As a part of our project we have written an R package, `SMC`, which is mostly written in C++ using the `Rcpp/RcppArmadillo` package. In this tutorial we provide an overview of the different algorithms implemented in the `SMC` package. For a description of the algorithms used within this tutorial, we refer the reader to the primary report.

The algorithms within `SMC` have been written in order to conduct analysis on data which can be modelled by a Stochastic Volatility Model (SVM):

$$X_t = \alpha X_{t-1} + \epsilon_t, \quad \epsilon \sim \mathcal{N}(0,1) \tag{1}$$

$$Y_t = \beta \exp(X_t/2)\zeta_t, \quad \zeta_t \sim \mathcal{N}(0,1) \tag{2}$$

We start by defining our model parameters $\theta = (\alpha, \beta, \sigma)$ and generating synthetic data from the associated SVM.

```r
theta <- c(0.91, 1, 1)
tmax <- 1000
set.seed(1234)
data <- SMC::stochastic_volatility(tmax, theta)
head(data)
```

```
##            x          y
## 1 -2.9113404  0.0647087
## 2 -1.5648786 -1.0726622
## 3 -0.9949148  0.3077198
## 4 -1.4801125 -0.2607910
## 5 -1.9113543 -0.3422655
## 6 -2.2165251 -0.3295994
```

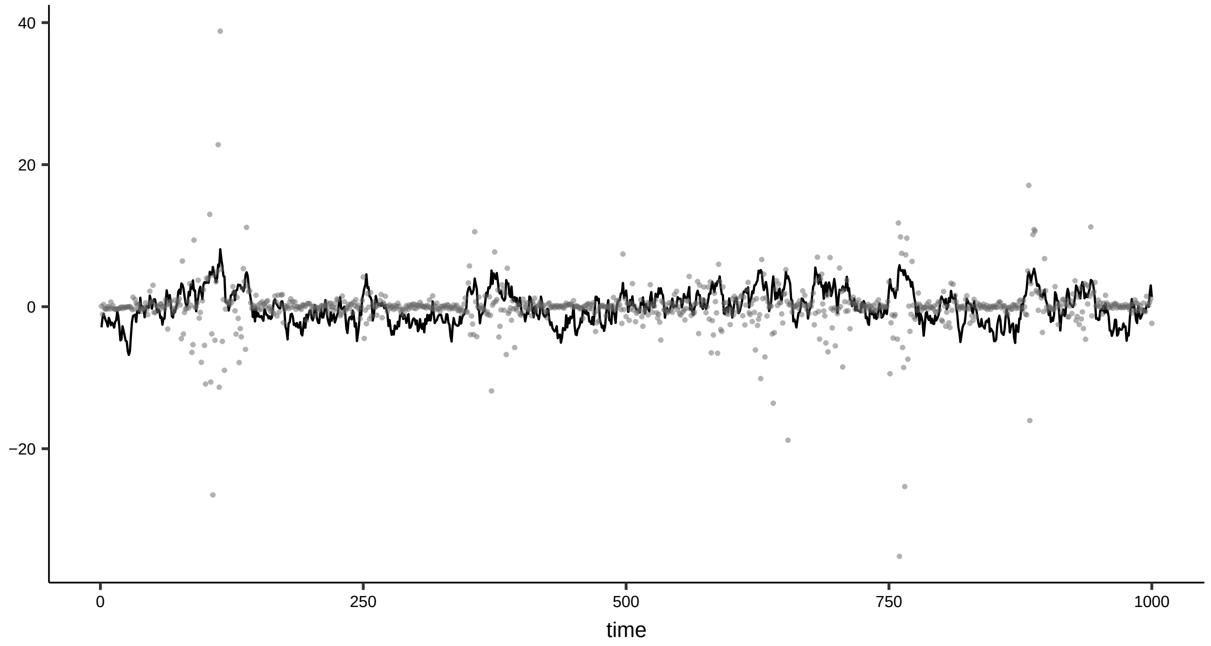

Figure 1: Synthetic data generated according to the Stochastic Volatility model with model parameters theta = (0.91, 1, 1).

Figure 1 shows a plot of the data generated using `SMC::stochastic_volatility()`. One can clearly see that the volatility of the observations, represented by the grey points, decreases as the states decrease. In practice we do not have access to the unknown states $x_{0:t}$, however we can estimate them through the use of the **Bootstrap Partcile Filter** (BSF) and the **Auxiliary Particle Filter**.

```r
obs <- data$y
N <- 400 # We use 400 particles.
BSF_fit <- SMC::BSF(obs, N, theta) # Assumes we know the true model paramters.
APF_fit <- SMC::APF(obs, N, theta)

# Compute the MSE:
cat("Bootstrap MSE: ", sum((BSF_fit$states[-1] - data$x)^2), "\n",
    "Auxiliary MSE: ", sum((APF_fit$states[-1] - data$x)^2), sep = "")
```

```
## Bootstrap MSE: 1553.523
## Auxiliary MSE: 1569.248
```

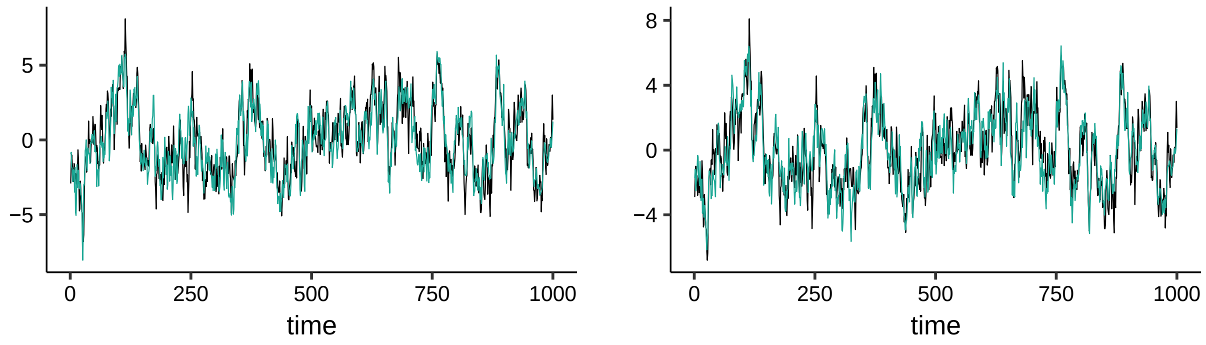

Figure 2: Results of running running the BSF and APF (Left/Right) on the synthetic data. The true states are given by the solid black line, and the filtered states are given by the dark green line.

Figure 2 shows the filtered states plotted against the true synthetic states. In this context there does not seem to be a huge difference in performance. We have of course taken for granted that we know the true model parameters used to generate the data. In practice we often must estimate the model parameters from the data. Working in the offline setting, we can do this using a Pseudo-Marginal Metropolis Hastings (PMMH) algorithm. In short, one can use the above particle filters to approximate the marinal likelihood of the observations

$$\hat{P}_N(y_{0:t}|\boldsymbol{\theta}) \approx p(y_{0:t}|\boldsymbol{\theta}). \tag{3}$$

We can incorporate this approximation into the Metropolis Hastings algorithm, and use it to accept/reject proposed values. For further details of this algorithm we once again refer the reader to the main report.

There are two versions of PMMH algorithm implemented within the `SMC` package. The first, `pmmh1`, estimates parameters α and σ **only**, using the prior/proposal

$$p(\alpha) = \text{Beta}\left((6, \frac{6}{0.8} - 6\right), \quad q(\alpha^*|\alpha) = \text{Beta}\left(100, \frac{100}{\alpha} - 100\right) \tag{4}$$

$$p(\sigma) = \text{Gamma}(5, \frac{0.5}{5}), \quad q(\sigma^*|\sigma) = \text{Beta}(200, \frac{\sigma}{200}). \tag{5}$$

The second, `pmmh2` estimates all three parameters $\theta = (\alpha, \beta, \sigma)$ and uses the prior

$$p(\alpha) = \text{Trunc-Normal}_{(-1,1)}(0.9, 0.5) \tag{6}$$

$$p(\beta) = p(\sigma) = \text{Gamma}(2, 2) \tag{7}$$

with the proposal

$$q(\theta^*|\theta) = \mathcal{N}(\theta, \sigma_{\text{prop}}\boldsymbol{I}). \tag{8}$$

Since all of the parameters are bounded (σ and β must be positive; $\|\alpha\| < 1$), we have applied reflection at each of their respective boundaries. This ensures proposals are only ever in feasible regions. In this tutorial we will show the use of `pmmh2`:

```r
pmmh2_fit <- SMC::pmmh2(10000, data$y, 800, c(0.5, 0.5, 0.5), 0.045)
```

This function outputs a list containing the number of accepted steps and a matrix whose columns represent a markov chain for each parameter.

```r
acceptance_rate <- pmmh2_fit$Accepted / nrow(pmmh2_fit$chain[-(1:2500),])
cat("Acceptance rate: ", acceptance_rate * 100, "%", sep = "")
```

```
## Acceptance rate: 20.98387%
```

The acceptance rate should ideally be around 23.4%. Let's examine the trace plots:

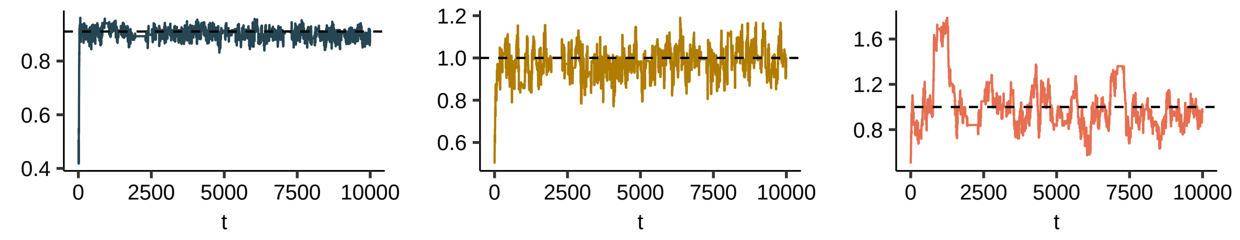

Figure 3: Trace plots for each model parameter (alpha/beta/sigma resp.) generated using pmmh2().

For α and σ, the traceplots in Figure 3 indicate that the chain has converged to its sationary regime. Another method of assessing convergence is through the ACF plots, which should also decay to 0. Figure 4 shows the ACF plots for each parameter after taking a burn in of 2500 iterations.

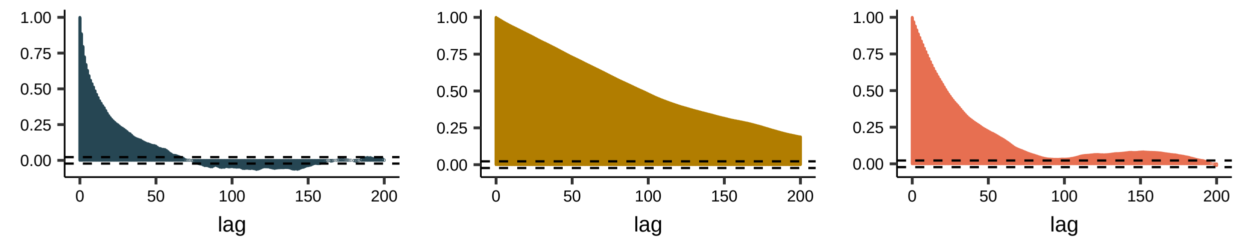

Figure 4: ACF plots for each model parameter (alpha/beta/sigma resp.) generated using pmmh2().

By lag 100, both α and σ's ACF appear to have decayed sufficiently. However, β's ACF plot suggestes that there is still a large level of correlation between values by lag 200. As a result, one might consider running the chain for longer. Finally, Figure 5 gives the histograms of the approximated posterior distributions, which are all roughly centred on the correct values.

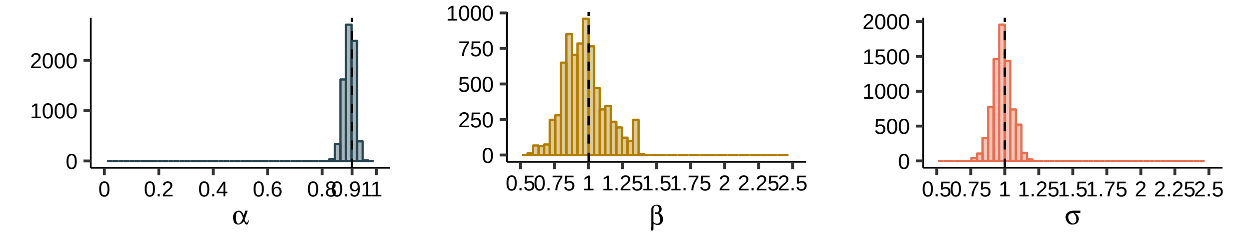

Figure 5: Sample histograms for each model parameter (alpha/beta/sigma resp.); generated using pmmh2().