Scientific Computation Project 2

Leon Wu 01190736

March 2, 2020

Part 1

In this discussion of the algorithms for part 1, N will denote the number of nodes, and M the number of edges between nodes for the input graphs. It is noted that all operations involving appending/popping from both ends of a deque, assigning/reading items from a list and removing/adding/searching items in a set/dictionary are $O(1)$.

1.1)

Description

The algorithm is a modification of breath-first search (BFS). First, lists are initialized, which store shortest distances to nodes and number of shortest paths to each node from the start node. A deque (double-ended queue) is also initialized, and is used to store the queue of nodes. It has $O(1)$ appends and pops at both the front and the back, and so is useful for BFS, where both ends of the queue are used.

After initialization ($O(N)$), the algorithm pops node x from the front of the deque, and iterates through its neighbours. If neighbour v hasn't been visited, it is added to the back of the deque. v is then marked as visited, and its depth (from the start node) is set as the 1 more than the depth of x. This is guaranteed to be the lowest distance from the start node since BFS exhausts each depth from the start node before moving on to lower depths. If v is at a depth 1 lower than x (meaning therefore that this edge is also used in the shortest path to v since the nodes can be organised by depth), the number of shortest paths to x is added onto the paths for v. In this way, the number of shortest paths from the start node, along with their distances, are stored for each visited node.

As soon as the destination has been visited, the number of paths for its neighbours with a depth one higher than itself are added to its number (apart from the node that it was visited from in the first place).

Each reachable node from the start node is added and removed from the deque, and its label and depth updated, and checks if it is dest (max $5N$ operations). The depths are compared, number of paths is potentially updated and a label is checked twice per edge (since each edge's nodes must both become explored, starting with the case that only one of the nodes is explored) (max $6M$ operations). The neighbours of the destination node are iterated through to potentially add to the number of paths, with 2 extra operations (max $2N+2$ operations). Overall, the worst-case running time is estimated to be around $(7N + 4M + 2)$, so that the asymptotic worst time complexity is estimated to be $O(N + M)$.

1.2i)

The algorithm uses a modification of Dijkstra's algorithm. Initialization is done in $O(N)$. A dictionary containing all the nodes with the corresponding safety and last nodes is created. The algorithm terminates when the destination node is removed. The node chosen to be removed is the one with the lowest safety.

For each node choice, there are 3 assignments, 2 condition checks, and a loop to find this chosen node by finding the minimum safety of the nodes (max $4N(5 + N)$ operations). If this safety = $dmin$, then the destination node must be disconnected, since it hasn't yet been found, and the rest of the graph is disconnected. The chosen node's current path is guaranteed to be the safest path to the start node, since the current path is the safest using the current explored nodes, and since it is the safest path to the start node out of all the current explored nodes (following from the properties of the given safety metric). Similar to part 1.1, each edge is looked at twice, checking whether the neighbour node is already explored, and if the new safety is less then the provisional safety, then the new safety and the last node are stored (max $5 * 2 * M$ operations) (there is some dependence on this complexity on how many nodes have already been visited). When the destination has

been reached, the path is obtained by walking backwards from the destination using the last nodes. A deque is used here for $O(1)$ appends to the front, and this path is found in $O(N)$, since the path can have a maximum length of N. (Also, constructing the list from the deque is maximum $O(N)$). This gives the whole algorithm an asymptotic runtime of $O(N^2 + M)$. If a binary heap is used in order to find the minimum of the queue, the search for the minimum is performed in $O(log(N))$ (since the heap must be reconstructed after the minimum is taken out). Adding items to this heap is also $O(log(N))$ so that the algorithm has an overall runtime of $O(Nlog(N) + Mlog(N))$, since there can be a maximum of M additions to the binary heap. This can result in significant time savings for most graphs.

1.2ii)

This algorithm is very similar to 1.2i. The differences are that the route is constructed at the end using a list rather than a deck, which saves the $O(N)$ operation of converting the deque to a list (however this section of code is still $O(N)$). This can be done since the length of the list is known and can be preallocated. Also, the distance metric used here is just the sum of the weights instead of maximum safety score. An extra comparison must be made for the case of equal time journeys. The journey with the smallest number of steps is chosen; this extra step is $O(5 * 2 * M)$. The algorithm has the same asymptotic run time as 1.2i. This algorithm is probably slightly slower than 1.2i because of the extra $10M$ operations.

1.3)

Initialization is done in $O(N)$. A node-tracking set is used to keep track of which nodes have been visited, which at the beginning contains all nodes. An initial search is performed on the nodes in order to find sets of connected nodes. A node is popped from a set, checked to see if it alone, added to the main set, and added to the set keeping track of connected nodes ($O(K)$, where K is the number of connected graphs). For each connected graph, its nodes are added (at random, since the order does not matter in order to traverse the whole graph) and removed from the main set, removed from the node-tracking set, and added to the set keeping track of connected nodes ($O(N)$). Each node is checked twice to see if one of its nodes has already been visited ($O(M)$). The connected graphs are then iterated through twice to find the 2 cheapest stations to return and arrive at ($O(KN)$). The cheapest journey for each connected graph is computed, dealing with the case that the cheapest entry and exit stations are the same by comparing the two costs that include a second minimum cost appropriately ($O(N)$). The costs are then compared across connected graphs in order to find the minimum ($O(K)$). Finally, the cheapest journey is returned. The estimate for the asymptotic runtime is therefore $O(N + M)$.

Part 2

2.1i)

Using a fully vectorized approach, for each time step, the array of degrees corresponding to the M walkers from the previous time-step is multiplied by an array of (precomputed) random numbers between 0 and 1. Rounding down these values gives indices for the neighbours of the M walkers for the next time step, which are then used to index the Adjacency List of the graph along with indices of the nodes at the previous time step to give the next time-step's nodes.

2.1ii)

Figure 1 shows the proportion of the end positions of M random walks after Nt time steps. For the large Nt and M results, the proportions are very close to the asymptotic result, which is defined for node i as $\frac{d_i}{2E}$ where E is the number of edges. Note that this is the stationary distribution of the random walk, which is a Markov Chain, since the future states of the random walk only depend on the present state and not the past. For smaller M and Nt, these points are less close to this line. The limiting distribution is linearly proportional to the degrees of the nodes, which makes sense since higher degree nodes have more walkers likely to move towards them.

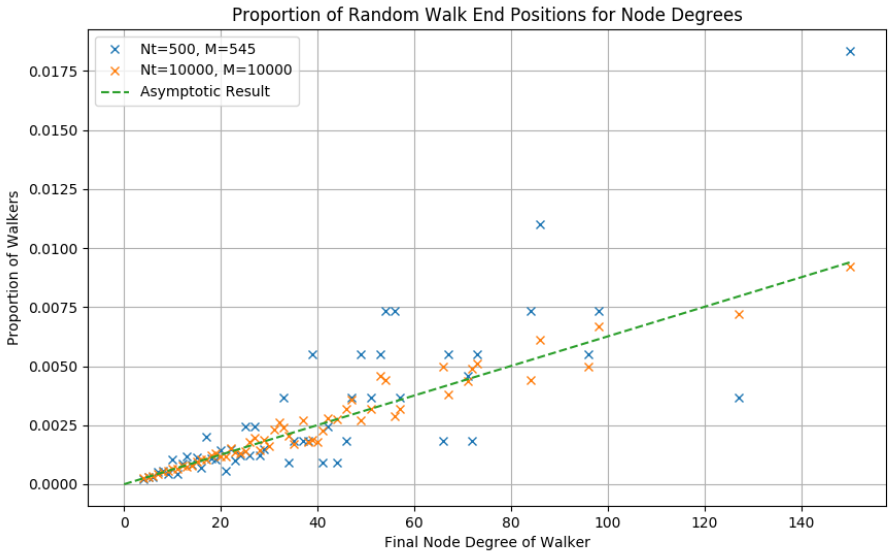

Figure 1: Proportion of Random Walks vs Degree

Figure 2 illustrates this result for varying Nt and M. The plot shows the convergence to the asymptotic result as M and Nt are increased. It is expected that as Nt and M tend to infinity, this difference will approach 0. The results as M increases (for fixed Nt) shows a clear decrease in error, whereas the plots as Nt increases (for fixed M) have a less clear trend slightly downwards. This suggests that even for relatively small values of Nt, the random walks seem to have stepped sufficiently far to display dynamics for a long term distribution. With these being said, values of M and Nt of roughly 10000 seem sufficiently large to display long-term distributions of the random walk.

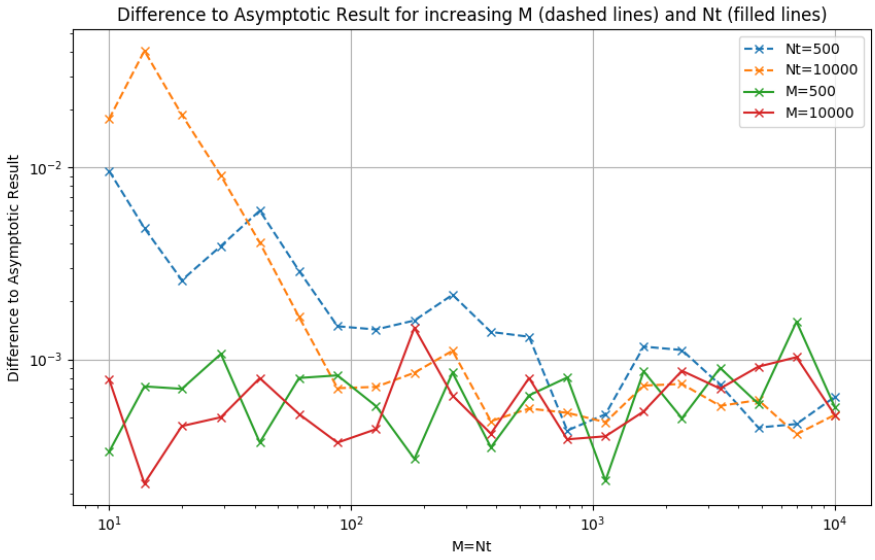

Figure 2: Convergence to Asymptotic Result

2.1iii)

Figure 3 shows the eigenvalues for the Laplacian and scaled Laplacian transposed. Note the eigenvalues for the scaled Laplacian are the same as those for the transpose. The eigenvalues are all negative and real-valued, except for a zero eigenvalue for the Laplacian operator. The eigenvalues for the Laplacian also get smaller slower than the other two operators. Each operator has one zero eigenvalue, meaning that as time approaches infinity, the solutions approach a fixed value and don't decay to zero. This is similar to the random walk in that there exists a limiting distribution.

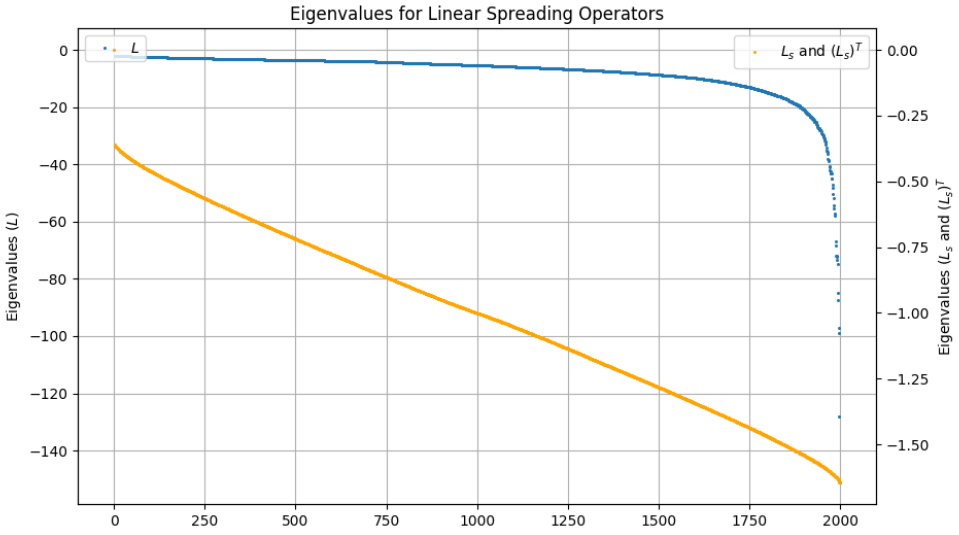

Figure 3: Eigenvalues for Linear Spreading Operators

Figure 4 shows the proportion of walkers at the end of the random walks, and the intensity of the Linear spreading operators, against the degrees of the final nodes. Both the Laplacian and the scaled Laplacian operator show a uniform asymptotic distribution, meaning that for large t the intensity is spread equally among all nodes in the graph. Contrast this with the scaled Laplacian transposed, which shows a limiting distribution proportional to the degrees of the nodes. This has the same limiting behaviour as the random walk, so that the

limiting intensity at these nodes is linear in degree. From these results it can be said that the Linear operator L_s^T has the most similar properties to that of the random walk.

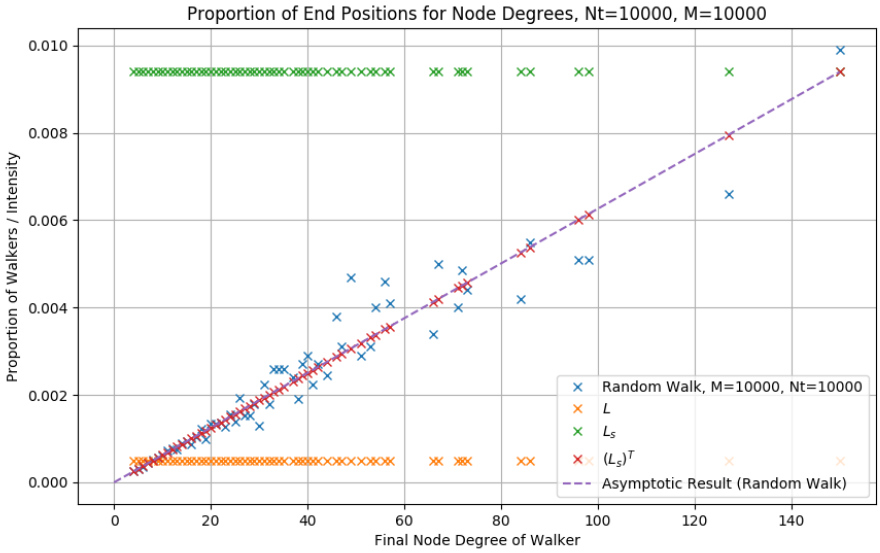

Figure 4: Proportion of End Positions for Node Degrees

2.2i)

Since gammaA is a sparse matrix, has $2M$ non-zero elements (where M is the number of edges) and $M > N$, this sparse matrix vector multiplication with y has $4M$ operations since there are $2M$ multiplications and $2M$ additions. mbeta*y is N operations, $(1 - y)$ is N operations and gammaAi * $(1 - y)$ is N operations. Overall, we have $4M + 3N$ operations.

2.2ii)

Figure 5 shows the intensity of N nodes over time for model A. The intensities for all nodes increases until they reach a steady value. Different nodes end up at different steady values; these values will later be seen to be related to the degree of the nodes.

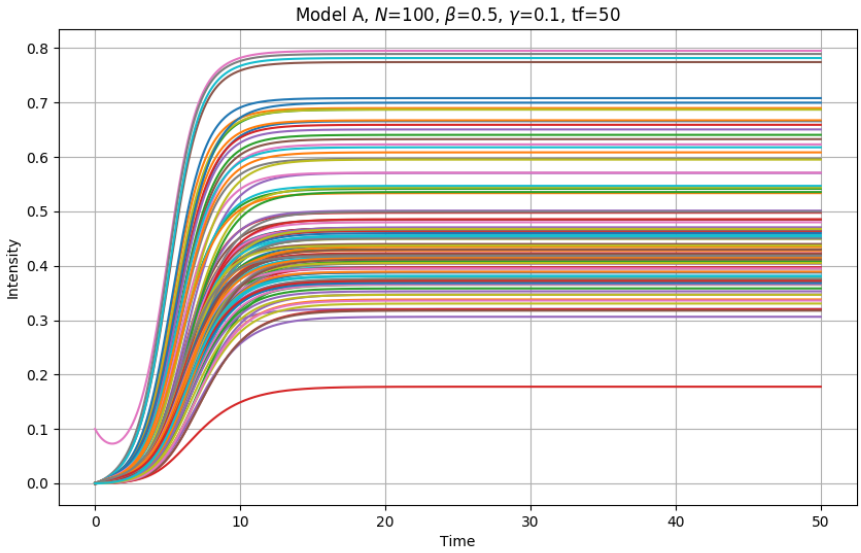

Figure 5: Model A Intensity over Time

Figure 6 shows averaged model A intensities for the N nodes over time. The different parameters clearly show different dynamics. For example, high values of γ and low values of β lead to high values of average intensity for large time values. Since model A can be used as an infection model, the dynamics with these parameters could represent a disease with a high rate of infection. The equilibrium is reached, with most of the population being infected after long periods of time.

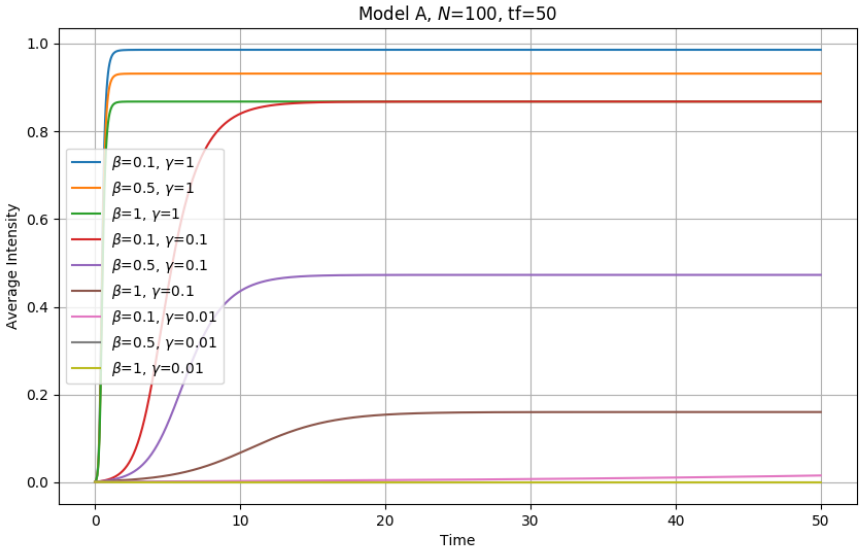

Figure 6: Model A Average Intensity over Time

Figure 7 shows the intensity of N nodes over time for model B. Computing the sum of the intensities across nodes gives 0 for all time. The solution for each node can be seen to oscillate about 0, with varying amplitudes across both nodes and time, although the variance of the amplitude across time can be seen to be small for high amplitudes. The maximum amplitude of these solutions can be related to the degree of the nodes, as shown in **Figure 8**. This figure shows this relationship, with a higher degree node corresponding to a larger maximum amplitude. The quantitative relationship is unclear when plotted on log plots; since for very large degree nodes, the maximum amplitude is especially high. Possibly for smaller degrees there seems to be a power-law relationship. Also, increasing the magnitude of alpha increases the rate of change of the system.

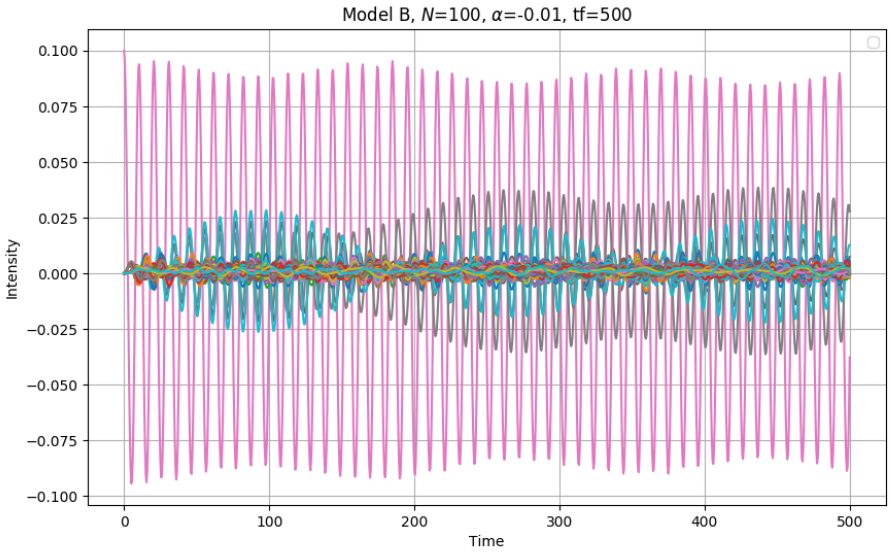

Figure 7: Model B Intensity over Time

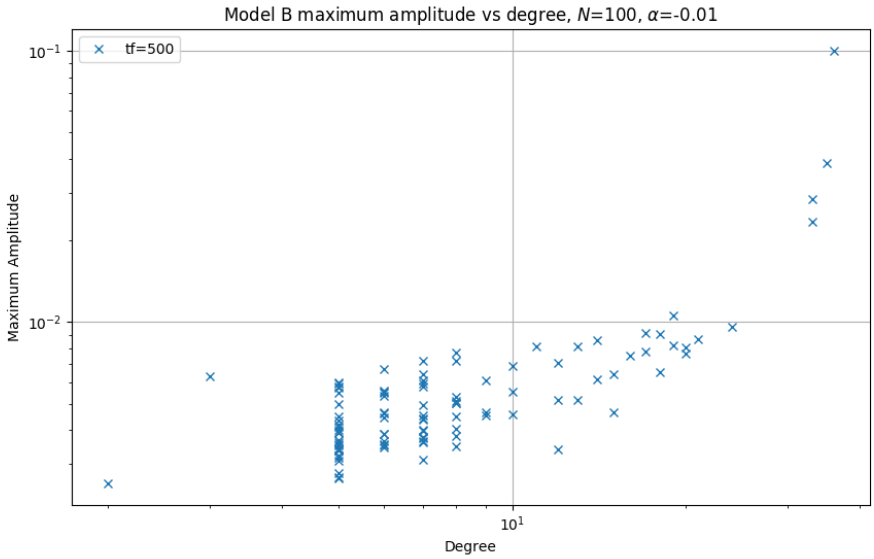

Figure 8: Model B Maximum Amplitude over Degree

Figure 9 is another plot of the intensity vs node degree; this time for model A with varying parameters. For the parameters $\beta = 0.5$ and $\gamma = 0.05$, the intensity can be seen to be roughly linearly increasing with node degree. This is the consistent with the dynamics seen in the random walk, and the linear operator L_s^T.

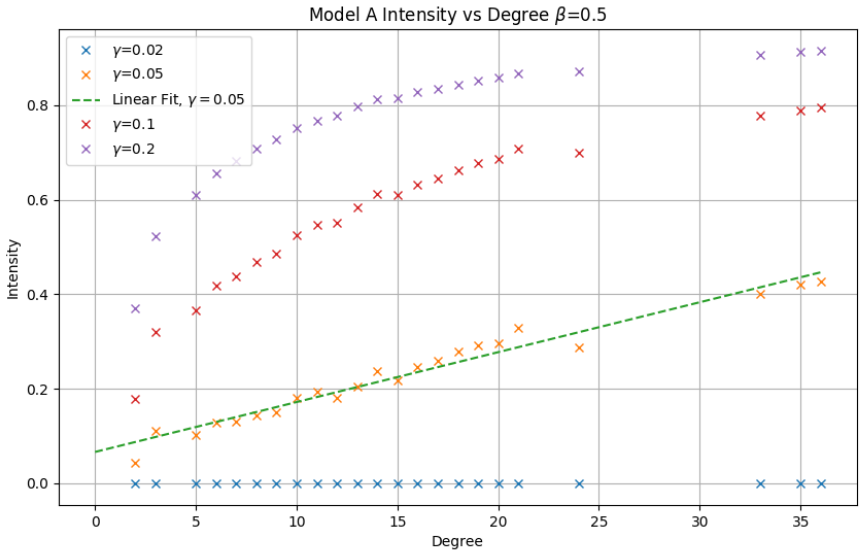

Figure 9: Model A Intensity vs Degree for varying Gamma

The variance of the intensity across the nodes of model A, model B and linear diffusion is plotted in **Figure 10**. The variance of model A varies at first, then converges to a fixed variance. This suggests the intensity at each of the nodes has also converged, but each node converges to a different intensity. The Linear Diffusion is similar, except the variance tends to 0, showing that the intensity of the nodes tends to a single value that is the same for all nodes. Finally the variance of model B seems not to converge to a single value, but oscillates at a fixed frequency.

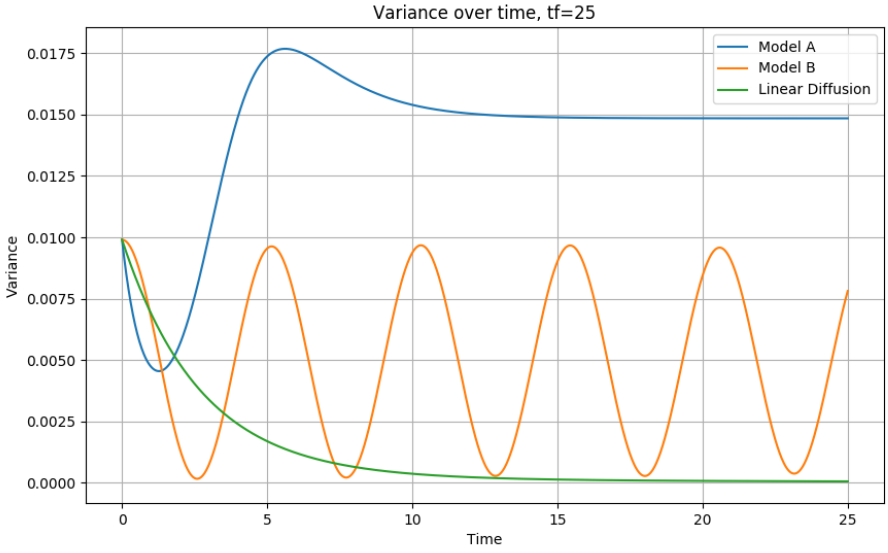

Figure 10: Variance of Models over Time