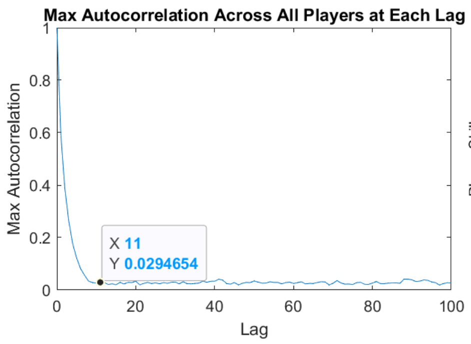
Question (a)

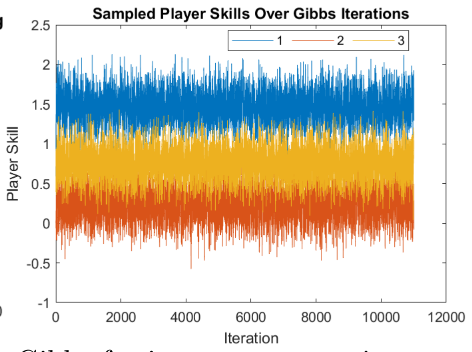
To compute the means `m` of the conditional skill distributions for the players, I added `m(p)=dot(t,(G(:,1)==p)-(G(:,2)==p))` because $\tilde{\mu}_i = \sum_{g=1}^{G} t_g\left(\delta\left(i - I_g\right) - \delta\left(i - J_g\right)\right)$. For the sum of the inverse precision matrices contributed by all the games `iS`, I computed them using:

```
iS(G(g,1),G(g,1)) = iS(G(g,1),G(g,1))+1; iS(G(g,2),G(g, 2)) = iS(G(g,2),G(g,2))+1;
iS(G(g,1),G(g,2)) = iS(G(g,1),G(g,2))-1; iS(G(g,2),G(g,1)) = iS(G(g,2),G(g,1))-1;
```

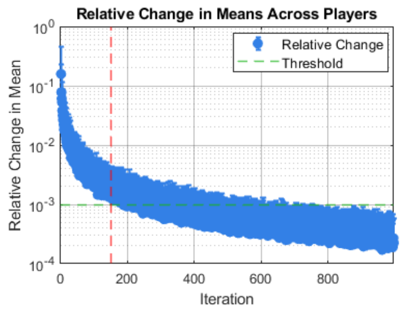
since $\tilde{\Sigma}_{ii}^{-1} = \sum_{g=1}^{G} \delta\left(i - I_g\right) + \delta\left(i - J_g\right)$ and $\tilde{\Sigma}_{i \neq j}^{-1} = -\sum_{g=1}^{G} \delta\left(i - I_g\right)\delta\left(j - J_g\right) + \delta\left(i - J_g\right)\delta\left(j - I_g\right)$ (winning or losing a game adds 1 to a player's diagonal entry and winning or losing a game between two players subtracts 1 from the non-diagonal entries corresponding to player i and j).

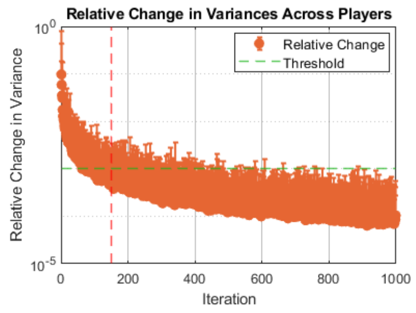
Below on the left graph are the sampled player skills for players 1 to 3 after running Gibbs sampling for 11000 iterations. We define convergence as the point where our sampled skills are independently and identically distributed (IID) and there are enough samples for the distribution to be inferred from the samples. We thinned the samples to make them IID by keeping only every 11th sample. We thin every 11th sample because the max autocorrelation of the sampled skills across all players at lags between 0 and 100 show that the autocorrelation is high before 11, and plateaus to a small value at around 11 – see the below graph on the right. Since the plot tells us the correlation between the samples for a player that are N iterations/lag apart, and takes the max autocorrelation across all the players, we use it to infer the number of iterations after which a new sample can be considered uncorrelated/independent to a previous sample. I validated the thinning by re-plotting the autocorrelation and it immediately plateaus at a lag of 1.





After thinning I determined how long to run Gibbs for it to converge using trace plots of the sampled skills: specifically by looking at the running mean and variance of the samples for each player at each iteration, which represents the evolution of the marginal skill distributions that Gibbs has been sampling from for the players. Convergence is when these distributions stop evolving, and so we compute the absolute difference in the mean and variance of the running marginal skill distribution for each player at each iteration and we seek when these approach 0. Since we have traces for each player, we get the overall picture of their differences over iterations by computing the mean and variance of the differences across all the players in an iteration, plotted below (the iteration numbers are divided by 11 now due to thinning):





Once the differences fall below 0.001, they are sufficiently small that the estimated marginals have converged, as the running mean skills vary roughly between 2 and -1 and 0.001 is very small relative to that range (also the performance variance of 1). The differences fall below 0.001 after roughly 865*11 = 9515 iterations (marked on the graph with the dotted green line), so we run Gibbs for 11,000 iterations so it can reinforce its collection of samples after having converged.

Finally, I determined the burn time by inspecting the difference graph and the autocorrelation. Since the burn time is the time during which the samples are too correlated to the first sample and are thus not IID from the underlying skill marginals, the burn time is lower bounded by the lag of 11. The relative change traces show significant changes in the estimated parameters of the underlying marginals in the first 150 * 11 = 1650 iterations (shown by the dotted red line), after which they have a constant gradient. In this region Gibbs may discovering new parts of the underlying distribution, so the samples may not be IID and we burn them,so our burn time is 1650.

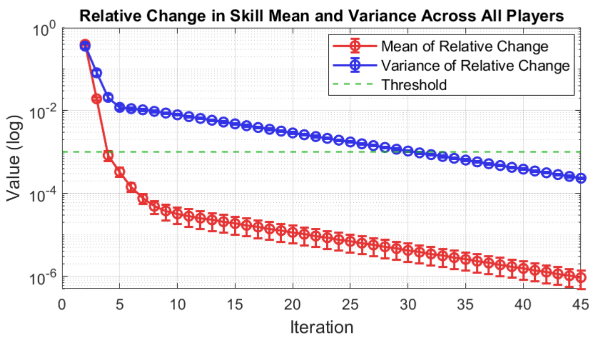
We get similar results repeating these steps with different seeds and thus different chains. Therefore, the chain didn't "get lucky" and reliably converges to the representative samples from the skill marginals. Ways to improve reliability could be to look at the differences over larger windows, monitor the evolution of the *joint* distribution of skills (rather than the individual marginals) as it has informaiton on the correlations between skills, or metrics like the Geweke Diagnostic.

Question (b)

For TrueSkill, Gibbs sampling converges to an IID collection of samples from the underlying marginals, since on every iteration it draws samples from them. As the number of iterations increases, these samples better cover/represent the full distribution of the underlying marginals. On the other hand, message passing and EP converges to the means and precisions of the players' skill marginals directly, since it iteratively refines its estimates of these parameters rather than drawing samples from them. So, Gibbs gives you samples from which you can estimate the distribution parameters, while message passing gives you the approximate distribution parameters directly.

Both algorithms have converged when the estimated parameters of the skill marginals have stabilised. For Gibbs sampling, this is when the obtained samples are IID from the underlying marginals and independent of the iteration number. We discard or "burn" the first N samples, since they are too dependent on the first sample (due to Gibbs sampling being a Markov chain). Low autocorrelation of the samples between consecutive iterations can be indicative of convergence and the burn time for Gibbs sampling, since it implies that the sample at τ is not correlated to the sample at $\tau + 1$. For both algorithms, the difference between the estimated parameters of the underlying marginals between each iteration can be indicative of convergence – this approaches 0 when the estimates from the algorithms have stabilised (for Gibbs sampling, we estimate the parameters via e.g. the running mean and variance). The "difference" measure could be the L1 distance for continuous parameters, or the KL divergence in other cases. One can also use statistical tests like the Geweke Diagnostic to determine if the samples after the burn time in Gibbs sampling are from a different distribution than the samples at the end of the Gibbs sampling run (a higher k-score with the Geweke Diagnostic indicates the samples come from different distributions). However, these metrics are only approximations and ideally one would run the algorithm for as long as possible (within the time and computational resources budget) to get the best accuracy.

For the number of iterations necessary, both algorithms must be run until they have converged. However, Gibbs sampling must be run briefly after convergence to obtain a representative collection of samples (this extra time is proportional to the variance of the underlying marginals). One must also run message passing briefly after convergence to ensure all of the parameters in the factor graph have converged. Changes in parameters lower down the graph could take a while to propagate up, so running the algorithm for longer ensures that isn't the case. Alternatively, one could monitor the



evolution of the estimated parameters lower in the graph, and stop earlier if the parameters at all levels have converged. In (a) we found from our trace of the mean and variance of the differences in marginal parameters that 11000 iterations were necessary to achieve our desired accuracy of differences in estimated means and variances ≤ 0.001, while 45 iterations were necessary to achieve this accuracy for message passing – see the below trace graph of parameter changes for message passing. Since 45 <<11000, message passing converges much faster than Gibbs in our context.

Question (c)

To compute the probability that player i has a higher skill than player j, $p(w_i > w_j|y)$ (the y is included implicity from here onwards), I ran `normcdf(m(i)-m(j),0,sqrt(1/pr(i)+1/pr(j))` where `m` is a vector of the mean skills for each player (from `Ms` at the last iteration of the algorithm) and `pr` is a vector of the precision of the skills for each player (from `Ps` at the last iteration of the algorithm) since we integrate over the region where $w_i - w_j > 0$ and the mean `mu` and variance σ^2 of the difference of two Gaussian random variables is $\mu = \mu_1 - \mu_2$, $\sigma^2 = \sigma_1^2 + \sigma_2^2$. Then, to compute to the probability that player i beats player j, $p(t_g > 0) = p(w_i - w_j + 1 > 0)$, I ran `normcdf(m(i)-m(j),0,sqrt(1/pr(i)+1/pr(j)+1))` since we integrate over the region where $t_g > 0$ and $t_g = w_i - w_j + n$ where $n = 1$ is the performance noise, so there is an additional variance in t_g. The tables for the probability of the top 4 players having higher skill and beating eachother are below (at row i and column j, the tables show $p(w_i > w_j)$ and $p(t_g > 0) = p(w_i - w_j + 1 > 0)$ respectively; a smaller index implies greater ranking in the ATP). Interestingly, our results imply that Federer (index 3) is likely more skilled than Nadal (index 2), disagreeing with the ATP.

$p(w_i > w_j)$:

–	0.9398	0.9089	0.9853
0.0602	–	0.4271	0.7665
0.0911	0.5729	–	0.8108
0.0147	0.2335	0.1892	–

$p(w_i - w_j + 1 > 0)$:

–	0.6554	0.6380	0.7198
0.3446	–	0.4816	0.5731
0.3620	0.5184	–	0.5909
0.2802	0.4269	0.4091	–

The key difference is that if a player is likely to be more skilled than another, then their probability of winning vs that opponent is less than the probability that they are more skilled, but still >0.5. On the other hand, if a player is likely to be less skilled than another, then their probability of winning vs that opponent is higher than the probability that they are more skilled, but still <0.5. This is due to the performance noise in the model causing strong players to sometimes perform worse than usual and weaker players to sometimes perform better than usual. This is explained as follows. The probability of the skill of player i being greater than j is $p(w_i > w_j) = \Phi(\frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2}})$, and the probability of player i winning against j in game g is $p(t_g > 0) = \Phi(\frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2 + 1}})$. By the definition of Φ, $p(w_i > w_j) > 0.5 \iff \frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2}} > 0 \iff \mu_i - \mu_j > 0$. When $p(w_i > w_j) > 0.5$, $\frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2}} > \frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2 + 1}}$, as the numerator is positive and the denominator is larger on the RHS.

Thus, $p(w_i > w_j) > p(t_g > 0)$, so the chance is winning is less than the chance of the skill being greater. On the other hand, when $p(w_i > w_j) < 0.5$, $\frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2}} < \frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2 + 1}}$, as the numerator is **negative** and the denominator is larger on the RHS. Thus, $p(w_i > w_j) < p(t_g > 0)$, so the chance is winning is greater than the chance of the skill being greater. Note that a player's skill being likely greater cannot imply their winning chance to be unlikely because $\text{sgn}(\frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2}}) = \text{sgn}(\frac{\mu_i - \mu_j}{\sqrt{\sigma_i^2 + \sigma_j^2 + 1}})$.

Question (d)

To approximate their marginal skills by Gaussians, I fitted a 1D Gaussian to the samples obtained for each player individually. I computed the mean and variance of the Gaussian for the player with index i by running `mean(skills(i, :))` and `var(skills(i, :))` (with $i = 1$ for Nadal and $i = 5$ for Federer). With these Gaussians, I computed the probability of Nadal's skill being greater than Federer's by using `normcdf` with the difference of the means and the sums of the variances, as in (c), and I obtained **0.4318**. To approximate their joint skills by a Gaussian, I fitted a 2D Gaussian to the samples obtained for all of the players at once. I computed the mean vector and covariance matrix of this Gaussian by running `[mean(nadal_skills), mean(federer_skills)]` and `cov([nadal_skills; federer_skills].')`. With this Gaussian representing $p(w_i, w_j)$ (i for Nadal and j for Federer), the two players' skills are no longer assumed independent and they have a covariance given by the bottom-left or bottom-right entry in the covariance matrix. Thus, the Gaussian for $p(w_i - w_j)$ has mean $\mu_i - \mu_j$ (μ_i = first entry in the mean vector of the Gaussian, μ_j = second entry) and variance $\sigma_i^2 + \sigma_j^2 - 2\text{Cov}(i, j)$ by the definition of the difference of two non-independent random variables. σ_i^2 and σ_j^2 are obtained as the two diagonal entries in the covariance matrix (representing the variance of the individual skills) and $\text{Cov}(i, j)$ is the bottom-left entry in the covariance matrix. We can then compute the probability of Nadal's skill being greater than Federer's using `normcdf` in the same way as before, to obtain **0.4220**. Finally, I calculated the probability of Nadal's skill being greater than Federer's directly from the samples in a monte-carlo fashion by calculating the number of iterations where the sampled skill of Nadal >the sampled skill of Federer, and normalising that by the total number of iterations, to obtain **0.4259**.

Out of these methods, comparing the skills by approximating their joint skills with a Gaussian is likely the best in our context because it takes into account correlations between the skills of the players, which isn't accounted for with the independent Gaussian per player (the variance of the skill difference Gaussian does not have the covariance subtracted as it's equal to 0 when the players' skills are assumed independent). These correlations could manifest themselves as the players having the same coach or the players both suffering similar injuries. Since in our TrueSkill context we assume the skills are Gaussian distributed, computing the probability directly from the samples does harness our prior knowledge of the players' skill distributions and is thus not likely to be accurate without a larger number of samples. However, it is more flexible for contexts without this Gaussian prior or even an intractable prior, so it does have merit, and is likely better than using individual Gaussians (supported by the fact that the computed probability directly from the samples lies in-between the probabilities via the Gaussians). Below are the tables for the probability of the top 4 players being more skilled when approximating their joint skills with a single Gaussian:

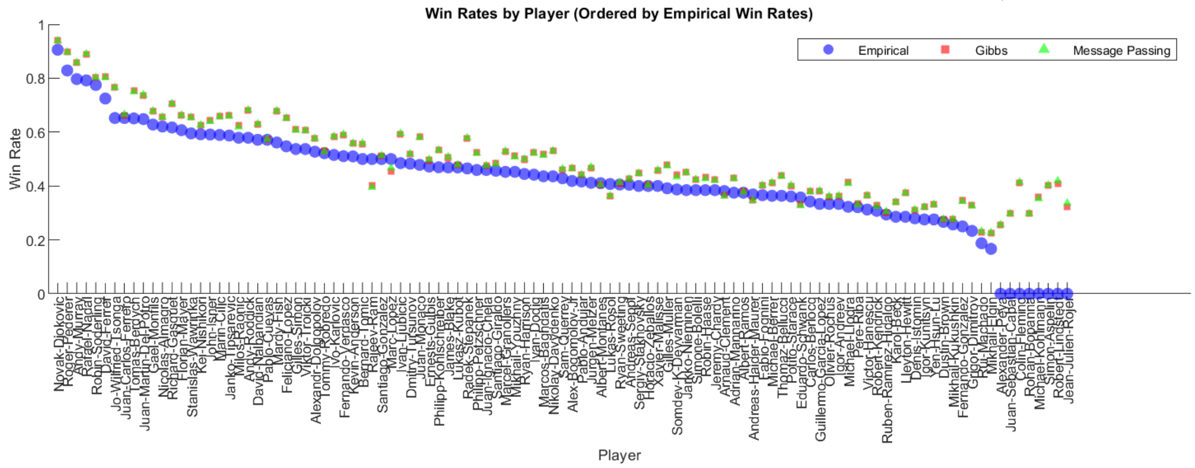
$p(w_i > w_j)$:

–	0.9571	0.9247	0.9865
0.0429	–	0.4220	0.7577
0.0753	0.5780	–	0.7992
0.0135	0.2423	0.2008	–

Compared to (c), the probabilities are very similar (the maximum absolute L1 difference is 0.0173), and thus the skill estimates from message passing and Gibbs sampling can be very similar in the context of TrueSkill. While message passing relies on approximations (moment matching and ignoring loops), its converged parameters are still very accurate.

Question (e)

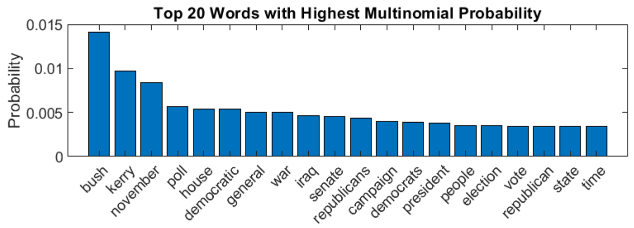
To calculate the rankings of the players using empirical averages, I calculated the ratio of the games a player won to the games that they played for each player from the dataset G. This gives us the players' win rates from the given outcomes. To calculate the rankings of the players using Gibbs sampling and message passing, I simulated each player playing against each other infinitely many times by, for each combination of players, assuming the obtained probability that the player beats the other is accurate. That probability is the expected outcome for games between the two players which is also equivalent to the expected win rate between the players due to the fact the outcomes are binary (0 or 1) and the win-rate is bounded by 0 and 1. Thus, for a particular player, we average their probabilities that they beat each other player, to obtain their expected win rate / outcome. To determine the probability of a player beating another via Gibbs, we compute the joint skill between the two players as a 2D Gaussian from the samples and use the CDF of the skill difference with the added noise as in (d). For message passing, we also use the CDF of the skill difference with the added noise, but we only have the parameters of the marginal skills for each player (no access to the joint), so the players' skills are assumed independent with our rankings from message passing. With these win rates computed for each player, we plotted them on a graph where the x-axis is the player and the y-axis is their win-rate via the 3 methods (the x-axis is ordered by empirical win-rates such that the highest empirical win-rate player is on the left):



Djokovic has the highest win rate with all 3 methods: a good validation sign. The win rates obtained via Gibbs and message passing are very close, however they do differ to the empirical win rates (but still follow the overall trend). Most clearly, the win rates are much higher with Gibbs and message passing for players with 0% empirical win rates. This is due to the performance noise in the TrueSkill model which adds randomness to their performances and can allow low skilled players to have "good days" or get lucky. Also, TrueSkill factors in the skill of their opponent, and so if the players with low win rates actually played against players with high skills, then their skill wouldn't be assumed low. The discrepancy is also due to the fact that we simulated players playing against every player with our TrueSkill (Gibbs / message passing) approximations. This might be why the win rates are generally higher with TrueSkill vs empirical, since the players can win against players they wouldn't normally play against, assessing their skill more comprehensively. Normally, players would play against players of similar skills, biassing the empirical win rates to gravitate around 50%. Finally, we look at differences in win rate rather than position in the ranking list because small changes in win rate can vastly change a player's ranking (e.g. in the middle of the graph), which can exaggerate differences and mislead analysis.

Question (a)

To set the scene, $\boldsymbol{k}$ is our observed/training data from `A` (the occurrences) and we intend to estimate the underlying word probabilities $\boldsymbol{\pi}$ across all documents in the corpus. The maximum likelihood multinomial over occurrences $\boldsymbol{k}$ is given by $p(\boldsymbol{k}|\boldsymbol{\pi}, n) \propto \prod_{i=1}^{m} \pi_i^{k_i}$. To find the maximum likelihood multinomial over words, we assume no prior and thus $p(\boldsymbol{w}|\boldsymbol{k}, \boldsymbol{\pi}, n) \propto p(\boldsymbol{k}|\boldsymbol{\pi}, n)$. Since we assume that the observed occurrences reflect the underlying word probabilities, $p(w_i|\boldsymbol{k}, \pi, n) = k_i / \sum_{j=1}^{m} k_j$, i.e. the maximum likelihood multinomial simply estimates the probability of a word by its relative frequency of occurrence. I implemented this by computing the word counts across all documents by running `word_id_counts = accumarray(word_ids, word_counts)` where `word_ids` and `word_counts` are the 2nd and 3rd dimension of `A` respectively. Then, I normalise the word counts by total number of words across all documents to obtain the probabilities by running `word_probs = word_id_counts / sum(word_id_counts)`. Plotting the probabilities of the top 20 words, I get:

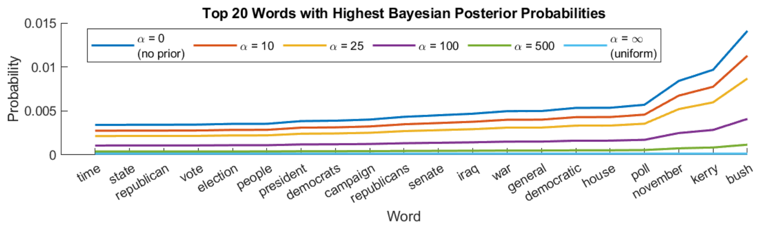


For our multinomial model ($\boldsymbol{\pi}$: the estimated probability of each word), if we let the word counts across the test documents be given by $\boldsymbol{k'}$ and $m = |\boldsymbol{k'}|$, then the test set log probability can be given by a categorical distribution using our multinomial probabilities for $\boldsymbol{\pi}$: $\log(p(\boldsymbol{k'}|\boldsymbol{\pi}, n)) = \log(\prod_{i=1}^{m} \pi_i^{k'_i}) = \sum_{i=1}^{m} k'_i \log(\pi_i)$. For a test set with n words, the maximal test set probability is n occurrences of the word with index $m = \arg\max_i(\pi_i)$ - the word which our multinomial model estimated as the highest probability - which from our graph is 'bush'. This is because every word with index j contributes $\log(\pi_j)$ to the total test set log probability, so we would get a maximal total probability by taking the maximal π_m every time. As such, the maximal log probability is $n\pi_m$. Similarly, the minimum test set log probability is obtained when the test set contains a word outside the corpus of words in the training set, since $\pi_j = 0$ for a word with index m' in the test set which does not exist in the training set and thus $\log(\pi_{m'})$ is $-\infty$, making the test set log probability $-\infty$. Since the test set log probability can be $-\infty$ if even only one of its words are outside of the training set, the test set log probability is highly dependent on the words present in the training set. Thus the simple multinomial model suffers from overfitting, since it has no prior to help it generalise to unseen test data. Another source of overfitting is that the most frequent words in the multinomial model are considered the most probable in the test data, which is overly simplistic and could be overestimating the true word probabilities if e.g. there is a large amount of repetitive noise in the training set. Note that, if we modelled the test set log probability with a multinomial over the test set, then the proportionality constant makes maximisation difficult and potentially intractable as it involves factorials in k' whose optimisation conflicts with the optimisation of the categorical part, which is why we have reasoned with a categorical test set log probability. The minimum test set log probability would still be minimal ($-\infty$) with a word outside the training set with a multinomial though, since the ∞ from the $\log(\pi_{m'})$ propagates.

Question (b)

To do Bayesian inference using a symmetric Dirichlet prior with a concentration parameter α on the word probabilities, we define the prior as $p(\boldsymbol{\pi}|\alpha) = \text{Dir}(\boldsymbol{\pi}|\alpha) \propto \prod_{i=1}^{m} \pi_i^{\alpha-1}$ and the likelihood as a multinomial: $p(\boldsymbol{k}|\boldsymbol{\pi}, n) \propto \prod_{i=1}^{m} \pi_i^{k_i}$. Using Bayes' rule, the posterior incorporates the observed counts ($\boldsymbol{k}$) and the prior belief / pseudocounts α, and is given by $p(\boldsymbol{\pi}|\boldsymbol{k}, \alpha, n) \propto p(\boldsymbol{k}|\boldsymbol{\pi}, n) * p(\boldsymbol{\pi}|\alpha)$. Since the Dirichlet distribution is a conjugate prior of the multinomial, the posterior is easy to compute: $p(\boldsymbol{\pi}|\boldsymbol{k}, \alpha, n) \propto (\prod_{i=1}^{m} \pi_i^{k_i})(\prod_{i=1}^{m} \pi_i^{\alpha-1}) = \prod_{i=1}^{m} \pi_i^{k_i+\alpha-1}$ (where the normalisation constant is a combination of the $B(\alpha)$ from the prior and the combinatorial factor from the likelihood), which is a Dirichlet distribution with parameters $k_i + \alpha$. Since the expected value of the Dirichlet distribution is given by $E(\pi_j) = \alpha_j / \sum_{i=1}^{m} \alpha_i$, we can compute the probability of each word directly: $p(w_j) = E(\pi_j) = (k_j + \alpha)/(\sum_{i=1}^{m}(k_i + \alpha))$.

Note that predicted probability of each word is same as in MLE but with added pseudocounts α to each k_i. As such, larger values of α cause the Bayesian predictions to diverge from the MLE predictions. Since the MLE relies on the occurrences to be accurate measures of words' probabilities, common words will have enough occurrences for the MLE to reliably predict their probability. However, for rare words, the probability predictions from MLE may be unreliable due to them having few occurrences in the training set. For a pseudocount $\alpha > 0$ we address this issue, as each word is assumed a baseline number of occurrences α in the Bayesian model, thus the predicted word probabilities are closer to the uniform distribution than with the MLE (since the MLE does not factor in these pseudocounts). Therefore, the Bayesian predicted word probabilities are a 'smoothed' version of the probabilities using just the MLE, and they also take into account rare words. When α is too large, the Bayesian estimates are too uniformly biased, and common words are too heavily penalised (since the distribution has to sum to 1). See the below graph for the estimated probabilities from our Bayesian model when varying α in the Dirichlet prior:



A balance needs to be found when setting α. It cannot be too small because otherwise rare words are not sufficiently accounted for, and it cannot be too large because otherwise the predicted probabilities are too uniform and do not express the true distribution (which we know is not uniform from Zipf's law). We choose $\alpha = 10$ as a middle-ground.

Question (c)

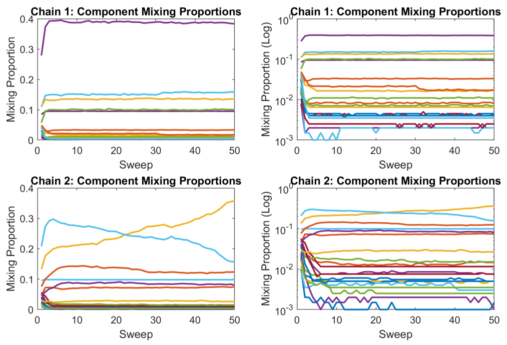
For the log probability of a single test document, since our data is a bag of words, the multinomial might technically be the correct distribution to use. However, for simplicity, we use the categorical distribution as it's computationally simpler than the multinomial distribution and it's proportionally the same, so comparisons of categorical probabilities preserve the ordering of the multinomial comparisons. Furthermore, our simple bag of words model is based entirely on word frequencies rather than their sequence, thus including the combinatorial factor to consider all possible sequences that generated the counts may cause probabilities to be overestimated, making the categorical model potentially more suitable. As such, letting $\boldsymbol{k'}$ denote the occurrences of the words in the test document, we compute the log probability using our Bayesian model from (b) - which estimates the underlying word probabilities $\boldsymbol{\pi}$ - by taking the log of the categorical distribution: $\log(p(\boldsymbol{k'}|\boldsymbol{\pi}, n)) = \log(\prod_{i=1}^{m} \pi_i^{k'_i}) = \sum_{i=1}^{m} k'_i \log(\pi_i)$. Note that $m = |\boldsymbol{k'}|$ and π_i is actually the word probability from the Bayesian model corresponding to the i'th unique word in document 2001. I implemented this using `log_prob_2001 = sum(log(bayes_word_prob(words_2001)) .* counts_2001)`, where `words_2001` contains the IDs of the words in document 2001, `counts_2001` contains the counts of the words in document 2001 and `bayes_word_prob` contains the Bayesian word probabilities obtained from (b). Doing this, I obtain **-3681** as the log probability for document 2001: a low probability, and thus our probabilities from our Bayesian model with the Dirichlet prior do not seem to generalise well to this document. We can verify this by comparing the log probability with other test documents. For example, the log probability is -321 for document 2002 and -1085 for document 2003: both larger than the log probability for document 2001.

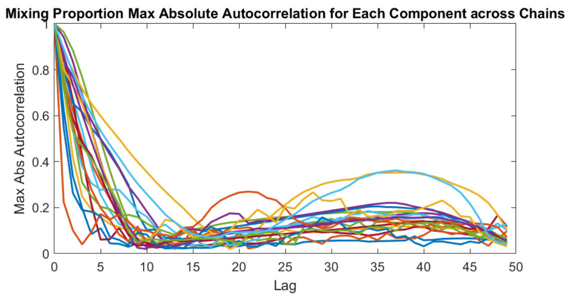
For the per-word perplexity of document 2001 in the test set, we compute the log joint probability over words as $\ell = \log(\prod_{i=1}^{m} \pi_i^{k'_i}) = \sum_{i=1}^{m} k'_i \log(\pi_i)$, the same as the log probability of the document. We use our previous Bayesian word probabilities $\boldsymbol{\pi}$ because the perplexity is a measure of how well our trained model generalises to the test data. Then, we compute the number of words in document 2001 as $n = \sum_{i=1}^{m} k'_i$ and the perplexity as $\exp(-\ell/n)$, by definition. I implemented this using `perplexity_2001 = exp(-log_prob_2001 / sum(counts_2001))` to obtain a per-word perplexity of **4294** for test document 2001 using our Bayesian model for the word probabilities. I computed the per-word perplexity over all documents in B in a similar way. In the previous equations for ℓ and n, adjust $\boldsymbol{k'}$ to be the count across all documents of every word in the corpus of test documents and then recompute $m = |\boldsymbol{k'}|$ and the equations still hold. To implement the per-word perplexity across all test documents, I used the same code to compute the log probability and perplexity of test document 2001, but used `B(:,2)` in place of `words_2001` and `B(:,3)` in place of `counts_2001` (with an `accumarray` to overcome duplicates), yielding **2729**. The perplexity is 57% higher for the words in just test document 2001 compared to the words across all the test documents, thus there is significantly greater uncertainty in the word counts for test document 2001 when using our Bayesian model when compared to the word counts across the entire corpus of test documents (they are 'generated' by a die with 57% more sides). This could be due to the words in test document 2001 being an 'outlier' and containing very different to the words in most of the training corpus of documents, or our Bayesian model being ineffective at representing the true word probabilities for documents similar to test document 2001 (it could come from a rare topic which uses unique words when compared to the rest of the corpus, and topics aren't an element of our simple model). Generally, we expect the perplexities to be different for different documents as they contain different distributions of words, which may align to varying degrees with the word counts in the training corpus from which the word probabilities were estimated.

Finally, when our model is a uniform multinomial, we would expect the perplexity to be higher than the perplexity with our Bayesian model from (b) because Zipf's law tells us that the probability distribution over words is not uniform. Further, since we use no information from the training data with a uniform multinomial (it's akin to randomly generating words), the Bayesian model would maximally underfit the true word probabilities. I verified that the perplexities are worse with a uniform multinomial by setting $\alpha \to \infty$ in the Dirichlet prior for our Bayesian model in (b), and then computing the perplexity with this uniform multinomial. Over only document 2001 and the entire test corpus, I obtain a perplexity of **6892** for both, which is larger than the perplexities from our original Bayesian model, validating that the perplexities are worse with a uniform multinomial.

Question (d)

To compute the mixture proportions, we normalise the counts c_k for each mixture component k, where c_k is the number of documents of ID d such that $z_d = k$, into probabilities by computing $c_k / \sum_{i=1}^{K} c_i$. In the code, `sk_docs` stores c_k and thus we compute the mixture proportions for each component as `sk_docs ./ sum(sk_docs)`. We store these mixture proportions after every Gibbs sweep. To measure convergence of the collapsed Gibbs sampler, we use traditional methods for measuring convergence of Gibbs sampling: analysing traces, autocorrelation and the Gelman-Rubin statistic. We can apply these methods to the mixture proportions, c_k, because they are a direct function of the variable we are sequentially sampling with our collapsed Gibbs sampler: z_d. To measure multiple chains, we extended the code to compute 5 different chains of the Gibbs sampler, by adding an outer loop to repeat the Gibbs sampler 5 times and storing the mixture proportions per iteration for each chain. With the mixing proportions for component k from multiple chains available in `chain_sk_docs` of size `nsweeps x nchains`, we compute the Gelman-Rubin $\hat{R}$ statistic to measure convergence of the sampled document components across the chains. It is given by $\hat{R} = \sqrt{(\frac{N-1}{N} + \frac{B}{W} * \frac{1}{N})}$ where N = number of sweeps = 50, B = the between-chain variance and W = the within-chain variance. We compute the within-chain variance as `mean(var(chain_sk_docs, [], 2))`, the between-chain variance as `var(mean(chain_sk_docs, 2))` and the $\hat{R}$ statistic as `R_hat(k) = sqrt((nsweeps - 1) / nsweeps + (B / W) * (1 / nsweeps))`. For the autocorrelation, we compute the maximum absolute value of the autocorrelation across the chains at each lag, computed via `xcov` with some post-processing, since we assess closeness to 0 of the autocorrelation when assessing convergence, so the max absolute value across all chains indicates divergence from 0 in any chain. Below on the left shows plots of the mixing proportions for the first two chains across Gibbs sweeps in both normal and log space. We show the mixing proportions in log space too because it is difficult to see variations in components with small mixing proportions on the normal plot. On the right, we show the max absolute value of the autocorrelation across the 5 chains at each lag.

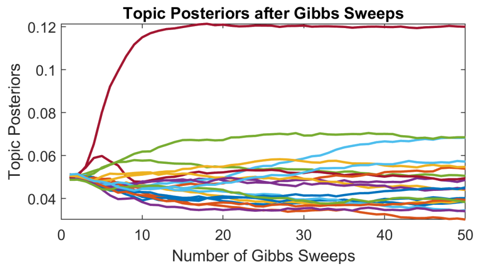


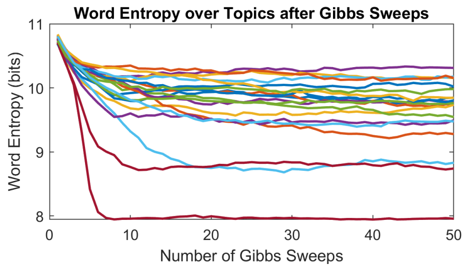


From the data there are several indications of non-convergence to the posterior. Starting with the traces, there are components whose mixing proportions vary after many sweeps. For the components that have seemingly converged, Gibbs appears to get stuck in local optima since the distribution of the mixing proportions across the components is very different between the chains. From the autocorrelation plot, the sampled document components for each component are roughly independent at a lag of 15, but it increases again at approx. 25. This may indicate that, during Gibbs (after at least 15 sweeps), Gibbs explores the posterior, but after that it gets stuck in a local optimum and stops exploring, since the sampled document components are subsequently correlated and non-independent. Gibbs appears to begin exploring again after approximately iteration 45, but we would need to run it for longer to see if it converges after iteration 50. On the other hand, the maximum Gelman-Rubin $\hat{R}$ statistic we get across the components is **0.9962**, which is significantly less than 1.1 and would indicate convergence of the chains (since all of the $\hat{R}$ values are less than 1.1 for each component). Despite this result, I still think Gibbs has generally not converged, and in the *best* case it has converged to a local optimum. It does appear to have explored the posterior, as the autocorrelation is low briefly (approx. between lags 10 and 25), but the rising autocorrelation and non-stable mixing proportion traces indicate to me that we have not converged to the posterior. Doing more sweeps of Gibbs would be necessary to determine convergence (also checking $\hat{R}$).

Question (e)

For the topic posteriors, we want the probability distribution of the topics across all documents in the corpus: $p(\text{topic}|\text{document})$. Since the topic assignments have a symmetric Dirichlet prior parameterised by α, we compute this likelihood by leveraging the conjugacy property of the Dirichlet distribution. This involves adding α to the total count of words assigned to topic k across all documents and then normalising the result. We implement this as `topic_posterior = (sk + alpha) ./ sum(sk + alpha, 1)` at the end of each iteration of the Gibbs sampling loop (`sk` contains the count of all words assigned to each topic across all documents). For the word entropies, we want the probability distribution of the words that are assigned a particular topic, $p(\text{word}|\text{topic})$. These are given by the topics of the observed words $\boldsymbol{\beta}_k$ in the LDA model. Since the topic assignments have a symmetric Dirichlet prior paramaterised by γ, we compute this likelihood by similarly leveraging conjugacy. We add γ to the counts of the topics assigned to each word w across all documents (as $\boldsymbol{\beta}_k$ only touches the word plate and not the document plate in the graphical model) and then normalise the result. We implemented this as `betas = (swk + gamma) ./ sum(swk + gamma, 1)` at the end of each iteration of the Gibbs sampling loop (`swk` contains the counts of the topics assigned to each word across all the documents). With these $\boldsymbol{\beta}_k$'s, we can compute the Shannon entropy across words for each topic by definition: for each topic k we compute $-\sum_{i=1}^{n} \boldsymbol{\beta}_{ki} \log_2 \boldsymbol{\beta}_{ki}$ where n is the number of words and $\boldsymbol{\beta}_{ki}$ is $p(\text{word} = i|\text{topic} = k)$. We use log base 2 here because Shannon entropy is an information-theoretic concept and it gives the uncertainty quantified in the number of bits of an outcome (a word in this case). As such, we calculate the entropy using `sum(-betas .* log2(betas), 1)`. Below shows the topic posteriors and word entropies (in bits) after each sweep:





Since our initial assignment of topics to words (values of z_{nd}) is random, the topic posteriors also begin roughly uniform. After approx. 15 sweeps, the topic posteriors have diverged from uniform and almost converged/plateaued for most topics. This happens because we update our estimates of z_{nd} after each sweep, since our collapsed Gibbs sampler is sequentially sampling the latent z_{nd}'s and the actual topic distribution is non-uniform. We see a similar trend for the word entropies. Initially, it is maximal since the initial word-topic distribution is uniform, and the uniform distribution has the highest Shannon entropy. After roughly 15 sweeps, the word entropies have decreased and mostly plateaued. LDA appears to converge to a large range of possible topic posteriors and word entropies, implying that there are more dominant topics than others in the corpus, both in terms of their frequency (indicated by the topic posterior) and distinguishability (indicated by the word entropy). This is particularly evident with the red topic that has both the maximum topic posterior and minimum word entropy, implying that it is estimated as respectively the most common topic among the documents in the corpus and the easiest topic to distinguish from the words. This topic could be, for example, 'Financial Market Trends', since there could be weekly reports for the topic (high frequency) and documents of the topic contain a lot of financial jargon (easy to distinguish).

Finally, we look at the perplexity for the test documents to evaluate our LDA model. After 50 sweeps we get a perplexity of **1658**, which is over 1000 less than the perplexity across all test documents with the Bayesian model trained in (b), so LDA appears to be better estimating the true distribution of the words in the corpus. This is not surprising since LDA uses 'topics' to categorise documents and their words: a concept absent from the Bayesian model. The convergence of the perplexity can also imply the convergence of LDA, since the perplexity is a function of the estimated word probabilities that we are sampling. With $10 \rightarrow 20 \rightarrow 30 \rightarrow 40 \rightarrow 50 \rightarrow 70$ sweeps we obtain perplexities of $1880 \rightarrow 1728 \rightarrow 1701 \rightarrow 1649 \rightarrow 1658 \rightarrow 1633$. With 20 more iterations than 50, our best measured perplexity decreases by only 13, so it appears that the perplexity with 50 iterations is adequate to get an optimal perplexity from our run of LDA. However, if one intends to run LDA until it has fully converged, then one should run it for more than 50 sweeps as the perplexity still technically decreases after 50 sweeps (more diagnostics could increase confidence, like in (d)).