifold from a sampling of points and approximate global dynamics from

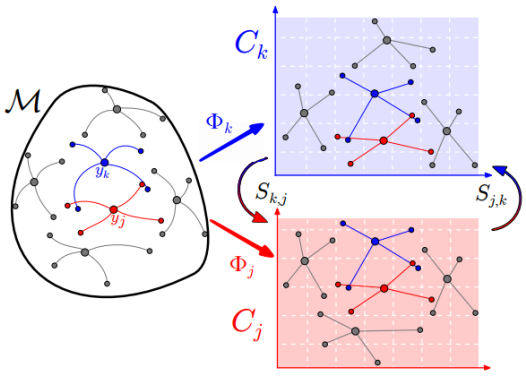


FIGURE 1. Learning the ATLAS: larger circles are net points, smaller circles are path endpoints [1]

the local dynamics at patches of the manifold. The output is a reduced simulator in lower dimensions with a coarser time scale and which approximates long term statistics of the original simulator.

The input stochastic system of interest $(X_t)_{t\geq 0}$ assumed to have state space in $\mathbb{R}^D$ but concentrate near a manifold (the *effective state space*) $\mathcal{M}$ of dimension $d \ll D$. The main inputs for the algorithm are the simulator $\mathcal{S}$ for $(X_t)_{t\geq 0}$, a sampling of the manifold $\mathcal{M}$, the effective state space dimension $d$, and spatial granularity parameter $\delta$.

The algorithm has three main steps:

(i) **Net Construction**: Find set of points $\Gamma = \{y_k\}$ on $\mathcal{M}$ which are well-distributed on $\mathcal{M}$ and which are at least $\delta$ away from each other
(ii) **Learning the ATLAS** (Figure 1):
  (a) For each $y_k$, learn mapping $\Phi_k$ from neighborhood of $y_k$ to local coordinate *chart* $C_k \subseteq \mathbb{R}^d$. The mapping $\Phi_k$ is learned via the Landmark Multi-Dimensional Scaling Algorithm (LMDS)
  (b) Learn pairwise *transition maps* for changing coordinates between overlapping charts
(iii) **Learning the Simulator**
  (a) For each $y_k$, run ensemble of paths from $y_k$, map paths to coordinate chart $C_k \subseteq \mathbb{R}^d$
  (b) Use mapped paths to learn a constant-coefficient SDE on the chart in $\mathbb{R}^d$, and use transition maps to stitch together the SDEs

1.1. **Net Construction.** A $\delta$-net is a set of points $\{y_k\}_{n\in\Gamma} \subseteq \mathcal{M}$ such that with $d(y_i, y_j) > \delta$ for each $i, j$ and such that for each $x \in \mathcal{M}$ there exists $y \in \{y_k\}_{n\in\Gamma}$ such that $d(x, y) < 2\delta$. Two net points $y_i, y_j$ are connected neighbors if $d(y_i, y_j) < 2\delta$, and we denote this by $i \sim j$.

Many algorithms exist for constructing covering nets in this fashion, though a simple brute force algorithm for finding the net is downsampling a sufficiently dense sampling of $\mathcal{M}$. Hence, construction of the $\delta$-net does assume the ability to sample $\mathcal{M}$.

1.2. **Learning the coordinate charts.** Given net point $y_k$, a mapping $\Phi_k$ is constructed from $y_k$ and the connected neighbors $\{y_j\}_{j\sim k}$ to $\mathbb{R}^d$. The chart $C_k$ is the image of the mapping $\Phi_k(B_{2\delta(y_k)}) \to \mathbb{R}^d$. For ATLAS, the $\Phi_k$ map is from the Landmark Multidimensional Scaling algorithm (LMDS) [2]. This is an extension of the Multidimensional scaling algorithm (MDS) [3].

Given $y_k$ in the $\delta$-net, the landmark points $\mathcal{A}_{\|} \subseteq \mathcal{M}$ for $y_k$ should be roughly $\delta$ away from $y_k$ and well-spread in $d$ directions. The landmarks are obtained from taking $m \geq d$ paths from the original simulator started at $y_k$. Then, all neighboring landmark points are collected in $L_k = \bigcup_{j\sim k} A_k$, and the chart $C_k$ is the image of $B_{2\delta}(y_k)$ under LMDS applied with landmarks $L$.

1.3. **Learning the Simulators.** For each $y_k$, the associated SDE for $C_k$, is learned from running $p$ sample paths from the original simulator $S$ up to time $t_0$ (on the order of $\delta$). Each chart $C_k$ has an associated simulator modeling an Ito diffusion SDE with constant coefficients:

$$d\bar{X}_t = \bar{b}_k dt + \bar{\sigma}_k dW_t \quad \bar{b}_k \in \mathbb{R}^d, \bar{\sigma}_k \in \mathbb{R}^{d\times d} \tag{1}$$

The solution to (1) is a Gaussian with mean $\bar{b_k} t_o$, covariance $\bar{\sigma_k}\bar{\sigma_k}^T t_o$. Thus, $b_k$ and $\sigma_k$ are estimated by taking sample mean $\bar{b}_k$ and covariance $\bar{\sigma}_k$ of the endpoints of the $p$ paths simulated.

After iterating through all net points $y_k$, the simulators $(\hat{S}_k)_{k\in\gamma}$ with parameters $(\bar{b}_k, \bar{\sigma}_k)_{k\in\Gamma}$ are obtained.

1.4. **Learning the Transition Maps and Simulating the SDE.** In order to combine the above simulators, transition maps are needed between different charts, in particular near landmarks which overlap, which are denoted as $L_{k,j} = A_k \cup A_j$ for $k \sim j$. The transition map $T_{kj}$ from $C_k$ to $L_j$ is defined as $YX^\dagger$, namely the solution to the least squares minimization $||TX - Y||_2$, where $X = \Phi_k((L_{k,j})$, $Y = \Phi_j(L_{k,j})$ and the rows of $X$ (respectively $Y$) are the data vectors. For $x \in C_k$, coordinates are changed through the transition

$$S_{k,j}(x) = T_{k,j}(x - \mu_{k,j}) + \mu_{j,k},$$

where $\mu_{k,j}$ is the mean of the columns of $X$ and $\mu_{j,k}$ is the mean of the columns of $Y$.

Finally, the simulations are extended to a global SDE which advances in time by the following steps:

- Given simulation at time $t$, at $x \in \mathbb{C}_i \subseteq \mathbb{R}^d$
- Find closest coordinate chart $C_{i'}$ to $x$
- Compute a forward Euler step for simulator $\hat{S}_i$ :

$$x \leftarrow x + \hat{S}.\bar{b}_{i'} + \eta \hat{S}.\bar{\sigma}_{i'}\sqrt{\delta t} \quad \eta \sim \mathcal{N}(0, I_d)$$

## 2. ATLAS implementation

The ATLAS algorithm assumes an input stochastic ODE simulator $S$ which can be started at a given initial condition and run for a given period of time. The simulator coded (`simulator.m`) uses the Euler-Maruyama method to simulate the SDE

$$dX_t = f(x)dt + dW_t, \quad X(0) = X_0,$$

where the function $f(x)$ can be specified by the user. To implement this simulator in ATLAS, we wrote a function for the simulator to be run by $S(X_0, m, T)$, which simulates $m$ paths starting at $X(0) = X_0$ for time $T$ and stores the endpoints of these paths.

The ATLAS code is organized around the main construction steps outlined in the algorithm (see Figure 1):

- Constructing the $\delta$-net $\{y_k\}_{n=1}^N$ ( `delta_net.m`)
- Creating landmarks $\mathcal{A}$ for each $y_k$ in the $\delta$-net and finding neighbors (`construction.m`)
- Constructing charts $C_k$ for each $y_k$ in the $\delta$-net (`construction.m`)
- Learning the simulator associated to each chart $C_k$ (`construct_local_sde()` in `construction.m`)
- Constructing a timestep function for the new approximate simulator (function `learned_simulator_step.m`)

The $\delta$-net construction follows a simple brute force method: given an initial dense sampling of $\mathcal{M}$, downsample until the $\delta$-net coarseness and covering requirement are met. The $\delta$-net is stored an array $net = [y_1\ y_2 \dots\ y_k]$, and a graph is created, where two net points are neighbors if $d(y_i, y_j) < 2\delta$. The edges are stored in `edges` and the neighbors in an array `neighbors` where each row is the length of the maximum degree, and row $i$ has first degree($i$) entries as the neighbors of $y_i$, with zeros for the rest of the row.

All landmarks are stored in a three-dimensional array `L` where `L(:,:,n)` lists all landmarks associated to the net point $y_k$. The paths sample paths $\{x_{n,l}\}_{l=1}^p$ associated to $y_n$ are similarly stored in `X(:,:,n)` where `X` is a three dimensional array storing sample paths from the net points.

For constructing charts, LMDS and MDS are implemented following the algorithm outlined in [2] and outlined in the ATLAS description earlier.

**input :** Simulator $\mathcal{S}$, Manifold sampling $\{x_j\}$, distance $\rho$
**output:** Reduced Simulator $\hat{\mathcal{S}}$

$\{y_k\}_{k\in\Gamma} = \delta - \text{net}(\{x_j\})$;
**for** $k \in \Gamma$ **do**
    *Create landmarks for LMDS around* $y_k$;
    $\{a_{k,l}\}_{l=1}^{m} = \mathcal{S}(y_k, m, t_0)$;
    $A_k = y_k \cup \{a_{k,l}\}$;
**end**
*Create charts and local simulators*;
**for** $k \in \Gamma$ **do**
    *Simulate paths for creating local simulator*;
    $\{x_{k,l}\}_{l=1}^{p} = \mathcal{S}(y_k, p, t_0)$;
    *Compute charts and chart centers*;
    $L_k = \bigcup_{i\sim k} A_i$;
    $[L'_k, \{x'_{k,l}\}_{l=1}^{p}] = \text{LMDS}(L_k, \{x_{k,l}\}_{l=1}^{p})$;
    $\{\mathcal{S}.c_{k,j}\} = \bigcup_{j\sim k}\{y'_j\}$ in $L'_k$;
    *Center chart* $C_k$ *and compute diffusion, drift*;
    $C_k = C_k - c_{k,k}$;
    $\hat{\mathcal{S}}.\bar{b}_k = \sum_l x'_{k,l}/pt_0$;
    $\hat{\mathcal{S}}.\bar{\sigma}_k = (\text{Cov}(x'_{k,l})/t_0)^2$;
    *Compute transition maps* $T$ *between charts*;
    **for** $j \sim k$, $j < k$ **do**
        $L_{k,j} = A_k \bigcup A_j$;
        $L'_{k,j} = L_{k,j}$ in chart $C_k$ coordinates;
        $L'_{j,k} = L_{j,k}$ in chart $C_j$ coordinates;
        $\hat{\mathcal{S}}.\mu_{k,j} = \mathbb{E}[L'_{k,j}]$;
        $\hat{\mathcal{S}}.\mu_{j,k} = \mathbb{E}[L'_{j,k}]$;
        $\hat{\mathcal{S}}.T_{k,j} = (L'_{j,k} - \mu_{j,k})(L'_{k,j} - \mu_{k,j})^{\dagger}$;
    **end**
**end**

**Algorithm 1:** Pseudocode for construction of the ATLAS, closely following [1] and implemented in `construction.m` function

Further, the notation in 1 uses $\hat{S}.b_k, \hat{S}.\sigma_k$, and $\hat{S}.c_{k,\cdot}$ to denote the drift vectors, diffusion coefficients and chart centers for chart $C_k$. Following this notation, we store these variables in a struct array $newS$ where $newS(n).b$ denotes the drift vector for chart $y_k$, and likewise for the other properties.

2.1. **Examples.** TO DO: Use language for a User Guide, instead of a report

(2) $$dX_t = -\nabla V(x)dt + dW_t$$

with potential

$$V(x) = 16x^2(x-1)^2 + \frac{1}{6}\cos(100\pi x),$$

also tested in [1].

A great visual example for the ATLAS algorithm is to apply to potential-driven problems, and to reconstruct potentials by the effective potentials obtained from the ATLAS algorithm. For 1D problems, $LMDS$ is trivial and undoing it just requires multiplying by $\pm 1$. Then, the effective potential in a neighborhood of net point $y_k$ is found by undoing MDS for $b_k$ associated to the simulator for chart $C_k$ (equation (1)) and then integrating. Unfortunately, a 1D problem is an extremely special case and in general we cannot construct the inverse for $\Phi_k$.

2.1.1. *Two-Well Rough Potential.* This example is a simple test case to see if the ATLAS algorithm can take a system with microscale interactions and output a homogenized system with dynamics approximating macroscale interactions of the original. The high-frequency component in $V(x)$ from (2) emulate such microscale interactions, and homogenization from ATLAS should ideally return a system with smooth potential $U(x) = 16x^2(x-1)^2$.

Following [1], the implementation for equation (2) has $\delta = 0.1$, $dt = 0.00005$ for the original simulator. The initial $\delta$-net was created from subsampling points from the set of linear spaced points with distance 0.01 in $[-.3, 1.3]$.

Figure 2(comparable to Figure 13 in [1]) shows that the effective potential $\hat{V}(x)$ from ATLAS is a homogenized version of the rough potential $V(x)$. The ATLAS simulator obtained in this example has timestep roughly 100 times the original, so long simulations from the ATLAS will run about 100 times faster than the original simulator.

## 3. Code Summary

`construction(S,init,delta,rho,m,p,t_0,d)`

- inputs
  - `S` - SDE simulator, see `simulator.m` for example
  - `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
  - `net_info` - struct for delta-net, contains

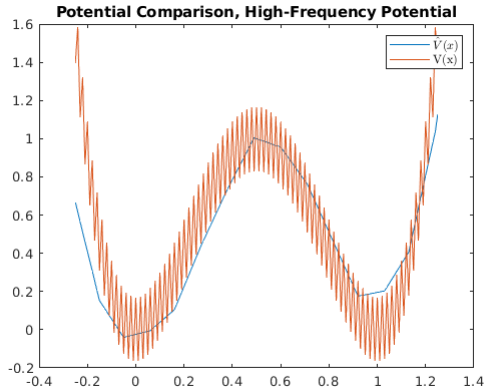


FIGURE 2. Effective potential $\hat{V}(x)$ from ATLAS plotted against true potential $V(x) = U(x) + \frac{1}{6}\cos(100\pi x)$

    - `net` - output $\delta-$net, columns are data points,
    - `deg` - $N \times 1$ vector, entry $j$ is the number of neighbors (degree) of net point $j$
    - `max_deg` - maximum entry of `deg`
    - `neighbors` - $N \times max_deg$ array, row $j$ is neighbor indices of net point $j$, with 0's after $deg(j)$ index
- outputs:
  - `new_Sim` - struct with key values that specify parameters for simulating a constant coefficient SDE

Constructs the landmarks, ATLAS charts, and local SDE simulators for a given delta net and simulator. Can be run with input parameters, or with no parameters and it will load from `current_driver.mat`.

The output is a struct `new_S` containing parameters for the family of local SDEs learned by the ATLAS algorithm. In particular, each net point $n$ has a corresponding local simulator for the constant coefficient SDE

$$dX_t = b_n dt + \sigma_n dW$$

, and any neighboring net points $j \sim k$ have associated transition maps $T_{jk}, T_{kj}$ which allow the global ATLAS simulator to switch between the two local SDE simulators.

The struct `new_S` has key values:

- `c` - $d \times N \times N$ array containing local coordinates of neighboring charts, updated each iteration of `construct_local_SDE`, `c(:,i,j)` is a $d \times 1$ vector, is $j$-th neighbor of net point $n$, expressed in chart $n$ coordinates
- `b` - $d \times 1$ vector, drift coordinate for $n$'s local SDE

- `sigma` - $d \times d \times N$ array, `sigma(:,:,n)` is $d \times d$ matrix of diffusion coords for n's local SDE, `mu` - $d \times N \times N$ array containing mean landmarks of local neighboring charts `mu(:,i,j)` is a $d \times 1$ vector, the average of chart $j$'s landmark expressed in chart $n$ coords

`learned_simulator`

- inputs:
  - `Yzero`- initial point for simulator
  - `mA`- number of simulations to run
  - `T`- right endpoint of time domain
  - `new_S` - struct with values of learned simulator
  - `neighbors` - as above
  - `deg` - as above
  - `d` - intrinsic dimension of dynamics
  - `dt` - desired time-step length
  - `delta` - as above
- outputs:
  - (multiple trajectories) `path_end` - array of all endpoints from all $m$ trajectories simulated
  - (multiple trajectories)`chart_end` - chart indices of all endpoints from `path_end`
  - (one trajectory) `Y` - column vector of trajectories in various charts
  - (one trajectory) `chart` - column vector of chart locations from `Y`)

This function simulates trajectories from learned the ATLAS simulator. The user inputs the number of trajectories $m$, and the endpoints (in coordinates of respective charts) of the $m$ trajectories run will be output, as well as the chart location of the endpoints. This function also contains the code for `learned_simulator_step()` below.

`learned_simulator_step`

- inputs:
  - `x` - initial point for the simulator, vector in $\mathbb{R}^d$
  - `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 $\mathbb{R}^d$
  - `j` - chart index of `x` above, used for future timesteps

This is a standalone function for the learned simulator step of the ATLAS algorithm, also implemented as part of the `learned_simulator.m` function. Propagates the learned simulator one step forward in time with an Euler step. The initial point `x` and it's corresponding chart `i` are required inputs. This initial chart `i` can come from a previous `learned_simulator_step()` call, or from knowledge of `x`'s chart prior to the LMDS mapping into chart space.

`delta_net(net,init,delta,rho,is_random)`

- 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,
  - `deg` - $N \times 1$ vector, entry $j$ is the number of neighbors (degree) of net point $j$
  - `max_deg` - maximum entry of `deg`
  - `neighbors` - $N \times maxdeg$ array, row $j$ is neighbor indices of net point $j$, with 0's after $deg(j)$ index

This function takes a set of points `init` and :

- Sub-samples the given set of points to create a delta-net, i.e. a set of points $\Gamma$ 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 $d(x, y) < \delta$.
- identifies neighboring points in the delta net, i.e net points with $d(y_i, y_j) < 2(\delta)$.

`LMDS(L,Z,rho,d)`

- inputs:
  - `L` - set of landmarks for $Z$, array with columns in $\mathbb{R}^D$ as landmark vectors
  - `Z` - set of data points, array with columns in $\mathbb{R}^D$ 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 $\mathbb{R}^d$ as projected landmark vectors
  - `embed_Z` - projected $Z$ data, array with columns in $\mathbb{R}^d$ as projected data vectors

Implementation of the Landmark Multi-Dimensional Scaling Algorithm[2]. For this implementation, the user must provide what the landmarks are ($L$) for the given data set ($X$).

## References


[1] M. Crosskey and M. Maggioni. Atlas: a geometric approach to learning high-dimensional stochastic systems near manifolds. *Multiscale Modeling & Simulation*, 15(1):110–156, 2017.

[2] V. De Silva and J. B. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Technical report, Stanford University, 2004.

[3] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, Dec 1952.