Class 14 Structured random effects: time series and related models

Andrew Parnell
andrew.parnell@mu.ie

Maynooth University
National University of Ireland Maynooth

Learning outcomes

- Fit some time series models in Stan and use these with hierarchical models
- Some new methods
    - Autoregressive models
    - Stochastic Volatility Models
    - State space and dynamic models
    - A repeated measures time series
- Do some model comparison with Stan
- Show we can do shrinkage rather than model selection

A short introduction to time series methods

Almost all of time series is based on two ideas:

1. Base your future predictions on previous values of the data
2. Base your future predictions on how wrong you were in your past predictions

Everything else in time series is just an extension of these!

In this class we will only discuss *discrete time series*, i.e. where $t = 1, 2, 3, \ldots$

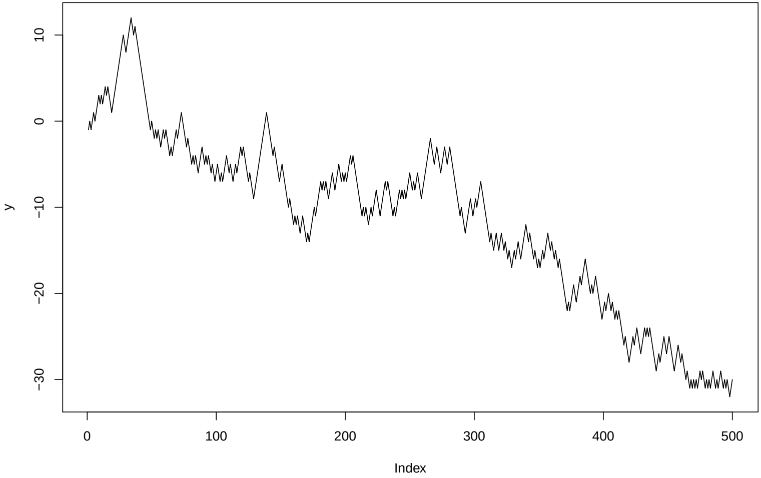
Decomposing time series

- We decompose time series commonly as:

$$y_t = \text{trend}_t + \text{seasonality}_t + \text{error}_t$$

- ... but sometimes it is not easy to separate these into different parts
- The concept of *stationarity* helps us decompose the time series

A time series with a big trend?



Generating the series

```
set.seed(123)
y <- cumsum(sample(c(-1, 1), size=1000, replace=TRUE))
```

- The `sample` command just produces a set of 1000 values either -1 or 1
- `cumsum` just cumulatively adds them up
- This is a *random walk* series

Autoregressive (AR) models

- Autoregressive models literally perform a linear regression of the time series against the previous lag of the series
- For example, an AR(1) process can be written as:

$$y_t = \alpha + \beta y_{t-1} + \epsilon_t$$

- where $\epsilon_t \sim N(0, \sigma^2)$ just like a linear regression.
- In a probability distribution format, we might write:

$$y_t \sim N(\alpha + \beta y_{t-1}, \sigma^2)$$

. . . and maximise the likelihood as normal

Interpretation of the AR parameters

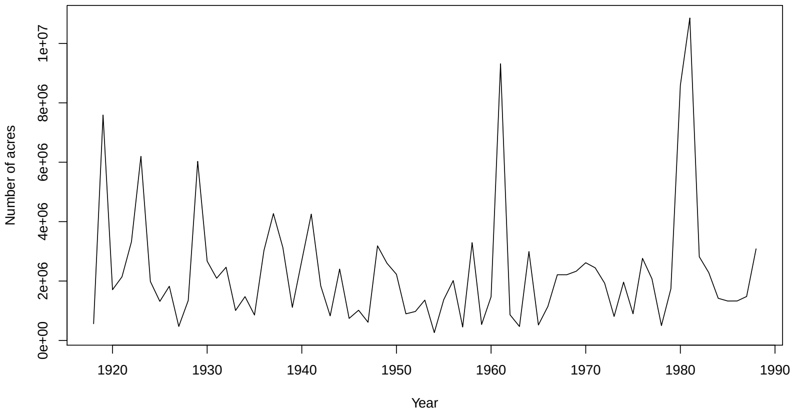
- α is an estimate of the stable mean of the process
- β is interesting:
    - Values close to 1 indicate that the series is almost like a random walk.
    - Values close to 0 indicate that the series is almost completely composed of random normally-distributed error terms
    - Values less than 0 indicate that the series is ‘repulsive’
    - Values greater than 1 (or less than -1) indicate that the series is chaotic

Fitting an AR model in rstan

```
stan_code = '
...
model {
  for (t in 2:N)
    y[t] ~ normal(alpha + beta * y[t-1], sigma);
  // Priors
  alpha ~ normal(0, 10);
  beta ~ normal(0, 10);
  sigma ~ uniform(0, 100);
}'
```

Forest fire data

```r
ff = read.csv('../data/forest_fires.csv')
with(ff, plot(year, acres, type = 'l',
                ylab = 'Number of acres',
                xlab = 'Year'))
```



Another way of running stan

- Set up a stan model

```
stan_mod_ar1 = stan_model(model_code = stan_code)
```

- Now choose either full MCMC. . .

```
stan_run_ar1 = sampling(stan_mod_ar1,
                        data = list(y = scale(ff$acres)[,1],
                                    N = nrow(ff)))
```

- . . . or just optimizing:

```
stan_opt_ar1 = optimizing(stan_mod_ar1,
                          data = list(y = scale(ff$acres)[,1],
                                      N = nrow(ff)))
```

```
print(stan_opt_ar1)
```

```
## $par
##      alpha       beta      sigma
## 0.01308191 0.16354467 0.98117769
##
## $value
## [1] -33.67004
##
## $return_code
## [1] 0
```

Changing the variance instead

- The AR model has a mean that changes but the variance is constant:
$$y_t \sim N(\alpha + \beta y_{t-1}, \sigma^2)$$
- Instead we could try:
$$y_t \sim N(\alpha, \sigma_t^2)$$
- Lots of different ways to model this:
  - Autoregressive Conditional Heteroskedasticity (ARCH)
  - Generalised Autoregressive Conditional Heteroskedasticity (GARCH)
  - Stochastic Volatility Models (SVM)
- They follow the same principles as AR models, but work on the standard deviations or variances instead of the mean

Stochastic Volatility Modelling

- A Stochastic Volatility Model (SVM) models the variance as its own *stochastic process*
- The general model structure is often written as:

$$y_t \sim N(\alpha, \exp(h_t))$$

$$h_t \sim N(\mu + \phi h_{t-1}, \sigma^2)$$

- You can think of an SVM being like a GLM but with a log link on the variance parameter

Mixing up models

- What if we wanted to fit an AR(1) model with stochastic volatility
- Impossible in almost any R package
- Simple to do in Stan!

Code for a an AR(1)-SVM

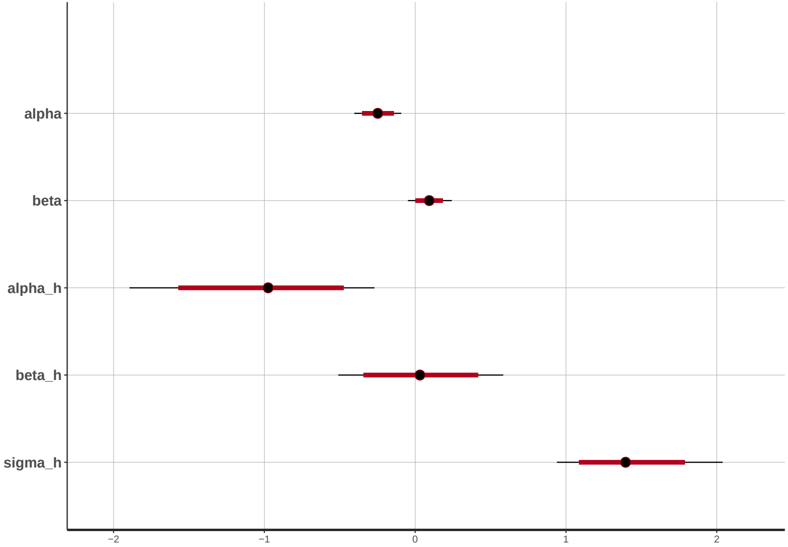
```
stan_code = '
data {
  int<lower=0> N; // number of observations
  vector[N] y; // response variable
}
parameters {
  real alpha; // intercept
  real beta; // AR parameter
  vector[N] h; // stochastic volatility process
  real alpha_h; // SVM mean
  real beta_h; // SVM AR parameter
  real<lower=0> sigma_h; // SVM residual SD
}
model {
  h[1] ~ normal(alpha_h, 1);
  for (t in 2:N) {
    y[t] ~ normal(alpha + beta * y[t-1], sqrt(exp(h[t])));
    h[t] ~ normal(alpha_h + beta_h * h[t-1], sigma_h);
  }
}'
```

Find the posterior distribution

```r
print(stan_run_svm)
```

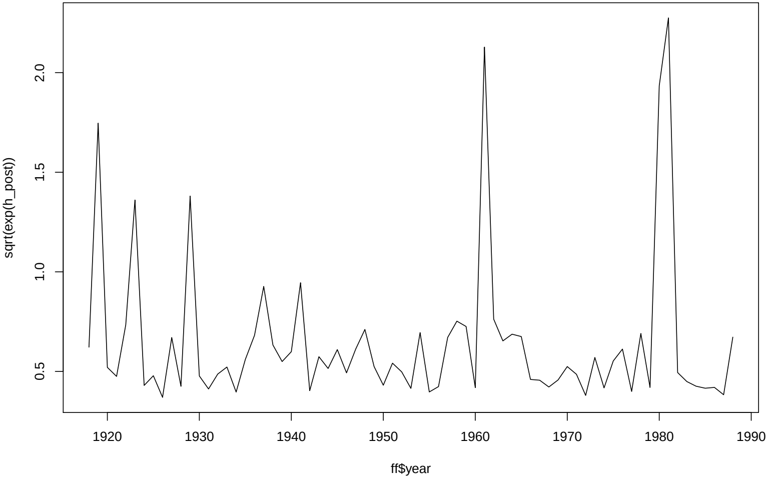
```
## Inference for Stan model: fe789435a2f878b474bc085314281fc3.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=400
##
##             mean se_mean   sd  2.5%   25%   50%   75% 97.
## alpha      -0.25    0.00 0.08 -0.40 -0.30 -0.25 -0.19 -0.
## beta        0.09    0.00 0.07 -0.05  0.05  0.09  0.14  0.
## h[1]       -0.97    0.03 1.18 -3.29 -1.78 -0.95 -0.16  1.
## h[2]        1.21    0.01 0.79 -0.10  0.65  1.12  1.68  2.
## h[3]       -1.30    0.03 1.36 -3.99 -2.20 -1.31 -0.37  1.
## h[4]       -1.47    0.02 1.31 -4.01 -2.35 -1.49 -0.62  1.
## h[5]       -0.54    0.02 0.99 -2.31 -1.23 -0.62  0.08  1.
## h[6]        0.69    0.01 0.80 -0.65  0.12  0.62  1.18  2.
## h[7]       -1.75    0.02 1.40 -4.58 -2.67 -1.69 -0.80  0.
## h[8]       -1.48    0.02 1.37 -4.02 -2.43 -1.47 -0.59  1.
## h[9]       -1.99    0.02 1.42 -4.81 -2.93 -1.99 -1.05  0.
## h[10]      -0.70    0.02 1.08 -2.55 -1.47 -0.80 -0.05  1.
## h[11]      -1.74    0.04 1.49 -4.66 -2.71 -1.71 -0.74  1.
```

Plot the important parameters



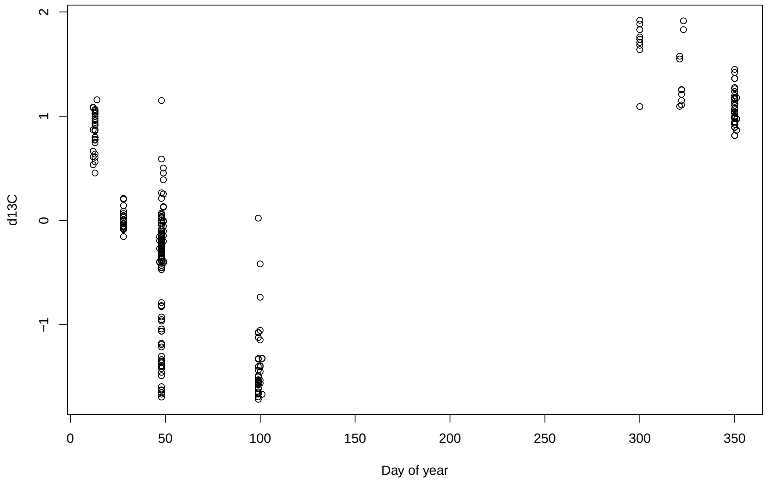
Plot the $\sqrt{\exp(h)}$ values

```
h_post = summary(stan_run_svm, pars = c("h"))$summary[,'50%
plot(ff$year, sqrt(exp(h_post)), type = 'l')
```



A repeated measures example

- Let's return to the Geese example all the way back on day 1:



What model would we like for these data?

- We have *repeated measures* - more than one observation at each time point.
- We would like the model to fill in the gaps and separate out the uncertainty due to the change over time from the uncertainty to do with repeated measurement
- We have to separate out the model into two layers:
  1. The observations and how they link to a single time series value on that day
  2. The underlying time series model defined at each time point
- A possible model:

$$y_t \sim N(\mu_{\text{day}_t}, \sigma^2)$$

$$\mu_{\text{day}} \sim N(\mu_{\text{day}-1}, \sigma^2_\mu)$$

Stan code for a repeated measures random walk model

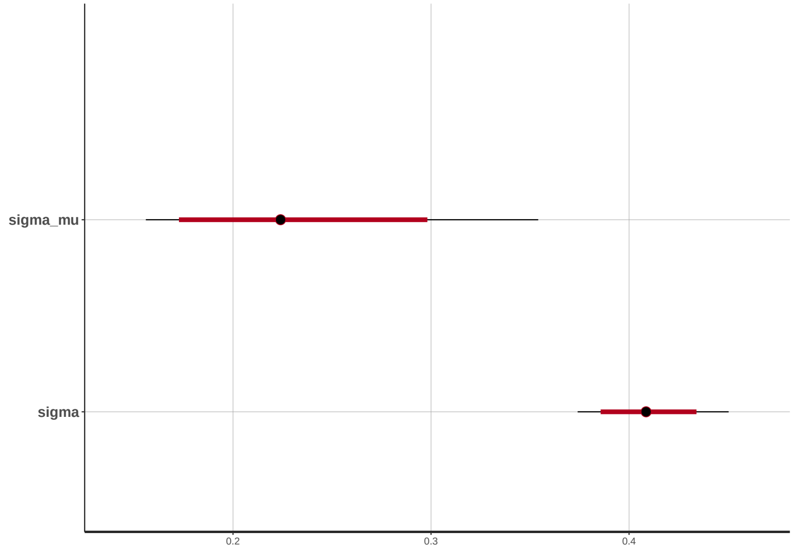
```
stan_code_rm = '
data {
  int<lower=0> N; // number of observations
  int<lower=0> N_day; // total number of days
  vector[N] y; // response variable
  int day[N]; // variable to match days to observations
}
parameters {
  real<lower=0> sigma; // st dev within day
  real<lower=0> sigma_mu; // st dev of RW
  vector[N_day] mu; // repeated measure parameter
}
model {
  mu[1] ~ normal(0, sigma_mu);
  for(t in 2:N_day) {
    mu[t] ~ normal(mu[t-1], sigma_mu);
  }
  sigma ~ uniform(0, 10);
  sigma_mu ~ uniform(0, 10);
  for (i in 1:N)
    y[i] ~ normal(mu[day[i]], sigma);
}'
```

Optimise the parameters

```r
print(stan_run_rm, pars = c('sigma_mu', 'sigma'))
```

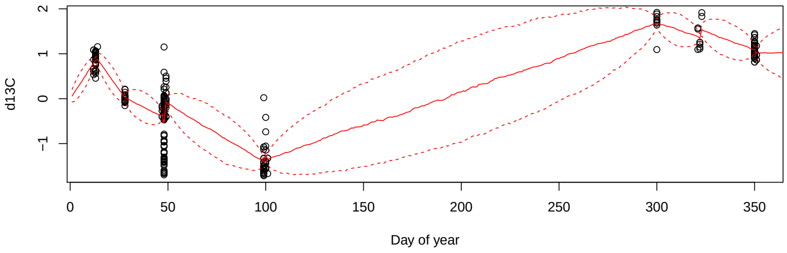
```
## Inference for Stan model: 9824a40eeca19f5c917c651042e0bc
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draw
##
##          mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_e
## sigma_mu 0.23    0.01 0.05 0.16 0.19 0.22 0.26  0.35
## sigma    0.41    0.00 0.02 0.37 0.40 0.41 0.42  0.45  40
##
## Samples were drawn using NUTS(diag_e) at Tue Oct  9 15:3
## For each parameter, n_eff is a crude measure of effecti
## and Rhat is the potential scale reduction factor on spl
## convergence, Rhat=1).
```

Plot the interesting parameters



Plot the best fit model

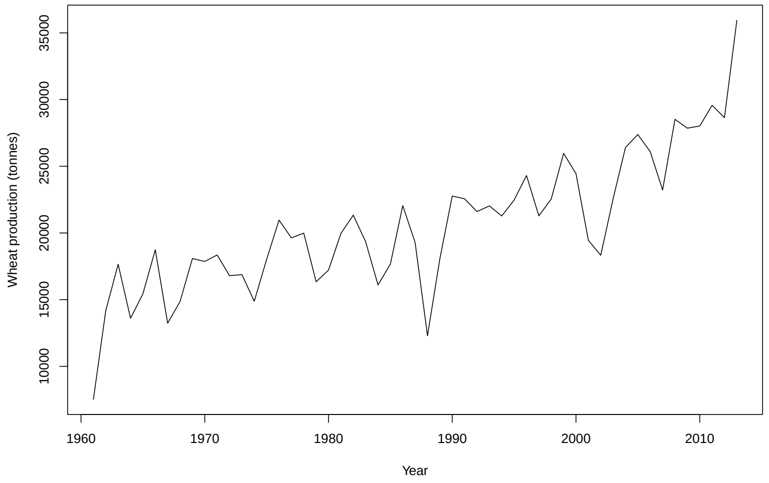
```
with(geese, plot(int_days[o], scale(d13CPl[o])[,1],
                                          ylab = 'd13C',
                    xlab = 'Day of year'))
mu_post = summary(stan_run_rm, pars = c("mu"))$summary[,c(
lines(1:365, mu_post[,1], col = 'red', lty = 2)
lines(1:365, mu_post[,2], col = 'red', lty = 1)
lines(1:365, mu_post[,3], col = 'red', lty = 2)
```



Shrinkage and AR models

- It's possible to do variable selection with AR models of multiple orders
- Suppose we want to choose the order of auto-regression p in an AR(p) model
- We would fit a model for a large number of p values and put a prior to reduce the size on the coefficients on them
- The normal isn't the only choice, an even more popular one is the double exponential (or Laplace) distribution

Reminder: wheat data



Fitting a shrinkage AR model

```
stan_code_ar_shrink = '
data {
  int<lower=0> N; // number of observations
  int<lower=0> max_P; // maximum number of AR lags
  vector[N] y; // response variable
}
parameters {
  real alpha; // intercept
  vector[max_P] beta; // AR parameter
  real<lower=0> sigma; // residual sd
}
model {
  for (t in (max_P+1):N) {
        real mu;
        mu = alpha;
        for(k in 1:max_P)
          mu = mu + beta[k] * y[t-k];
        y[t] ~ normal(mu, sigma);
  }
  // Priors
  alpha ~ normal(0, 10);
  for (k in 1:max_P) {
    beta ~ double_exponential(0, 1);
  }
  sigma ~ uniform(0, 100);
}'
```

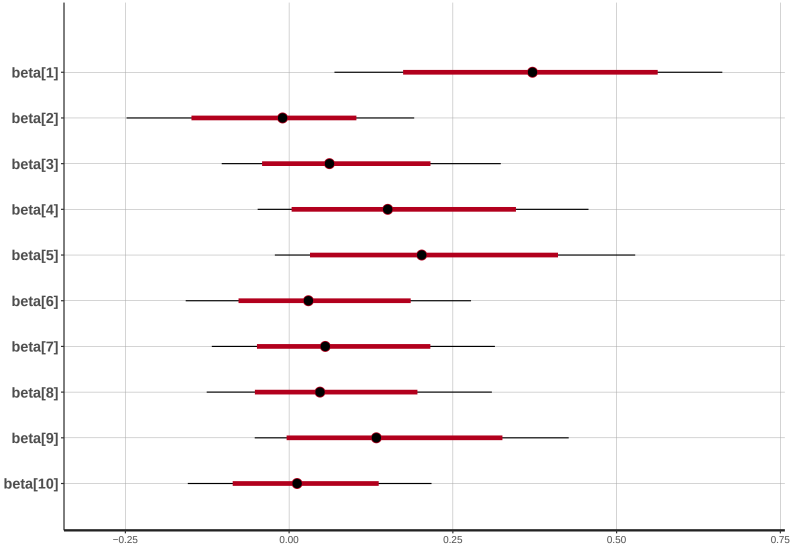
Fitting the model

```r
print(stan_run_ar_shrink)
```

```
## Inference for Stan model: ea24da5bc3933a959f2a5cf4093b89
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draw
##
##            mean se_mean   sd  2.5%   25%   50%   75% 97.
## alpha      0.27    0.00 0.10  0.08  0.21  0.27  0.34  0.
## beta[1]    0.37    0.00 0.15  0.07  0.27  0.37  0.48  0.
## beta[2]   -0.02    0.00 0.11 -0.25 -0.08 -0.01  0.04  0.
## beta[3]    0.08    0.00 0.11 -0.10  0.00  0.06  0.14  0.
## beta[4]    0.17    0.00 0.13 -0.05  0.06  0.15  0.26  0.
## beta[5]    0.22    0.00 0.15 -0.02  0.11  0.20  0.31  0.
## beta[6]    0.04    0.00 0.11 -0.16 -0.02  0.03  0.10  0.
## beta[7]    0.07    0.00 0.11 -0.12  0.00  0.06  0.13  0.
## beta[8]    0.06    0.00 0.11 -0.13 -0.01  0.05  0.12  0.
## beta[9]    0.15    0.00 0.13 -0.05  0.05  0.13  0.23  0.
## beta[10]   0.02    0.00 0.09 -0.15 -0.04  0.01  0.07  0
```

Plot the AR parameters

```r
plot(stan_run_ar_shrink, pars = c('beta'))
```



Mixing up state space models, multivariate time series, Gaussian processes

- We can extend the simple state space model we met earlier to work for multivariate series
- We would have a state equation that relates our observations to a multivariate latent time series (possibly of a different dimension)
- We could change the time series model of the latent state to be an ARIMA model, an O-U process, a Gaussian process, or anything else you can think of!

Dynamic linear models

- So far in all our models we have forced the time series parameters to be constant over time
- In a *Dynamic Linear Model* we have a state space model with :

$$y_t = F_t x_t + \epsilon_t,\ \epsilon_t \sim MVN(0, \Sigma_t)$$

$$x_t = G_t x_{t-1} + \gamma_t,\ \gamma_t \sim N(0, \Psi_t)$$

- The key difference here is that the transformation matrices F_t and G_t can change over time, as can the variance matrices Σ_t and Ψ_t, possibly in an ARCH/GARCH type framework
- These are very hard models to fit in JAGS/Stan but simple versions can work

Latent factor time series models

- If we have very many series, a common approach to reduce the dimension is to use Factor Analysis or Principal components
- In a latent factor model we write:

$$y_t = Bf_t + \epsilon_t$$

  where now B is a *numseries* $\times$ *numfactors* factor loading matrix which transforms the high dimensional y_t into a lower dimensional f_t.
- f_t can then be run using a set of univariate time series, e.g. random walks
- The B matrix is often hard to estimate and might require some tight priors

Summary

- Time series analysis involves using regression-type models to predict new data from previous time points
- With a Bayesian model you can do the model selection inside the modelling step
- Some really cool and flexible models can be fitted with advanced tools like state space models