Project1

Markov Chain Monte Carlo techniques

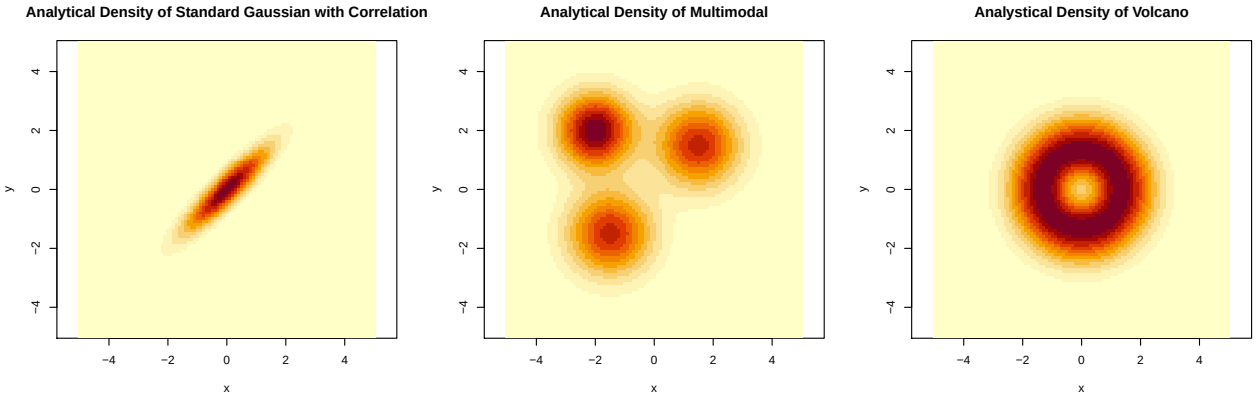
Florian Beiser, Yaolin Ge

1 Metropolis-Hastings for Bivariate Densities

We consider three different bivariate densities. 1. Standard Gaussian with correlation 2. Multimodal as muixture of three Gaussians 3. volcano (unnormalized)

1.1 Plotting

We start the project with the visulaisation of the respective densities in $[-5, 5] \times [-5, 5]$. (Whyever R likes to put more white space left and right of the plots, but the middle areas are still representative.)



For the subsequent part, where different Metropolis Hastings MCMC algorithms will be implemented, we prepare with the skeleton that was discussed in the exercise class and uploaded. This defines the frame for all MCMC implementations under consideration which differ in the `proposal_func` and `acceptance_func`.

These implementation is not tweaked for optimal performance, e.g. log-scale could lead to better scaling or reduction of redundant calculations in the individual MCMC versions could lead to fast execution, but we want to the same skeleton for all algorithms to higlight similarities and differences. For the analysis of the outcomes, we use below's plotting functionality.

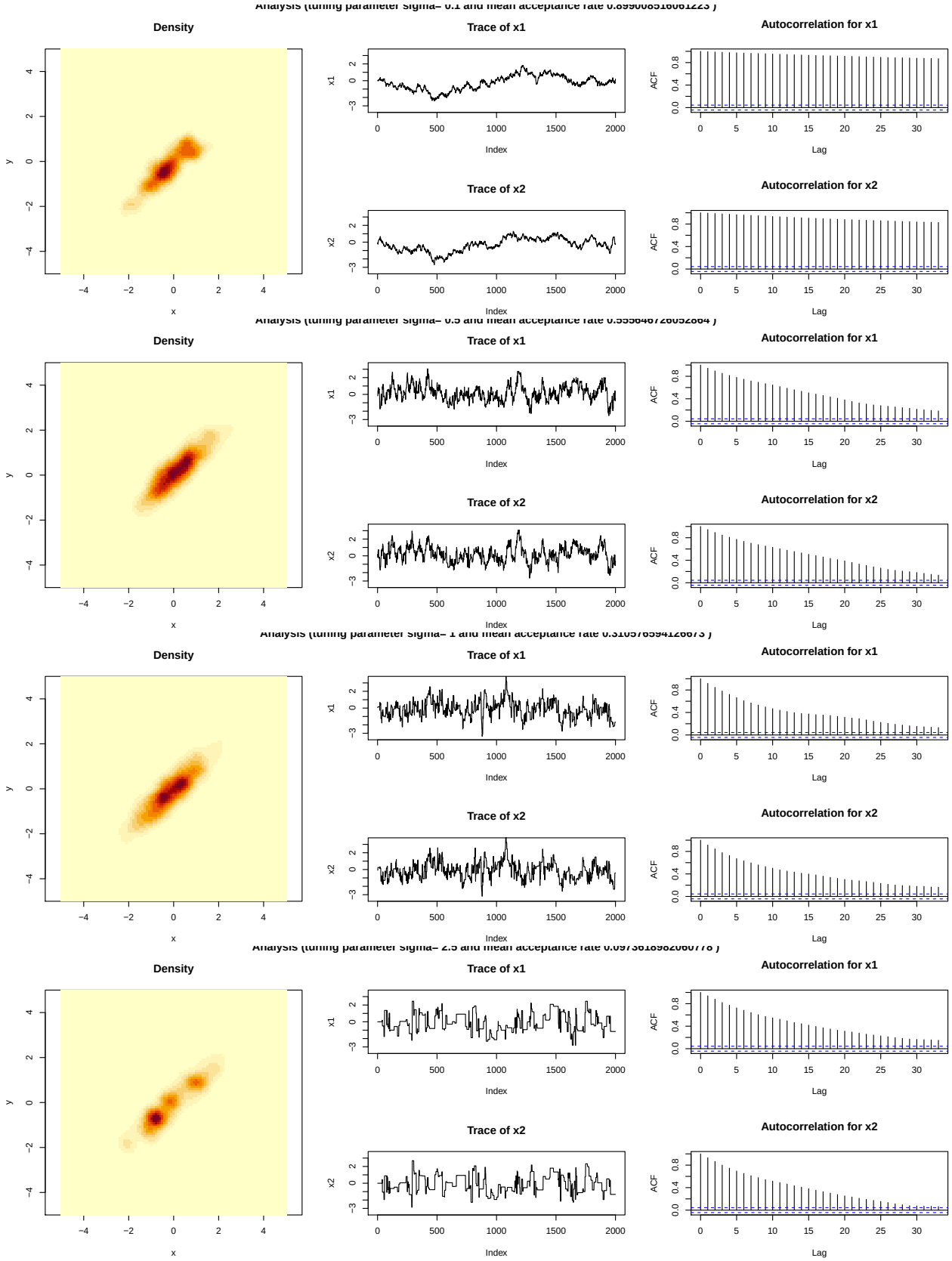
We will use the same input parameters for all subsequent MCMC calculations, the tuning parameters are of course not influenced.

1.2 Random walk MH

The random walk MH uses a symmetric $\mathcal{N}(x_{i-1}, \sigma^2)$ distribution to generate the new proposal. Moreover, the general MH acceptance probability simplifies.

Standard Gaussian

We first test the Random Walk MH for the Gaussian with different tuning parameters σ in the proposal distribution.

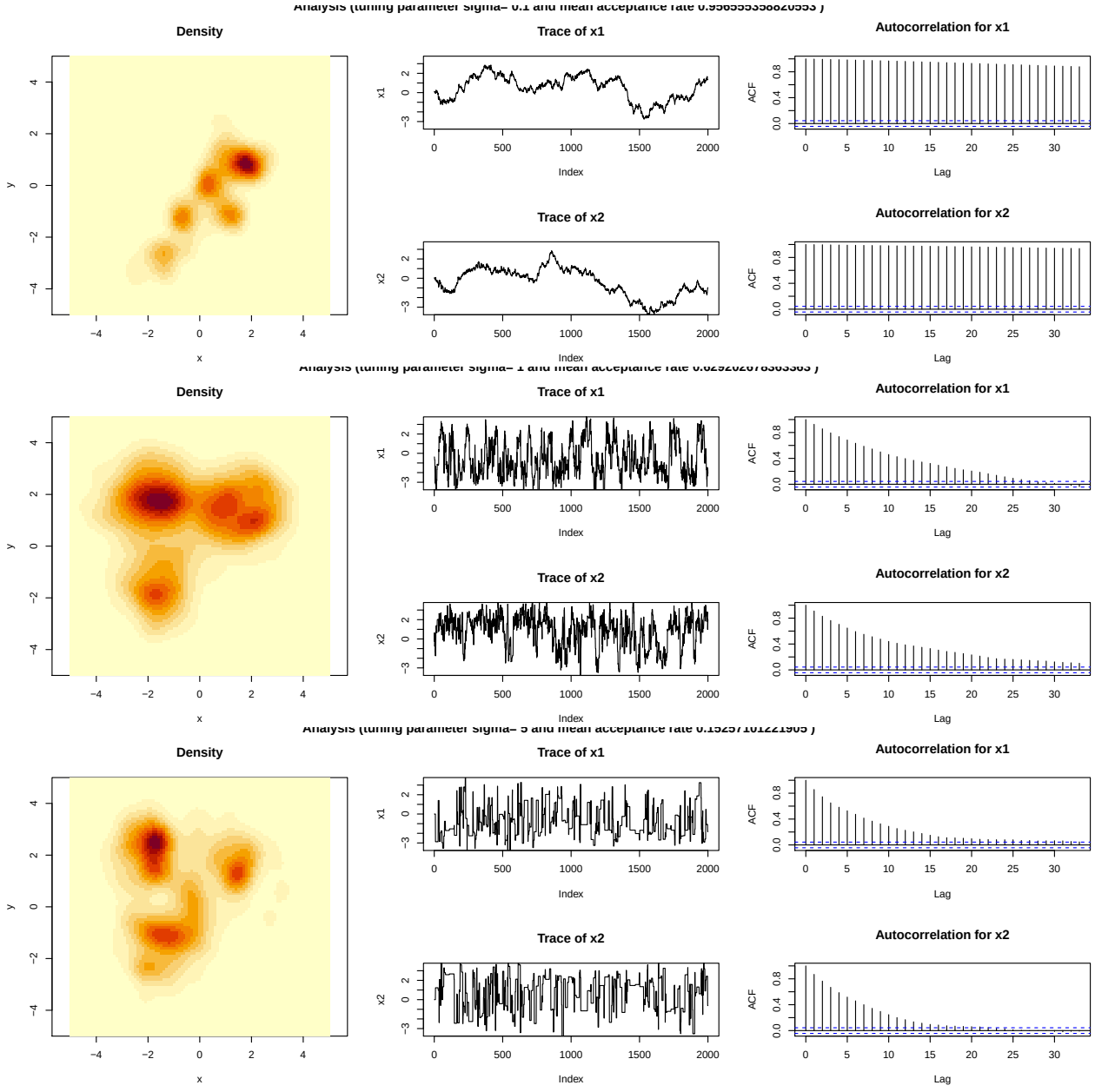


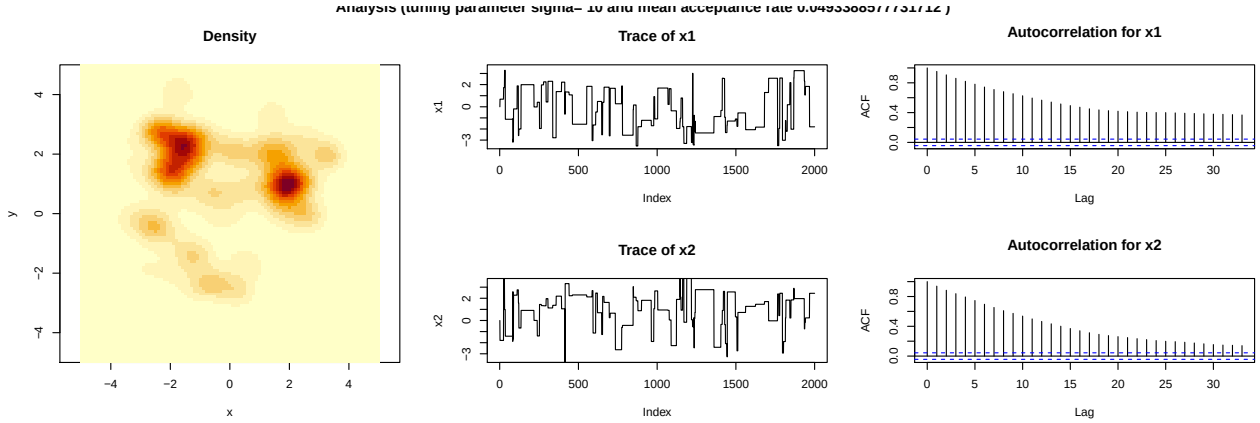
Depending on the choice of the tuning parameter the MCMC algorithms show different efficiency. Actually all

tuning parameters explore the state-space rather sedately. However, for values $\sigma < 0.5$ the mean acceptance rate gets bigger than recommended and the traces change in too little step, for values $\sigma > 1.5$ the traces start to pause in a level due to a too small acceptance rate. We recommend values $\sigma \in (0.5, 1.5)$ for the Gaussian, since for those values the autocorrelation has t he smallest lag and shrinks at least after 20 steps (what is still much).

Multimodal

We continue with application of the Random Walk MH to the Multimodal with different tuning parameters σ in the proposal distribution.

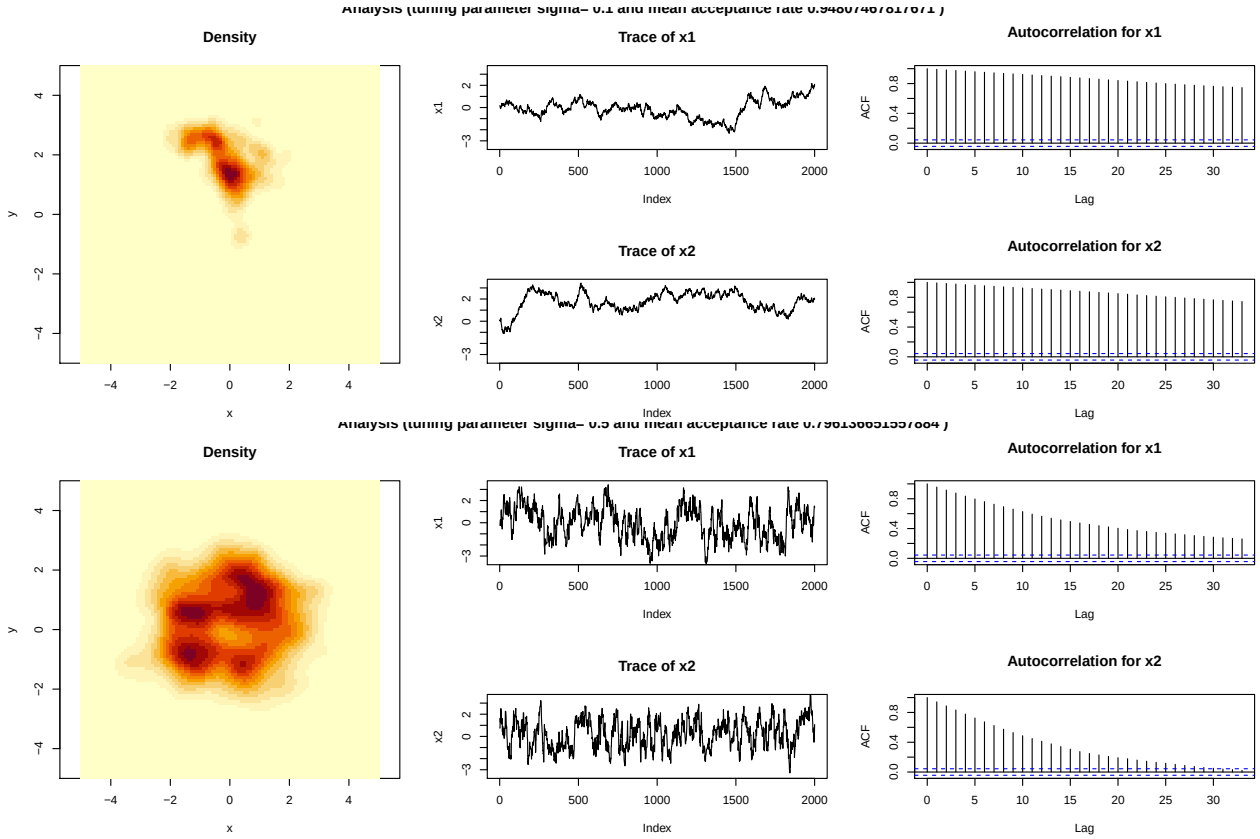


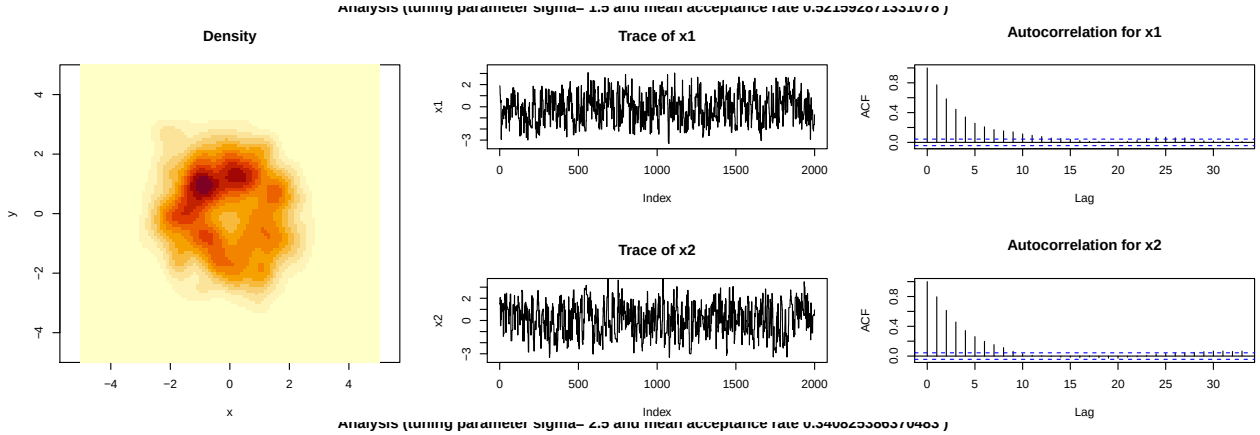


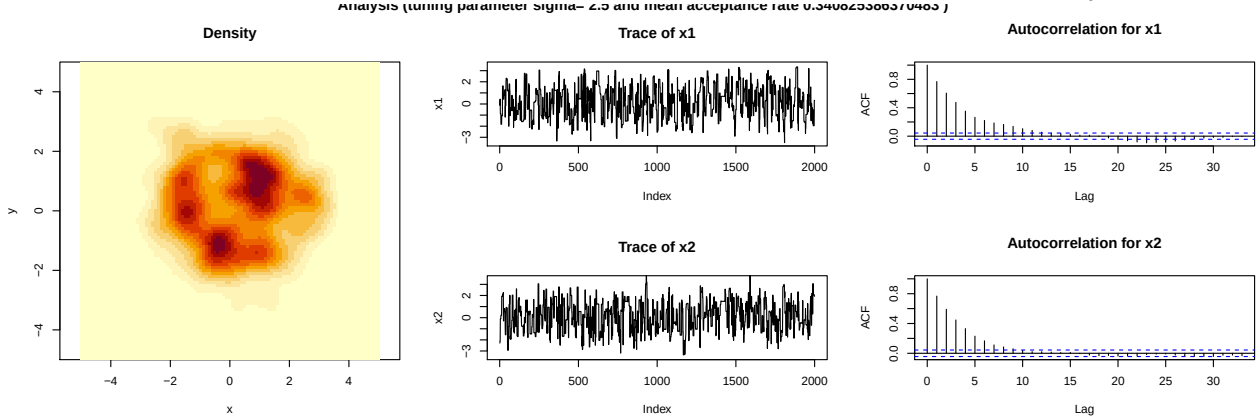
For the smallest choice of $\sigma = 0.1$ the chain does not really explore all modes and stays for quite long in one mode when it is there, therefore the autocorrelation is very high. When increasing up to $\sigma = 5.0$ we improve the exploration of the different modes and reduce the autocorrelation significantly. Moreover, the mean acceptance rate reaches the recommended range. For very high tuning parameter choices $\sigma = 10.0$, we start to wildly jump from one mode to the other, but the traces start to pause too long between the jumps.

Volcano

Finally, we try the random walk MH for the volcano shaped density.







If the tuning parameter is chosen too small $\sigma = 0.1$ the chain does not explore the entire ring, but get stuck. When increasing σ the chain starts to walk along the circle, but for higher parameters like $\sigma = 1.5$ the chain explores the ring with a short autocorrelation (figuratively speaking, it can also jump from one side toe the other and does not need to walk along the circle).

Conclusion

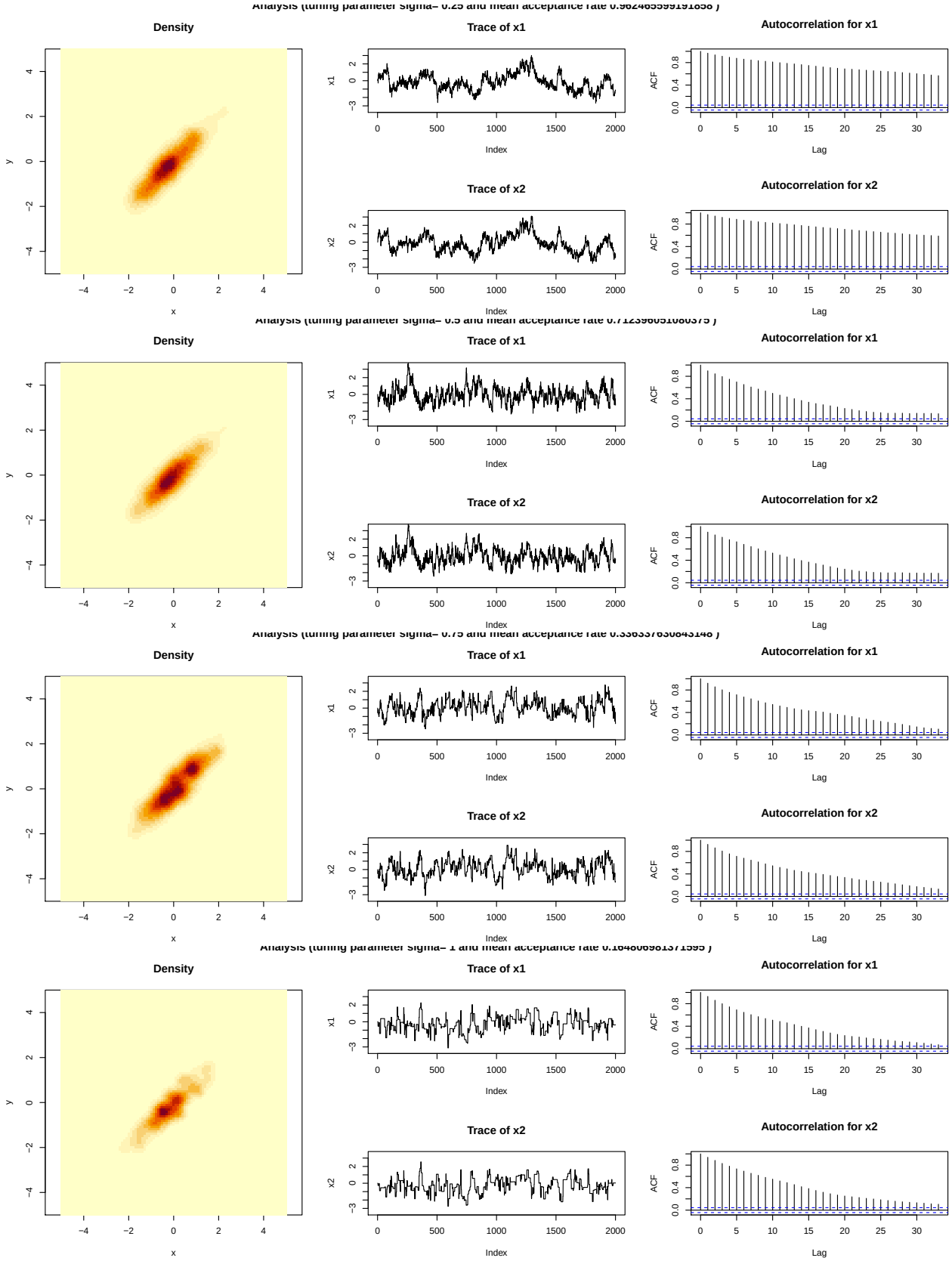
For all examples, there is a range of runing parameters which explore the state-space, but the autocorrelation is still pretty high.

1.3 Langevin MH

Based on the Langevin dynamics and its Euler-Maruyama discretisation, the MALA algorithm uses gradient information to define the proposal density.

Standard Gaussian

We first test the Langevin MH for the Gaussian with different tuning parameters σ in the proposal distribution.

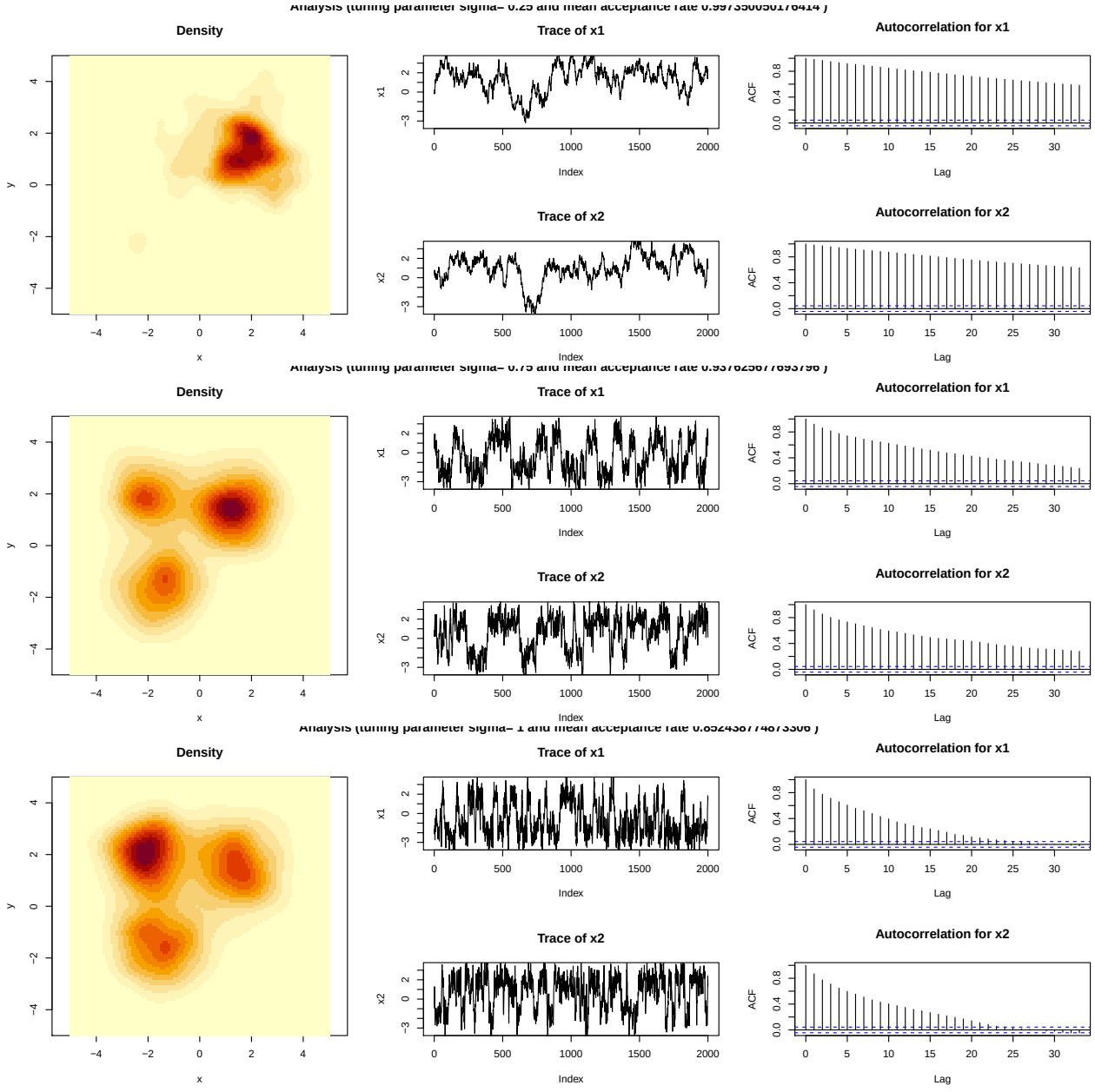


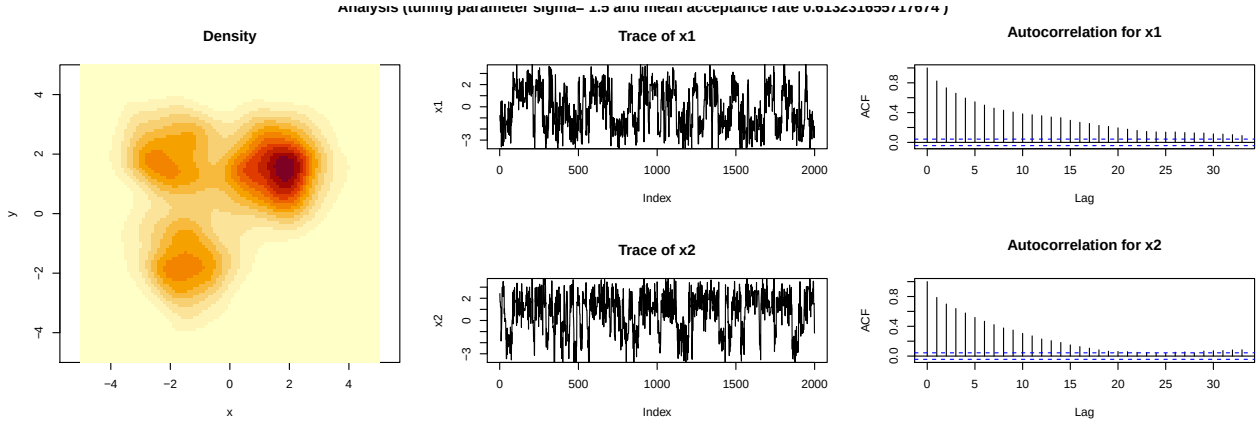
Even if the mean acceptance rate for small parameters $\sigma \leq 0.5$ is in the asymptotically optimal range, the trace evolves to slow and with too high correlation. For $\sigma = 0.75$ the target density is replicated - the

autocorrelation decreases and the explorance is at least acceptable. Already for $\sigma = 1.0$ the acceptance rate is too small and the chain pauses too long to explore the space.

Multimodal

We continue with application of the Langevin MH to the Multimodal with different tuning parameters σ in the proposal distribution.

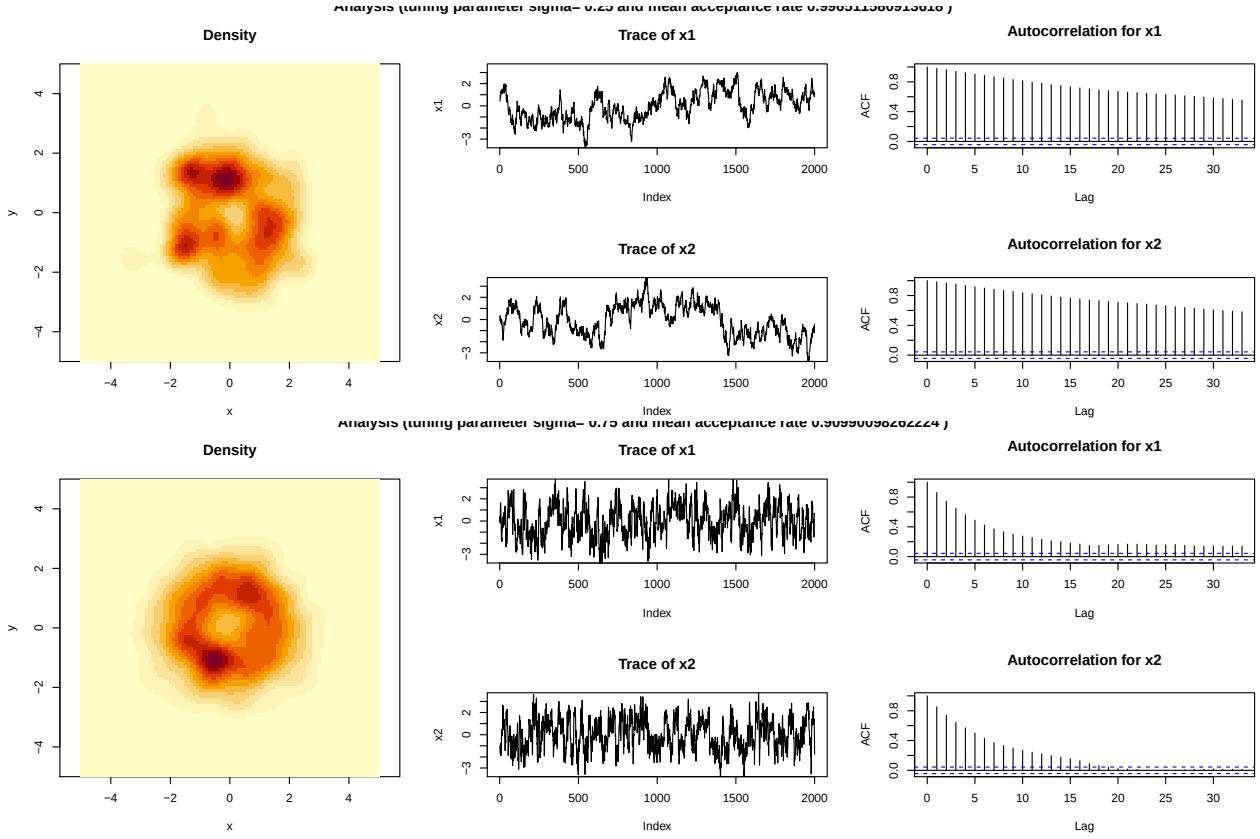


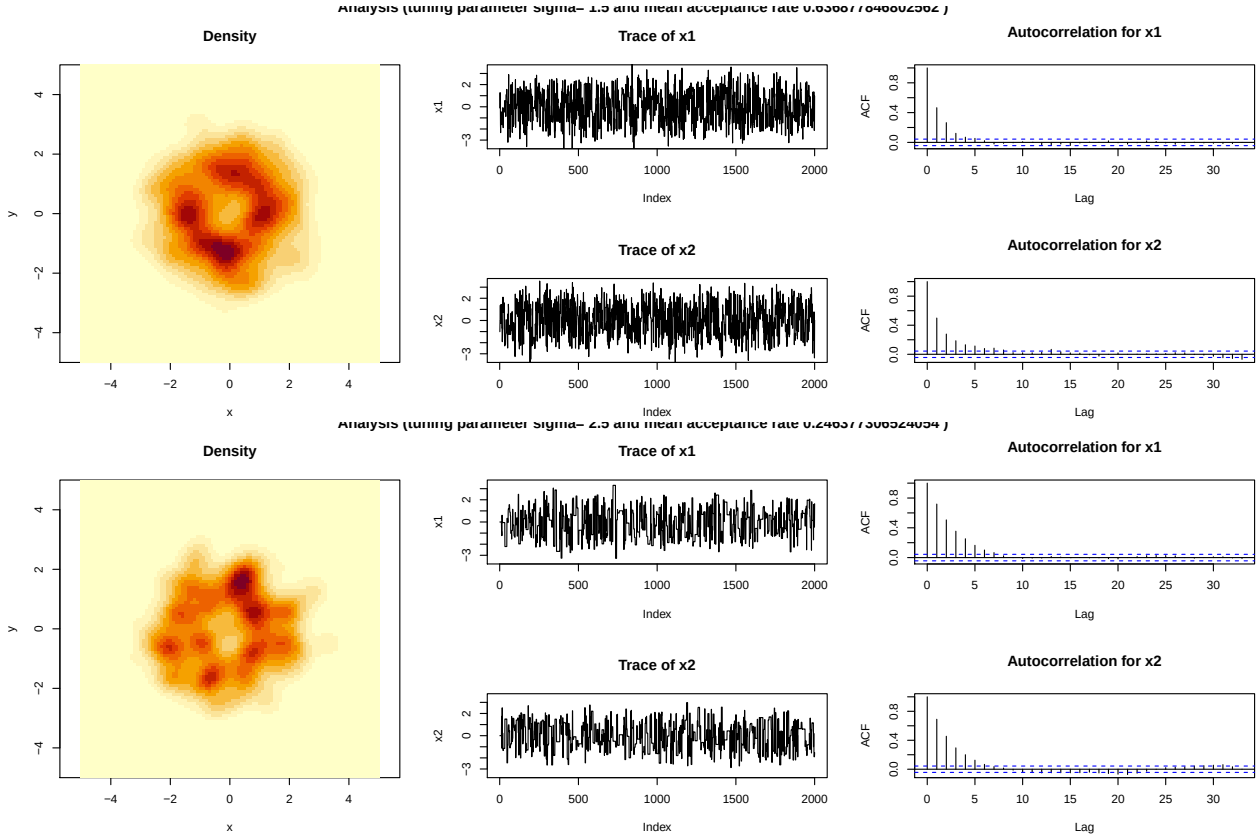


For small tuning parameter $\sigma \leq 0.75$ the Langevin chain get stuck in the local maxima of the multimodal dsitribution too long and hence does explore the other modes too badly. For higher parameters $\sigma \geq 1.0$ the chains reduce their autocorrelation and start to explore all modes.

Volcano

Finally, we try the Langevin MH for the volcano shaped density.





For resonably big choices of $\sigma = 1.5$ a chain can be generated which does explore the entire ring with small autocorrelation and an acceptance rate close to the asymptotic optimum. For too small and too big choices the same as for the Random Walk holds: Only a fraction of the ring is explored for too small $\sigma = 0.25$, then the chains starts to walk along the circle for $\sigma = 0.5$. For too big proposal spread with $\sigma = 2.5$ too many large steps will be rejected.

Conclusion

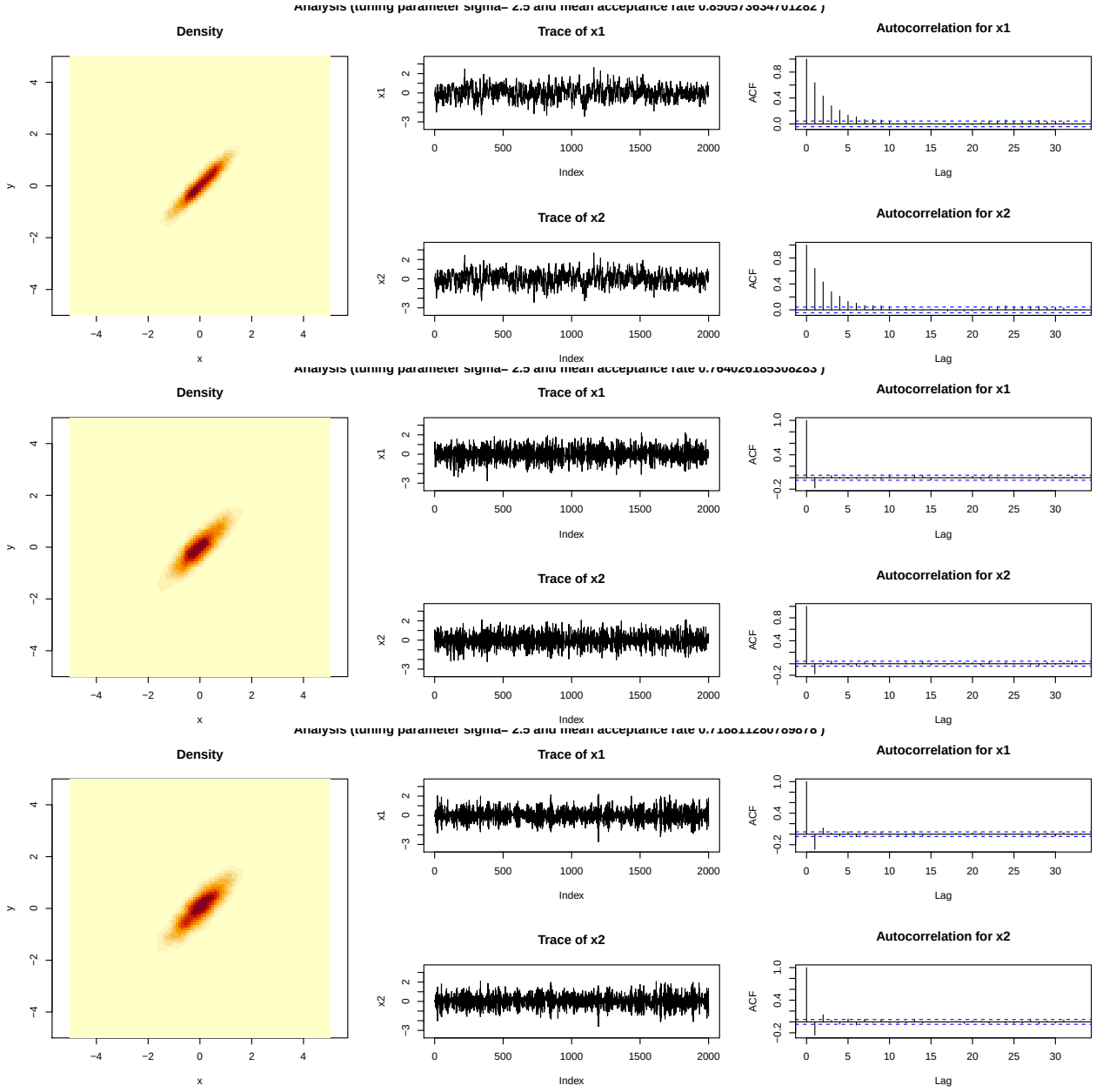
We saw that the MALA algorithm is very sensitive to small changes in tuning parameter σ. For too small choices it tends to get stuck in local maxima, what is a known issue and we realized that behaviour here as well.

For the Gaussian distribution, the Langevin MCMC version works nicely, but the Random Walk does not perform bad here either. In the Multimodal case, the Random walk with suitable proposal spread does not get stuck in a maximum as the Langevin did. The circular density without clear mode is difficult for both MCMC versions, but the Langevin can be parametrised to handle is better. However, in general it depends a lot on the proper tuning for both.

1.4 Hamiltonian MH

Standard Gaussian

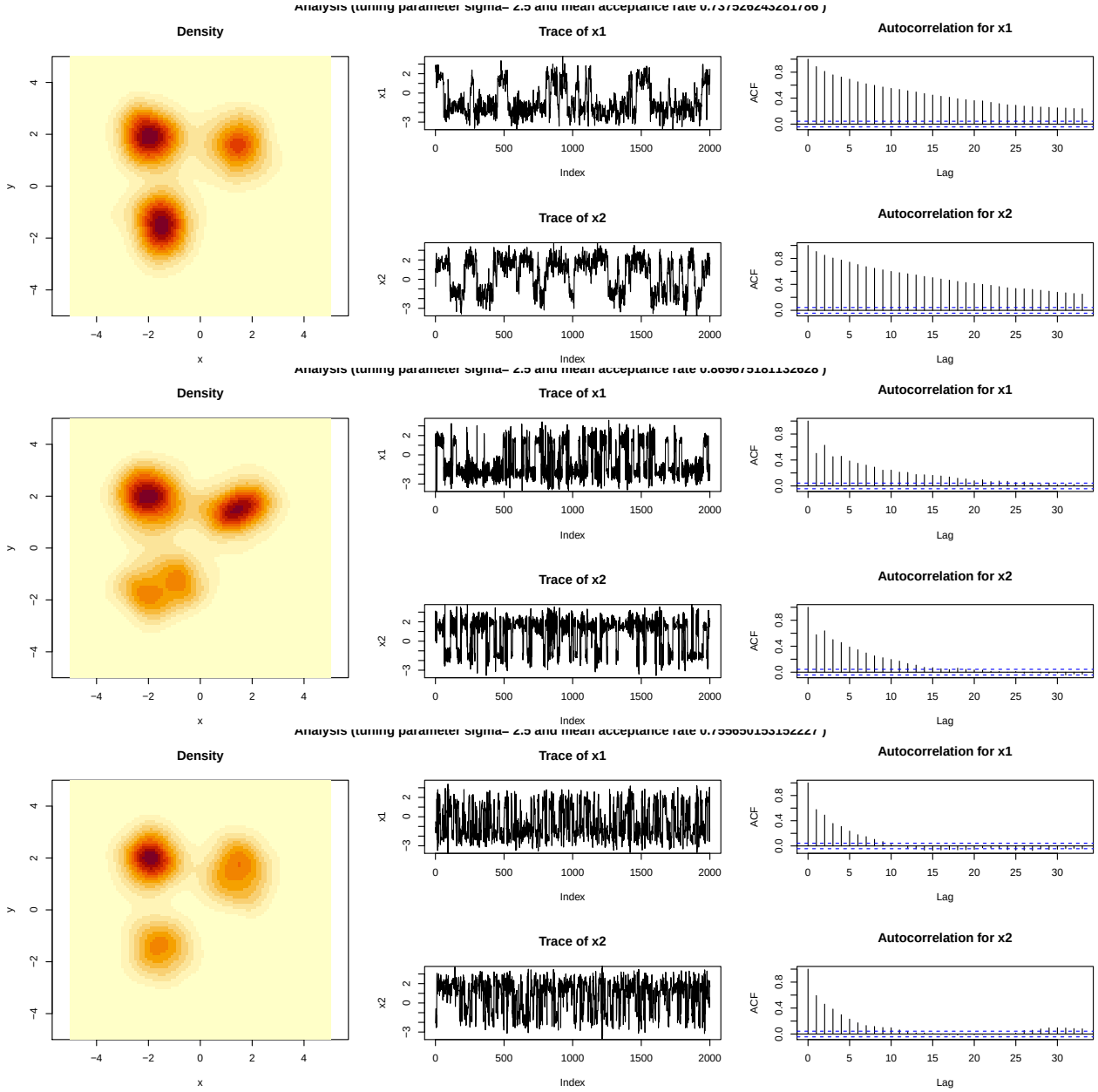
We first test the Hamiltonaian MH for the Gaussian with different tuning parameters δ and L for step sizes and step numbers in the leapfrog integration algorithm.



For the standard normal case, when $\delta = 0.3, L = 10$ gives reasonably good results on the target distribution. It can be shown also that Hamiltonian MH outperforms both random walk and langevin MH in terms of its independent samples and well mixing in the target space.

Multimodal

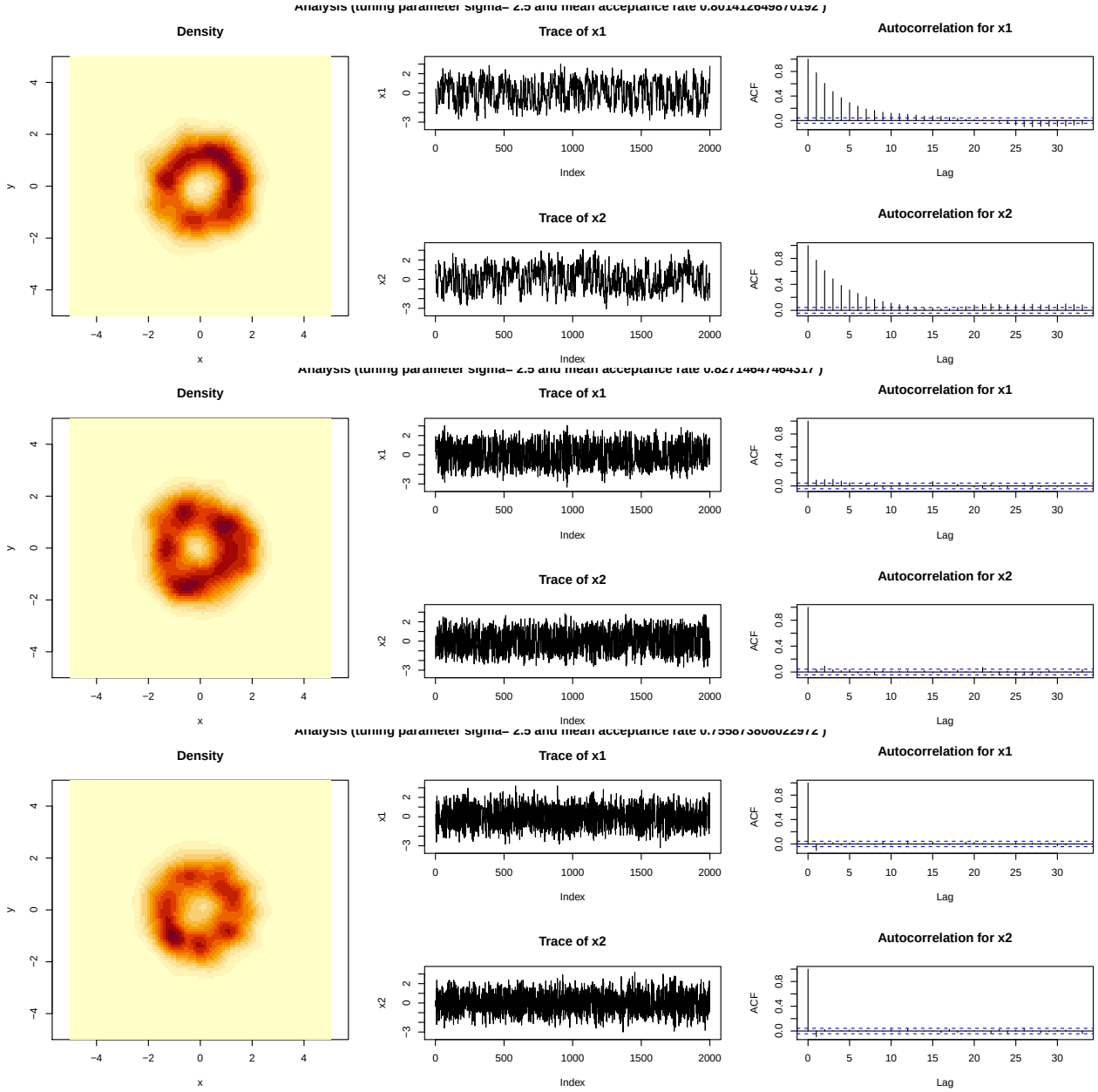
We continue with application of the Hamiltonian MH to the Multimodal with different tuning parameters δ and L for step sizes and step numbers in the leapfrog integration algorithm.



Hamiltonian MH in this case is sensitive to the tuning parameters, one can tell that it is easier to get stuck with one modal region when δ is too small. As one increases the step size, it gets more easier to explore the full target density space. From the autocorrelation plot, it seems that Hamiltonian MH behaves better than the other two in this case as well.

Volcano

Finally, we try the Hamiltonian MH for the volcano shaped density.



In this case, it truly tells where Hamiltonian idea was from, the volcano density is essentially the planet for the algorithm to explore. Hamiltonian MH explores pretty well already when $\delta = 0.3, L = 10$. Better than the other two since it is less independent for those samples generated.

Conclusion

We have considered 3 different MCMC versions and tested them for 3 examples with different features. The standard Gaussian distribution is the easist case, the multimodal challanges the chain to explore all modes and the volcano serves as bench mark whether the chain is flexible to discover the fire ring efficiently. All versions have in common that they require tuning parameters, that have to be chosen by the user to make the outcoming chain usable for sampling from the target distribution. Here, we kept some parameters (like the inital state or if applicable initial kinetic energy) fixed, trained the algorithm-specfic parameters by trial-and-error and present result for a collection of those parameter choices. Moreover, common for MCMC algorithms is to discard the samples in the so-called "burn-in" phase. However, in our trace plots we do not exclude those samples from the chain, since it anyways did not seem to be an major issue in our experiments.

While we have compared the results of the 3 MCMC version, we saw that all methods can handle the Gaussian case (assuming we use proper tuning parameters), whereas Random Walk and Langevin can get troubles with Multimodal and Volcano. It stands out that the Hamiltonian works very well for all cases and has by far the smallest autocorrelation. Moreover, the HMC seems less sensitive to the parameter choices, which makes it the most attractive MCMC version from the quality perspective. The only disadvantage, is the perceptible longer calculation time.

2 RStan

We consider the example of George et al. For a set of $N = 10$ pumps we observe the values - y_i: number of times that pump i failed - t_i: operation time of pump i whose data is used as input in the next chunk.

The numbers of failures per pump are modelled by a $Poisson(\lambda_i t_i)$ likelihood together with a $Gamma(\alpha, \beta)$ distributed prior for the λ_i with always $i = 1, \ldots, N$. Additionally, the hyper-prior for α is $Exp(1.0)$ and for β it is $Gamma(0.1, 1.0)$ respectively. Therewith, we define a `stan` model.

NB! We do not use a separate `.stan`-file but generate the model named "pump" directly in the environment. The syntax for the model generation is exactly the same as in files, but later the function call for the model fit deviates! The `pump.stan` file is only included for the seek of accordance with the exercise sheet.

```
// generates a stan model named pump in the current environment

data{
  int<lower=0> N;       // number of pumps
  int<lower=0> y[N];    // number of failures
  real<lower=0> t[N];   // operation times of pumps
}

parameters{
  real<lower=0> lambda[N];
  real<lower=0> alpha;
  real<lower=0> beta;
}

transformed parameters{
  real<lower=0> eta[N];
  for (i in 1:N)
    eta[i] = lambda[i] * t[i];
}

model{
  target += exponential_lpdf( alpha | 1.0 );      // hyper-prior log-density
  target += gamma_lpdf( beta | 0.1, 1.0 );        // hyper-prior log-density
  target += gamma_lpdf( lambda | alpha, beta );   // prior log-density
  target += poisson_lpmf( y | eta );              // likelihood log-density
}
```

We sample from the posterior distribution using `stan`.

NB! The function call is different since we have already a stan model called "pump" in the cache and can sample from that directly.

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)
```

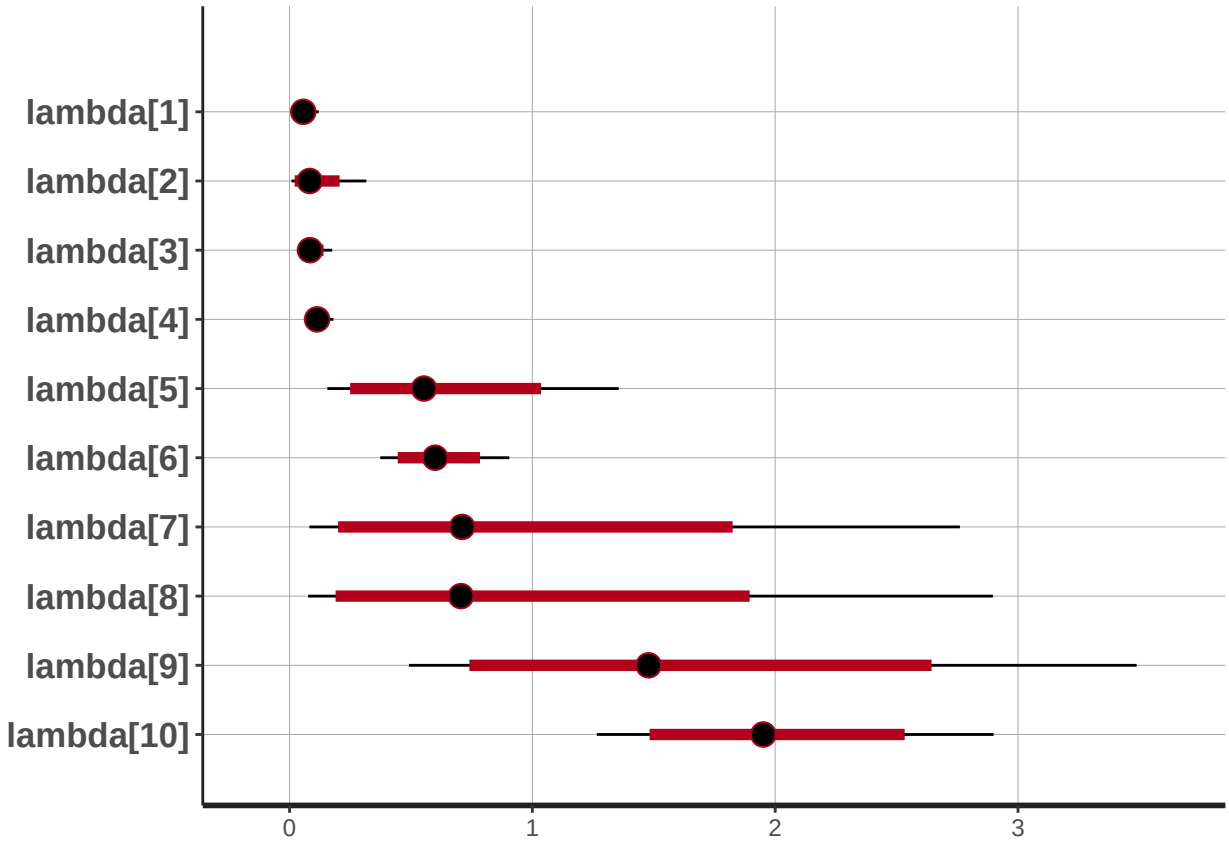
```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
##
## SAMPLING FOR MODEL '459e77192bc13dc921eabddda5d77169' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.2 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.04187 seconds (Warm-up)
## Chain 1:                0.039732 seconds (Sampling)
## Chain 1:                0.081602 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '459e77192bc13dc921eabddda5d77169' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 9e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.041077 seconds (Warm-up)
## Chain 2:                0.030755 seconds (Sampling)
## Chain 2:                0.071832 seconds (Total)
```

```
## Chain 2:
##
## SAMPLING FOR MODEL '459e77192bc13dc921eabddda5d77169' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.046043 seconds (Warm-up)
## Chain 3:                0.044496 seconds (Sampling)
## Chain 3:                0.090539 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '459e77192bc13dc921eabddda5d77169' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 8e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.043109 seconds (Warm-up)
## Chain 4:                0.039403 seconds (Sampling)
## Chain 4:                0.082512 seconds (Total)
## Chain 4:
```

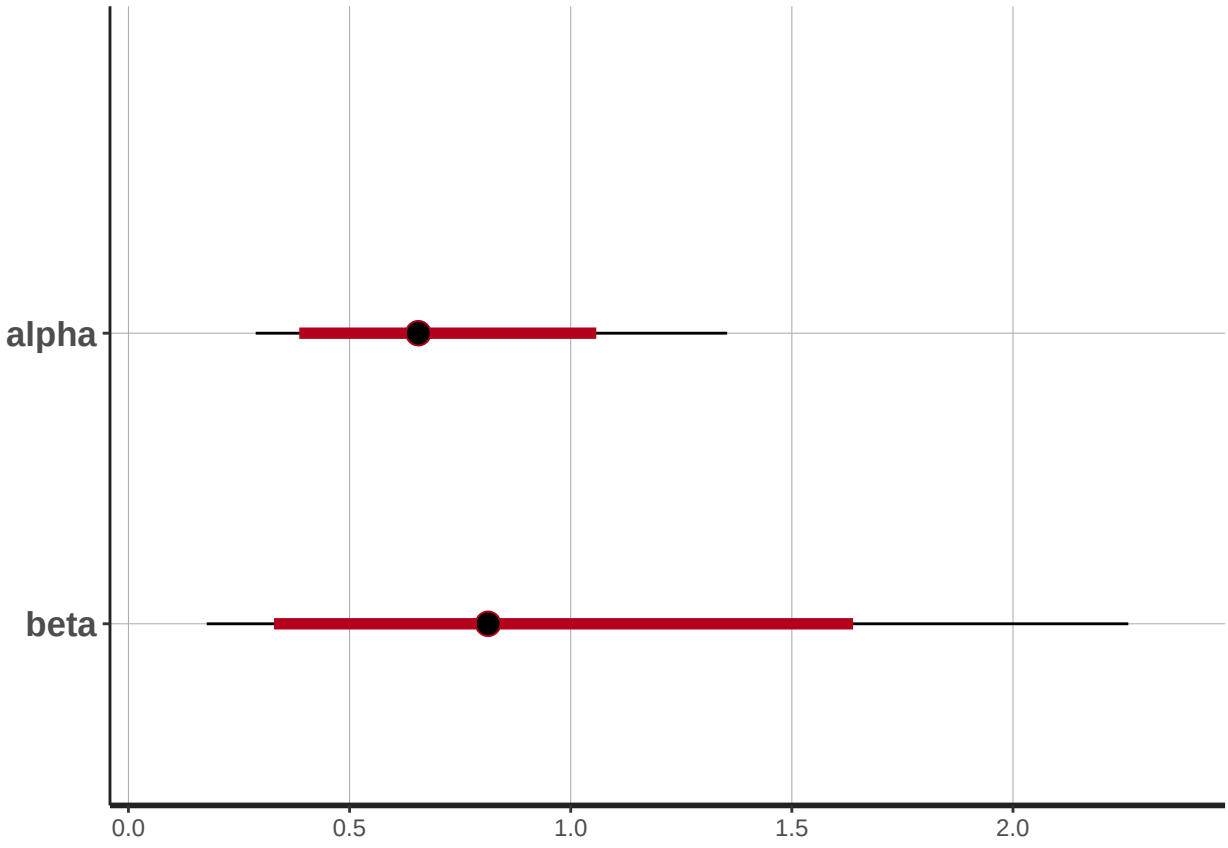
Finally, we investigate the results of the stan fit.

```
## Inference for Stan model: 459e77192bc13dc921eabddda5d77169.
```

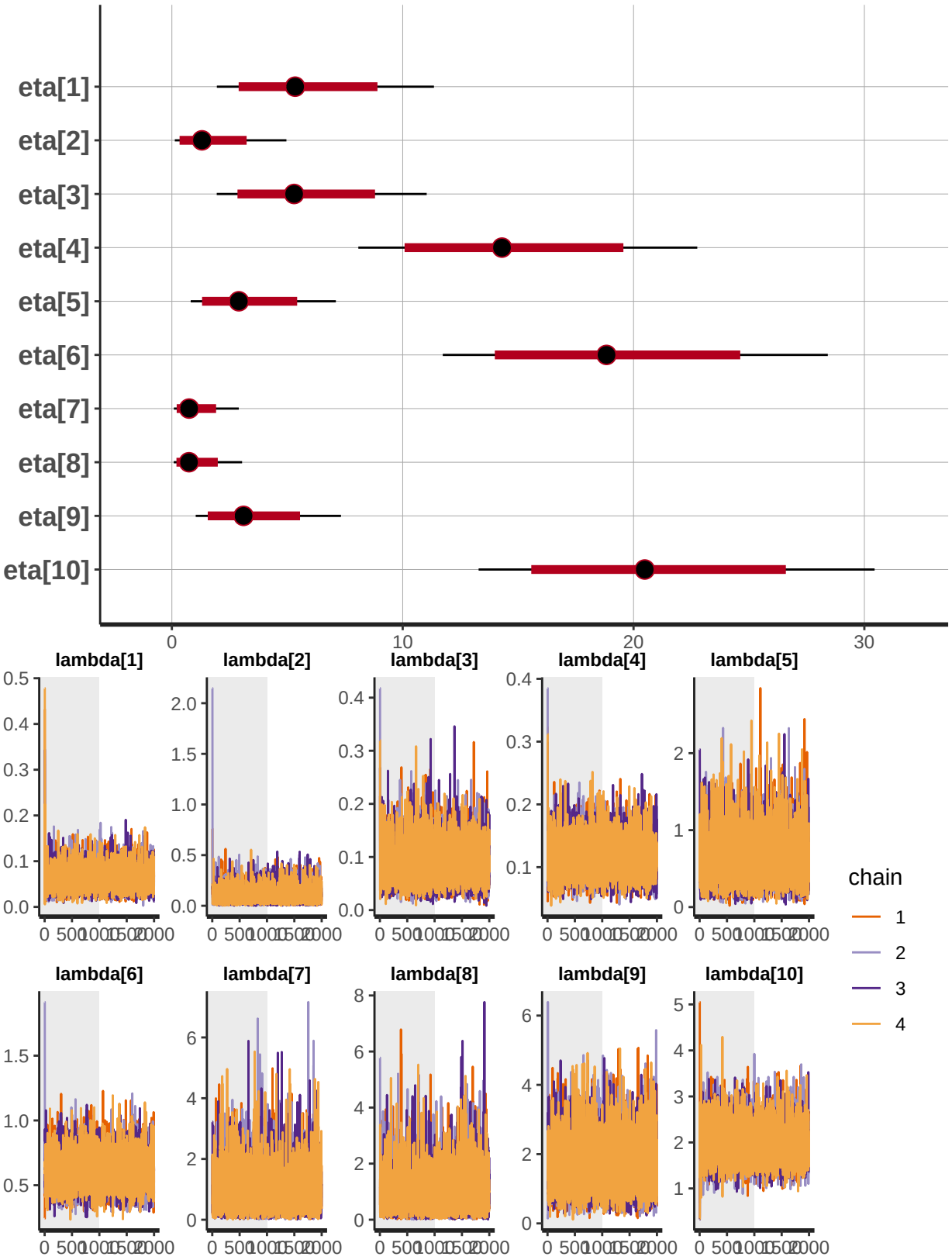
```
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##              mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## lambda[1]    0.06    0.00 0.03   0.02   0.04   0.06   0.08   0.12  4971    1
## lambda[2]    0.10    0.00 0.08   0.01   0.04   0.08   0.14   0.32  4460    1
## lambda[3]    0.09    0.00 0.04   0.03   0.06   0.08   0.11   0.18  4804    1
## lambda[4]    0.12    0.00 0.03   0.06   0.09   0.11   0.13   0.18  4574    1
## lambda[5]    0.61    0.00 0.32   0.15   0.37   0.55   0.77   1.36  5638    1
## lambda[6]    0.61    0.00 0.13   0.37   0.51   0.60   0.70   0.90  5525    1
## lambda[7]    0.90    0.01 0.73   0.08   0.38   0.71   1.21   2.77  4980    1
## lambda[8]    0.91    0.01 0.76   0.08   0.37   0.71   1.21   2.90  5427    1
## lambda[9]    1.61    0.01 0.77   0.49   1.05   1.48   2.04   3.49  4484    1
## lambda[10]   1.99    0.01 0.42   1.27   1.68   1.95   2.25   2.90  4243    1
## alpha        0.70    0.01 0.28   0.29   0.50   0.66   0.85   1.35  2484    1
## beta         0.92    0.01 0.55   0.18   0.52   0.81   1.19   2.26  2613    1
## eta[1]       5.68    0.03 2.40   1.95   3.94   5.34   7.08  11.35  4971    1
## eta[2]       1.60    0.02 1.26   0.13   0.68   1.31   2.20   4.96  4460    1
## eta[3]       5.61    0.03 2.39   1.94   3.84   5.30   7.02  11.04  4804    1
## eta[4]      14.60    0.06 3.74   8.08  11.96  14.30  16.93  22.76  4574    1
## eta[5]       3.18    0.02 1.66   0.81   1.96   2.90   4.05   7.10  5638    1
## eta[6]      19.15    0.06 4.24  11.74  16.16  18.84  21.83  28.42  5525    1
## eta[7]       0.95    0.01 0.77   0.09   0.40   0.75   1.28   2.91  4980    1
## eta[8]       0.95    0.01 0.80   0.08   0.39   0.74   1.27   3.04  5427    1
## eta[9]       3.38    0.02 1.62   1.03   2.20   3.11   4.28   7.33  4484    1
## eta[10]     20.85    0.07 4.42  13.29  17.68  20.49  23.66  30.44  4243    1
## lp__       -43.97    0.07 2.67 -50.16 -45.53 -43.57 -42.00 -39.90  1524    1
##
## Samples were drawn using NUTS(diag_e) at Wed Feb 17 16:53:28 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).

## ci_level: 0.8 (80% intervals)

## outer_level: 0.95 (95% intervals)
```

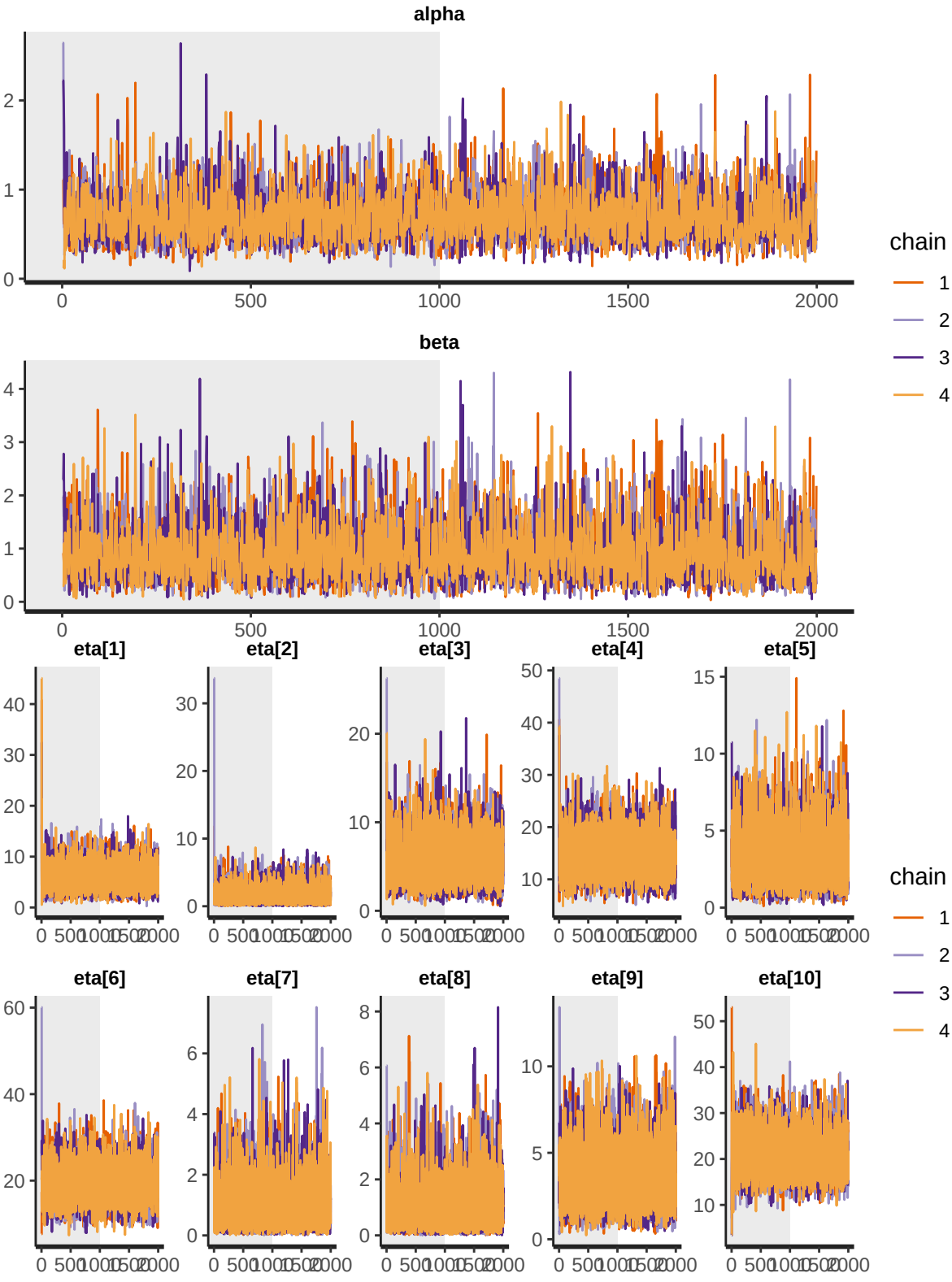



```
## ci_level: 0.8 (80% intervals)
## outer_level: 0.95 (95% intervals)
```




```
## ci_level: 0.8 (80% intervals)
## outer_level: 0.95 (95% intervals)
```


eta[1]
eta[2]
eta[3]
eta[4]
eta[5]
eta[6]
eta[7]
eta[8]
eta[9]
eta[10]
0
10
20
30
lambda[1]
lambda[2]
lambda[3]
lambda[4]
lambda[5]
lambda[6]
lambda[7]
lambda[8]
lambda[9]
lambda[10]
chain
1
2
3
4



From the print output of the fitted model, we analyse the effective sample size `n_eff`. With only 4.000 post-warmup draws in total, for the λ (and η) the effective samples size is around 5.000 and thereby surprisingly higher as the actual number of samples! In contrast for α and β, the effective sample size is only around 2500 and 3000, respectively, such that the quality of those samples is not as good as for the oother quantities.

In the trace plots for λ (and eta) we see that the state space is explored very effectively throughout the entire chains. Maybe for $i \in \{5, 7, 9\}$ the range of lower probability is not perfectly covered, but that is hard/impossible to say without further histograms. In general, we can assume that the chains reached a behaviour of the limit distribution already after a few dozen steps. For α and β we observe some higher autocorrelation and a less effective exploration of the target space. For a few dozen steps the sampels appear to remain close before the next jump comes, what is an undesired behavior.