Statistical Methods in Applied Computer Science (DD2447) - Assignment 2

Marcus Nystad Öhman 19900517-1419
Jacob Malmberg 19850806-0558

December 2018

1 Introduction

We have completed tasks 2.1, 2.2, and 2.3. Our Github usernames are jacobmalmberg and marcusNystad. The repository can be found at `https://github.com/jacobmalmberg/kth-statmeth-18`.

2 Gibbs sampler for magic word

2.1 Question 1

In this assignment a Gibbs sampler was implemented to estimate the posterior over start positions of "magic" words after having observed N sequences. Each position in the sequence is sampled from a categorical distribution, where all the positions in the background is sampled from the same categorical with a given Dirichlet prior and hyperparameter α'. The j:th positions in the magic words are each sampled from their own categorical distribution with a corresponding Dirichlet prior θ_j. Where each θ_j has the same hyperparameter α.

To generate the synthetic data we used the model given for the magic words that generates N sequences of length M, s^1, ..., s^N where $s^n = s^n_1$, ..., s^n_M. These sequences are all over the alphabet[K]. Hidden in a sequence is a "magic" word if length W, and the rest of the sequence is just called background.

We used the following parameters:

N = 20, M = 10, K = 4, W = 5, seed = 123, num_iters = 500, α = [0.9, 0.9, 0.9, 0.9] and α' [12,7,3,1].

Generation of data:

- For each sequence n, a start position r_n is generated by sampling uniformly from [M - W +1].
- Then we draw theta from the Dir($\theta|\alpha^{'}$) prior distribution for the background, and for the "magic" word we draw θ_j from the Dir($\theta_j|\alpha$) prior (drawing five thetas given W = 5).
- With the set start positions and the sampled thetas we sample all our data from the categorical distributions for each respective position in the "magic" words and the background.

After generating the synthetic data the gibbs sampler was implemented by using the marginal likelihoods. Where the marginal likelihood for the j:th positions in the "magic" words is given by the following:

$$p(D_j|R) = [\Gamma(\sum_k \alpha_k)/\Gamma(N + \sum_k \alpha_k)] \prod_k \Gamma(N^j_k + \alpha_k)/\Gamma(\alpha_k) \tag{1}$$

For the background positions the marginal likelihood is given by:

$$p(D_B|R) = [\Gamma(\sum_k \alpha'_k)/\Gamma(B + \sum_k \alpha'_k)] \prod_k \Gamma(B_k + \alpha'_k)/\Gamma(\alpha'_k) \tag{2}$$

By combining these the full conditional $p(r_n|R_{-n}, D)$ can be given as:

$$p(r_n|R_{-n}, D) \propto p(D_B|R_{-n} \cup r_n) \prod_j p(D_j|R_{-n} \cup r_n) \tag{3}$$

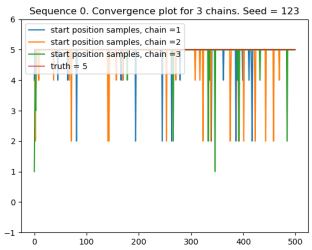
To estimate the posterior using the gibbs sampler we implemented the following:

- For each iteration:
  - For each sequence:
    - For each starting position:
      - Compute $p(r_n|R_{-n}, D)$ and store these.
    - Build a distribution from calculated $p(r_n|R_{-n}, D)$ probabilities.
    - Sample start position r_n from this distribution.
    - Update R with r_n.

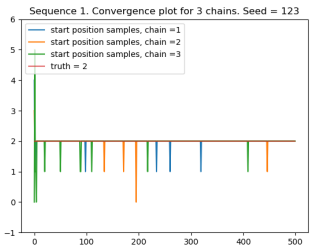
By sampling each variable in turn, conditioned on the values of all the other variables in the distribution, we can thus estimate the posterior using this Gibbs sampler.

2.2 Question 2

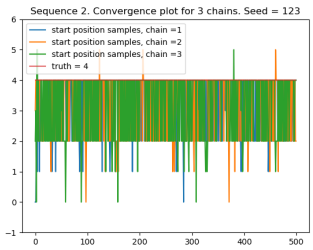
To assess convergence, three chains was run with different starting positions. If the different chains converge to the same starting position, we have reached convergence. Since we have 20 sequences and thus 20 starting positions, but only so much space in the report, plots will only be shown for 3 sequences. This is shown in figure 1. As can be seen in the figure, the three chains behave similarly and converge to the same value, which proves convergence.



(a) Sequence 0



(b) Sequence 1



(c) Sequence 2

Figure 1: Convergence plot for various starting positions

Further, upon inspecting the sample vectors for the different starting positions for the different chains an interesting pattern was observed. For sequence 1, we had very fast convergence for all three chains: within 5 iterations all three chains had converged to the true starting postion 2. This is shown in table 1.

Chain	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Sample 7
Chain 1	4	1	1	2	2	2	2
Chain 2	3	2	2	2	2	2	2
Chain 3	0	5	1	1	0	2	2

Table 1: Table showing convergence for sequence 1

To assess accuracy, all the samples after the burn-in period was used and for each sequence, the most commonly occurring sample was used as the guess for that sequence. For example, if we had 10 samples for a sequence and 4 of them were 1's, 3 were 2's and 3 were 5's, 4 was the most commonly occurring sample and thus would be as the guess for that sequence. As the burn-in period observed during the convergence testing was shown to be short, the first 50 samples were discarded and the rest was used. In order to avoid correlation between samples, we used a lag of 10.

Using this framework, the accuracy was shown to be 95%. Further, the most commonly occurring sample occurred over 60% in all the sequences. This is shown in table 2.

Start position truth	5	2	4	2	1	3	2	3	1	1	0	1	1	0	0	1	3	5	4	0
Start position guessed	5	2	4	2	1	3	2	4	1	1	0	1	1	0	0	1	3	5	4	0

Table 2: Table showing accuracy for alpha_bg = [12,7,3,1]

Now, instead of using alpha = [12,7,3,1] as our prior for the background distribution, accuracy was also measured with a alpha = [1,1,1,1] prior for the background distribution. Since the magic word distribution prior is [0.9, 0.9, 0.9, 0.9], we think that the accuracy should be lower since the distributions for the background and the magic word are very similar and it should thus be harder to discern the magic word. Using the [1,1,1,1] prior for the background distribution, the accuracy was measured to be 70%. This is shown in table 3. That the accuracy is higher for two very different distributions for the background and the magic word makes sense and is further evidence that we have succeeded with our implementation.

Start position truth	5	2	4	2	1	3	2	3	1	1	0	1	1	0	0	1	3	5	4	0
Start position guessed	3	2	4	2	1	3	5	3	1	5	0	1	1	2	0	1	3	5	5	3

Table 3: Table showing accuracy for alpha_bg = [1,1,1,1]

3 MCMC for the train

3.1 Question 3

The algorithm is implemented in q2.py. The runtime complexity of the algorithm is O(T), since given that we know the set of all switch settings as well as the start position, we know the path that the train will take. Thus, we do not need to check if the train visits all the vertices in all the timesteps.

3.2 Question 4

To be able to implement the Gibbs update for s_1, we need to be able to compute $p(s_1|G, \boldsymbol{o}, \boldsymbol{x})$. $p(s_1|G, \boldsymbol{o}, \boldsymbol{x}) = \frac{p(\boldsymbol{o}|G,s_1,\boldsymbol{x})p(s_1)}{p(\boldsymbol{o},\boldsymbol{x})}$, where $p(o, x) = \sum_{s_1} p(o|x, s_1)p(s_1)$. In the text, it is given that $p(s_1) = 1/3$. Since we are making a categorical distribution over the possible starting positions s_1 to sample from, $p(o, x)$ is constant as well and we do not need to compute it either. Thus $p(s_1|G, \boldsymbol{o}, \boldsymbol{x}) \propto p(\boldsymbol{o}|s_1, \boldsymbol{x})$.

To be able to use the MH algorithm for the switch states, we propose a uniform distribution q for the switch states. This distribution is symmetric. We also need to derive an expression for $p(\boldsymbol{x}|G, \boldsymbol{o}, s_1)$. $p(\boldsymbol{x}|G, \boldsymbol{o}, s_1) = \frac{p(\boldsymbol{o}|s_1,x)p(x)}{p(\boldsymbol{o}|x)}$. Given that we have a symmetric proposal and that we have a uniform prior for the switch settings, $p(x') = p(x)$, the acceptance probability becomes $r = min(1, \frac{p^*(x')}{p^*(x)}) = min(1, \frac{\frac{p(\boldsymbol{o}|s_1,x')p(x')}{p(\boldsymbol{o}|x)}}{\frac{p(\boldsymbol{o}|s_1,x)p(x)}{p(\boldsymbol{o}|x)}}) = min(1, \frac{p(\boldsymbol{o}|s_1,x')}{p(\boldsymbol{o}|s_1,x)}$ which is just the ratio of the conditional likelihoods.

The algorithm is first to randomly create an $\mathbf{x}$ vector. Then, a s_1 sample is sampled using Gibbs. This is done by first creating a distribution over the possible starting positions and a s_1 sample is then sampled from this distribution. This s_1 sample is then used in the MH algorithm to be able to sample a $\mathbf{x}$ sample. Once this is done, we use this $\mathbf{x}$ sample in the Gibbs sampler and continue on like this. For the sake of brevity, the standard parts of the Gibbs sampler and the MH algorithm are not described.

G, X_truth and o was generated using seed = 17, seed_lattice =3, T=100, p=0.1, num_iter=1000. The sampled $\mathbf{x}$ have very low accuracy since the train only goes through 4 vertices. The accuracy of s_1, that is how many samples of s_1 after the burn in period (and with a lag of 10) that matches the true s_1 value, is 28.4%. In this case, we used a burn-in period of 100 samples. The acceptance probability for MH was 88% which obviously is very high. This could possibly be because the train only goes through 4 vertices (depending on the seed) which leaves no information about the 9-4=5 remaining vertices.

3.3 Question 5

One way to assess convergence is to run multiple different chains for the same G, X_truth and O and see if the histogram for s_1 looks similar. The idea is that even if we have different starting guesses for s_1 and $\mathbf{x}$, we should end up with the same distribution. In order to avoid correlation between samples, we used a lag of 10. A burn-in period of 100 samples was used as well.

G, X_truth and o was generated using seed = 17, seed_lattice =3, T=100, p=0.1, num_iter=1000. Seeds 4, 8, 12 was then used for the different chains. The true s_1 was (2,1), that is 8 in our histogram. Since the histogram for the different chains looks similar, we conclude that we have reached convergence. This is displayed in figure 2.

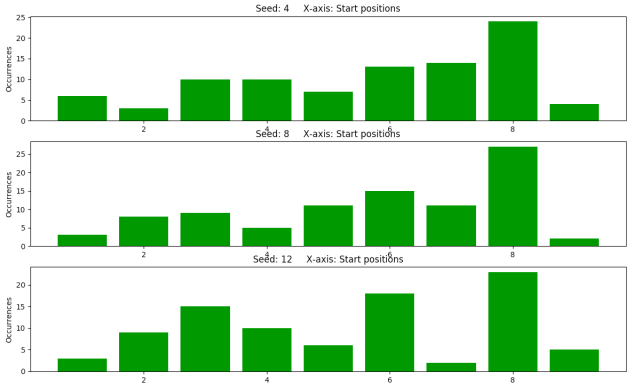


Figure 2: MH within Gibbs convergence, 3 chains

3.4 Question 6

The conditional distribution of X_v can be derived similarly to $p(r_1|R_n, D)$:

$$p(X_v|X_{-v}, G, s_1, \boldsymbol{o}) = \frac{p(X_v \cup X_{-v}, G, s_1, \boldsymbol{o})}{p(X_{-v}, G, s_1, \boldsymbol{o})} \propto p(X_v \cup X_{-v}, G, s_1, \boldsymbol{o}) \propto p(\boldsymbol{o}|s_1, X_v \cup X_{-v}, G) \tag{4}$$

Gibbs sampling for the switch states is implemented in the function gibbs in q2.py.

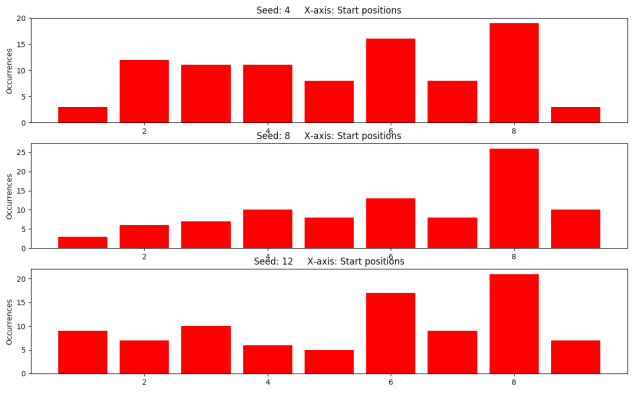
3.5 Question 7

The block size was determined with a little help from Seong-Hwan Jun during the help session on 19/12. While Seong suggested two blocks of non-connecting vertices, we found it easier to implement three blocks of non-connecting vertices. The

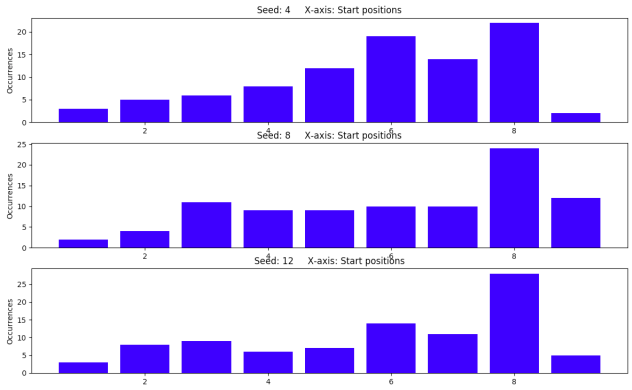
block size is therefore 3. Blocked gibbs sampling is implemented in the function block_gibbs in q1.py.

3.6 Question 8

Using the way to assess convergence laid out in section 3.3, we find that the three variants converge in a similar manner. This is displayed in figure 3, compare with figure 2. The true s_1 was (2,1), that is 8 in our histogram. The accuracy for s_1 was calculated in the way laid out at the end of section 3.2, and was 28.2% for Gibbs, 28.8% for blocked Gibbs and 28.4% for MH within Gibbs. The accuracy for $\mathbf{x}$ was low for all three variants, possibly for reasons laid out in section 3.2. Further, since the train did not travel through all the vertices, there is no data on many of the switches. Thus, the accuracy for $\mathbf{x}$ is heavily reliant on randomness and therefore not deemed relevant or interesting. To conclude, in terms of convergence and accuracy these methods have very similar results.



(a) Gibbs sampling for switch states, 3 chains

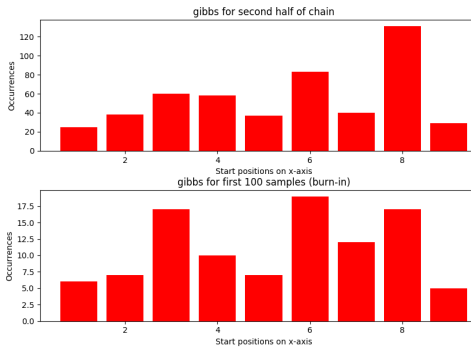


(b) Blocked gibbs sampling for switch states, 3 chains

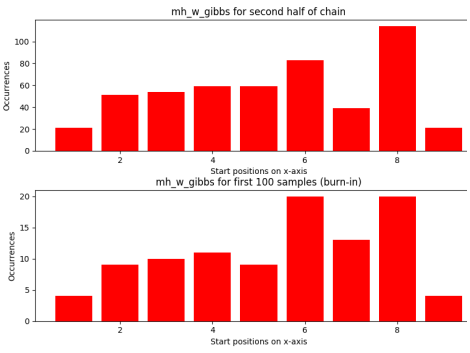
Figure 3: Convergence plot for various starting positions

Convergence rate was analyzed using a modified Geweke diagnostic, found in `http://goo.gl/Vk4BYS` (page 10). Essentially, we make a histogram for the starting positions using the last half of the chain (where we assume we are sampling from the stationary distribution) and compare it to a histogram of the first 100 samples of the chain. If the histogram do not look similar, we have not converged yet. This is displayed in figure 4. As can be seen, the Gibbs sampler has not converged to the stationary distribution during the first 100 samples, while MH within Gibbs looks reasonably good, and blocked gibbs looks very good. Thus, blocked Gibbs converged faster than MH within Gibbs, which in turn converged faster than Gibbs.

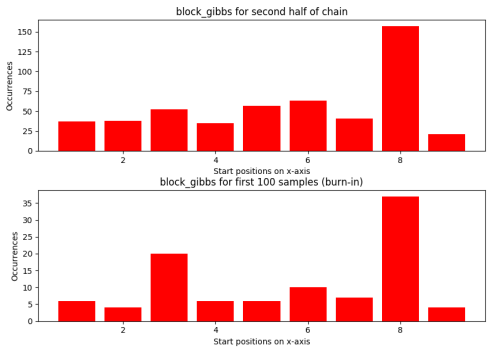
Figure 4: Convergence rate for the MCMC variants



(a) Gibbs sampling conv. rate



(b) MH within gibbs sampling conv. rate



(c) Blocked gibbs sampling conv. rate

The mixing rate for MH was diagnosed using the acceptance probability, which was 88%. This is very high, however since the train does not go through all the vertices, there are many switches which we have no data on. This could increase the acceptance probability, since we just randomly propose a new switch setting for which there is no data. Interpreting Seong-Hwan Jun's messages on Slack on the 6th of January, the intention of this question is to study convergence. For the Gibbs sampler this is done throughout this question.

4 SMC for the stochastic volatility model

4.1 Question 9 and 10

We first generated the SV parameters and data according to the problem description with T = 100, $\phi = 1.0$, $\sigma = 0.16$ and $\beta = 0.64$, seed = 57832, and num˙particles=100. The observations, Y_t, which are the observed scaled log-returns from the asset is given by:

$$\begin{aligned} x_1 &\sim \mu(x_1) \\ x_t|x_{t-1} &\sim f(x_t|x_{t-1}) \qquad for\ t = 2, ..., T \\ Y_t|(X_t = x_t) &\sim \mathcal{N}(y_t|0, \beta^2 exp(x_t)) = g(y_t|x_t) \qquad for\ t = 1, ..., T \end{aligned} \tag{5}$$

However, in the original code provided the function generator uses the variance in numpy.random.normal. Since numpy.random.normal takes the standard deviation rather than the variance as an argument, this was changed. When sampling from the proposals in the various schemes the standard deviation was used rather than the variance.

We choose the proposal distribution q to be the same as the prior distribution for the transition, f, since it makes the weight update function much simpler. For $t = 1$ we simply sample $X_i^1 \sim \mathcal{N}(0, \sigma^2)$. For $t >= 2$ we use $q_i^t = f(x_i^t|x_i^{t-1})$. By doing this, we simplify the problem and get that our weight function alpha gets the same form as g:

$$\begin{aligned} \gamma(\boldsymbol{x}) = p(\boldsymbol{x}, \boldsymbol{y}) \text{ and } \nu(x_{1:t}) = \textstyle\prod_{i=1}^t q_i(x_i|x_{1:i-1}) \\ w(x_{1:t}) = \frac{\gamma_t(x_{1:t})}{\nu_t(x_{1:t})} = \frac{\gamma_{t-1}(x_{1:t-1})}{\nu_{t-1}(x_{1:t-1})} \frac{f(x_t|xt-1)g(y_t|x_t)}{q_t(x_t|x_{1:t-1})} = \\ w(x_{1:t-1})\alpha(x_{1:t-1}, x_t) \Rightarrow \alpha(x_{1:t-1}, x_t) = g(y_t|x_t) \end{aligned} \tag{6}$$

The mean squared error of the estimate to the truth was calculated as laid out by Seong-Hwan Jun on Slack on the 22nd of December: $E[f(X_T)] \sim \sum_{k=1}^K \bar{w}_k (x_T^k - x_T^*)^2$. To calculate a point estimate $\hat{x}_T$ of x_T we took the normalized weights in the last timestep multiplied with the X vector in that timestep and summed them up: $\hat{x}_T = \sum_{i=1}^N w_T^i * x_T^i$. The histogram of the weights for SIS, SIS with multinomial and SIS with stratified resampling is shown in figure 5 for space reasons.

4.1.1 Implementing SIS without resampling

We implemented sequential importance sampling (SIS) without resampling. The algorithm for SIS is

For t = 1:

- Sample from the proposal: $x_1^k \sim q_1(x_1)$
- Calculate the weights: $w(x_1^k) = \alpha(x_1^k)$

For $t >= 2$:

- Sample from the proposal: $x_t^k \sim q_t(x_t|x_{1:t-1}^i)$
- Extension: $\boldsymbol{x}^k = (x_{1:t-1}^i, x_t^k)$
- Calculate the weights: $w(x_{1:t}^k) = \alpha(x_{1:t-1}^i, x_t^k)$

Point estimate of x_T: $\hat{x}_T$ = -0.3733923717821612 (x_{truth} = -0.5078903908094016)
Empirical variance: 0.001199499525294217
Mean squared error:

No. particles	200	400	600
MSE	0.21110255368454361	0.156603760388357	0.05894792761451937

Table 4: Table for SIS MSE

4.2 Question 11 - Implementing SIS with multinomial resampling

The multinomial resampling is implemented according to the following algorithm in the function smc˙multinomial in q3.py.

For t =1:

- Sample from the proposal: $x_1^k \sim q_1(x_1)$
- Calculate the weights: $w(x_1^k) = \alpha(x_1^k)$
- Normalize the weights: $\bar{w}_1^k = \frac{w(x_1^k)}{\sum_{k'} w(x_1^{k'})}$

For $t >= 2$:

- Resampling: $i \sim Multinomial(\bar{w}_{t-1}^1, ..., \bar{w}_{t-1}^K)$
- Sample from the proposal: $x_t^k \sim q_t(x_t|x_{1:t-1}^i)$
- Extension: $\boldsymbol{x}^k = (x_{1:t-1}^i, x_t^k)$
- Calculate the weights: $w(x_{1:t}^k) = \alpha(x_{1:t-1}^i, x_t^k)$
- Normalize the weights: $\bar{w}_t^k = \frac{w(x_{1:t}^k)}{\sum_{k'} w(x_{1:t}^{k'})}$

Point estimate of x_T: $\hat{x}_T$ = -0.39431713173771993 (x_{truth} = -0.5078903908094016)
Empirical variance: 1.8075552967854315e-05
Mean squared error:

No. particles	200	400	600
MSE	0.34828795714742355	0.3499576861589596	0.44830003151149245

Table 5: Table for Multinomial MSE

4.3 Question 12 - Implementing SIS with stratified resampling

For the SIS with stratified resampling we used the same algorithm as we used for multinomial resampling, however we changed the resampling step to allow for stratified resampling as described in slide 10 of `https://www.cs.ubc.ca/~arnaud/samsi/samsi_lec3.pdf`. The resampling step of the algorithm becomes:

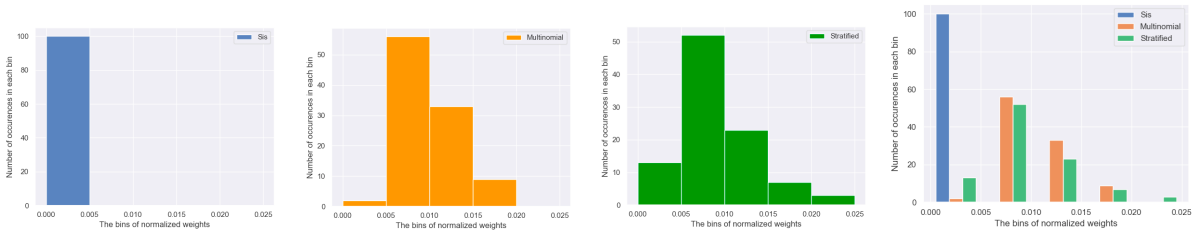
- First we uniformly draw U_1 uniformly:
  $U_1 \sim [1, N]$
- For every timestep >= 2:
  - Then for i = 2, ..., N:
    $U_i = U_1 + \frac{i-1}{N} = U_{i-1} + \frac{1}{N}$
  - After that we set:
    $N_n^{(i)} = \#\{U_j : \sum_{m=1}^{i-1} W_n^{(m)} \leq U_j < \sum_{m=1}^{i} W_n^{(m)}\}$, where $\sum_{m=1}^{0} = 0$

Point estimate of x_T: $\hat{x}_T$ = -0.2571731364313156 (x_{truth} = -0.5078903908094016)
Empirical variance: 2.573152738347793e-05
Mean squared error:

No. particles	200	400	600
MSE	0.38042687895988003	0.5004250478505947	0.45229182439438065

Table 6: Table for Stratified MSE

In Figure 5 a histogram of the weights in the last timestep is plotted for each of the three schemes.



(a) SIS (b) Multinomial (c) Stratified (d) Three Schemes

Figure 5: Histogram of weights in each scheme and all in one plot in (d).

4.4 Question 13

As can be seen in the weight histogram for SIS, the weights decay and almost all of them become close to zero, which does not happen for the other two schemes. When T increases only a few samples becomes relevant, therefore leading to waste of computational resources and resampling is therefore more suitable. Lastly, according to R. Douc and O. Cappé. in their 2005 paper "Comparison of resampling schemes for particle filtering", resampling methods can reduce variance of the estimator, which is consistent with the results in this report.