Chapter 14. Case study: POMP modeling to investigate financial volatility

Objectives

1. Introduce financial volatility and some models used to study it.
2. Define ARCH, GARCH, and stochastic volatility models.
3. Provide a case study in using `pomp` to study a stochastic volatility model.
4. Discuss this case study as an example of a broader class of nonlinear POMP models, derived from adding stochastically time varying parameters to a linear or nonlinear time series model.
5. Discuss this case study as an example of an extension of the POMP framework which allows feedback from the observation process to the state process.

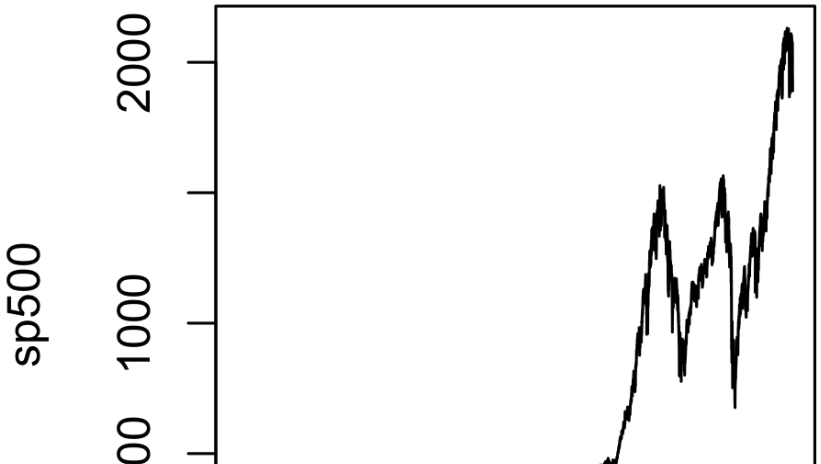
Introduction

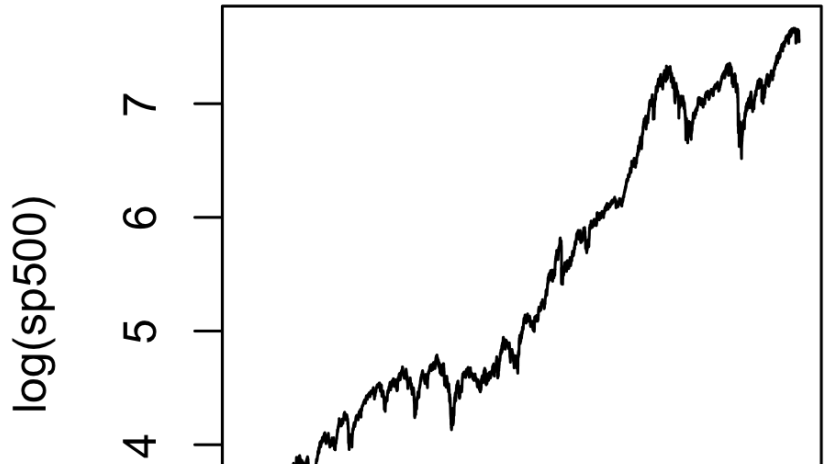
- Returns on investments in stock market indices or large companies are often found to be approximately uncorrelated.
- If investment returns are substantially correlated, that provides an opportunity for investors to study their time series behavior and make money.
- If the investment is non-liquid (i.e., not reliably tradeable), or expensive to trade, then it might be hard to make money even if you can statistically predict a positive expected return.
- Otherwise, the market will likely notice a favorable investment opportunity. More buyers will push up the price, and so the favorable situation will disappear.
- Consequently, most liquid and widely traded investments (such as stock market indices, or stock of large companies) have close to uncorrelated returns.
- However, the variability of the returns (called the volatility) can fluctuate considerably. Understanding this volatility is important for quantifying and managing the risk of investments.

Data

- Recall the daily S&P 500 data that we saw earlier, in Chapter 3, downloaded from `yahoo.com`.

```
dat <- read.table("sp500.csv",sep=",",header=TRUE)
N <- nrow(dat)
sp500 <- dat$Close[N:1] # data are in reverse order in sp500.csv
par(mfrow=c(1,2))
plot(sp500,type="l")
plot(log(sp500),type="l")
```





Returns, absolute returns, and autocorrelation

- We write $\{z_n, n = 1, \ldots, N\}$ for the data.
- We write the return on the S&P 500 index, i.e., the difference of the log of the index, as

$$y_n = \log(z_n) - \log(z_{n-1}).$$

  * We saw in Chapter 3 that $y_{2:N}$ has negligible sample autocorrelation.

- However, the absolute deviations from average,

$$a_n = \left| y_n - \frac{1}{N-1} \sum_{k=2}^{N} y_k \right|$$

  have considerable sample autocorrelation.

Question 14.1. Fitting a stationarity model to stock market returns.

- Look at the plots for the S&P data when we considered the random walk model in Chapter 3 of the notes.
- Is it appropriate to fit a stationary model to $y_{2:N}$, or do we have evidence for violation of stationarity? Explain.

ARCH and GARCH models

- The ARCH and GARCH models have become widely used for financial time series modeling. Here, we follow @cowpertwait09 to introduce these models and corresponding computations in R. See also, Section 5.4 of Shumway and Stoffer (3rd edition).
- An order p autoregressive conditional heteroskedasticity model, known as ARCH(p), has the form

$$Y_n = \epsilon_n \sqrt{V_n},$$

where

$$V_n = \alpha_0 + \sum_{j=1}^{p} \alpha_j Y_{n-j}^2$$

and $\epsilon_{1:N}$ is white noise.
- If $\epsilon_{1:N}$ is Gaussian, then $Y_{1:N}$ is called a Gaussian ARCH(p). Note, however, that a Gaussian ARCH model is not a Gaussian process, just a process driven by Gaussian noise.
- If $Y_{1:N}$ is a Gaussian ARCH(p), then $Y_{1:N}^2$ is AR(p), but not Gaussian AR(p).

Fitting a GARCH model

```
require(tseries)
fit.garch <- garch(sp500,grad = "numerical", trace = FALSE)
L.garch <- logLik(fit.garch)
```

- This 3-parameter model has a maximized log likelihood of NA.

Question 14.2. It is usually inappropriate to present numerical results to seven significant figures. Is this appropriate for reporting the log likelihood here? Why?

- We are now in a position to employ the framework of likelihood-based inference for GARCH models. In particular, profile likelihood, likelihood ratio tests, and AIC are available.
- We can readily simulate from a fitted GARCH model, if we want to investigate properties of a fitted model that we don't know how to compute analytically.
- However, GARCH is a black-box model, in the sense that the parameters don't have clear interpretation. We can develop an appropriate GARCH(p,q) model, and that may be useful for forecasting, but it won't help us understand more about how financial markets work.
- To develop and test a hypothesis that goes beyond the class of GARCH models, it is useful to have the POMP framework available.

Financial leverage

- It is a fairly well established empirical observation that negative shocks to a stockmarket index are associated with a subsequent increase in volatility.
- This phenomenon is called **leverage**.
- Here, we formally define leverage, R_n on day n as the correlation between index return on day $n-1$ and the increase in the log volatility from day $n-1$ to day n.
- Models have been proposed which incorporate leverage into the dynamics (Bretó; 2014).
- We present a pomp implementation of @breto14, which models R_n as a random walk on a transformed scale,

$$R_n = \frac{\exp\{2G_n\}-1}{\exp\{2G_n\}+1},$$

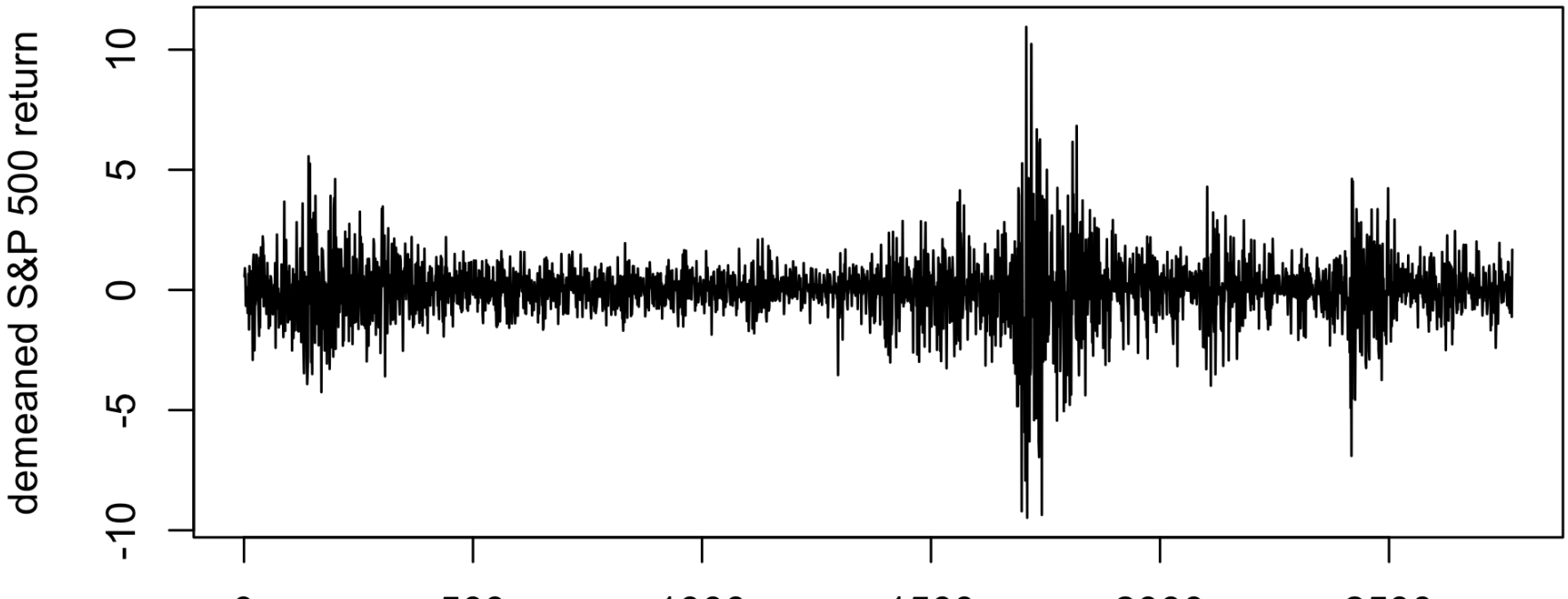
where $\{G_n\}$ is the usual, Gaussian random walk.

Time-varying parameters

- A special case of this model, with the Gaussian random walk having standard deviation zero, is a fixed leverage model.
- The POMP framework provides a general approach to time-varying parameters. Considering a parameter as a latent, unobserved random process that can progressively change its value over time (following a random walk, or some other stochastic process) leads to a POMP model. The resulting POMP model is usually nonlinear.
- Many real-world systems are non-stationary and could be investigated using models with time-varying parameters.
- Some of the midterm projects discovered examples of this.
- This is one possible thing to explore in your final project.

- We fit the time-varying leverage model to demeaned daily returns for the S&P 500 index, using the data that were downloaded, detrended and analyzed by Bretó (2014).
- Here, we just analyze 2002-2012, though it is also interesting to fit to longer intervals, or to compare fits to different intervals.

```
load(file="sp500-2002-2012.rda")
plot(sp500.ret.demeaned,xlab="business day (2002-2012)",ylab="demean
```



Building a POMP model

```
require(pomp)
```

- A complication is that transitions of the latent variables from (G_n, H_n) to (G_{n+1}, H_{n+1}) depends on the observable variable Y_n.
- Formally, therefore, we use the state variable $X_n = (G_n, H_n, Y_n)$ and model the measurement process as a perfect observation of the Y_n component of the state space.
- To define a recursive particle filter, we write filter particle j at time $n-1$ as

$$X^F_{n-1,j} = (G^F_{n-1,j}, H^F_{n-1,j}, y_{n-1}).$$

- Now we can construct prediction particles at time n,

$$(G^P_{n,j}, H^P_{n,j}) \sim f_{G_n,H_n|G_{n-1},H_{n-1},Y_{n-1}}(g_n|G^F_{n-1,j}, H^F_{n-1,j}, y_{n-1})$$

with corresponding weight

$$w_{n,j} = f_{Y_n|G_n,H_n}(y_n|G^P_{n,j}, H^P_{n,j}).$$

- Resampling with probability proportional to these weights gives an SMC representation of the filtering distribution at time n.

- We can coerce the basic sequential Monte Carlo algorithm, implemented as `pfilter` in `pomp`, into carrying out this calculation by building two different `pomp` objects, one to do filtering and another to do simulation.
- For the implementation in `pomp`, we proceed to write Csnippet code for the two versions of `rprocess`.

```
sp500_statenames <- c("H","G","Y_state")
sp500_rp_names <- c("sigma_nu","mu_h","phi","sigma_eta")
sp500_ivp_names <- c("G_0","H_0")
sp500_paramnames <- c(sp500_rp_names,sp500_ivp_names)
```

```
rproc1 <- "
  double beta,omega,nu;
  omega = rnorm(0,sigma_eta * sqrt( 1- phi*phi ) * sqrt(1-tanh(G)*ta
  nu = rnorm(0, sigma_nu);
  G += nu;
  beta = Y_state * sigma_eta * sqrt( 1- phi*phi );
  H = mu_h*(1 - phi) + phi*H + beta * tanh( G ) * exp(-H/2) + omega
"
rproc2.sim <- "
  Y_state = rnorm( 0,exp(H/2) );
 "


rproc2.filt <- "
  Y_state = covaryt;
 "
sp500_rproc.sim <- paste(rproc1,rproc2.sim)
sp500_rproc.filt <- paste(rproc1,rproc2.filt)
```

```
sp500_rinit <- "
  G = G_0;
  H = H_0;
  Y_state = rnorm( 0,exp(H/2) );
"
```

```
sp500_rmeasure <- "
   y=Y_state;
"

sp500_dmeasure <- "
   lik=dnorm(y,0,exp(H/2),give_log);
"
```

Parameter transformations

- For optimization procedures such as iterated filtering, it is convenient to transform parameters to be defined on the whole real line.
- We therefore write transformation functions for σ_η, σ_ν and ϕ,

```
sp500_partrans <- parameter_trans(
  log=c("sigma_eta","sigma_nu"),
  logit="phi"
)
```

- We can now build a pomp object suitable for filtering, and parameter estimation by iterated filtering or particle MCMC.
- Note that the data are also placed in a covariate slot.
- This is a devise to allow the state process evolution to depend on the data. In a POMP model, the latent process evolution depends only on the current latent state. In `pomp`, the consequence of this structure is that `rprocess` doesn't have access to the observation process.
- However, a POMP model does allow for the possibility for the basic elements to depend on arbitrary covariates. In `pomp`, this means `rprocess` has access to a covariate slot.
- The code below gives an example of how to fill the covariate slot and how to use it in `rprocess`.

```
sp500.filt <- pomp(data=data.frame(
    y=sp500.ret.demeaned,time=1:length(sp500.ret.demeaned)),
  statenames=sp500_statenames,
  paramnames=sp500_paramnames,
  times="time",
  t0=0,
  covar=covariate_table(
    time=0:length(sp500.ret.demeaned),
    covaryt=c(0,sp500.ret.demeaned),
    times="time"),
  rmeasure=Csnippet(sp500_rmeasure),
  dmeasure=Csnippet(sp500_dmeasure),
  rprocess=discrete_time(step.fun=Csnippet(sp500_rproc.filt),delta.
  rinit=Csnippet(sp500_rinit),
  partrans=sp500_partrans
)
```

- Simulating from the model is convenient for developing and testing the code, as well as to investigate a fitted model. We can do this as follows:

```
expit<-function(real){1/(1+exp(-real))}
logit<-function(p.arg){log(p.arg/(1-p.arg))}
params_test <- c(
     sigma_nu = exp(-4.5),
     mu_h = -0.25,
     phi = expit(4),
     sigma_eta = exp(-0.07),
     G_0 = 0,
     H_0=0
  )

sim1.sim <- pomp(sp500.filt,
  statenames=sp500_statenames,
  paramnames=sp500_paramnames,
  rprocess=discrete_time(
    step.fun=Csnippet(sp500_rproc.sim),delta.t=1)
)
```

- Now. to build the filtering object from `sim1.sim`, we need to copy the new simulated data into the covariate slot, and put back the appropriate version of `rprocess`.

```
sim1.filt <- pomp(sim1.sim,
  covar=covariate_table(
    time=c(timezero(sim1.sim),time(sim1.sim)),
    covaryt=c(obs(sim1.sim),NA),
    times="time"),
  statenames=sp500_statenames,
  paramnames=sp500_paramnames,
  rprocess=discrete_time(step.fun=Csnippet(sp500_rproc.filt),delta.
)
```

Filtering on simulated data

- Let's check that we can indeed filter and re-estimate parameters successfully for this simulated data.
- To develop and debug code, it is nice to have a version that runs extra quickly, for which we set `run_level=1`. Here, `Np` is the number of particles (i.e., sequential Monte Carlo sample size), and `Nmif` is the number of iterations of the optimization procedure carried out below. Empirically, `Np=5000` and `Nmif=200` are around the minimum required to get stable results with an error in the likelihood of order 1 log unit for this example; this is implemented by setting `run_level=2`. One can then ramp up to larger values for more refined computations, implemented here by `run_level=3`

```
run_level <- 1
sp500_Np <-           switch(run_level,100,1e3,2e3)
sp500_Nmif <-         switch(run_level,10, 100,200)
sp500_Nreps_eval <-   switch(run_level,4,  10,  20)
sp500_Nreps_local <-  switch(run_level,10, 20, 20)
sp500_Nreps_global <- switch(run_level,10, 20, 100)
```

- We'll carry out replications to assess Monte Carlo error. Since most modern machines have multiple cores, it is convenient to do some parallelization when generating replications.

```
require(doParallel)
registerDoParallel()
```

- By default, `doParallel` enrolls half the available cores. Now, we filter.

```
stew(file=sprintf("pf1.rda",run_level),{
  t.pf1 <- system.time(
    pf1 <- foreach(i=1:sp500_Nreps_eval,.packages='pomp',
                   .options.multicore=list(set.seed=TRUE)) %dopar% -
                     pfilter(sim1.filt,Np=sp500_Np)
                   )
  )
},seed=493536993,kind="L'Ecuyer")
(L.pf1 <- logmeanexp(sapply(pf1,logLik),se=TRUE))

##                       se
## -3664.273257     1.925115
```

- In 1.4 seconds, we obtain a log likelihood estimate of -3664.27 with a Monte standard error of 1.93. Notice that the replications are averaged using the `logmeanexp` function, since the likelihood estimate is unbiased on the natural scale but not the log scale.
- We could test the numerical performance of an iterated filtering likelihood maximization algorithm on simulated data.
- We could also study the statistical performance of maximum likelihood estimators and profile likelihood confidence intervals on simulated data.
- However, here we are going to cut to the chase and start fitting models to data.

Fitting the stochastic leverage model to S&P500 data

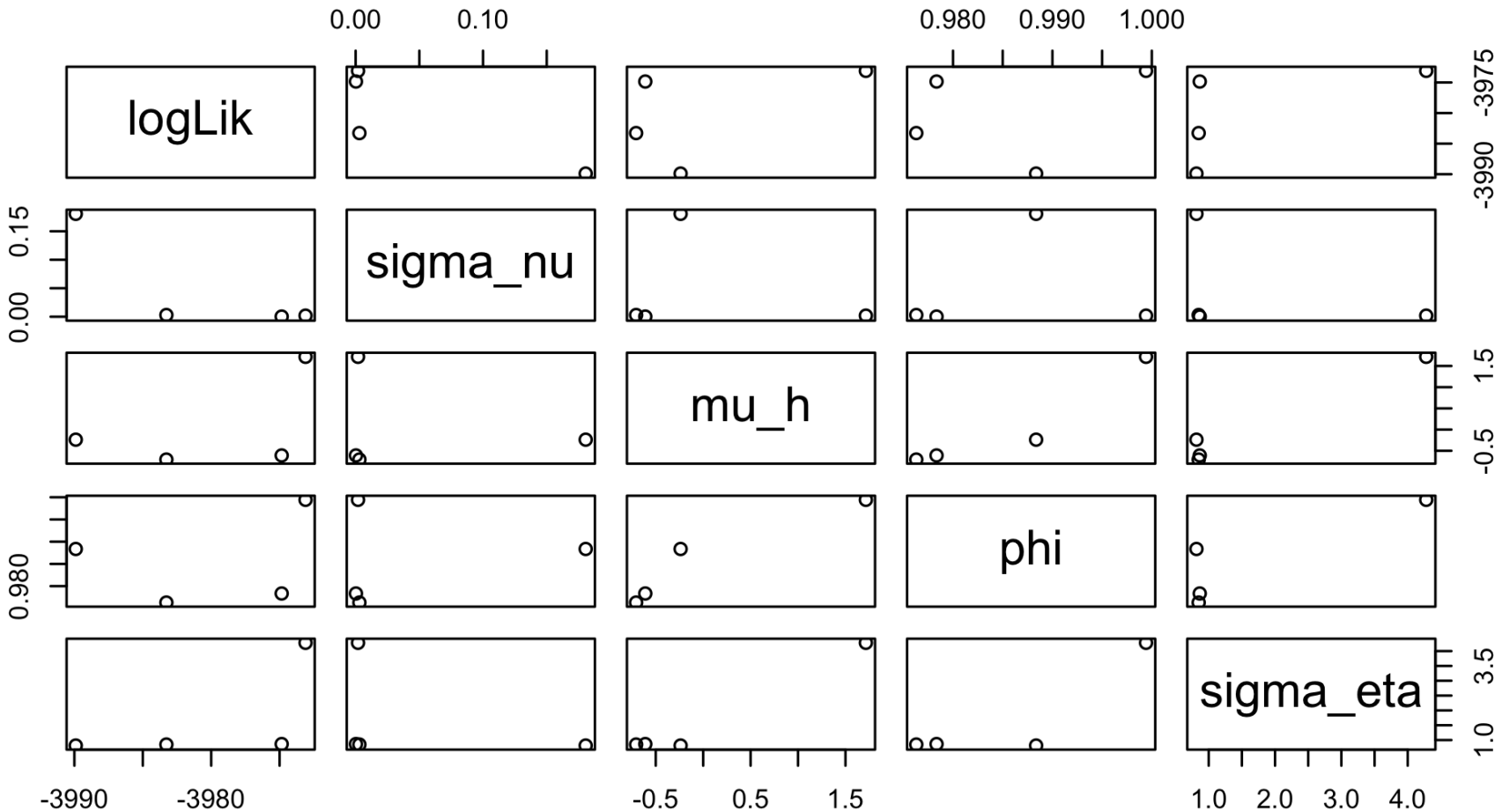
- We are now ready to try out iterated filtering on the S&P500 data. We will use the IF2 algorithm of @ionides15, implemented by `mif2`.

```
sp500_rw.sd_rp <- 0.02
sp500_rw.sd_ivp <- 0.1
sp500_cooling.fraction.50 <- 0.5

stew("mif1.rda",{
   t.if1 <- system.time({
   if1 <- foreach(i=1:sp500_Nreps_local,
     .packages='pomp', .combine=c) %dopar% mif2(sp500.filt,
          params=params_test,
          Np=sp500_Np,
          Nmif=sp500_Nmif,
          cooling.fraction.50=sp500_cooling.fraction.50,
          rw.sd = rw.sd(
                              sigma_nu  = sp500_rw.sd_rp,
                              mu_h      = sp500_rw.sd_rp,
                              phi       = sp500_rw.sd_rp,
                              sigma_eta = sp500_rw.sd_rp,
```

- This investigation took 0.3 minutes.
- The repeated stochastic maximizations can also show us the geometry of the likelihood surface in a neighborhood of this point estimate:

```
pairs(~logLik+sigma_nu+mu_h+phi+sigma_eta,data=subset(r.if1,logLik>
```



Likelihood maximization using randomized starting values

- When carrying out parameter estimation for dynamic systems, we need to specify beginning values for both the dynamic system (in the state space) and the parameters (in the parameter space).
- Practical parameter estimation involves trying many starting values for the parameters. One can specify a large box in parameter space that contains all parameter vectors which seem remotely sensible. If an estimation method gives stable conclusions with starting values drawn randomly from this box, this gives some confidence that an adequate global search has been carried out.
- For our volatility model, a box containing reasonable parameter values might be

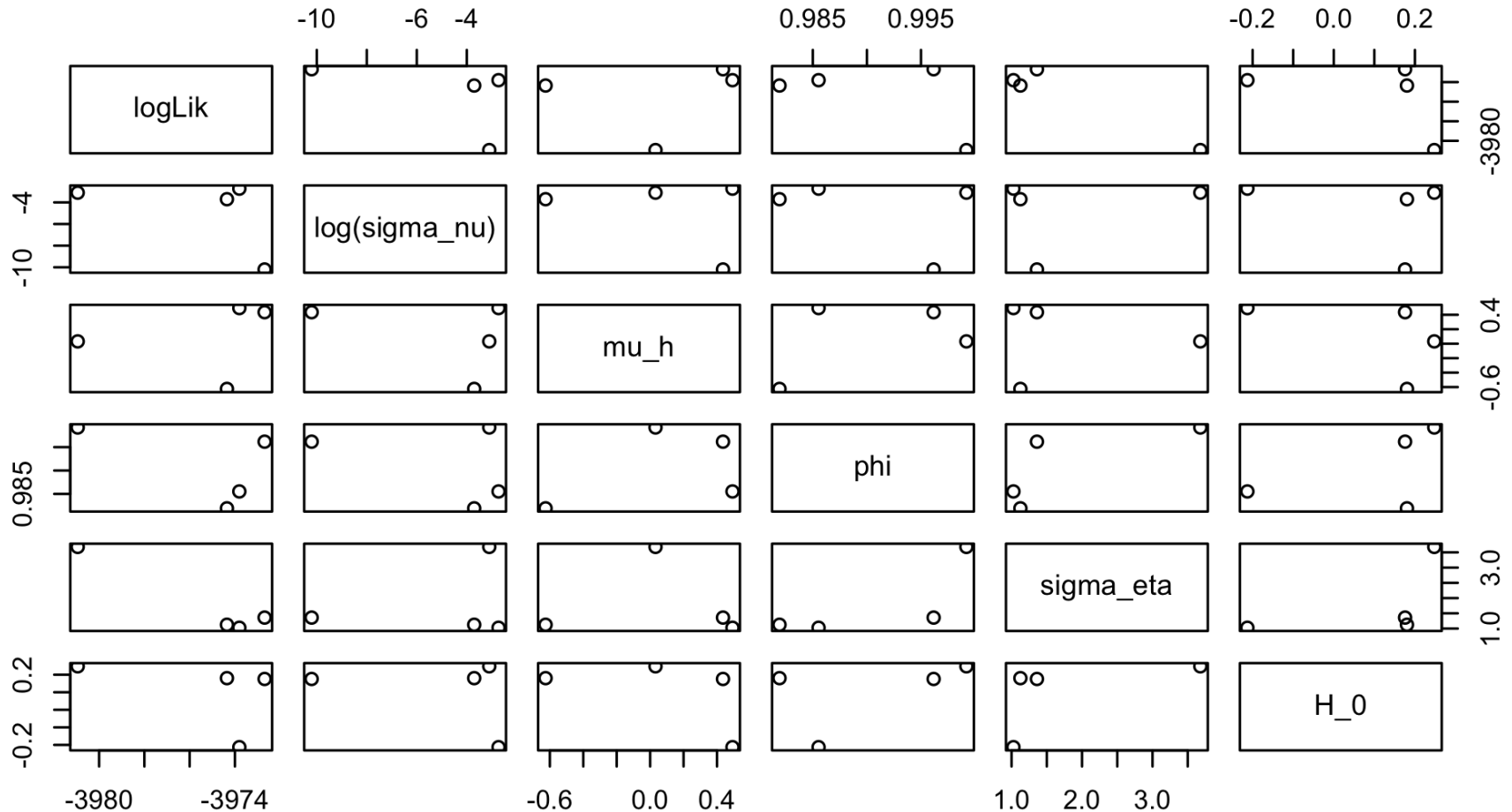
```
sp500_box <- rbind(
 sigma_nu=c(0.005,0.05),
 mu_h    =c(-1,0),
 phi = c(0.95,0.99),
 sigma_eta = c(0.5,1),
 G_0 = c(-2,2),
 H_0 = c(-1,1)
```

```
stew(file="box_eval.rda",{
  t.box <- system.time({
    if.box <- foreach(i=1:sp500_Nreps_global,
      .packages='pomp',.combine=c) %dopar%
      mif2(
        if1[[1]],
        params=apply(sp500_box,1,function(x)runif(1,x))
      )

    L.box <- foreach(i=1:sp500_Nreps_global,
      .packages='pomp',.combine=rbind) %dopar% {
                        set.seed(87932+i)
                        logmeanexp(
                          replicate(sp500_Nreps_eval,
                                    logLik(pfilter(sp500.filt,params
                          ),
                          se=TRUE)
                      }
  })
},seed=290860873,kind="L'Ecuyer")
```

- This search took 0.3 minutes.
- The best likelihood found was -3972.7 with a standard error of 1.6.
- We see that optimization attempts from diverse remote starting points can approach our MLE, but do not exceed it. This gives us some reasonable confidence in our MLE.
- Plotting these diverse parameter estimates can help to give a feel for the global geometry of the likelihood surface

```
pairs(~logLik+log(sigma_nu)+mu_h+phi+sigma_eta+H_0,data=subset(r.bo
```



- This preliminary analysis does not show clear evidence for the hypothesis that $\sigma_\nu > 0$.
- That is likely because we are studying only a subset of the 1988 to 2012 dataset analyzed by @breto14.
- Also, it might help to refine our inference be computing a likelihood profile over σ_ν.

Benchmark likelihoods for alternative models

- To assess the overall success of the model, it is helpful to put the log likelihoods in the context of simpler models
- This use of fitting a simpler model is called a **benchmark**.
- Benchmarks provide a complementary approach to residual analysis, and studying simulations from the fitted model.
- Our stochastic volatility model, with time-varying leverage, model has a maximized log likelihood of -3972.7 with 6 fitted parameters.
- Refitting the GARCH(1,1) model to this exact dataset,

```
library(tseries)
fit.garch.benchmark <- garch(sp500.ret.demeaned,grad = "numerical",
L.garch.benchmark <- logLik(fit.garch.benchmark)
```

- The GARCH(1,1) model has a maximizes likelihood of NA with 3 fitted parameters.
- Here, AIC favors the stochastic volatility model.
- A model which both fits better and has meaningful interpretation has clear advantages over a simple statistical model.
- The disadvantage of the sophisticated modeling and inference is the

Can a mechanistic model be helpful if it loses to a non-mechanistic alternative?

- Sometimes, the mechanistic model does not beat simple benchmark models. That does not necessarily mean the mechanistic model is entirely useless.
- We may be able to learn about the system under investigation from what a scientifically interpretable model fails to explain.
- We may be able to use preliminary results to improve the model, and subsequently beat the benchmarks.
- If the mechanistic model fits disastrously compared to the benchmark, our model is probably missing something important. We must reconsider the model, based on clues we might obtain by carrying out residual analysis and looking at simulations from the fitted model.

Appendix: Proper weighting for a partially plug-and-play algorithm with a perfectly observed state space component

- Suppose a POMP model with latent variable $X_n = (U_n, V_n)$ and observed varaible Y_n having conditional density $f_{Y_n|V_n}(y_n|v_n)$ depending only on V_n.
- The proper weight (Liu and Chen; 1998) for an SMC proposal density $q_n(x_n|x_{n-1})$ is

$$w_n(x_n|x_{n-1}) = \frac{f_{Y_n|X_n}(y_n|x_n) f_{X_n|X_{n-1}}(x_n|x_{n-1})}{q_n(x_n|x_{n-1})}.$$

- Consider the proposal,

$$q_n(u_n, v_n|x_{n-1}) = f_{U_n|X_{n-1}}(u_n|x_{n-1}) g_n(v_n).$$

  This is partially plug-and-play, in the sense that the U_n part of the proposal is drawn from a simulator of the dynamic system.
- Computing the weights, we see that the transition density for the U_n component cancels out and does not have to be computed, i.e.,

$$w_n(x_n|x_{n-1}) = \frac{f_{Y_n|V_n}(y_n|v_n) f_{U_n|X_{n-1}}(u_n|x_{n-1}) f_{V_n|U_n,X_{n-1}}(v_n|u_n, x_{n-}}{f_{U_n|X_{n-1}}(u_n|x_{n-1}) g_n(v_n)}$$

- Now consider the case where the V_n component of the state space is perfectly observed, i.e., $Y_n = V_n$. In this case,

$$f_{Y_n|V_n}(y_n|v_n) = \delta(y_n - v_n),$$

  interpreted as a point mass at v_n in the discrete case and a singular density at v_n in the continuous case.
- We can choose $g_n(v_n)$ to depend on the data, and a natural choice is

$$g_n(v_n) = \delta(y_n - v_n),$$

  for which the proper weight is

$$w_n(x_n|x_{n-1}) = f_{Y_n|U_n,X_{n-1}}(y_n|u_n, x_{n-1}).$$

- This is the situation in the context of our case study, with $U_n = (G_n, H_n)$ and $V_n = Y_n$.

Acknowledgments and License

- Produced with R version 3.6.2 and `pomp` version 2.3.
- These notes build on previous versions at `ionides.github.io/531w16` and `ionides.github.io/531w18`.

References I

Bretó, C. (2014). On idiosyncratic stochasticity of financial leverage effects, *Statistics & Probability Letters* **91**: 20–26.
URL: *http://www.sciencedirect.com/science/article/pii/S0167715214001345*

Cowpertwait, P. S. and Metcalfe, A. V. (2009). *Introductory time series with R*, Springer Science & Business Media.

Liu, J. S. and Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems, *Journal of the American Statistical Association* **93**: 1032–1044.