Lab #X: Applying Markov chain Monte Carlo to sea-level data (Kelsey L. Ruckert, Tony E. Wong, Yawen Guan, and Murali Haran)

Learning objectives

After completing this exercise, you should be able to:

- perform a Markov chain Monte Carlo analysis that accounts for autocorrelation and heteroskedasticity
- apply this approach to "real" data (sea level) and use a "real" physical model (Rahmstorf 2007)
- explain why accounting for the error structure is important

Introduction

Suppose you have a physical model that characterizes the relationships within a system. However, this model is not perfect. There is an additional element of uncertainty. Adding a residual term is intended to account for this inaccuracy or uncertainty by accounting for model structural uncertainty, parametric uncertainty, physical process variability, and observational uncertainty not represented in the physical model. For example, sea-level data may have years when measurements are less or more certain simply because the instruments making these measurements may periodically break down, technological advancements increase the accuracy of these measurements over time, and more measurements become available. Further, deviations from the main trend of the data (residuals) could be dependent on previous observations. Testing the data for interdependent (autocorrelated) residuals and time-varying observation error (heteroskedasticity) is important because these can impact what assumptions you make about the data and the methods you choose to fit a physical model to the data.

In the previous chapters, you performed a Markov chain Monte Carlo (MCMC) calibration for a linear physical model, using data generated with non-correlated residuals. However, in many situations, observations in the Earth sciences (e.g., sea-level data) have a different and often more complicated error structure. Hence, a different approach for performing a MCMC analysis must be applied if you want to analyze data with a more complicated error structure. For instance, sea-level data is heteroskedastic, meaning that the error of the measurements varies over time. Additionally, sea-level residuals are autocorrelated, meaning that the current value is dependent upon previous values. In order to perform a MCMC analysis using sea-level data, it is necessary to check for evidence of these properties in the data. If evidence exists for this error structure, then it is important to account for autocorrelation and heteroskedasticty in the MCMC analysis that you perform.

How do we account for the error structure?

The error structure is defined within the residual term. It is important to distinguish the difference between residuals and errors. An error is the difference of the observed value from the "unobservable" true value. On the other hand, a residual is the deviation of an observation from its estimated value or modeled value,

$$\overbrace{R_t}^{\text{residual}} = \overbrace{y_t}^{\text{observation}} - \overbrace{f(\theta, t)}^{\text{physical model}} ,$$

where t is time and θ is the physical model parameters. The point of the residuals is to describe recognized amounts of unexplained variation in observations that are not always represented in a physical model. This variation can include a variety of uncertainties including model structure uncertainty, parametric uncertainty, physical process variability, and observational uncertainty. In this case, we approximate the residuals as the sum of the model error ω_t and observational/ measurement errors ϵ_t,

$$R_t = \omega_t + \epsilon_t.$$

The model error is a time series from a multivariate normal distribution. This multivariate normal distribution is described with the variance σ^2_{AR1} and the correlation structure given by the first-order autoregressive parameter ρ,

$$\omega_t = \rho \times \omega_{t-1} + \underbrace{\delta_t}_{\text{white noise}},$$

$$\omega_0 \sim N(0, \frac{\sigma^2_{AR1}}{1-\rho^2}), \qquad \delta_t \sim N(0, \sigma^2_{AR1}), \qquad \epsilon_t \sim N(0, \sigma^2_{\epsilon,t}).$$

This model is also recognized as a lag-1 autoregressive process or *AR1*. An autoregressive model is a representation of random time-varying processes in nature. In an autoregressive model, the output depends on its previous values and white noise. White noise is a sequence of continuously uncorrelated random variables with a zero mean and a constant variance σ^2_{AR1}. For AR1, only the previous term in the process, the autocorrelation coefficient, and the white noise are used to create the output. If the autocorrelation runs deeper than just an AR1 process, then it may be necessary to fit more sophisticated models (e.g., lag-2 autoregressive process or other types of autoregressive integrated moving average models).

When you account for an AR1 process, you not only modify the equation of how to approximate the residuals, but also modify the likelihood function (check out **Rahm_obs_likelihood_AR.R** and compare it to **iid_obs_likelihood_AR.R**). For instance, the modified likelihood function accounts for the observational errors ϵ_t and the autocorrelation coefficient ρ so that the likelihood function is then,

$$L(y \mid \theta) = \Big(\frac{1}{\sqrt{2\pi}}\Big)^N \times \mid \Sigma \mid^{\frac{-1}{2}} \exp\Big\{\frac{-1}{2}\big[y - f(\theta,t)\big]^\tau \Sigma^{-1}\big[y - f(\theta,t)\big]\Big\},$$

where $\sum$ is the sum of the variance of the AR1 process and the observation errors. The variance of the AR1 process and the variance of the observation errors are both diagonal matricies of size N x N where the diagonal has the variance/ error element (see the supplementary material of Ruckert et al. 2017 for details). Additionally, ρ is added as an uncertain parameter in the MCMC analysis.

In the previous chapter, we make a key assumption in the MCMC framework that the residuals are normally distributed with a mean of 0. However, in the example of calibrating changes in sea level, we fail this key assumption. Instead, the AR1 process is a Gaussian process, which is *white* (statistically uncorrelated), thereby the residuals satisfy the MCMC assumptions. Hence, in this example, we fit the AR1 error model and use the AR1 residuals. This effect of going from non-normally distributed and potentially non-mean 0 residuals to those that are normally distributed with mean 0 is called "whitening" the residuals, referring to the fact that often the signal strength of *white noise* is assumed to be normally distributed. There are actually many situations where we do not know the most appropriate error model to use, so there are many different ways we can "whiten" the residuals ($R_{t+1} - \rho R_t$) in order to satisfy the assumptions underlying the MCMC framework. Once we whiten the residuals we can then estimate the log likelihood as the sum of the normal density of the non-correlated (whitened) residuals with a mean of 0 and a standard deviation of the square root of the variance σ^2 plus the observation errors squared.

```
# Estimate the log likelihood of the AR1 process
logl.ar1 <- function(r, sigma1, rho1, eps1 = y.meas.err){
    n <- length(r) # r is the residuals

    logl <- 0
    if(n>1) {
      # Whiten the residuals (w)
        w <- r[2:n] - rho1*r[1:(n-1)]
        logl <- logl + sum(dnorm(w, sd=sqrt((sigma1)^2 + (eps1[c(-1)])^2), log=TRUE))
    }
    return(logl)
}
```

Here, we treat the estimation of the log prior distribution function and the log posterior distribution function the same as in the previous chapter. In this tutorial, you will use uniform distributions as priors for the

physical and statistical parameters. Hence, the log prior function evaluates whether the parameter values are within the boundary and if so returns zero to represent a uniform distribution. As before, the log posterior function estimates the posterior distribution based on the log likelihood and log prior (posterior $\propto$ likelihood $\times$ prior).

Tutorial

If you have not already done so, download the .zip file containing the scripts associated with this book from www.scrimhub.org/raes. Put the file labX_sample.R in an empty directory. Open the R script labX_sample.R and examine its contents. This chapter builds off of the previous chapters about MCMC, so it is highly recommended that you complete those chapters before you proceed. In this exercise, you will calibrate the Rahmstorf (2007) model to sea-level data while accounting for the autocorrelated residuals and the heteroskedastic observation errors.

Before you proceed, first clear away any existing variables or figures and install packages. Install the `adaptMCMC` package. The `compiler` package should already exist in the system library and therefore does not need to be installed.

In the code block below, the command `enableJIT()` is used to increase the efficiency of R by enabling or disabling "just-in-time" compilation. Enabling "just-in-time" compilation causes the closures to be compiled before they are first used (`enableJIT(1)`), closures to be compiled before they are duplicated (`enableJIT(2)`), and all `for()`, `while()`, and `repeat()` loops to be compiled before they are executed (`enableJIT(3)`). "Just-in-time" compilation is turned off if the argument is set to 0. Test out the speed gain by tracking the time it takes to run a set of code with `proc.time()`.

```
# Clear away any existing variables or figures.
rm(list = ls())
graphics.off()

# Install and read in packages.
# install.packages("adaptMCMC")
library(adaptMCMC)
library(compiler)
enableJIT(3)

# Set the seed.
set.seed(111)

# Make directory for this lab
dir.create("lab_X")
dir.path <- paste(getwd(), "/lab_X/", sep="")
scrim.git.path <- "https://raw.githubusercontent.com/scrim-network/"
```

Next, download and read in the sea-level observations, temperature observations, and the Rahmstorf (2007) global sea-level model. Specific details on these data sources can be found within the code block below and in previous chapters.

```
# Global mean sea level from Church and White (2011)
# year, time in years from 1880 to 2013 at the half year mark
# slr, global mean sea level in mm
# err.obs, global mean sea level measurement error in mm
url <- paste(scrim.git.path, "BRICK/master/data/GMSL_ChurchWhite2011_yr_2015.txt", sep="")
download.file(url, paste(dir.path, "GMSL_ChurchWhite2011_yr_2015.txt", sep=""))
church.data <- read.table("lab_X/GMSL_ChurchWhite2011_yr_2015.txt")
year <- church.data[ ,1] - 0.5
```

```
slr <- church.data[ ,2]
err.obs <- church.data[ ,3]

# Calculate sea level -/+ observation errors.
err_pos <- slr + err.obs
err_neg <- slr - err.obs

# Read in the information from the Smith (2008) temperature file.
# project.years, time in years from 1880 to 2100.
# hist.temp, historical global mean temperatures from 1880 to 2013 in C
# rcp85, merged historical + rcp 8.5 temperatures from 1880 to 2100 in C
url <- paste(scrim.git.path,
             "Ruckertetal_SLR2016/master/RFILES/Data/NOAA_IPCC_RCPtempsscenarios.csv",
             sep="")
download.file(url, paste(dir.path, "NOAA_IPCC_RCPtempsscenarios.csv", sep=""))
temp.data <- read.csv("lab_X/NOAA_IPCC_RCPtempsscenarios.csv")
project.years <- temp.data[1:match(2100, temp.data[,1]), 1]
hist.temp <- temp.data[1:match(2013, temp.data[,1]), 2]
rcp85 <- temp.data[1:match(2100, temp.data[,1]), 4]

# The sea level and temperature data have a yearly timestep.
tstep = 1

# Load the sea-level model (Rahmstorf, 2007)
url <- paste(scrim.git.path, "BRICK/master/R/gmsl_r07.R", sep="")
download.file(url, paste(dir.path, "gmsl_r07.R", sep=""))
source("lab_X/gmsl_r07.R")
```

Characterizing the error structure

In the introduction, we discuss how scientists often must make assumptions about the data and the error structure of the data. In practice, how do we know what assumptions to make? To answer this, you can examine the observations and inform assumptions based on what information and details are known about the observations. For example, the sea-level observations `slr` start from the year 1880 and you can assume that since 1880 the measurements of sea level have become more accurate and more available. Since the data from Church and White (2011) includes error estimates, you can test that assumption of whether the data has time-varying observation error (heteroskedasticity) by plotting the errors over time. The resulting plot shows that the errors indeed vary over time.

```
par(mfrow = c(1,1))
plot(year, err.obs, type = "l", ylab = "Observation errors [mm]", xlab = "Time")
points(year, err.obs, pch = 20)
```

Additionally, you can assume that changes in sea level this year will impact what sea level is like next year. Hence, this implies that the residuals are autocorrelated. Before testing for autocorrelation, you must calculate the residuals. Using an optimization procedure (`optim` or `DEoptim`), you can estimate the parameter values. The model parameters include the sensitivity of sea level to changes in temperature, α; the temperature when then sea-level anomaly is zero, T_{eq}; and the initial sea-level anomaly value (the value in 1880), SL_0. These values can be plugged into the model to generate a "best fit" simulation and be later used as initial parameter estimates. Using the resulting "best-fit" simulation, you estimate the residuals by subtracting the model simulation from the observations. Taking the standard deviation of the residuals can serve as the initial estimate of σ later on.

```r
# Generate fit to sea-level observations using General-purpose Optimization.
# by calculating the root mean squared error.
fn <- function(pars, tstep, Temp, obs){
  a   <- pars[1]
  Teq <- pars[2]
  SL0 <- pars[3]

  np      <- length(Temp)
  gmsl    <- rep(NA, np)
  gmsl[1] <- SL0

  # Use Forward Euler to estimate sea level over time.
  for (i in 2:np) {
    gmsl[i] <- gmsl[i-1]  +  tstep * a * ( Temp[i-1]-Teq )
  }
  resid <- obs - gmsl
  # Return the root mean square error
  rmse <- sqrt(mean(resid^2))
  return(rmse)
}

# Plug in random values for the parameters.
parameter_guess <- c(3.4, -0.5, slr[1])

# Optimize the model to find initial starting values for parameters.
result <- optim(parameter_guess, fn, gr=NULL, tstep, Temp = hist.temp, obs = slr)
start_alpha <- result$par[1]
start_Teq <- result$par[2]
start_SL0 <- result$par[3]

# Make a plot of observations and fit.
slr.est <- gmsl_r07(a = start_alpha, Teq = start_Teq, SL0 = start_SL0, tstep,
                    Tg = hist.temp)
plot(year, slr, pch = 20, ylab = "Sea-level Anomaly [mm]", xlab = "Year")
lines(year, slr.est, col = "blue", lwd = 2)

# Calculate residuals and initial sigma value.
res <- slr - slr.est
start_sigma <- sd(res)
```

You can use the resulting residuals to test for autocorrelation with the `acf()` function. "Autocorrelation" refers to the correlation between two points within a time series, where the time difference between them is called the "lag". In the autocorrelation plot, the correlation is shown at each time lag. If the correlation exceeds the dashed blue lines (a 95% confidence interval), then the correlation of the data at that lag is considered to be "statistically significant". Values below the dashed lines are considered insignificant correlations. The correlation at lag 0 should always equal 1 because each observation should be 100% correlated to itself. The resulting plot shows that the residuals are autocorrelated with the highest correlation at the lag-1 autoregressive process or *AR1*. Accounting for the AR1 process requires adding an uncertain statistical parameter, ρ. ρ is the first-order or lag-1 autocorrelation coefficient. For later use, you can save `rho[2]` as the initial ρ estimate to set up the MCMC calibration.

```r
# Apply the auto-correlation function to determine correlation coefficients.
rho <- rep(NA,3)
ac <- acf(res, lag.max = 5, plot = TRUE, main = "", xlab = "Time lag",
```

```
          ylab = "Autocorrelation function")
rho[1] <- ac$acf[1]
rho[2] <- ac$acf[2]
rho[3] <- ac$acf[3]
rho[4] <- ac$acf[4]
rho[5] <- ac$acf[5]
start_rho <- rho[2]
```

Setting up the prior information

In the previous blocks, you read in the data, learned about what to account for in the error structure, and estimated some good initial parameter estimates. The next goal is to learn about the distribution of the parameters (α, T_{eq}, SL_0, σ, and ρ) given the observed data. First, you will specify a prior distribution for each model parameter and statistical parameter. Like the previous MCMC chapter, you will use uniform prior distributions. For the model parameters, you will use the prior distributions based on Ruckert et al. (2017; see supplementary material). Hence, the prior for α will be set as 0 to 20 $[mm \cdot yr^{-1} \cdot {}^{\circ}C^{-1}]$, -3 to 2 $[^{\circ}C]$ for T_{eq}, and as the sea-level observation [mm] in 1880 plus and minus the observation error in 1880 (1880 is the year of the first sea-level value) for SL_0. Also, set σ as 0 to 10 [mm] because in this example the σ cannot be negative and that it is small based on observing the residuals. ρ is set to have a prior range of -0.99 to +0.99. This is because ρ is a percent represented as a decimal and therefore cannot have a positive or negative correlation greater than 100%. As an added challenge (recommended in the Appendix), test out different nonuniform prior distributions for the model parameters.

The difference with the approach shown in the previous MCMC chapter is that the data have residuals that are correlated and show heteroskedasticity. To account for the differences in the error structure, we update the likelihood function to add ρ as an uncertain parameter and to set the measurement errors equal to the known observational errors, `err.obs` (as stated in the introduction).

```
# Set up MCMC calibration.
# Set up priors.
bound.lower <- c( 0, -3, err_neg[1], 0, -0.99)
bound.upper <- c(20,  2, err_pos[1], 10,  0.99)

# Name the parameters and specify the number of physical model parameters.
# Sigma and rho are statistical parameters and will not be counted in the number.
parnames <- c("alpha", "Teq", "SL0", "sigma", "rho")
model.p <- 3

# Set the measurement errors.
y.meas.err <- err.obs

# Load the likelihood model accounting for the error structure.
source("Rahm_obs_likelihood_AR.R")

# Setup initial parameters.
p <- c(start_alpha, start_Teq, start_SL0, start_sigma, start_rho)

# Set p0 to a vector of values to be used in optimizing the log-posterior function.
# We optimize the negative log-posterior function because "optim" only knows how to
# minimize a function, while we would like the log-posterior to be maximized.
p0 <- c(3.4, -0.5, slr[1], 6, 0.5)
p0 <- optim(p0, function(p) -log.post(p))$par
print(round(p0, 4))
```

Running MCMC

For this chapter, you will use a more sophisticated MCMC algorithm with an adaptive algorithm for the MCMC sampler (Vihola, 2011). This adaptive MCMC uses Metropolis-Hastings updates with joint multivariate normal proposal, which tunes the covariance matrix for these proposals to "learn" what the step sizes and directions ought to be, in order to achieve a desired acceptance. The purpose of using an adaptive MCMC over the "vanilla" MCMC version shown in the previous chapters on MCMC because 1) you will get the opportunity to learn both versions and 2) speed is a priority for this lab exercise and using adaptation can lessen the number of iterations needed to show evidence of convergence.

In `R`, the package `adaptMCMC` includes the `MCMC` function, which implements adaptive MCMC. For this function, you must specify several arguments. First, you will specify your desired acceptance rate and set adapt to `TRUE` so adaptive sampling is used. You learned from the first chapter on MCMC that "the optimal acceptance rate varies depending on the number of dimensions in the parameter space." You also learned that for a single parameter the optimal acceptance rate is about 44%, but as the number of parameters increases, the acceptance rate stabilizes around 23.4%. For the sea-level rise model, there are in total five parameters; three model parameters (α, T_{eq}, SL_0) and two statistical parameters (σ, and ρ). Hence, the desired acceptance rate is likely around 23.4%. Second, set the rate of adaptation `gamma.mcmc` to equal 0.5; this is because a lower γ leads to a faster adaptation. Next, you will set the step size based on step sizes used in the calibration of the Rahmstorf model in Ruckert et al. 2017. Even though the step sizes adapt, better initial estimates of the proposal covariance matrix yield faster convergence of the algorithm. In practice, you can play around with the "vanilla" MCMC function to observe how changing the step size for each parameter impacts the acceptance rate and use the results to inform the initial step sizes for the adaptive version. Lastly, you will set the MCMC analysis to run for 4×10^5 iterations and set the analysis to start adapting the step size after 1% of the iteration has run. Note that you do not want to start adapting too early because estimates of the parameter correlations will be weak if not enough proposals have been accepted yet. This analysis may take several minutes to complete.

```
# Set optimal acceptance rate as # parameters->infinity
# (Gelman et al, 1996; Roberts et al, 1997)
# Set number of iterations and rate of adaptation
# (gamma.mcmc, between 0.5 and 1, lower is faster adaptation)
accept.mcmc = 0.234
gamma.mcmc = 0.5
NI <- 4e5
step <- c(0.2, 0.02, 1, 0.1, 0.01)

# Run MCMC calibration. Note: Runs for several minutes.
mcmc.out <- MCMC(log.post, NI, p0, scale = step, adapt = TRUE, acc.rate = accept.mcmc,
                 gamma = gamma.mcmc, list = TRUE, n.start = round(0.01*NI))
mcmc.chains <- mcmc.out$samples
```

Determine the burn-in period

In the previous chapter, you applied a 1% burn-in period, but how do you decide the appropriate length of the burn-in period? Besides visual inspection, you could use the Heidelberger and Welch diagnostic or the Gelman and Rubin diagnostic. The Gelman and Rubin diagnostic technique is useful because it can also be used to check for evidence of convergence as was described in the previous chapter on MCMC. First, you will run the MCMC analysis two more times and check for convergence at different points along the chains. Once the chains show evidence for convergence, then you can determine the burn-in. For this, look at how many iterations are required for the Gelman and Rubin potential scale reduction factor to get and stay below 1.1 for all parameters. It is only after the chains are converged that you should use the parameter values as posterior draws, for analysis. Therefore, the iterations prior to the point at which the scale reduction factor falls below 1.1 is what is thrown away as the burn-in period.

```r
## Gelman and Rubin's convergence diagnostic:
set.seed(1708)
p0 <- c(1.9, -0.9, -145, 4, 0.7) # Arbitrary choice.
mcmc.out2 <- MCMC(log.post, NI, p0, scale = step, adapt = TRUE, acc.rate = accept.mcmc,
                  gamma = gamma.mcmc, list = TRUE, n.start = round(0.01*NI))
mcmc.chains2 <- mcmc.out2$samples

set.seed(1234)
p0 <- c(2.9, 0, -160, 5, 0.8) # Arbitrary choice.
mcmc.out3 <- MCMC(log.post, NI, p0, scale = step, adapt = TRUE, acc.rate = accept.mcmc,
                  gamma = gamma.mcmc, list = TRUE, n.start = round(0.01*NI))
mcmc.chains3 <- mcmc.out3$samples

# Turn the chains into an mcmc object.
# This is a requirement for the gelman.diag and gelman.plot commmand.
mcmc1 <- as.mcmc(mcmc.chains)
mcmc2 <- as.mcmc(mcmc.chains2)
mcmc3 <- as.mcmc(mcmc.chains3)

# Test for convergence.
set.seed(111) # revert back to original seed
mcmc_chain_list <- mcmc.list(list(mcmc1, mcmc2, mcmc3))
gelman.diag(mcmc_chain_list)
gelman.plot(mcmc_chain_list)

# Create a vector to test the statistic at several spots throughout the chain.
# Test from 5000 to 4e5 in increments of 5000.
niter.test <- seq(from = 5000, to = NI, by = 5000)
gr.test <- rep(NA, length(niter.test))

# Once the statistic is close to 1 (adopting the standard of < 1.1), the between-chain
# variability is indistinguishable from the within-chain variability, and hence converged.
for (i in 1:length(niter.test)){
mcmc1 = as.mcmc(mcmc.chains[1:niter.test[i],])
mcmc2 = as.mcmc(mcmc.chains2[1:niter.test[i],])
mcmc3 = as.mcmc(mcmc.chains3[1:niter.test[i],])
mcmc_chain_list = mcmc.list(list(mcmc1, mcmc2, mcmc3))
gr.test[i] = gelman.diag(mcmc_chain_list)[2]
}

# Plot Gelman and Rubin statistics as a function of iterations. The iterations prior
# to the point in which convergence happens is what we set as the burn-in.
plot(niter.test, gr.test, pch=20)
abline(h = 1.1, col = "red")
gr.test < 1.1; first.stat = max(which((gr.test < 1.1) == FALSE))+1; print(first.stat)

# Set the burn-in to the 1st statistic that remains under 1.1 and remove it.
burnin <- seq(1, niter.test[first.stat], 1)
slr.burnin.chain <- mcmc.chains[-burnin, ]
```

Hindcasting and projecting sea-level rise

The parameters from the chains (burn-in removed) can be plugged into the model to generate multiple simulations of the data, called an "ensemble". The residuals can be added to the simulations of the data to generate hindcasts and projections for further analysis. In the exercise below, you will generate hindcasts and projections with this process.

If you insert the following lines at the bottom of your R script, then you will make sea-level rise projections as a matrix (`SLR.projections`), using the sea-level model and multiple `for` loops. In `SLR.projections`, each row represents a state of the world while the columns represent time. Because you are using all 4×10^5 iterations, this code block will take several minutes to fully generate the projections. For an additional exercise, rather than using multiple `for` loops try to improve on the code by using the `apply` or `sapply` function in R, or vectorize the internal `for` loop in the `SLR.residuals` calculation. Does using any of these other options increase the speed of the code?

```
# Extract parameter vectors from the chain to enhance code readability.
alpha.chain <- slr.burnin.chain[ ,1]
Teq.chain <-   slr.burnin.chain[ ,2]
SL_0.chain <-  slr.burnin.chain[ ,3]
sigma.chain <- slr.burnin.chain[ ,4]
rho.chain <-   slr.burnin.chain[ ,5]

# Set up empty matrices for sea-level output.
# Loop over the sea level model to generate a distribution of sea level simulations.
NI_length <- length(slr.burnin.chain[ ,1])
SLR.model.sim <- mat.or.vec(NI_length, length(project.years))
for(n in 1:NI_length) {
  SLR.model.sim[n, ] = gmsl_r07(a = alpha.chain[n], Teq = Teq.chain[n],
                                SL0 = SL_0.chain[n], tstep, Tg = rcp85)
}

# Estimate the residuals with the AR(1) coefficient and sigma.
SLR.residuals <- mat.or.vec(NI_length, length(project.years)) #(nr,nc)
for(n in 1:NI_length) {
  for(i in 2:length(project.years)) {
    SLR.residuals[n,i] <- rho.chain[n]*SLR.residuals[n,i-1] +
      rnorm(1, mean = 0, sd = sigma.chain[n])
  }
}

# Estimate the hindcasts and projections: add the residuals onto the model simulations.
SLR.projections <- mat.or.vec(NI_length, length(project.years)) #(nr,nc)
for(i in 1:NI_length) {
  SLR.projections[i,] <- SLR.model.sim[i,] + SLR.residuals[i,]
}
```

Exercise

Part 1. Using the diagnostics specified in the previous chapter, check the analysis for evidence of convergence. In your lab report, provide evidence to support or reject whether the chains are converged. If the chains are not converged, then explain how you would go about getting the chains to converge. The goal of this chapter is to understand how to apply a Markov chain Monte Carlo analysis to "real" data and use a "real" model, so getting the chains to converge—while extremely important in a physical application—is not a top priority in this chapter due to time constraints.

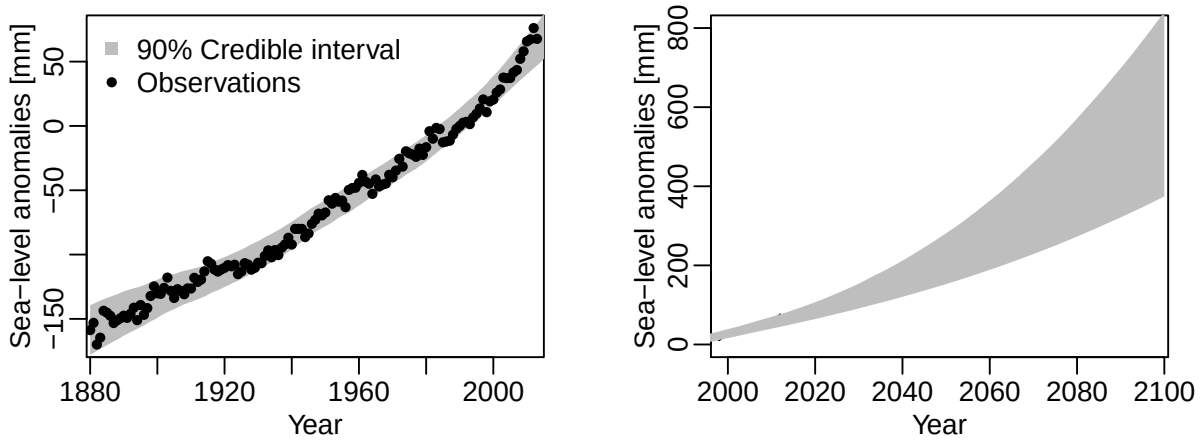

Figure 1: The sea level anomaly data from Church and White (2011; left) with its associated 90% credible interval, and the 90% credible interval projections of sea level anomalies from 2000 to 2100 in mm (right).

Part 2. Modify your script from Part 1 to:

- calculate and plot the 90% credible interval of the sea-level projections (to the year 2100) using the RCP8.5 temperature scenario (hint: consider using `quantile()` and `polygon()`),
- and to make density plots of the parameters α, T_{eq}, SL_0, and ρ.

It is highly recommended that you do not plot all 4×10^5 projections as individual lines at once, but rather plot credible intervals. Ultimately, you should end up with a plot similar to Figure 1. If not, check your script against the codes listed through the tutorial to make sure your script is working properly.

Part 3. Save a copy of your script from Part 2 using a different file name. In this script, modify the MCMC analysis to assume the errors are homoskedastic (constant through time) by changing `y.meas.err <- err.obs` to a vector where the values equal 0, `y.meas.err <- rep(0, length(slr))`. Setting the measurement errors to 0 combines the model error and observational error into one term.

Your scripts should make the following plots:

1. a two-panel plot displaying the autocorrelation function and the time dependent measurement error of the sea-level data.
2. a plot comparing the 90% credible interval of projections made from considering heteroskedasticity (part 2) versus assuming homoskedasticity (part 3).
3. a four-panel plot comparing the densities of parameters α, T_0, H_0, and ρ made from considering heteroskedasticity (part 2) versus assuming homoskedasticity (part 3).

Questions

1. Compare the sea-level projections from *part 2* and *part 3*. How do the results differ? How much larger or smaller is the 90% credible interval in *part 1* from *part 2*? How might the assumptions made about the data impact "real-life" decisions regarding flood protection?
2. Compare the parameter distributions from *part 2* and *part 3*. How do the results differ? Do both methods produce the same mode and the same width in distribution?
3. Refer to the paper Ruckert et al. (2017) (link) and the results from *part 2* and *part 3*. Does it matter how you account for the error structure of data? Explain your reasoning.

Appendix

Comparing un-whitened raw residuals to the whitened residuals

In the exercise above, you whiten the residuals from the sea-level data in order to satisfy the MCMC assumptions. This process changes the residuals to be normally distributed. Using the initial residuals and *rho* calculate the whitened residuals and compare them to the un-whitened raw residuals and to a normal distribution. For this case, does whitening the residuals make a difference?

```
# Whiten the residuals
n <- length(res)
w_res <- res[2:n] - start_rho*res[1:(n-1)]

par(mfrow = c(1,1))
plot(density(w_res), col="black", xlab="Residuals", yaxt="n", main="")
lines(density(res), col="red")
lines(density(rnorm(1e5, sd=5)), col="gray", lty=2)
abline(v=0, col="gray", lty=2)
legend("topleft", c("Whitened", "Un-whitened", "Normal distribution\nwith mean 0"),
       lty=c(1,1,2), col=c("black", "red", "gray"), bty="n")
```

Using different prior distributions

Modify the prior parameter distributions (priors do not have to be uniform distributions). Explain your reasoning behind the modifications and rerun the MCMC calibration. How do the results differ?

Thinning the chain

Thinning the chain is the process of taking a subsample of the chain instead of all of the chain. For example, you can take every "n'th" observation from the chain. Note that thinning the chain is optional and often not necessary. Here, we give an example of how it is done in practice.

Thinning can be useful for two reasons: (1) to reduce storage burden and (2) to increase computational speed. A good way to inform the choice of thinning is to look at the autocorrelation function of the Markov chains. We adopt the standard that the chains should be no smaller in length than the lag at which the autocorrelation function is below 0.05 or (5%), so the thinned parameters are not substantially autocorrelated (Link and Eaton, 2012). The plots show the parameters are relatively uncorrelated by roughly the 5,000th lag (this number may vary because the chains are not converged). This means that to properly thin the chain only every 5,000th iteration (or more) should be saved and that the samples are in general highly correlated with one another.

```
# The diagonal ACF plots are the plots of interest
par(mfrow = c(3,2))
for(i in 1:5){
  acf(slr.burnin.chain[,i], lag.max = 20000,
      main = paste('Parameter = ', parnames[i], sep = ''))
  abline(h = 0.05, col = "red")
}
```

When you thin the chains, visually check to make sure the full chain and the subset have almost identical posterior parameter distributions, so no important information is lost. The plot created below shows that the distributions are similar in shape, however, they do not visually match in shape.

```
# Thin the chain to a subset by taking every 5000th iteration (this number
# may vary if the chains are not converged). Another option is to
# use the sample() command.
```

```
subset_length <- round(nrow(slr.burnin.chain)/5000, 0)
sub_chain = slr.burnin.chain[seq(1, nrow(slr.burnin.chain), 5000), ]
# sub_chain2 <- slr.burnin.chain[sample(nrow(slr.burnin.chain), size = subset_length,
#                                       replace = FALSE), ]

# Check for similarities between full chain and the subset.
par(mfrow = c(3,2))
for(i in 1:5){
  plot(density(slr.burnin.chain[ ,i]), type="l",
       xlab = paste('Parameter =',' ', parnames[i], sep=''), ylab="PDF", main="")
  lines(density(sub_chain[ ,i]), col="red")
}
```

In this chapter, the output was not thinned. Why was thinning not necessary as well as inappropriate for this example?

Applying MCMC to other data and models

In previous exercises, you used the sea-level data from Jevrejeva et al (2008). Apply MCMC analysis accounting for autocorrelation and heteroskedastic residuals on the Jevrejeva et al (2008) data. For this, use the second order polynomial from earlier chapters as the model to fit to the data. You can extract the third vector from `sl.data` as your measurement error, `err.sl <- sl.data[ ,3]` and do not forget to set `y.meas.err <- err.sl`. Examine your sea-level rise estimate in 2100. How do your estimates differ from Exercise #7? How do your model parameter distributions differ from Exercise #7?

Bootstrapping the sea-level model

Apply the Bootstrap method to the Rahmstorf (2007) model with the data used in this chapter. How would one account for autocorrelation? Can one account for heteroskedasticity using the Bootstrap method?

References

Bayes, T. 1764. An essay toward solving a problem in the doctrine of chances. Philosophical Transactions of the Royal Society on London 53, 370-418. Available online at http://rstl.royalsocietypublishing.org/content/53/370.full.pdf+html

Church, J. A. and White, N. J. 2011. Sea-Level Rise from the Late 19th to the Early 21st Century. Surv Geophys, 32, 585-602. Available online at https://link.springer.com/article/10.1007/s10712-011-9119-1.

Gelman, A. and Rubin, D. B. 1992. Inference from iterative simulation using multiple sequences. Statistical Science, 7, 457-511. Available online at https://projecteuclid.org/euclid.ss/1177011136

Gilks, W. R. 1997. Markov chain Monte Carlo in practice. Chapman & Hall/CRC Press, London, UK

Hastings, W. K. 1970. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57(1):97–109. doi:10.1093/biomet/57.1.97. Available online at https://www.jstor.org/stable/2334940?seq=1#page_scan_tab_contents

Jevrejeva, S., Moore, J. C., Grinsted, A., and Woodworth, P. L., 2008. Recent global sea level acceleration started over 200 years ago? Geophysical Research Letters 35, L08715. Available online at http://onlinelibrary.wiley.com/doi/10.1029/2008GL033611/full

Link, W. A. and Eaton, M. J. 2012. On thinning of chains in MCMC. Methods in Ecology and Evolution 3, 112-115. Available online at http://onlinelibrary.wiley.com/doi/10.1111/j.2041-210X.2011.00131.x/abstract

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of state calculations by fast computing machines. J. Chem. Phys. 21(6), 1087-1092. Available online at http://aip.scitation.org/doi/abs/10.1063/1.1699114

Rahmstorf, S. 2007. A Semi-empirical approach to projecting future sea-level rise. Science 315, 368–370. Available online at http://science.sciencemag.org/content/315/5810/368

Roberts, G. O., Gelman, A., and Gilks W. R. 1997. Weak convergence and optimal scaling of random walk Metropolis algorithms. Ann. Appl. Prob. 7, 110–120. Available online at http://projecteuclid.org/download/pdf_1/euclid.aoap/1034625254

Rosenthal, J. S. 2010. "Optimal Proposal Distributions and Adaptive MCMC." Handbook of Markov chain Monte Carlo. Eds., Brooks, S., Gelman, A., Jones, G. L., and Meng, X.-L. Chapman & Hall/CRC Press. Available online at https://pdfs.semanticscholar.org/3576/ee874e983908f9214318abb8ca425316c9ed.pdf

Ruckert, K. L., Guan, Y., Bakker, A. M. R., Forest, C. E., and Keller, K. The effects of non-stationary observation errors on semi-empirical sea-level projections. Climatic Change 140(3), 349-360. Available online at http://dx.doi.org/10.1007/s10584-016-1858-z

Smith, T. M., Reynolds, R. W., Peterson, T. C., and Lawrimore, J. 2008. Improvements to NOAA's historical merged land-ocean surface temperature analysis (1880-2006). J. Clim. 21, 2283–2296. Available online at http://journals.ametsoc.org/doi/abs/10.1175/2007JCLI2100.1

Vihola, M. 2011. Robust adaptive Metropolis algorithm with coerced acceptance rate. Statistics and Computing. Available online at http://www.springerlink.com/content/672270222w79h431/

Zellner, A. and Tiao, G. C. 1964. Bayesian analysis of the regression model with autocorrelated errors. J. Am. Stat. Assoc. 59, 763–778. available online at http://www.sciencedirect.com/science/article/pii/S0167947301000846