

RATS Programs for Dynamic Models of Infections

Thomas A. Doan
Estima

Draft
April 21, 2023

Contents

Preface	ii
1 Simulation Models	1
1.1 The SEIR Model	1
1.2 SEIR With Errors	3
1.3 With (Deterministic) Time-Varying beta	7
1.1 Fixed beta, No Errors	12
1.2 Fixed beta, With Errors	13
1.3 Varying beta, No Errors	15
2 Dealing with Data	17
2.1 Day of the Week Cycles with Exponential Smoothing	18
2.2 Multiplicative State-Space Model	21
2.1 Weekly Cycles by Smoothing	27
2.2 Weekly Cycles by Unobserved Components	27
3 SEIR Model for Estimation	30
3.1 With Fixed Beta	30
3.2 With Deterministic Time-Varying Beta	36
3.3 With Endogenous Beta	38
3.4 With Endogenous Beta and Day-of-Week Effects	43
3.1 State-Space Model with Fixed Beta	51
3.2 State-Space Model with Deterministic Time-Varying Beta	53
3.3 State-Space Model with Endogenous Time-Varying Beta	55
3.4 Model with Endogenous Beta and Day-of-Week Effects	58
Index	62

Preface

This examines the use of RATS for analyzing time series models of epidemics. The first chapter looks at (non-data) simulation methods which are done primarily using the **FORECAST** and **SIMULATE** instructions applied to non-linear models. The second chapter looks at different ways to handle the characteristics of the observed data. The third chapter uses state-space modeling techniques to estimate the unobservable data (such as the number of people currently infectious) given observables. This is done using the **DLM** instruction, and requires use of the various forms of the non-linear (“extended”) Kalman filter.

We use bold-faced Courier (for instance, **DLM**) for any use of RATS instruction names within the main text, and non-bolded Courier (%SCALAR) for any other pieces of code, such as function and variable names. For easy reference, the full text of each example is included. The running examples as separate files are also available.

Simulation Models

This chapter deals with pure simulation models, examining the behavior of an epidemic without attempting to fit it directly to data. This demonstrates both purely deterministic models (Section 1.1) and stochastic models (Section 1.2) and also looks at the effect of changes in the rate of transmission of infection due to either voluntary or enforced behavior changes (Section 1.3).

1.1 The SEIR Model

The basic dynamic model for studying epidemics is called SEIR. The population is partitioned into four groups:

- S represents the *susceptible* population (those who could still become infected)
- E are the *exposed* population (those who have been exposed but are not yet infectious)
- I is the *infectious* population (those currently capable of infecting others)
- R are the *recovered* (alternatively, *removed*) (those assumed to be no longer either infectious or susceptible).

The basic equations governing these¹ are:

$$E_t = (1 - \sigma)E_{t-1} + \beta I_{t-1}S_{t-1}/N \quad (1.1)$$

$$I_t = (1 - \gamma)I_{t-1} + \sigma E_{t-1} \quad (1.2)$$

$$R_t = R_{t-1} + \gamma I_{t-1} \quad (1.3)$$

Susceptible people are potentially exposed to infectious people in the final term of (1.1)—the β parameter governs how easily infectious people can expose others. The only non-linearity in the model is in that final term. A person who is exposed will eventually become infectious—the σ parameter governs how quickly that happens. And infectious people recover—the rate of that transition is governed by γ .

¹These are usually written as differential equations, but this is the feasible implementation using discrete time series data.

At the start, when (almost) the entire population is still susceptible, S_{t-1}/N is (effectively) 1, so the system becomes fully linear. If $\beta > \gamma$, the system is explosive and sees exponential growth (at least at first). If $\beta < \gamma$, the system is stable and the infection eventually burns out. The ratio of β to γ is known as R_0 (pronounced “R-naught”) so $R_0 > 1$ means the infection is explosive and $R_0 < 1$ means it is under control— β is the number of people (per day) that an infectious person exposes and $1/\gamma$ is the (average) number of days a person would be infectious.

It’s important to note that the dynamics of the system aren’t controlled by R_0 as much as by the difference (rather than the ratio) of β and γ . The largest eigenvalue of the linear system is approximately

$$1 + \frac{\sigma}{\sigma + \gamma} (\beta - \gamma)$$

For a given value of R_0 , the ones with larger values of γ (thus shorter infectious periods) will have a more extreme exponent—more rapid exponential growth when $R_0 > 1$ and more rapid decay when $R_0 < 1$.

As noted, this version has three parameters that govern the behavior of the process. However, trying to estimate these using time series data is extremely difficult. First of all, E is effectively unobservable. Even I is only observable if there is a really massive testing regime—note that it is supposed to represent the people who are *infectious* (currently), not the people who have been *infected*, so it would require repeated testing for active infection of people until they are recovered. In practice, both γ and σ are estimated using cross-sectional data, looking at patient histories. This leaves β , but β is the one part of the model which depends upon the behavior of humans, rather than just the pathogen. For instance, if you have a perfect quarantine, β will be zero, regardless of how infectious the disease itself is, because you will have no meetings of infectious people with susceptible ones. In practice, this means that β will be time-varying due to changes in behavior and government policies.²

The SEIR model is usually simulated deterministically. This is relatively easily done with RATS using the **FRML** and **GROUP** instructions to put together the model, and **FORECAST** to solve it. You just have to make sure to give initial values to the four variables for period 1 (which requires at least one person to be either in category I or E), and do the forecasts from period 2 on. This is part of example program `seir_wo_errors.rpf`. Note that the series are all named with double letters, which is done to avoid the reserved name conflict with I .

² σ and γ in all the models come from the settings using in the MIT model: https://www.covidanalytics.io/projections_documentation

```

compute pop=1000
compute nper=200
all nper
*
set ss = pop-1
set ee = 1.0
set ii = 0.0
set rr = 0.0

*
frml ssf ss = ss{1}-beta*ss{1}*ii{1}/pop
frml eef ee = ee{1}+beta*ss{1}*ii{1}/pop-sigma*ee{1}
frml iif ii = ii{1}+sigma*ee{1}-gamma*ii{1}
frml rrf rr = rr{1}+gamma*ii{1}
*
group seir ssf>>ss eef>>ee iif>>ii rrf>>rr
forecast(model=seir,from=2,to=nper)

```

That program then computes the point at which the rate at the infection is transferred falls below 1 as a result of the decline in the susceptible population—the infection will eventually die out once that point is reached, though a considerable number of people may still be infected after that point.

```

set herd = (beta*ss{1}/pop) < gamma
*
graph(key=below,shade=herd,$
  klabels=||"Susceptible","Exposed","Infectious","Recovered"||) 4
# ss
# ee
# ii
# rr

```

1.2 SEIR With Errors

A non-stochastic model may be reasonable for some uses, but not in a situation where the number of infectious people is relatively small. The three transitions in the model (from S to E , from E to I and from I to R) are (in theory) probabilistic and when the expected number of people making those transitions is relatively small, the randomness can swamp the signals which indicate the growth (or decay) of the infection. The model is more easily analyzed by adding state variables which directly model those transitions, so the expanded model becomes

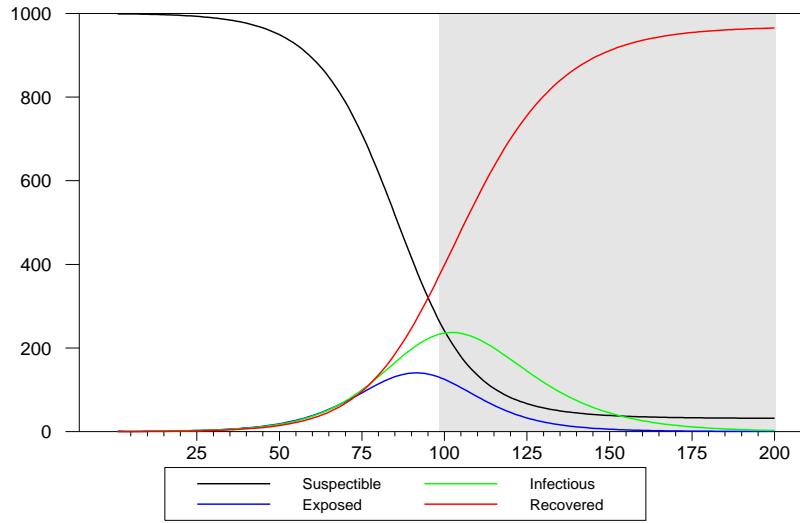


Figure 1.1: SEIR Non-Stochastic Simulation

$$\begin{aligned}
 SE_t &= \beta I_{t-1} S_{t-1} / N \\
 EI_t &= (1 - \sigma) E_{t-1} \\
 IR_t &= (1 - \gamma) I_{t-1} \\
 E_t &\equiv E_{t-1} + SE_t - EI_t \\
 I_t &\equiv I_{t-1} + EI_t - IR_t \\
 R_t &\equiv R_{t-1} + IR_t
 \end{aligned} \tag{1.4}$$

The probability for a person to make the S to E transition at time t is

$$p_t \equiv \beta I_{t-1} / N$$

If we look at this as a binomial process applied to a population of size S_{t-1} , it has expected value $S_{t-1}p_t$ (which is what is used in the deterministic model) with variance $S_{t-1}p_t(1 - p_t)$. If p_t is small, the variance is roughly equal to the mean. When for instance, the expected value is 100, the standard error is 10, so even a 10% day-to-day growth can get masked by a low, but not unreasonable, random value.

The program `seir_with_errors.rpf` simulates the process using Normal approximations to the binomials. (It uses a bigger population to make that a bit more realistic). Because the variances are time-varying (depending as they do on the population counts for the category in question, which change from simulation to simulation), the random shocks are to placeholder equations:

```

frml seuf seu = 0.0
frml eiuf eiu = 0.0
frml iruf iru = 0.0

```

The work of computing the draws for the transition is done with these three, which take the $N(0,1)$ draws for the SEU, EIU and IRU variables and convert them into the approximate binomial draws:

```
frml(identity) sef se = p=beta*ii{1}/pop,p*ss{1}+sqrt(ss{1}*p*(1-p))*seu
frml(identity) eif ei = p=sigma,p*ee{1}+sqrt(ee{1}*p*(1-p))*eiu
frml(identity) irf ir = p=gamma,p*ii{1}+sqrt(ii{1}*p*(1-p))*iru
```

Note that these will be continuously valued reals, rather than integers, but that should make little difference in practice. (And, of course, the non-stochastic simulations are similarly continuously valued reals).

The conversion of the transition states to the SEIR variables includes a floor at 0, just in case a wild draw produces (for instance) E going negative (unlikely except in the first few periods).

```
frml(identity) ssf ss = %max(0.0,ss{1}-se{0})
frml(identity) eef ee = %max(0.0,ee{1}-ei{0}+se{0})
frml(identity) iif ii = %max(0.0,ii{1}+ei{0}-ir{0})
frml(identity) rrf rr = %max(0.0,rr{1}+ir{0})
```

The full model includes the ten equations above, with a 3×3 identity matrix for the first three equations that take the $N(0,1)$ shocks:

```
group(cv=%identity(3)) seir_with_errors seuf eiuf iruf sef eif irf $
ssf>>ss eef>>ee iif>>ii rrf>>rr
```

The program then does NDRAWS (set to 1000) simulations of the model, saving the full simulated history of the model for each draw. Note, again, that this needs to start at period 2 since period 1 is needed for the initial values.

```
dec vect[series] iie(ndraws) eee(ndraws) ree(ndraws)
*
do draw=1,ndraws
  simulate(model=seir_with_errors,from=2,to=nper)
  set iie(draw) = ii(t)
  set eee(draw) = ee(t)
  set ree(draw) = rr(t)
end do draws
```

The program uses these in two ways. First, it does a fan chart showing the median and 95% confidence interval (from the .025 to the .975 quantiles at each time period).

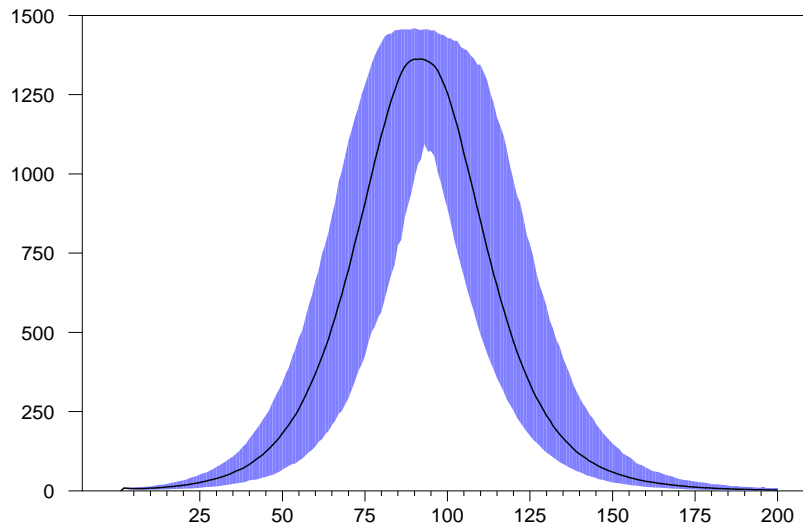


Figure 1.2: Fan Chart for Exposed Population

```
clear(zeros) ii025 iimed ii975
clear(zeros) ee025 eemed ee975
do t=2,nper
  *
  * Get the median, .025 and .975 percentiles
  *
  compute iit=%xt(iie,t)
  compute iiq=%fractiles(iit,||.025,.50,.975||)
  compute ii025(t)=iiq(1)
  compute iimed(t)=iiq(2)
  compute ii975(t)=iiq(3)
  *
  compute eet=%xt(eee,t)
  compute eeq=%fractiles(eet,||.025,.50,.975||)
  compute ee025(t)=eeq(1)
  compute eemed(t)=eeq(2)
  compute ee975(t)=eeq(3)
end do t
```

This does the graph (Figure 1.2) for the exposed population.

```
graph(style=line,overlay=fan,ovcount=2,ovsamescale,$
  footer="Exposed Population") 3
# eemed
# ee025
# ee975
```

The second is a “shot-gun” graph (Figure 1.3) which shows all the simulations.

```
graph(series=ree,footer="Recovered Population-Shotgun Graph")
```

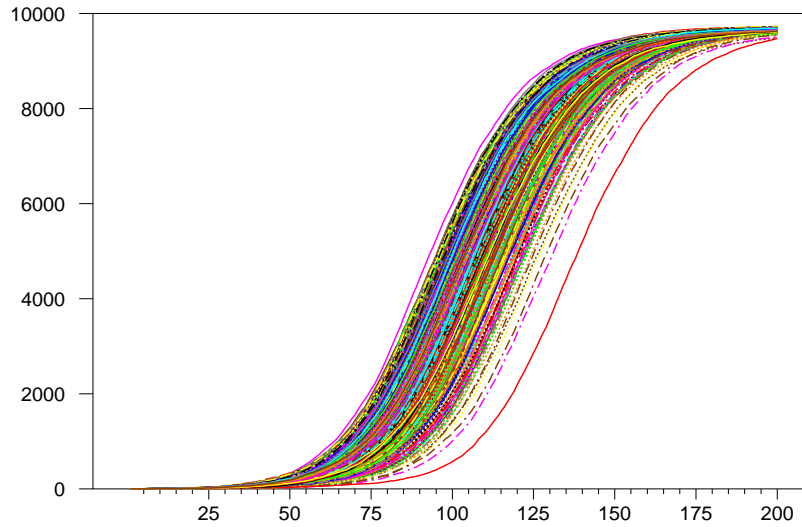


Figure 1.3: Shotgun Graph of Recovered Population

Note that almost all the simulations end up with the same pattern, but they are spread across 30-40 days on the timing of the eventual significant upward movement.

1.3 With (Deterministic) Time-Varying beta

It's expected that if the contagion is sufficiently dangerous there will be a combination of voluntary and government-enforced behavioral changes that will reduce the rate at which the disease is spread. This can be incorporated into the model using a time-varying function of t , β_t , to replace the fixed β parameter. A major problem with using this for projecting the course of the epidemic is that the form of β_t is at best an educated guess. It would appear that the typical assumption needs to specify a target (lower) level and some parameter which governs the speed of the transition to that target. A functional form which is relatively easy to handle for this is based upon the Gaussian density. The function

$$1 - \exp \left(- \left(\frac{t - t_0}{d} \right)^2 \right) \quad (1.5)$$

has value 0 at $t = t_0$, 1 at $t = \infty$, and .5 at (roughly) $t_0 + .85d$. This function also has a zero derivative at $t = t_0$ and zero derivative asymptotically. Using this

$$\beta_0 \left(1 + (g - 1) \left(1 - \exp \left(- \left(\frac{\max(t - t_0, 0)}{d} \right)^2 \right) \right) \right) \quad (1.6)$$

will have value β_0 until $t = t_0$, and will then transition to an asymptotic value of $\beta_0 g$, with the d parameter governing the rate of transition. It's relatively simple to splice multiple cases of this together to get transitions to different

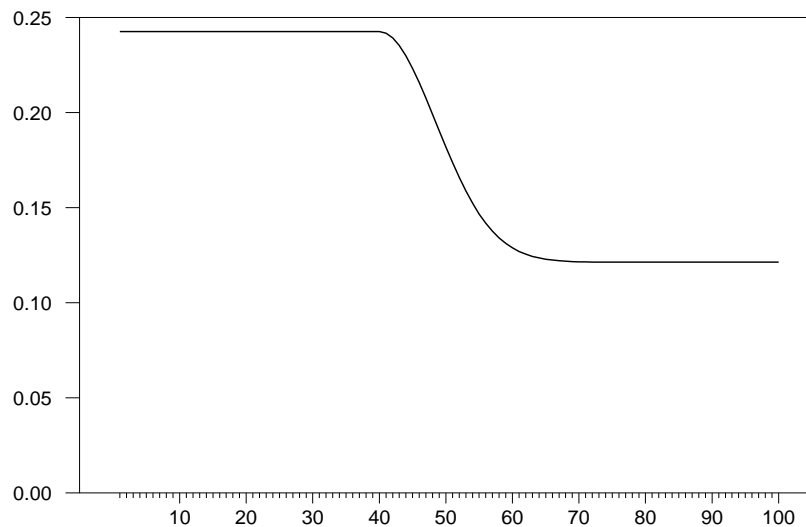


Figure 1.4: Transition Function

target levels as time goes on (such as a tighter initial response followed by loosening).

The program `seir_with_tvbeta.rpf` (Example 1.3) adds a deterministic time-varying β to the non-stochastic model from Section 1.1. It uses a population of 100000 with a initial exposed population of 200 (which is chosen to reduce the length of time before the infection takes hold).

The controls for the transition are $t_0 = 40$, $d = 12$ (so about 10 days for the half-transition), with $g = .5$. The resulting function is shown in Figure 1.4.

```
compute d0=12.0,t0=40,maxg=.50
set betat = beta*(1+(maxg-1)*(1-exp(-(t-t0)/d0)^2))
```

The only change to the model itself is to replace the `SSF` and `EEF` formulas to use `BETAT{0}` rather than `BETA`:

```
frml ssf ss = ss{1}-betat{0}*ss{1}*ii{1}/pop
frml eef ee = ee{1}+betat{0}*ss{1}*ii{1}/pop-sigma*ee{1}
```

After solving the model, this computes and saves the implied number of new cases and the number of people who have recovered:

```
set ccl = ss{1}-ss
set rrl = rr
```

The same calculations are done using the model with the original high value of β by resetting the `BETAT` series and solving the model again:

```
set betat = beta
forecast (model=seir, from=2, to=nper)
```

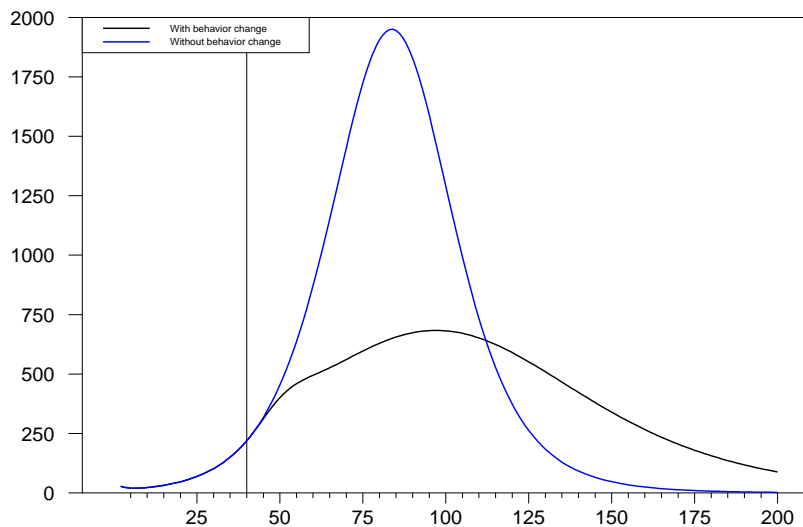


Figure 1.5: New Cases with Behavior Change

Graphs of the new cases (Figure 1.5) and the number of recoveries (Figure 1.6) are done comparing the progress with and without change:

```
graph(footer="New Cases",grid=(t==t0),key=upleft,$
      klabels=||"With behavior change","Without behavior change"||) 2
# cc1
# cc0
*
graph(footer="Recovered",grid=(t==t0),key=upleft,$
      klabels=||"With behavior change","Without behavior change"||) 2
# rr1
# rr0
```

Figure 1.5 demonstrates what has been known as “bending the curve” (reducing the peak number of new infections, while increasing the length of time until the peak is reached), though from Figure 1.6, this also reduces the overall number of people who get infected before the contagion is contained.

The example then adds in a second branch where β makes a transition to a middle level (due to relaxation of the original restrictions). The parameters governing this are:

```
compute d2=14.0,t2=80,maxg2=.70
```

so a transition is made beginning at $t = 80$ up to the level which is .70 times the original β . The spliced β function is computed and graphed (Figure 1.7) with

```
set betat = beta*(1+(maxg-1)*(1-exp(-(t-t0)/d0)^2))+$
            (maxg2-maxg)*(1-exp(-(t-t2)/d2)^2))
graph(min=0.0,footer="Spliced deterministic beta")
# betat
```

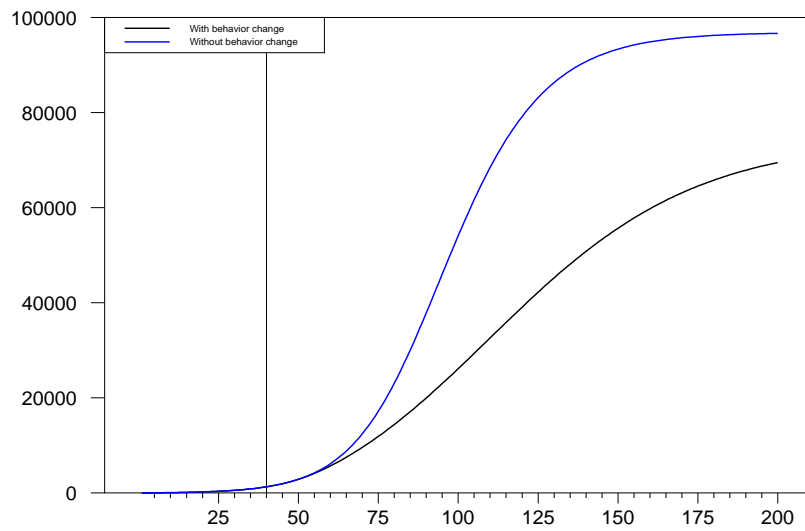


Figure 1.6: Recoveries with Behavior Change

The resulting change to the new case measure is shown in Figure 1.8.

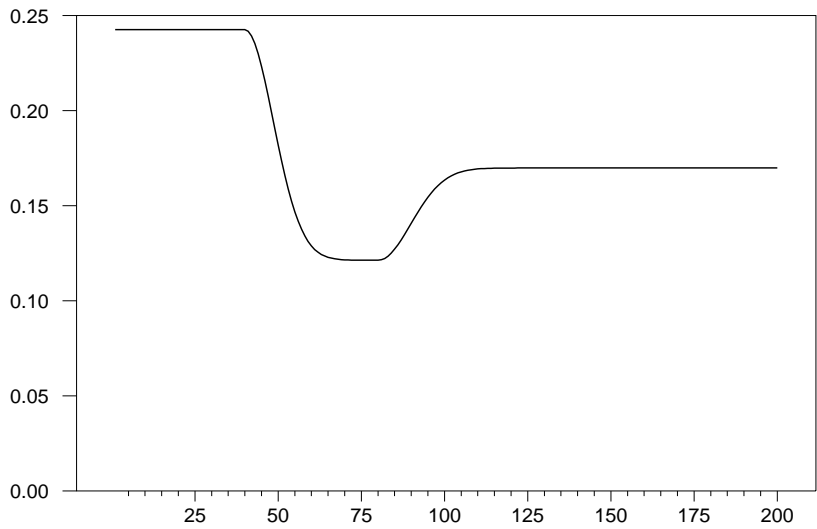


Figure 1.7: Deterministic Beta with Two Branches

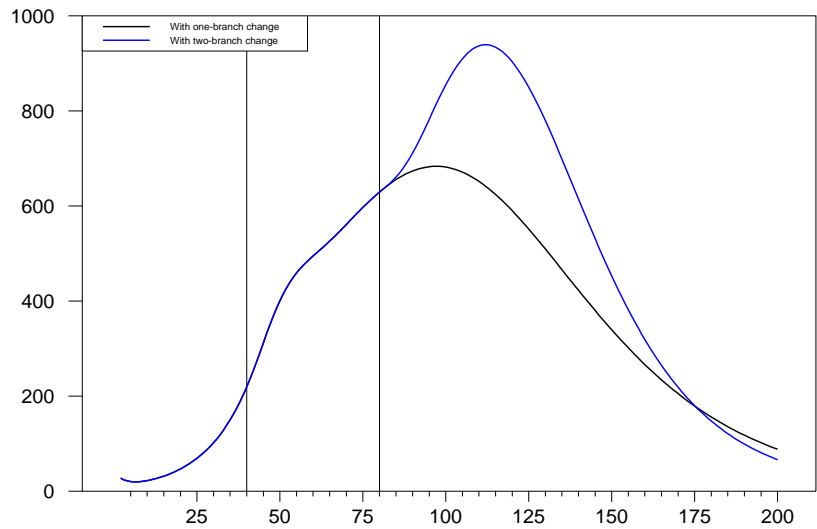


Figure 1.8: Effect on New Cases of Two Branches

Example 1.1 Fixed beta, No Errors

This is the `seir_wo_errors.rpf` program, which does a non-stochastic simulation of the SEIR model.

```
dec series ss ;* Susceptible
dec series ee ;* Exposed
dec series ii ;* Infectious
dec series rr ;* Recovered
*
* Control parameters
* sigma and gamma come from the settings using in the MIT model:
* https://www.covidanalytics.io/projections\_documentation
*
* beta is fixed from gamma to give an R0 of 3.5
*
compute sigma=log(2)/5.0 ;* Exponential distribution with 5 day median
compute gamma=log(2)/10.0 ;* Exponential distribution with 10 day median
compute beta=3.5*gamma ;* Transmission rate from infectious to susceptible
disp "R0=" beta/gamma
*
* This starts with one individual who has been exposed out of the
* population.
*
compute pop=1000
compute nper=200
all nper
*
set ss = pop-1
set ee = 1.0
set ii = 0.0
set rr = 0.0
*
frml ssf ss = ss{1}-beta*ss{1}*ii{1}/pop
frml eef ee = ee{1}+beta*ss{1}*ii{1}/pop-sigma*ee{1}
frml iif ii = ii{1}+sigma*ee{1}-gamma*ii{1}
frml rrf rr = rr{1}+gamma*ii{1}
*
group seir ssf>>ss eef>>ee iif>>ii rrf>>rr
forecast (model=seir, from=2, to=nper)
*
* Determine the point at which "herd immunity" takes effect (the
* percentage of susceptible people drops low enough that each infectious
* person would be expected to expose less than one person).
*
set herd = (beta*ss{1}/pop) < gamma
*
graph (key=below, shade=herd, $
  klabels=|| "Susceptible", "Exposed", "Infectious", "Recovered" ||) 4
# ss
# ee
# ii
# rr
```

Example 1.2 Fixed beta, With Errors

This is the `seir_with_errors.rpf` example, which simulates the SEIR model with errors drawn from a Normal approximation to the implied binomial distributions.

```
compute ndraws=1000
*
dec series ss ;* Susceptible
dec series ee ;* Exposed
dec series ii ;* Infectious
dec series rr ;* Recovered
*
* Control parameters
* sigma and gamma come from the settings using in the MIT model:
* https://www.covidanalytics.io/projections\_documentation
*
* beta is fixed from gamma to give an R0 of 3.5
*
compute sigma=log(2)/5.0 ;* Exponential distribution with 5 day median
compute gamma=log(2)/10.0 ;* Exponential distribution with 10 day median
compute beta=3.5*gamma ;* Transmission rate from infectious to susceptible
disp "R0=" beta/gamma
*
* This starts with ten individuals exposed out of the population.
*
compute pop=10000
compute nper=200
all nper
*
* There are three (probabilistic) transitions:
* From S to E: probability beta*i(t-1)/pop, applied to S(t-1) individuals
* From E to I: probability sigma, applied to E(t-1) individuals
* From I to R: probability gamma, applied to I(t-1) individuals
*
clear(zeros) se ei ir ss ee ii rr seu eiu iru
*
set ss = pop-10.0
set ee = 10.0
set ii = 0.0
set rr = 0.0
*
* These get the N(0,1) shocks
*
frml seuf seu = 0.0
frml eiuf eiu = 0.0
frml iruf iru = 0.0
*
* These are the transitions (using the variances from the binomial
* process).
*
frml(identity) sef se = p=beta*ii{1}/pop,p*ss{1}+sqrt(ss{1}*p*(1-p))*seu
frml(identity) eif ei = p=sigma,p*ee{1}+sqrt(ee{1}*p*(1-p))*eiu
```



```

frml(identity) irf ir = p=gamma,0.0,p*ii{1}+sqrt(ii{1}*p*(1-p))*iru
*
* Don't let any go negative
*
frml(identity) ssf ss = %max(0.0,ss{1}-se{0})
frml(identity) eef ee = %max(0.0,ee{1}-ei{0}+se{0})
frml(identity) iif ii = %max(0.0,ii{1}+ei{0}-ir{0})
frml(identity) rrf rr = %max(0.0,rr{1}+ir{0})
*
group(cv=%identity(3)) seir_with_errors seuf eiuf iruf sef eif irf $
    ssf>>ss eef>>ee iif>>ii rrf>>rr
*
dec vect[series] iie(ndraws) eee(ndraws) ree(ndraws)
*
do draw=1,ndraws
    simulate(model=seir_with_errors,from=2,to=nper)
    set iie(draw) = ii(t)
    set eee(draw) = ee(t)
    set ree(draw) = rr(t)
end do draws
*
clear(zeros) ii025 iimed ii975
clear(zeros) ee025 eemed ee975
do t=2,nper
    *
    * Get the median, .025 and .975 percentiles
    *
    compute iit=%xt(iie,t)
    compute iiq=%fractiles(iit,||.025,.50,.975||)
    compute ii025(t)=iiq(1)
    compute iimed(t)=iiq(2)
    compute ii975(t)=iiq(3)
    *
    compute eet=%xt(eee,t)
    compute eeq=%fractiles(eet,||.025,.50,.975||)
    compute ee025(t)=eeq(1)
    compute eemed(t)=eeq(2)
    compute ee975(t)=eeq(3)
end do t
*
graph(style=line,overlay=fan,ovcount=2,ovsamescale,$
    footer="Exposed Population") 3
# eemed
# ee025
# ee975
graph(series=ree,footer="Recovered Population-Shotgun Graph")

```

Example 1.3 Varying beta, No Errors

This is the `seir_with_tvbeta.rpf` example, which does a solution of the SEIR model with a deterministic time-varying beta.

```
dec series ss ;* Susceptible
dec series ee ;* Exposed
dec series ii ;* Infectious
dec series rr ;* Recovered
*
* Control parameters
* sigma and gamma come from the settings using in the MIT model:
* https://www.covidanalytics.io/projections_documentation
*
* beta is fixed from gamma to give an R0 of 3.5
*
compute sigma=log(2)/5.0 ;* Exponential distribution with 5 day median
compute gamma=log(2)/10.0 ;* Exponential distribution with 10 day median
compute beta=3.5*gamma ;* Transmission rate from infectious to susceptible
disp "R0=" beta/gamma
*
* This uses a population of 100000 to make it simpler to look at this
* using the standard epidemiological ratios of numbers per 100K.
*
compute pop=100000
compute nper=200
all nper
*
set ss = pop-200
set ee = 200.0
set ii = 0.0
set rr = 0.0
*
* This does a declining beta using a half-Gaussian density function
* shape starting at t0, with an asymptotic fraction of beta given by
* maxg.
*
compute d0=12.0,t0=40,maxg=.50
set betat = beta*(1+(maxg-1)*(1-exp(-(t-t0)/d0)^2)))
graph(min=0.0,footer="Deterministic Time-varying beta")
# betat 1 100
*
frml ssf ss = ss{1}-betat{0}*ss{1}*ii{1}/pop
frml eef ee = ee{1}+betat{0}*ss{1}*ii{1}/pop-sigma*ee{1}
frml iif ii = ii{1}+sigma*ee{1}-gamma*ii{1}
frml rrf rr = rr{1}+gamma*ii{1}
*
group seir ssf>>ss eef>>ee iif>>ii rrf>>rr
*
* Solve with the time-varying beta
*
forecast(model=seir,from=2,to=nper)
*
```

```

* Compute and save the implied number of new infections and the number of
* recovered.
*
set cc1 = sigma*ee{1}
set rr1 = rr
*
* Redo the calculations with the flat beta
*
set betat = beta
forecast(model=seir,from=2,to=nper)
*
* Compute and save the same statistics
*
set cc0 = sigma*ee{1}
set rr0 = rr
*
graph(footer="New Cases",grid=(t==t0),key=upleft,$
      klabels=||"With behavior change","Without behavior change"||) 2
# cc1
# cc0
*
graph(footer="Recovered",grid=(t==t0),key=upleft,$
      klabels=||"With behavior change","Without behavior change"||) 2
# rr1
# rr0
*
* With secondary increase at t=80.
*
compute d2=14.0,t2=80,maxg2=.70
set betat = beta*(1+(maxg-1)*(1-exp(-(plus(t-t0)/d0)^2))+$(
      (maxg2-maxg)*(1-exp(-(plus(t-t2)/d2)^2)))
graph(min=0.0,footer="Spliced deterministic beta")
# betat
*
* Redo the calculations with the spliced beta
*
forecast(model=seir,from=2,to=nper)
*
* Compute and save the same statistics
*
set cc2 = ee{1}
set rr2 = rr
*
graph(footer="New Cases",grid=(t==t0).or.(t==t2),key=upleft,$
      klabels=||"With one-branch change","With two-branch change"||) 2
# cc1
# cc2

```

Dealing with Data

As mentioned at the beginning, σ and γ parameters are generally estimated from patient histories and really can't be sensibly estimated from time series data, as they measure the transitions between (effectively) unobservable states. The observables are typically

1. Number of tests
2. Number of diagnosed cases
3. Hospitalization (new and/or continuing)
4. Deaths
5. Recoveries
6. Positive antibody tests

Not all countries or regions routinely report all of these—in the U.S. the antibody tests aren't, at this point, done in a systematic fashion, and some states do not test for recoveries (by negative virus test) but list a recovery as someone who is neither dead nor hospitalized after a certain length of time after diagnosis.¹

All of these can be subject to considerable uncertainty, particularly with respect to timing. For instance, different states in the U.S. have very different day-of-the-week patterns in the statistics—tests and diagnosed cases (which are generally reported together) can be batched in some jurisdictions and reported primarily on certain days. Even deaths, which one would assume would be fairly evenly spread, don't necessarily get reported that way.

A common method that has been used in reporting statistics subject to day-of-week fluctuations is to report seven-day averages. This *does* level out the seasonality, but makes it more difficult to spot trends that are less than a week

¹For instance, from the Illinois Department of Public Health (<https://www.dph.illinois.gov/covid19/covid19-statistics>), “Recovered cases are defined as persons with initial positive specimen collection date >42 days who have not expired.”

old.² However, there is really no standard seasonal adjustment algorithm that is designed for the combination of wildly varying non-negative levels (starting from zero, possibly growing rapidly to large numbers, then (one hopes) declining back to near zero) and a seasonal which changes (in some way) with the magnitude of the data.

2.1 Day of the Week Cycles with Exponential Smoothing

Example 2.1 looks at time series data for the State of Illinois through 14 December 2020. The data start on 10 March, though there is relatively little to see before 30 March.³

The data provided are all cumulative, so they are translated into daily changes for further analysis:

```
set(first=0.0) new_tests = total_tests-total_tests{1}
set(first=0.0) new_cases = confirmed-confirmed{1}
set(first=0.0) new_deaths = deaths-deaths{1}
```

The simple 7-day moving average can be computed using the **FILTER** instruction with the options `TYPE=LAGGING` and `WIDTH=7`. The raw data and smoothed values are shown in Figure 2.1.

```
filter(type=lagging,width=7) new_cases / cases_7day
graph(footer="New Cases with 7-day Average", $
  key=upleft,klabels=|| "Actual", "Smoothed" ||) 2
# new_cases
# cases_7day
```

A somewhat *ad hoc* but relatively simple procedure to “seasonally” adjust the data is to do exponential smoothing on the square root of the data, with an additive seasonal.⁴ The advantage of this over the simple smoothing is that it can predict the seasonal factor for coming days, so it’s possible to determine how much of a day-to-day change is new information versus just the repetition of a day-of-week pattern. The “smoothed” data (that is, the data with the day-of-week pattern removed) can be obtained by squaring the output series from **ESMOOTH**.

²The difference between today’s seven-day average and yesterday’s collapses to $(1/7)(y_t - y_{t-7})$ which, when y can be subject to fairly considerable variance, is not an especially informative statistic. Today’s seven-day average and the same average from a week earlier is likely to be a more useful number, but changes rather slowly.

³These are extracted from the `state_testing_results` array out of <https://www.dph.illinois.gov/sitefiles/COVIDTestResults.json>.

⁴Multiplicative seasonality won’t work because the smoothed level of the data can be zero (or is effectively zero for at least the early part of the sample). Taking the square root of the data largely flattens out the fluctuations, and is suggested by the behavior of the errors in Section 1.2.

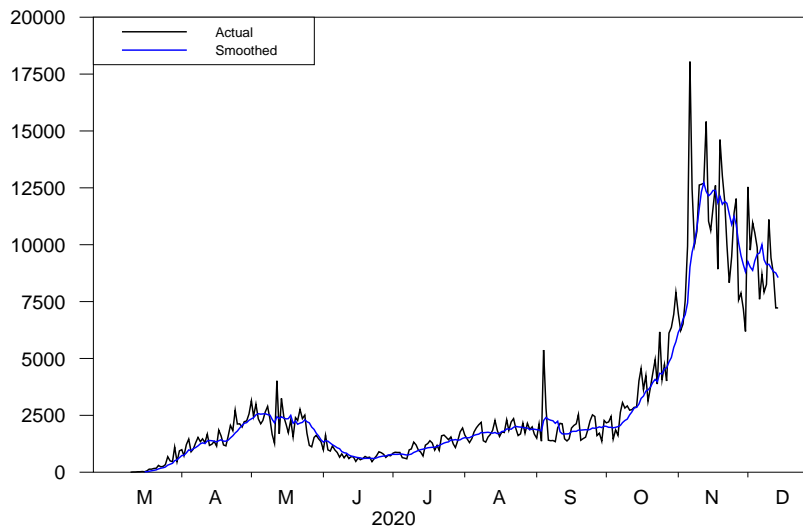


Figure 2.1: New Cases with 7-day Average

```
set sqrt_new = sqrt(new_cases)
esmooth(trend=linear, seasonal=additive, span=7, estimate, $
  factors=sf, smoothed=smsqrt, initial=start) sqrt_new
set esmoothed = smsqrt^2
graph(footer="New Cases with Exponential Smoothing", $
  key=upleft, klabels=|| "Actual", "Smoothed" ||) 2
# new_cases
# esmoothed
```

A couple of notes on the **ESMOOTH** instruction: this uses **TREND=LINEAR** even though the series has no consistent trend—it trends up at the beginning and largely down after the middle of May and then rises rapidly in October. However, the trending exponential smoothing model allows for a moving trend rate, which can include the sign of the trend rate changing. The **INITIAL=START** option is used so that only the data at the start of the sample (nine elements are needed) are used to initialize the recursion.

Figure 2.2 shows the data with the output from the smoothing procedure and Figure 2.3 does a comparison of the exponential smoothed output and the simple smoothing.

The daily “factors” (additive adjustments to the square root) are shown in Figure 2.4.

```
graph(footer="Seasonal Factor (Mondays highlighted)", $
  grid=%weekday(t)==1)
# sf
```

Pretty much throughout the range where the data values are non-trivial, the Mondays are the lowest values (by a substantial margin) and Fridays the highest. The predicted daily adjustments out-of-sample are the last values in sam-

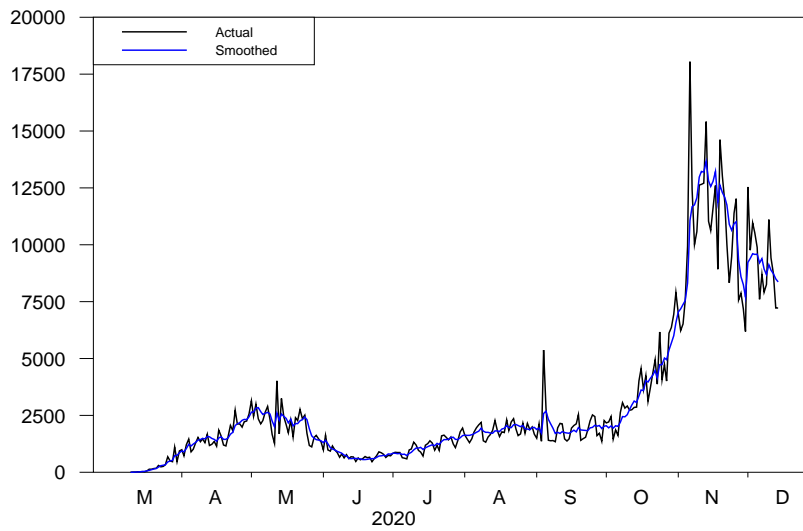


Figure 2.2: Data with ESmoothed Output

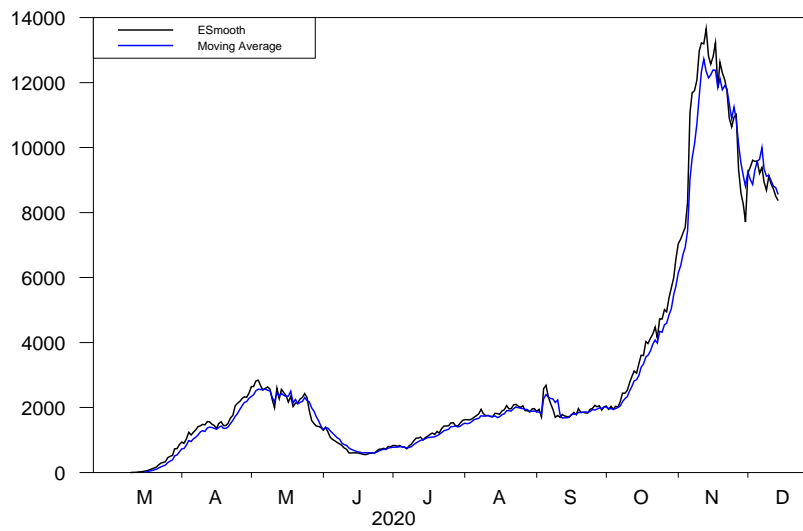


Figure 2.3: ESmoothed with Simple Average

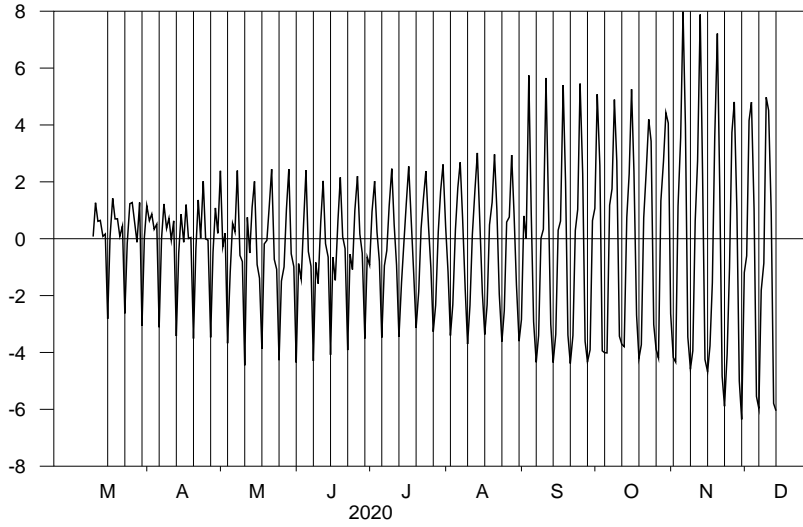


Figure 2.4: Daily Factors/Mondays Highlighted

ple, so with the baseline numbers at roughly 10000, you would expect a Monday to be at around $(\sqrt{10000} - 5)^2 \approx 9000$ and a Friday at $(\sqrt{10000} + 4)^2 \approx 10800$; an apparent roughly 20% difference.

2.2 Multiplicative State-Space Model

An alternative approach to decomposing the data to extract signal from the daily pattern is a state-space model with a multiplicative measurement equation.

The following non-linear state-space model can provide a decomposition of a series into an underlying level and a seasonal noise:

$$x_t = x_{t-1} + \varepsilon_{x,t} \quad (2.1)$$

$$d_t + d_{t-1} + \dots + d_{t-6} = 7 + \varepsilon_{d,t} \quad (2.2)$$

$$u_t = 1 + \varepsilon_{u,t} \quad (2.3)$$

$$y_t = x_t^2 d_t u_t \quad (2.4)$$

The model has the *square root* of the signal as a constant variance random walk.⁵ The daily factors in (2.2) average roughly 1—for the state-space model, this needs to be rearranged to put the lagged d on the right side.

The measurement equation (2.4) is non-linear, requiring use of a non-linear (extended) Kalman filter, linearizing that equation. The linearized measurement equation is

$$\begin{aligned} y_t &\approx \tilde{x}_t^2 \tilde{d}_t \tilde{u}_t + 2\tilde{x}_t \tilde{d}_t \tilde{u}_t (x_t - \tilde{x}_t) + \tilde{x}_t^2 \tilde{u}_t (d_t - \tilde{d}_t) + \tilde{x}_t^2 \tilde{d}_t (u_t - \tilde{u}_t) \\ &= -3\tilde{x}_t^2 \tilde{d}_t \tilde{u}_t + [2\tilde{x}_t \tilde{d}_t \tilde{u}_t, \tilde{x}_t^2 \tilde{u}_t, \tilde{x}_t^2 \tilde{d}_t] [x_t, d_t, u_t]' \end{aligned} \quad (2.5)$$

⁵There is obviously a minor complication with this that the sign of the x process isn't identified.

where \tilde{x}_t , \tilde{d}_t and \tilde{u}_t are the expansion points for the linearization. The most common choices for expansion points are the conditional expectation based upon $t - 1$, that is, $\tilde{x}_t = x_{t|t-1}$, $\tilde{d}_t = d_{t|t-1}$ and $\tilde{u}_t = u_{t|t-1}$

The notation for this (linearized) state-space model in RATS is:

$$\mathbf{X}_t = \mathbf{A}\mathbf{X}_{t-1} + \mathbf{Z} + \mathbf{F}\mathbf{W}_t \quad (2.6)$$

$$y_t = \mu_t + \mathbf{C}'\mathbf{X}_t \quad (2.7)$$

This eliminates any component which isn't required—for instance, all the shocks are included in the state equation (2.6), so there is no error term in the measurement equation (2.7). Other than being combined multiplicatively, this is a fairly classical unobserved components model. The system matrices of the three components are combined: by diagonal concatenation for the **A** and **F** matrices, vertical concatenation for **C** and **Z**. We call the three components `level`, `daily` and `irreg`. The level and daily components can be set up using the standard `@LocalDLM` and `@SeasonalDLM` procedures:

```
@LocalDLM(type=level,a=alevel,c=clevel,f=flevel)
compute [vector] zlevel=%zeros(%rows(alevel),1)
@SeasonalDLM(span=7,type=additive,a=adaily,c=cdaily,f=fdaily)
compute [vector] zdaily=7.0*unitv(%rows(adaily),1)
```

The only adjustment is the addition of a **Z** component which is 7 (to average one across a week) in the daily component.

The irregular is straightforward: the **A** component is zero, the others 1.

```
compute [rect] airreg=||0.0||
compute [vect] zirreg=||1.0||
compute [rect] firreg=||1.0||
compute [rect] cirreg=||1.0||
```

The **A**, **F** and **Z** matrices are all fixed, so we can just do the concatenations as described above:

```
compute a=alevel~\adaily~\airreg
compute f=flevel~\fdaily~\firreg
compute z=zlevel~~zdaily~~zirreg
```

C on the other hand, is a time-varying function of the expansion points, so it needs to be a `FRML[VECT]`. Note that the state-space model needs current and five lags of d to handle the dynamics of the seasonal—that's incorporated into `CDAILY`, which is a unit vector of size 6.

```
dec frml[vect] cf
clear(zeros) xtilde dtilde utilde
frml cf = clevel*2*xtilde*dtilde*utilde~~$
          cdaily*xtilde^2*utilde~~$
          cirreg*xtilde^2*dtilde
```

Finally, the MU component is also a FRML which depends upon the expansion points.

```
frml muf = -3.0*xtilde^2*dtilde*utilde
```

For initialization of the states, the initial value of the signal is clearly 0, though to avoid some numerical problems, it's given the slightly positive value of .01. The variance is set to 1 (to give it a non-zero value).

```
compute [vect] x0level=%fill(1,1,.01)
compute [symm] sx0level=%ones(1,1)
```

Without any prior information about what days will tend to be higher or lower, the daily weights start at the mean of 1, with a fairly high variance (.25 for a standard deviation of .5):

```
compute [vect] x0daily=%ones(6,1)
compute [symm] sx0daily=%diag(%fill(6,1,.25))
```

The irregular starts at its mean of 1, with a variance of 0. (Since it's serially uncorrelated, the pre-sample variance doesn't matter).

```
compute [vect] x0irreg=%ones(1,1)
compute [symm] sx0irreg=%zeros(1,1)
```

The variances of the shocks in the three components are unknown and need to be estimated. This sets up the parameter sets and gives them guess values:⁶

```
nonlin(parmset=varparms) swlevel swdaily swirreg swdaily>=0.0
compute swdaily=.01
compute swlevel=1
compute swirreg=1.0
```

The one remaining step is to define the three `tilde` series. This will be done by using the `FPRE` option on `DLM`.⁷ `FPRE` calls a function which takes the filtered state vector and covariance matrix given the data through time period $T-1$ as arguments. We want to choose as the expansions the predicted values for the states at T given $T-1$. For `XTILDE` and `UTILDE`, those are easy because x is a random walk, so $t|t-1$ is just the same as $t-1$ and u is serially uncorrelated so $t|t-1$ is just the unconditional mean of 1. To avoid some problems at the start with zero derivatives, `XTILDE` is not allowed to be less than 1. $t|t-1$ for `D` is 7 minus the sum of the other day factors. The value of the variable `T` is equal to the current entry being analyzed, so the function puts the computed

⁶This includes a restriction which bounds the daily shock variance as non-negative, which turns out to be necessary.

⁷The `FPRE`, `FPOST` and `FSPPOST` options were added to `DLM` with version 9.2. These allow interventions at various points in the Kalman filter/smoothing calculations.

values into entry T of the series. Because this gets computed before the C and MU frmls, the values from the current evaluation of the model will get used for the filter at time T .

```
function MakeTildes x sx
type vector *x
type symm    *sx
*
compute xtilde(t)=%max(1.0,x(1))
compute dtilde(t)=7-%sum(%xsubvec(x,2,7))
compute utilde(t)=1
end
```

The actual estimation is done using **DLM**.

```
dlim(y=new_cases,a=a,f=f,sw=%diag(||swlevel,swdaily,swirreg||),$
c=cf,mu=muf,z=z,x0=x0level~~x0daily~~x0irreg,$
sx0=sx0level~\sx0daily~\sx0irreg,fpre=MakeTildes,type=smooth,$
parmset=varparms,pmethod=simplex,piters=10,method=bfgs) / $
xstates vstates
```

Because we want actual pre-sample values for states, we need the $X0$ and $SX0$ options, which do a vertical concatenation of the $X0$ vectors and a diagonal concatenation of the $SX0$ covariance matrices.

The estimated states and covariance matrices are from Kalman smoothing.⁸ x is in slot 1, current d is in slot 2 and u is in slot 8 (allowing for the augmenting lags of d):

```
set xsmooth = xstates(t)(1)^2
set dsmooth = xstates(t)(2)
set usmooth = xstates(t)(8)
```

Figure 2.5 shows the smoothed estimate, and Figure 2.6 shows the factors.⁹ Unlike the exponential smoothing in Example 2.1, these are direct multiplicative factors, so 1.15 means a 15% scale up.

```
graph(footer="Smoothed Estimates from UCM") 2
# new_cases
# xsmooth
graph(footer="Seasonal Factors from UCM\Mondays highlighted",$
grid=%weekday(t)==1)
# dsmooth
```

⁸The parameter estimates are done using Kalman filtering to evaluate the likelihood; then Kalman smoothing is done using the converged values.

⁹The factors are effectively identical across the sample since the shock variance is roughly zero.

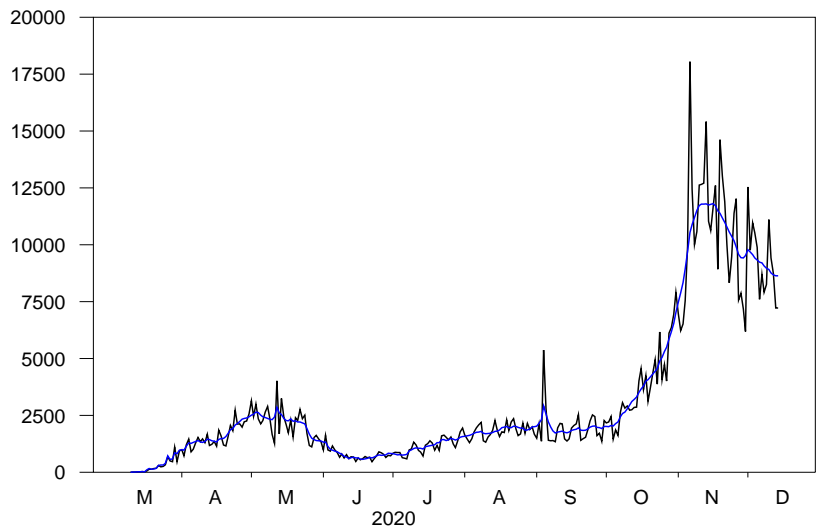


Figure 2.5: Smoothed Estimates from UCM Model

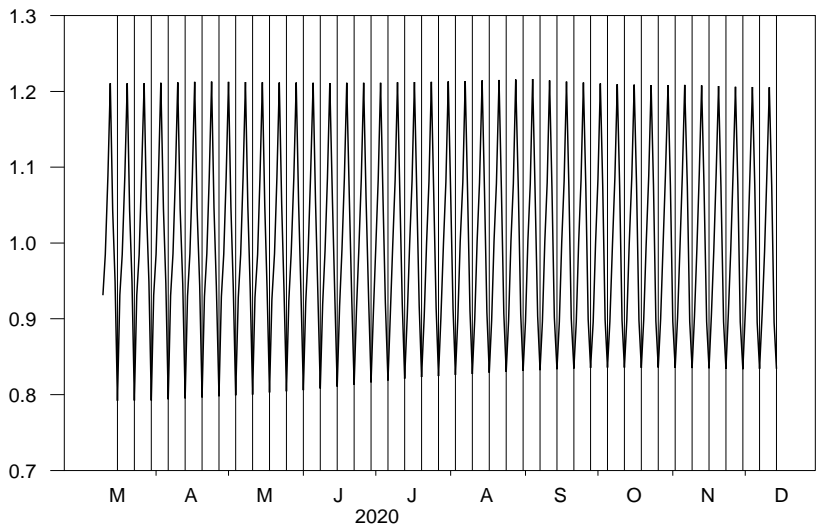


Figure 2.6: Seasonal Factors from UCM

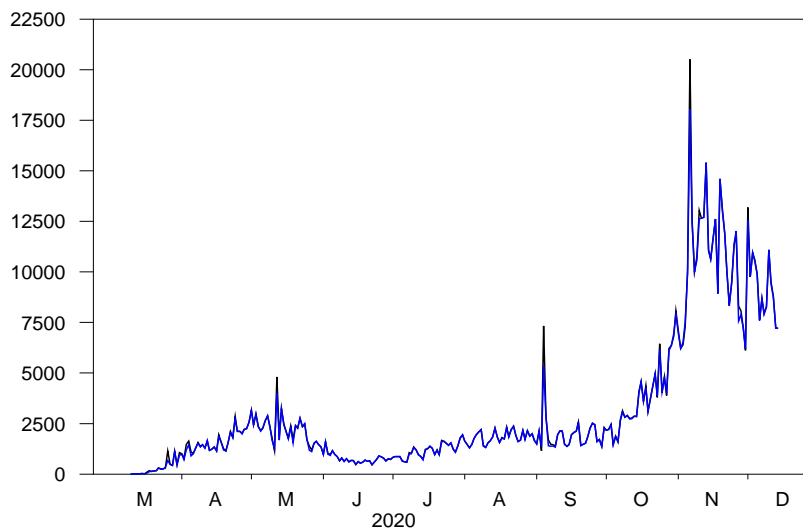


Figure 2.7: Comparison of Approximate and Actual Data

The product should be similar to the actual data, but won't be exact. This does a comparison to see how closely they match (Figure 2.7)—actually quite well except for a few sizable spikes:

```
set test = xsmooth*dsmooth*usmooth
graph(footer="Comparison of Approximate and Actual Data") 2
# test
# new_cases
```

Example 2.1 Weekly Cycles by Smoothing

This is `covidweeklycycles.rpf`. It does an analysis of day-of-week effects in data using relatively simple methods. This uses Illinois data (the `state_testing_results` array from <https://www.dph.illinois.gov/sitefiles/COVIDTestResults.json>).

```
open data "covidIllinois.txt"
calendar(7) 2020:3:10
*
* These are all cumulative numbers
*
data(format=prn,nolabels,org=columns,dateform="m/d/y") 2020:03:10 * $
    total_tests confirmed deaths
*
* Transform into new data
*
set(first=0.0) new_tests = total_tests-total_tests{1}
set(first=0.0) new_cases = confirmed-confirmed{1}
set(first=0.0) new_deaths = deaths-deaths{1}
*
filter(type=lagging,width=7) new_cases / cases_7day
graph(footer="New Cases with 7-day Average",$
    key=upleft,klabels=||"Actual","Smoothed"||) 2
# new_cases
# cases_7day
*
* Additive exponential smoothing on square root of data
*
set sqrt_new = sqrt(new_cases)
esmooth(trend=linear,seasonal=additive,span=7,estimate,$
    factors=sf,smoothed=smsqrt,initial=start) sqrt_new
set esmoothed = smsqrt^2
graph(footer="New Cases with Exponential Smoothing",$
    key=upleft,klabels=||"Actual","Smoothed"||) 2
# new_cases
# esmoothed
*
graph(footer="Exponential Smoothing and Simple Moving Average",$
    key=upleft,klabels=||"ESmooth","Moving Average"||) 2
# esmoothed
# cases_7day
*
graph(footer="Seasonal Factor (Mondays highlighted)",grid=%weekday(t)==1)
# sf
```

Example 2.2 Weekly Cycles by Unobserved Components

This is `covidssmcycles.rpf`. This does an analysis of day-of-week effects in data using a non-linear state-space model.

```

open data "covidIllinois.txt"
calendar(7) 2020:3:10
*
* These are all cumulative numbers
*
data(format=prn,nolabels,org=columns,dateform="m/d/y") 2020:03:10 * $
    total_tests confirmed deaths
*
* Transform into new data
*
set(first=0.0) new_tests = total_tests-total_tests{1}
set(first=0.0) new_cases = confirmed-confirmed{1}
set(first=0.0) new_deaths = deaths-deaths{1}
*
* Multiplicative state-space model
*
* Signal is  $x(t)=x(t-1)+ex(t)$ , where  $x(t)$  is the square root of the signal.
* Observable is  $y(t)=x(t)^2*d(t)*u(t)$ 
*
* The daily factor follows
*
*  $d(t)+d(t-1)+\dots+d(t-6)=7+ed(t)$ , that is, the daily factors average 1.
*
* This is basically an additive seasonal model, but summing to 7 rather
* than 0. The 7 requires a "Z" component in the state space model.
*
* The irregular factor follows
*
*  $u(t)=1+eu(t)$ 
*
@LocalDLM(type=level,a=alevel,c=clevel,f=flevel)
compute [vector] zlevel=%zeros(%rows(alevel),1)
@SeasonalDLM(span=7,type=additive,a=adaily,c=cdaily,f=fdaily)
compute [vector] zdaily=7.0*unitv(%rows(adaily),1)
compute [rect] airreg=||0.0||
compute [vect] zirreg=||1.0||
compute [rect] firreg=||1.0||
compute [rect] cirreg=||1.0||
*
* Concatenate the A and F matrices and Z vectors
*
compute a=alevel~\adaily~\airreg
compute f=flevel~\fdaily~\firreg
compute z=zlevel~~zdaily~~zirreg
*
declare real swlevel swdaily swirreg
*
* Generate the formulas for the expanded measurement equation (given
* xtilde, dtilde and utilde).
*
dec frml[vect] cf
clear(zeros) xtilde dtilde utilde
frml cf = clevel*2*xtilde*dtilde*utilde~~$
          cdaily*xtilde^2*utilde~~$

```

```

        cirreg*xtilde^2*dtilde
frml muf = -3.0*xtilde^2*dtilde*utilde
*
* Start the level at .01 (that is, slightly positive) with a variance of
* 1
*
compute [vect] x0level=%fill(1,1,.01)
compute [symm] sx0level=%ones(1,1)
*
* Start the day weights at 1's with a .25 variance
*
compute [vect] x0daily=%ones(6,1)
compute [symm] sx0daily=%diag(%fill(6,1,.25))
*
* Start the irregular at 1 with a zero variance
*
compute [vect] x0irreg=%ones(1,1)
compute [symm] sx0irreg=%zeros(1,1)
*
nonlin(parmset=varparms) swlevel swdaily swirreg swdaily>=0.0
compute swdaily=.01
compute swlevel=1
compute swirreg=1.0
*
function MakeTildes x sx
type vector *x
type symm *sx
*
compute xtilde(t)=%max(1.0,x(1))
compute dtilde(t)=7-%sum(%xsubvec(x,2,7))
compute utilde(t)=1
end
*
dlim(y=new_cases,a=a,f=f,sw=%diag(||swlevel,swdaily,swirreg||),$
      c=cf,mu=muf,z=z,x0=x0level~~x0daily~~x0irreg,$
      sx0=sx0level~\sx0daily~\sx0irreg,fpre=MakeTildes,type=smooth,$
      parmset=varparms,pmethod=simplex,piters=10,method=bfgs) / $
      xstates vstates
*
set xsmooth = xstates(t)(1)^2
set dsmooth = xstates(t)(2)
set usmooth = xstates(t)(8)
*
graph(footer="Smoothed Estimates from UCM") 2
# new_cases
# xsmooth
graph(footer="Seasonal Factors from UCM\\Mondays highlighted",$
      grid=%weekday(t)==1)
# dsmooth
*
set test = xsmooth*dsmooth*usmooth
graph(footer="Comparison of Approximate and Actual Data") 2
# test
# new_cases

```


SEIR Model for Estimation

To bring the actual SEIR model to data (or at least something close to it), we need to start with the form with the additional “transition” states (1.4) since that is what gives us the required errors, and also gives us an (imperfectly) observable datum: EI is at least theoretically the “new case” count. For simplicity, we will first assume that $S = N$ which is appropriate in the early stages when a relatively small percentage of the population has been infected. That eliminates one non-linearity. We also don’t need to include R since nothing in the dynamics of the model depends upon it. Thus, we reduce the model to

$$\begin{aligned} EI_t &= \sigma E_{t-1} + \varepsilon_{I,t} \\ SE_t &= \beta I_{t-1} + \varepsilon_{E,t} \\ IR_t &= \gamma I_{t-1} + \varepsilon_{R,t} \\ E_t &\equiv E_{t-1} + SE_t - EI_t \\ I_t &\equiv I_{t-1} + EI_t - IR_t \end{aligned} \tag{3.1}$$

The data being used are U.S. aggregate data¹

```
open data "uscovid.xlsx"
calendar(7) 2020:1:21
data(format=xlsx,org=columns,left=4) 2020:01:21 * total_cases $
    new_cases total_deaths new_deaths total_cases_per_million $
        new_cases_per_million total_deaths_per_million $
            new_deaths_per_million total_tests new_tests $
                total_tests_per_thousand new_tests_per_thousand
*
compute pop=330e+6
```

3.1 With Fixed Beta

As written, (3.1) is not in the proper form for a state-space model because the final two equations depend upon *current* (rather than lagged) states. Example 3.1 shows how to use the **DSGE** instruction to convert the system (3.1) into the proper form. This works as long as the parameters are not time-varying—they can be unknown, but have to be fixed across time.

¹From <https://covid.ourworldindata.org/data/owid-covid-data.xlsx>

We'll take a starting guess for β based upon early estimates of reproduction rate:

```
compute rawR0=3.5
compute beta=rawR0*gamma
```

For use with **DSGE**, any **FRML** which does *not* have an **IDENTITY** option is assumed to have its own (implied) error. Since this is the situation with the first three equations in (3.1), we can set it up as:

```
dec series se ei ir ee ii

frml eieq ei = sigma*ee{1}
frml seeq se = beta*ii{1}
frml ireq ir = gamma*ii{1}
frml(identity) eedef ee = ee{1}+se{0}-ei{0}
frml(identity) iidef ii = ii{1}+ei{0}-ir{0}
```

(The series have be declared first so they will be recognized as series in the **FRML** definitions).

A **GROUP** instruction is used to bind these into a **MODEL**:²

```
group seirdata eieq seeq ireq eedef iidef
```

This function will be called as a **START** option function on the **DLM** instruction—this allows β (and γ or σ if desired) to be estimated.

```
function SolveOut
dsge(model=seirdata,a=a,f=f,cutoff=%na) ei se ir ee ii
end
```

Next are the variances implied by the model, using feasible estimates of **EE** and **II**. These are truncated below at 1, which keeps them from collapsing to zero in the early stages when cases are very sparse.

```
clear eetilde iitilde
frml sweifrm1 = p=sigma,%max(1.0,eetilde*p*(1-p))
frml swirfrm1 = p=gamma,%max(1.0,iitilde*p*(1-p))
frml swsefrm1 = p=beta*iitilde/pop,%max(1.0,pop*p*(1-p))
```

We now have two “intervention” functions which are called at different points in the Kalman filter iteration. As in Section 2.2 **MakeTildes** is called as an **FPRE** function at the start of the time **T** calculation, and saves the values of the **EE** (state 4) and **II** (state 5) as the feasible estimates for the variance calculation—at the start of time **T**, those will have the filtered estimates through **T-1**.

²The order of listing is used in determining the order of the shocks (so here the three shocks will be to **EI**, **SE** and **IR** in that order). The order of the states themselves is determined by the order of listing of variables on the **DSGE**.

```

function MakeTildes x sx
type vector *x
type symm    *sx
compute eetilde(t)=x(4)
compute iitilde(t)=x(5)
end

```

`FixSigns` is called after the filter step (as an `FPOST` function), and is a crude but effective way to force the states to be non-negative—the shocks can take either sign so a filtered value might go negative. Eventually, all states will naturally be positive once the positive values for infections start to accumulate, but at the early stages, it won't be uncommon to have some filtered states be slightly negative.

```

function FixSigns x sx
type vector *x
type symm    *sx
*
compute x=%plus(x)
end

```

The `A` and `F` matrices are going to be computed using the `SolveOut` function above. The `C` vector is straightforward, since the observable is (theoretically) just state 1, so `C` is a unit vector for the first position.

```

compute [vect] c=%unitv(5,1)

```

The initialization for the states is not as clear. In the examples from Chapter 1, all that was required was for a small percentage of the population to be in the “exposed” category for the infection to eventually take root and grow exponentially. Here, however, there are only sporadic cases for the first five weeks, many of which are probably the result of “seeding” from outside the population by travel from areas with infections. Because the basic system is explosive, there is no “ergodic” solution for the initial conditions. The following seems to work reasonably with this data set: all the states are initialized to be mean of zero, but `EE` (exposed) and `II` (infectious) are given non-zero variances with a high correlation (which is what would be expected). Here the variances are 10000 (standard deviation of 100), so it's not unreasonable that the number of exposed people at the start is on the order of 100 to 200. Note that that's across the U.S. population of over 300 million.

```

compute [vect] x0=%zeros(5,1)
compute rhoeeii=.9
compute eevar=10000.0,iivar=10000.0
compute [symm] sx0=%diag(||0.0,0.0,0.0||)~\$
               %corrto cv(||rhoeeii||,||eevar,iivar||)

```

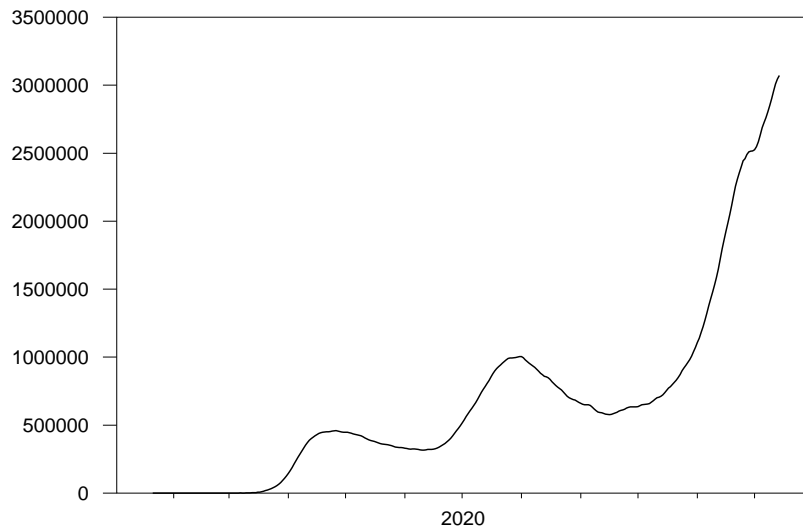


Figure 3.1: Estimate of Exposed from Fixed Beta

The **DLM** instruction to solve the model is:

```
compute beta=rawR0*gamma
d1m(y=new_cases, start=SolveOut(), fpre=MakeTildes, fpost=FixSigns, $
    a=a, f=f, c=c, sw=%diag(||sweifrml, swseifrml, swirifrml||), $
    x0=x0, sx0=sx0, type=filter, yhat=yhat, svhat=svhat, $
    save=seirdlm, likelihoods=ll) * * xstates vstates
```

The **START** option calls **SolveOut** to solve for the backwards state-space representation. **FPRE** calls the **MakeTildes** function to compute the “tilde” variables that are used in the variance **FRML**’s, and **FPOST** calls the function that makes all the states non-negative. **SW** makes the covariance matrix for the three shocks using the variance **FRML**’s. Note that there is no **SV** option in this because the shocks to **EI** are included in the state shocks and (if the model is correct) **EI** is observed without error.³

This pulls out and graphs the (filtered) estimates of the number exposed (Figure 3.1) and number infectious (Figure 3.2):

```
set ee = xstates(t) (4)
set ii = xstates(t) (5)
graph(footer="Estimate of Exposed")
# ee
graph(footer="Estimate of Infectious")
# ii
```

These don’t look at all unreasonable given the behavior of the new cases series. However, we can also look at the actual vs predicted case counts (Figure 3.3) with:

³From what we’ve seen of the observables in Chapter 2, this seems rather optimistic, but for simplicity we’ll stick with this for now.

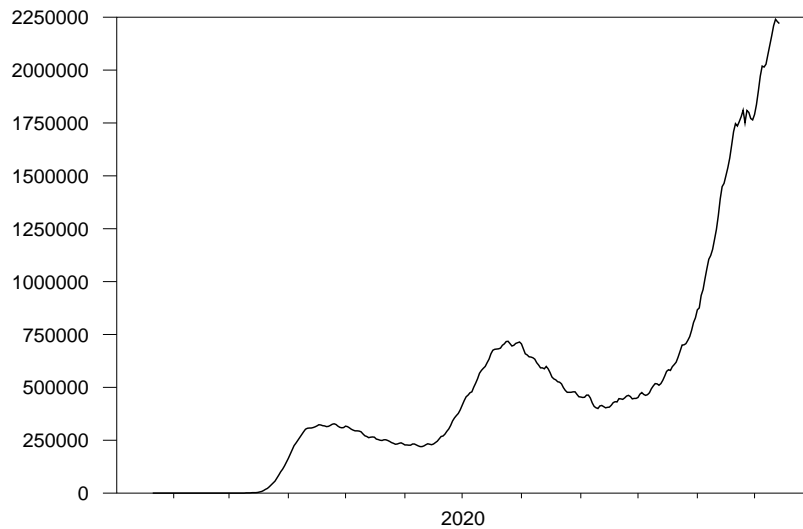


Figure 3.2: Estimate of Infectious from Fixed Beta

```
set yhatx = yhat(t)(1)
graph(footer="Actual and Predicted New Cases", $
      key=upleft, klabels=|| "Actual", "Predicted" ||) 2
# new_cases
# yhatx
```

which shows that the fixed beta model does a fairly good job of predicting the data through April 6, but is consistently high after that. This is not unexpected—the raw beta value is the assumed natural rate at which the infection spreads with normal behavior in the population, but would be expected to decline if people change their behavior to reduce the spread.

Another problem is that the model is far too optimistic about its accuracy. This computes and graphs (Figure 3.4) the standard error of prediction:

```
set svhatx = sqrt(svhat(t)(1,1))
graph(footer="Standard error of prediction")
# svhatx
```

In places where the actual errors are on the order of 20000-30000, the model is predicting standard errors of 200-300, so the actual errors are roughly 100 model standard errors, that is, effectively zero probability. The `LIKELIHOOD` option on `DLM` returns a series with the cumulated log likelihoods—the first difference of that can show the element by element log likelihood (Figure 3.5):

```
set lldiff = ll-ll{1}
graph(overlay=line, vlabel="Log Likelihood Element", $
      ovlabel="New Cases") 2
# lldiff
# new_cases
```

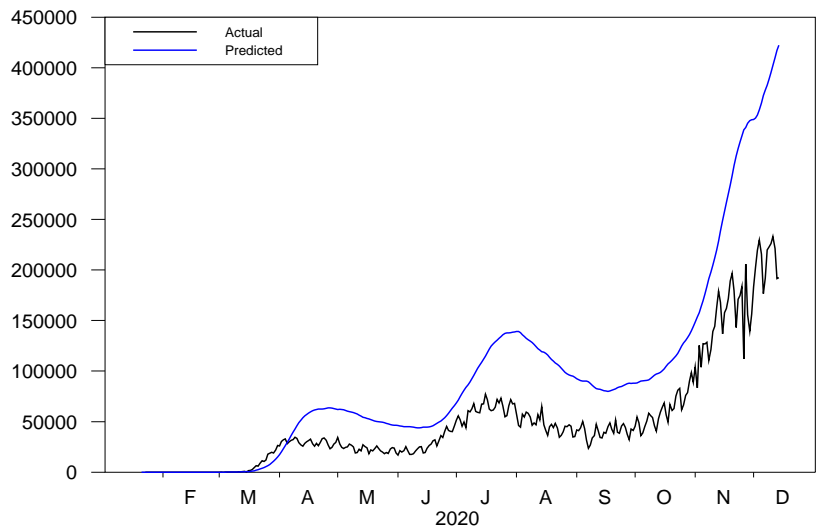


Figure 3.3: Actual/Predicted for Fixed Beta

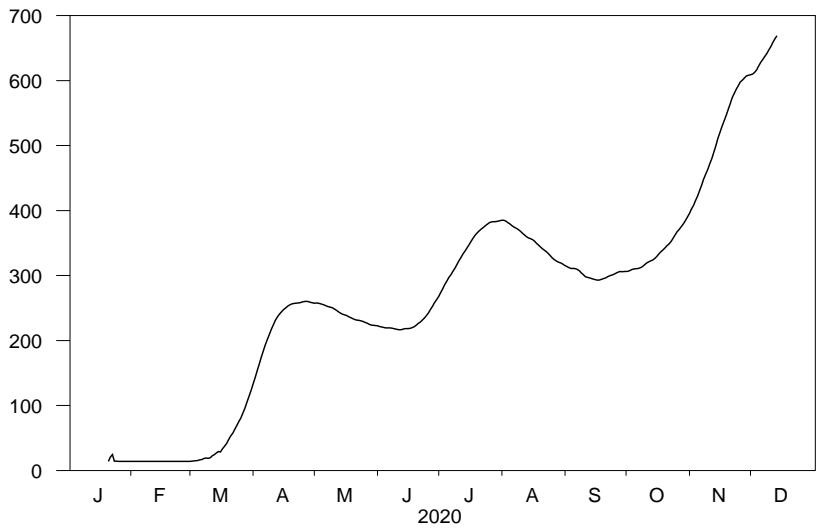


Figure 3.4: Standard Errors of Prediction

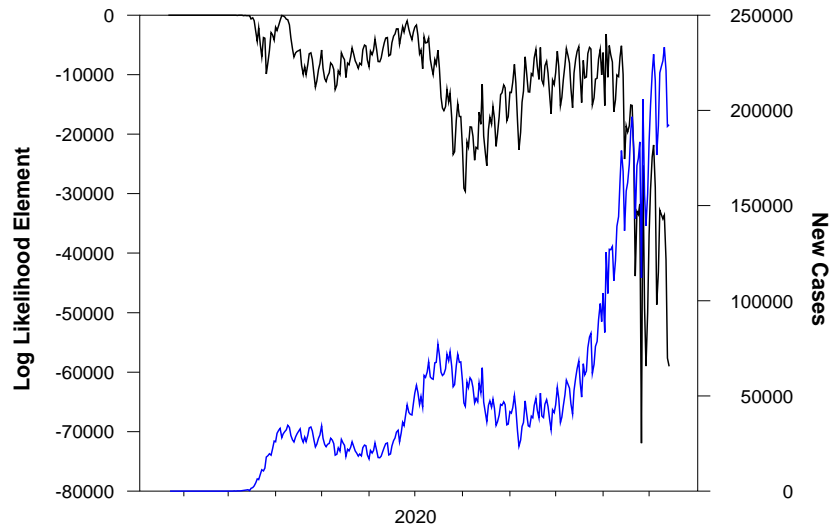


Figure 3.5: Log Likelihood Elements vs New Cases

Even where the model predicts the point values fairly well (March into early April) the log likelihoods are around -3000 (per entry) as the standard deviations are still way too small.

Higher values of beta give a higher likelihood, not because they predict the data better (they don't—a higher beta will greatly overestimate the cases through much of the sample), but because with a higher beta the standard error of prediction increases dramatically so even large estimation errors don't generate such extremely low log likelihoods, which indicates that the assumption EI being observed without error is unrealistic.

3.2 With Deterministic Time-Varying Beta

The above method of using **DSGE** to solve for a proper state-space representation only works if the β is fixed over the sample. If β is time-varying, as in Section 1.3, then the **A** matrix is time-varying as well. To handle that, it's necessary to solve the system out to get rid of the contemporaneous variables on the right side. That's relatively straightforward, yielding the system:

$$\begin{bmatrix} EI_t \\ SE_t \\ IR_t \\ E_t \\ I_t \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 & \gamma \\ 0 & 0 & 0 & 1 - \sigma & \beta \\ 0 & 0 & 0 & \sigma & 1 - \gamma \end{bmatrix} \begin{bmatrix} EI_{t-1} \\ SE_{t-1} \\ IR_{t-1} \\ E_{t-1} \\ I_{t-1} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} \varepsilon_{I,t} \\ \varepsilon_{E,t} \\ \varepsilon_{R,t} \end{bmatrix} \quad (3.2)$$

Much of the start of Example 3.2 is the same as Example 3.1. There are two main differences: a **BETAT** series is created with the assumed changes to the transmission rate over time, and the **A** and **F** matrices are created differently.

The first is done with

```
compute rawR0=3.5
compute beta=rawR0*gamma
*
compute d0=21.0,t0=2020:03:23,maxg=.30
set betat = beta*(1+(maxg-1)*(1-exp(-(t-t0)/d0)^2)))
```

where the parameters governing this are chosen by an “eyeball” fit to the data.⁴

The FRML for the *SE* variance needs to be changed slightly since the beta value is time-varying rather than fixed. That’s

```
frml swsefrml = p=betat{0}*iitilde/pop,%max(1.0,pop*p*(1-p))
```

replacing the fixed BETA with BETAT{0}.

F is a fixed matrix, but **A** is time-varying (as it depends upon BETAT). The **A** will be handled by the function `AMatrix`, which will “poke” the correct values into an otherwise all zeros matrix.

```
dec rect a(5,5) f(5,3)
compute a=%zeros(5,5)
input f
  1  0  0
  0  1  0
  0  0  1
-1  1  0
  1  0 -1

function AMatrix t
type integer t
type rect AMatrix
*
compute a(1,4)=sigma
compute a(2,5)=betat(t)
compute a(3,5)=gamma
compute a(4,4)=1-sigma
compute a(4,5)=betat(t)
compute a(5,4)=sigma
compute a(5,5)=1-gamma
compute Amatrix=A
end
```

⁴Trying to estimate these would be very difficult, particularly with the model in its current form. First, `T0` isn’t continuous, so would have to be chosen using a search. Second, as explained in the previous example, higher values of beta (which would require *negative* values of `MAXG`) produce higher likelihoods because they produce higher variances, not better point predictions.

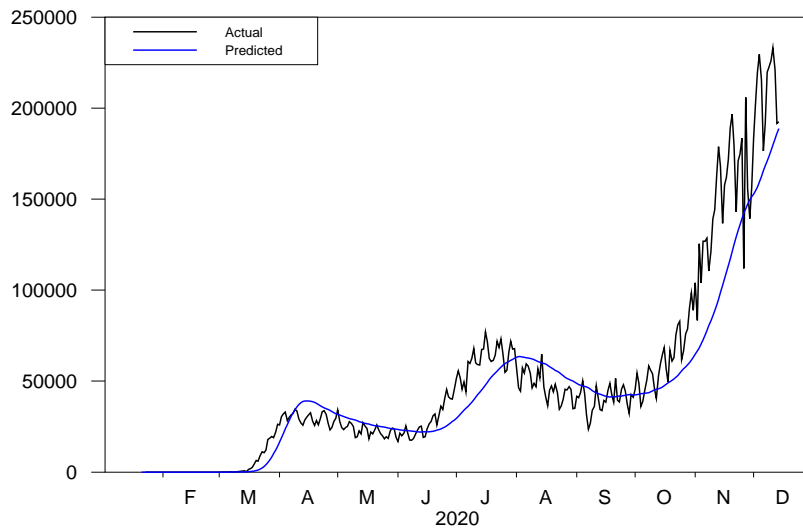


Figure 3.6: Actual/Predicted for Deterministic Time-Varying Beta

The **DLM** instruction is nearly the same as before: the differences are no `START` option since that was to solve out for `A` and `F`, and `A` is now set with `a=AMatrix(t)`:

```
dln(y=new_cases, fpre=MakeTildes, fpost=FixSigns, a=AMatrix(t), $
    f=f, c=c, sw=%diag(||sweifrml,swseifrml,swirifrml||), $
    x0=x0, sx0=sx0, type=filter, yhat=yhat, svhat=svhat, $
    likelihoods=ll) * * xstates vstates
```

The actual vs predicted graph (Figure 3.6) tracks the data fairly well through late June,⁵ but then fails to predict the second increase that came with the relaxations of restrictions in many states. If one were actually trying to model the full series, there would need to be a second transition to a higher beta value from late June through late July, and then a third transition to probably an intermediate beta beginning in late July as restrictions began to be re-imposed.

3.3 With Endogenous Beta

An alternative to exogenously imposed time-varying betas is to add beta to the model and allow it to be estimated. A simple way to do that is to allow it to be a time-varying random walk:

$$\beta_t = \beta_{t-1} + \varepsilon_{\beta,t} \quad (3.3)$$

This adds an additional state to the model, and adds a non-linearity to the dynamics, since

$$SE_t = \beta I_{t-1} + \varepsilon_{E,t}$$

⁵which was the point of the “eyeball fit”.

now will involve the product of two states. The simplest way to handle this is to date the β state variable so this would be

$$SE_t = \beta_{t-1}I_{t-1} + \varepsilon_{E,t}$$

Linearizing this around $\tilde{\beta}_{t-1}$ and \tilde{I}_{t-1} produces

$$SE_t \approx \tilde{\beta}_{t-1}I_{t-1} + \tilde{I}_{t-1}\beta_{t-1} - \tilde{\beta}_{t-1}\tilde{I}_{t-1} + \varepsilon_{E,t}$$

The $-\tilde{\beta}_{t-1}\tilde{I}_{t-1}$ term will be part of the \mathbf{Z} vector, while the other terms will be in the (time-varying) \mathbf{A} matrix. When rendered as a proper state-space model, we get

$$\begin{aligned} \begin{bmatrix} EI_t \\ SE_t \\ IR_t \\ \beta_t \\ E_t \\ I_t \end{bmatrix} &\approx \begin{bmatrix} 0 & 0 & 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & \tilde{I}_{t-1} & 0 & \tilde{\beta}_{t-1} \\ 0 & 0 & 0 & 0 & 0 & \gamma \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \tilde{I}_{t-1} & 1 - \sigma & \tilde{\beta}_{t-1} \\ 0 & 0 & 0 & 0 & \sigma & 1 - \gamma \end{bmatrix} \begin{bmatrix} EI_{t-1} \\ SE_{t-1} \\ IR_{t-1} \\ \beta_{t-1} \\ E_{t-1} \\ I_{t-1} \end{bmatrix} \\ &+ \begin{bmatrix} 0 \\ -\tilde{I}_{t-1}\tilde{\beta}_{t-1} \\ 0 \\ 0 \\ -\tilde{I}_{t-1}\tilde{\beta}_{t-1} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \varepsilon_{I,t} \\ \varepsilon_{E,t} \\ \varepsilon_{R,t} \\ \varepsilon_{\beta,t} \end{bmatrix} \end{aligned} \quad (3.4)$$

This is done in Example 3.3. In comparison with Example 3.2, this will have six states rather than five and will need a \mathbf{Z} component as well.

```

dec rect a(6,6)
dec vect z(6)
dec rect f(6,4)
*
compute a=zeros(6,6)
compute z=zeros(6,1)
*
input f
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
-1 1 0 0
1 0 -1 0

```

```

*
* Loadings on shocks to states
*
compute c=%unitv(6,1)

```

It needs three “tilde” variables, adding one for $\tilde{\beta}$ —we’ll also add one for EI , which will be needed in a more complicated version.

```

*
* Expansion points
*
clear(zeros) iitilde eetilde eitilde betatilde

```

As with Example 3.2, the **A** matrix is time-varying, and so will have a function which computes it (by re-setting the non-zero elements), while **F** is fixed. **Z** is time-varying as well, so it will have its non-zero elements set using its own function.

```

*
* Poke values into A
*
function AMatrix t
type rect    AMatrix
type integer t
*
compute a(1,5)=sigma
compute a(2,4)=iitilde(t)
compute a(2,6)=betatilde(t)
compute a(3,6)=gamma
compute a(4,4)=1.0
compute a(5,4)=iitilde(t)
compute a(5,5)=1-sigma
compute a(5,6)=betatilde(t)
compute a(6,5)=sigma
compute a(6,6)=1-gamma
compute AMatrix=A
end

*****
function ZVector t
type vector ZVector
type integer t
compute z(2)=-betatilde(t)*iitilde(t)
compute z(5)=z(2)
compute ZVector=z
end

```

```

*****
function MakeTildes x sx
type vector *x
type symm *sx
compute eitilde(t)=x(1)
compute betatilde(t)=x(4)
compute eetilde(t)=x(5)
compute iitilde(t)=x(6)
end

*****
function FixSigns x sx
type vector *x
type symm *sx
*
compute x=%plus(x)
end

```

The “tilde” variables at T are all the filtered versions from $T-1$, which is what we want throughout.

The pre-sample means and covariance matrix for the SEIR states will be the same as in the two other examples in this chapter (page 32). The pre-sample mean for beta (which is now state 4) is the base value used in the other two examples (roughly .25), and its standard deviation is .1 (variance .01).

```

compute x0=%zeros(6,1)+beta0*%unitv(6,4)
compute rhoeeii=.9
compute eevar=10000.0,iivar=10000.0
compute [symm] sx0=%diag(||0.0,0.0,0.0||)~\.01~\$
               %corrto cv(||rhoeeii||,||eevar,iivar||)

```

The formula for the SE variance needs to change slightly to allow for the fact that the beta is now given by its tilde value:

```

frml swsefrml = p=betatilde*iitilde/pop,%max(1.0,pop*p*(1-p))

```

There’s now a fourth transition variance (for the shock $\varepsilon_{\beta,t}$) and it has no special theoretical value. It’s possible to leave that free and estimate it, though that is unlikely to produce satisfactory results, at least with the model as it exists—it would likely produce a β which varies too much to be useful. We found that pegging the variance at 10^{-8} gives a generally fairly reasonable set of estimates. Note that that is extremely small: the natural values of β will range from around .1 to .3, while a random walk with an increment variance of 10^{-8} would have a standard deviation after a full year of only about .002.⁶ However, the model otherwise so severely underestimates the variances that changes in β

⁶ $(365 \times 10^{-8})^{1/2}$

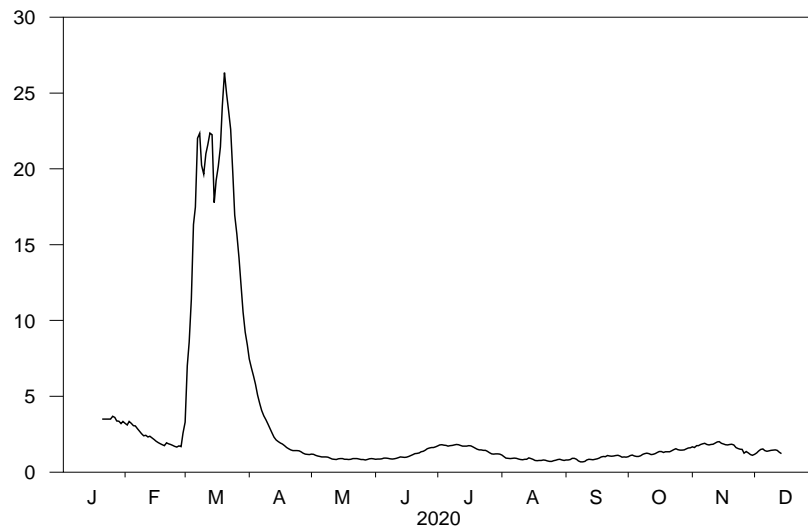


Figure 3.7: Estimates of R_0 from Endogenous Beta

that are much larger than what would be expected from its own random walk model end up being optimal.

```
compute swbeta=1.e-8
```

This is the **DLM** instruction for this model. Changes are the **z** option (which calls the **ZVector** function) and the additional variance in the **SW** option:

```
dln(y=new_cases, fpre=MakeTildes, fpost=FixSigns, a=AMatrix(t), f=f, $
  c=c, z=ZVector(t), sw=%diag(||sweifrm1, swsefmr1, swirfmr1, swbeta||), $
  x0=x0, sx0=sx0, type=filter, yhat=yhat, svhat=svhat) $
  * * xstates vstates
```

EE and **II** are now states 5 and 6 and the **BETA** is state 4:

```
set ee = xstates(t) (5)
set ii = xstates(t) (6)
set betax = xstates(t) (4)
```

The β can be converted into the equivalent R_0 by dividing by γ :

```
set r0 = betax/gamma
```

When graphed over the full range (Figure 3.7), this is seen to produce rather bizarre values during the early stages when the data are fairly thin.

Graphed over the portion of the sample beginning in late April (Figure 3.8), you see a more reasonable looking pattern, with R_0 declining slightly below 1 during May, then roughly doubling from there through the second peak in mid-July followed by another decline (as there were some imposition of restrictions in states that peaked in summer), and then another increase beginning in September.

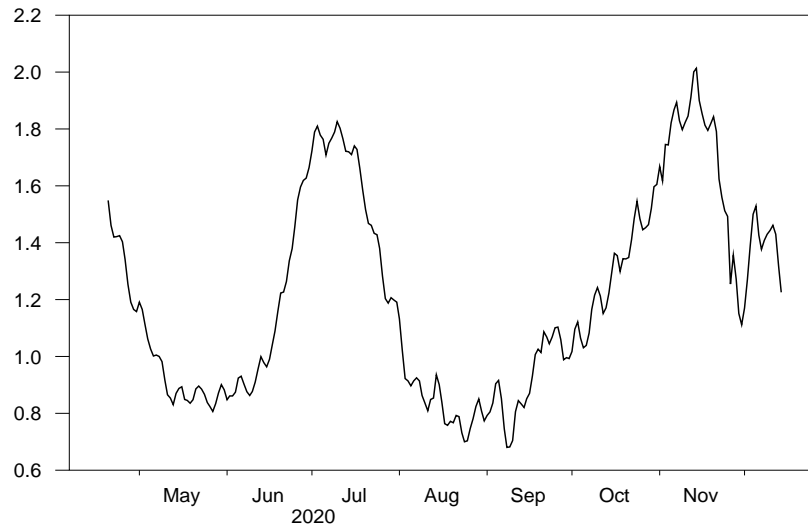


Figure 3.8: Estimates of R_0 from Endogenous Beta, Late Sample

One thing to note is that these are the *filtered* estimates and they are subject to rather considerable error—particularly at the beginning of the sample when there is very little information to help estimate them. If you do this with `TYPE=SMOOTH` rather than `TYPE=FILTER` on the **DLM**, you get Figure 3.9.

Not too surprisingly, the actual vs predicted graph (Figure 3.10) looks much better than with the fixed and deterministic betas. It fairly closely tracks the overall curve, but doesn't (and basically can't) deal with the day-of-the-week cycles. Note that if you loosen up the variance on shock 4 (say with 10^{-4}), it will track the actual data more closely, but will do that by substantial day-of-the-week variations in β , which would make little sense.

3.4 With Endogenous Beta and Day-of-Week Effects

All of the models so far in this chapter have faced the problem that the actual observable is far from the relatively simple large-sample binomial process posited by the model. In fact, as we saw in Chapter 2, the day-to-day changes in the data are dominated by day-of-the-week effects which are substantially greater than the systematic changes predicted by the model. We can create an improved model by combining the endogenous beta model from Section 3.3 with the unobserved components model for the day-to-day fluctuations from Section 2.2. However the x will now come from the SEIR model rather than being just a “trend” component.

The measurement equation will now take the form:

$$y_t = EI_t \times d_t \times u_t \quad (3.5)$$

where EI comes from the model (3.4), while d and u are as in (2.2) and (2.3). As before, the multiplicative measurement equation needs to be linearized:

$$y_t \approx \tilde{d}_t \tilde{u}_t EI_t + \widetilde{EI}_t \tilde{u}_t d_t + \widetilde{EI}_t \tilde{d}_t u_t - 2\widetilde{EI}_t \tilde{d}_t \tilde{u}_t \quad (3.6)$$

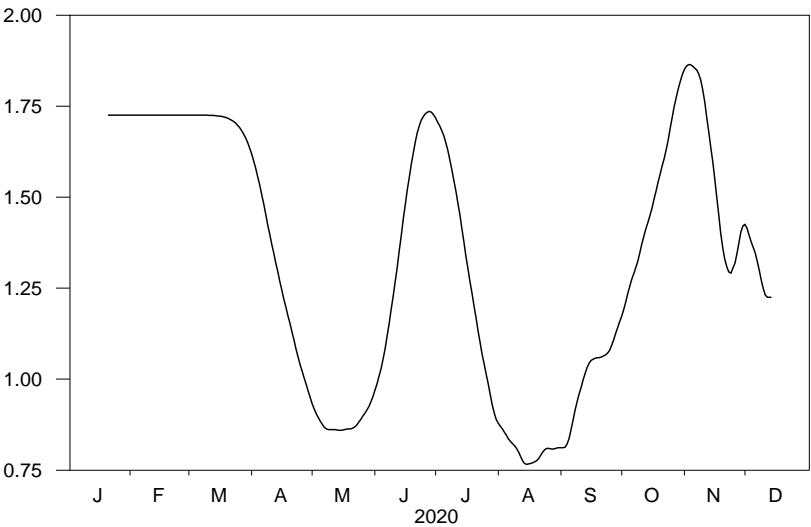


Figure 3.9: Smoothed Estimates of R_0 from Endogenous Beta

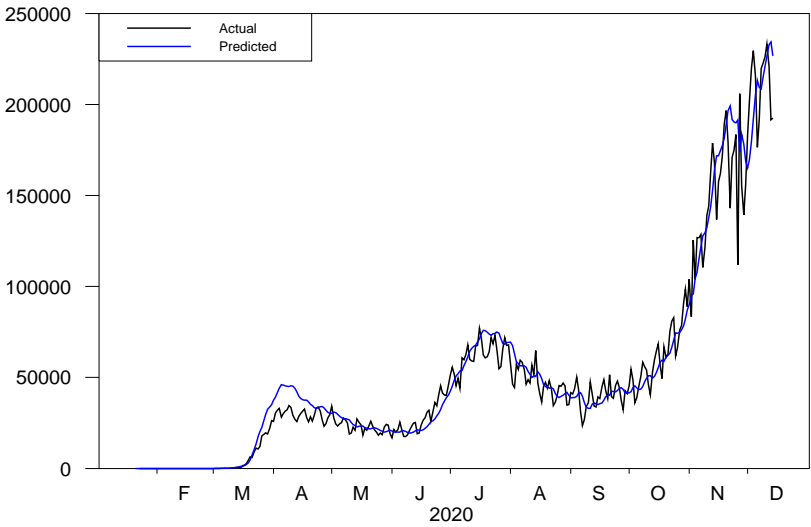


Figure 3.10: Actual/Predicted for Endogenous Time-Varying Beta

The last term will be the **MU** component in the state-space model, while the first three terms will give a (now time-varying) **C** component.

Example 3.4 starts out largely the same as Example 3.3. The additional states for the seasonal and irregular will be “added” to the model using concatenation operators.

As with Example 2.2, we need to set up components for the “seasonal” (and for the irregular), where the seasonal approximately adds to seven over the days of the week and the irregular is a mean 1 serially uncorrelated process:

```
@SeasonalDLM(span=7,type=additive,a=adaily,c=cdaily,f=fdaily)
compute [vector] zdaily=7.0*unitv(%rows(adaily),1)
*
compute [rect] airreg=||0.0||
compute [vect] zirreg=||1.0||
compute [rect] firreg=||1.0||
compute [rect] cirreg=||1.0||
```

The overall **C** vector will come from combining the **C** pieces for the components of the model using the “tilde” factors from (3.6).

```
dec frml[vect] cfs
frml cfs = c*dtilde*utilde~~cdaily*eitilde*utilde~~cirreg*eitilde*dtilde
```

This will have three non-zero elements: the first will be on the *EI* component of the SEIR model (and its value should be close to 1); the second will be on the current value of *d*, and the last will be on the current value of the irregular.

The **MU** formula is also taken out of (3.6):

```
dec frml mufs
frml mufs = -2.0*dtilde*eitilde*utilde
```

The following will help us navigate the (now) 13 states.

```
compute nbase=%rows(f)
compute dpos=nbase+1
compute ipos=dpos+6
```

The `MakeTildesS` function will save the `BETATILDE`, `EETILDE` and `IITILDE` as before. The variables needed for the expansion of the measurement equation are feasible expectations for time *T* (rather than *T*-1) and so need to be the predicted values given the information through *T*-1.


```

function MakeTildesS x sx
type vect *x
type symm *sx
*
* These need to be the filtered value at the end of t-1
*
compute betatilde(t)=x(4)
compute eetilde(t)=x(5)
compute iitilde(t)=x(6)
*
* These need to be the predicted value at t given t-1
*
compute eitilde(t)=sigma*x(5)
compute dtilde(t)=7-%sum(%xsubvec(x,dpos,dpos+5))
compute utilde(t)=1.0
end

```

The biggest issue is coming up with a reasonable way to handle the variances that aren't predicted by the model: the variance for $\varepsilon_{\beta,t}$, the pre-sample and increment variances for the daily component and the variance for the irregular component. There are two complications for this compared to the unobserved components model in Example 2.2:

1. With aggregate U.S. data, the daily component isn't as regular (since it's a mix of daily cycles from different states) and the full data set starts long before there is any discernible day-to-day pattern.
2. The "trend" component here is model-driven rather than data-driven, and (as we've seen) has a relatively low variance.

If you give the daily and irregular components what would seem to be objectively reasonable values for the variances, they will end up absorbing almost all the changes in the data since they would be seen as the source of most of the variance. On the other hand, if we guess that the model's predictions of the variance of EI is off by a factor of 25 (that is, the standard errors are understated by a factor of 5) then scaling the other variances by .04 should end up with the means being close to what they should be. This sets the variance of the change in beta to the same value used in Example 3.3, and fixes the variance scale adjustment for the others:

```

compute swbeta=1.e-8
compute scalefac=.04

```

If we think a natural range for the evolution of a seasonal factor is $\pm .01$ (that is, if we are at 1.12, we would think it unlikely it would go outside (1.11, 1.13) the next week. Taking .01 as two standard deviations gives us:

```
compute swdaily=.005^2*scalefac
frml swdailyf = swdaily
```

We've written this as a `FRML` to allow for the possibility that it might need to be adjusted for different values in different ranges.

Similarly, if we want the irregular factor to be largely limited to $(.98, 1.02)$, taking $.02$ as two standard deviations gives us:

```
compute swirreg=.01^2*scalefac
frml swirregf = %if(t>=2020:04:22.and.t<=2020:04:28,9.0*swirreg,swirreg)
```

with the slight twist that the irregular variance is scaled up during a particularly noisy period in the data (at least in this particular data set).⁷

The extra states are initialized using:

```
compute [vect] x0daily=%ones(6,1)
compute [symm] sx0daily=%diag(%fill(6,1,.05^2*scalefac))
*
compute [vect] x0irreg=%ones(1,1)
compute [symm] sx0irreg=%zeros(1,1)
```

Again, we need to scale down the variance of the daily cycles—this would have $(.90, 1.10)$ as the natural range for the daily factors.

The `DLM` instruction to fitting the model is:

```
dln(y=new_cases, fpre=MakeTildesS, fpost=FixSigns, $
a=AMatrix(t)~\adaily~\airreg, z=ZVector(t)~\zdaily~\zirreg, c=cfs, $
mu=mufs, f=f~\fdaily~\firreg, x0=x0~\x0daily~\x0irreg, $
sx0=sx0~\sx0daily~\sx0irreg, $
sw=%diag(||sweifrml, swseifrml, swirfrml, swbeta, swdailyf, swirregf||), $
type=filter, yhat=yhat, svhat=svhat) * * xstates vstates
```

The `A`, `Z`, `F`, `X0` and `SX0` options are all done by combining the separate matrices for the three submodels: with diagonal concatenation for `A`, `F` and `SX0` and vertical concatenation for `Z` and `X0`. Similarly, the `SW` option is handled as the diagonal matrix of the original three SEIR component variance, followed by the (fixed) value of `SWBETA` and the `FRML` values for `SWDAILY` and `SWIRREG`.

The estimates of the exposed, infectious, and the R_0 are very similar to the ones from Example 3.3 (R_0 is a *bit* smoother, since it's not called upon to pick up seasonal effects).

The model's predictions for the new cases series is again done with

⁷This is right at the first peak in cases. 2020:04:26 at 48529 is greater than the sum of its surrounding days (21352 and 26857) which is a far greater difference than is reasonable for just the daily effects. An alternative to a temporary increase in variance is to smooth out the raw numbers, which clearly have a strong “batching” component.

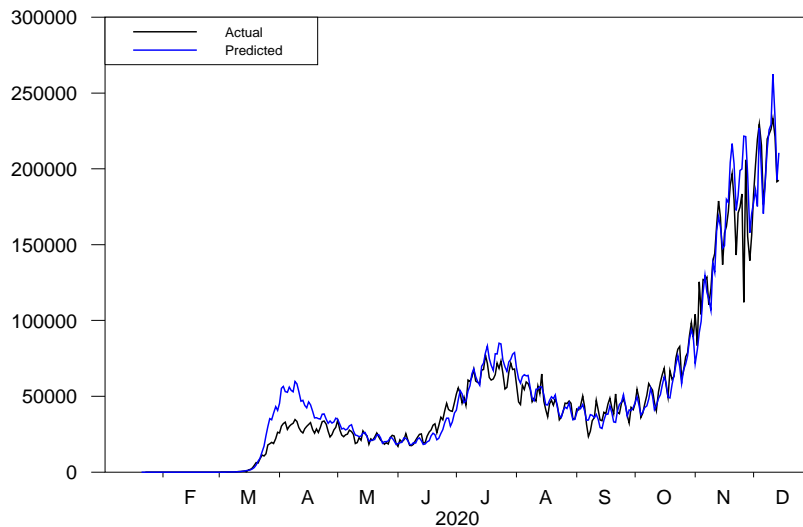


Figure 3.11: Actual and Predicted New Cases

```
set yhatx = yhat(t) (1)
graph(footer="Actual and Predicted New Cases", $
      key=upleft, klabels=|| "Actual", "Predicted" ||) 2
# new_cases
# yhatx
```

which yields Figure 3.11. Not surprisingly, this tracks the data better since it includes the day-of-week effects. A more direct analogue of Figure 3.10 is obtained by looking at the filtered estimates (Figure 3.12) of EI , which is the underlying signal:

```
set eihat = xstates(t) (1)
graph(footer="Actual New Cases and Model New Infections") 2
# new_cases
# eihat
```

The seasonal factors (Figure 3.13) and irregular factors (Figure 3.14) are computed and graphed with

```
set dx = xstates(t) (dpos)
set ux = xstates(t) (ipos)
graph(footer="Seasonal Factors from Combined Model", $
      grid=%weekday(t)==1)
# dx
graph(footer="Irregular Factor from Combined Model")
# ux
```

The filtered seasonal from mid to late March is not particularly well estimated which is why the combined model overpredicts from mid March through early April. (Note that, in real-time, one probably wouldn't even think to include a

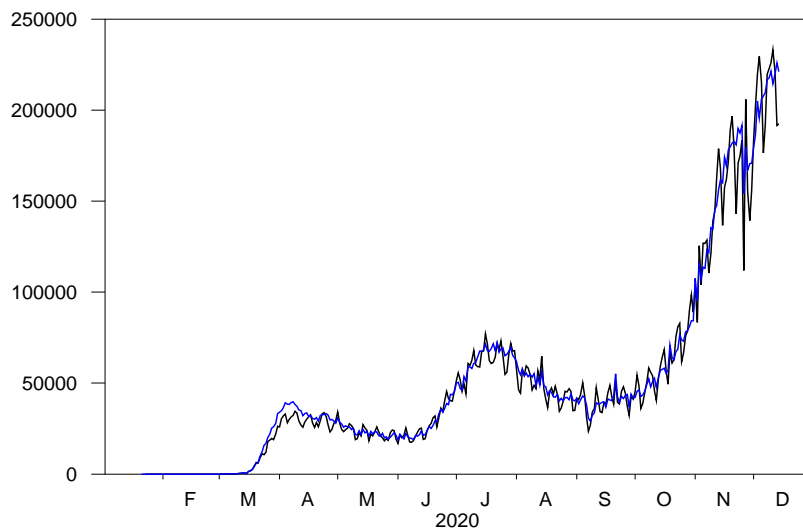


Figure 3.12: Actual New Cases and Model New Infections

seasonal effect until late April when the weekly cycles start to become clear). If a more careful approach to this were desired, it would require giving the seasonal effect a lower variance in the early part of the data set.

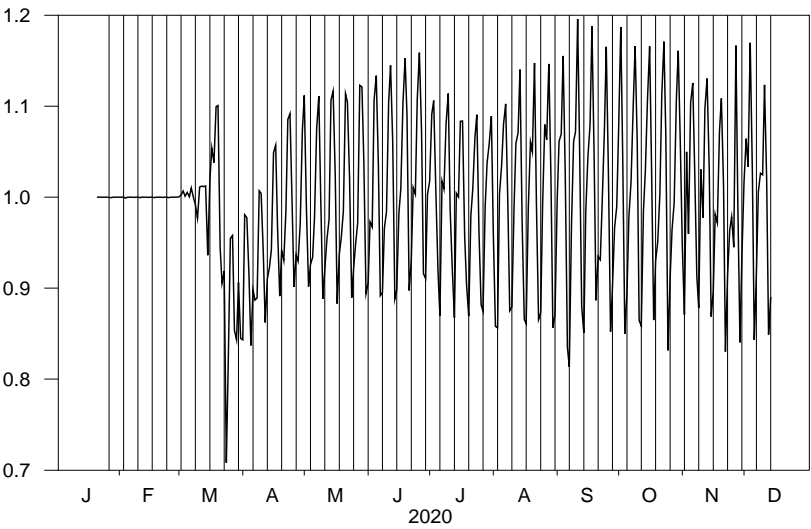


Figure 3.13: Daily Factors from Combined Model

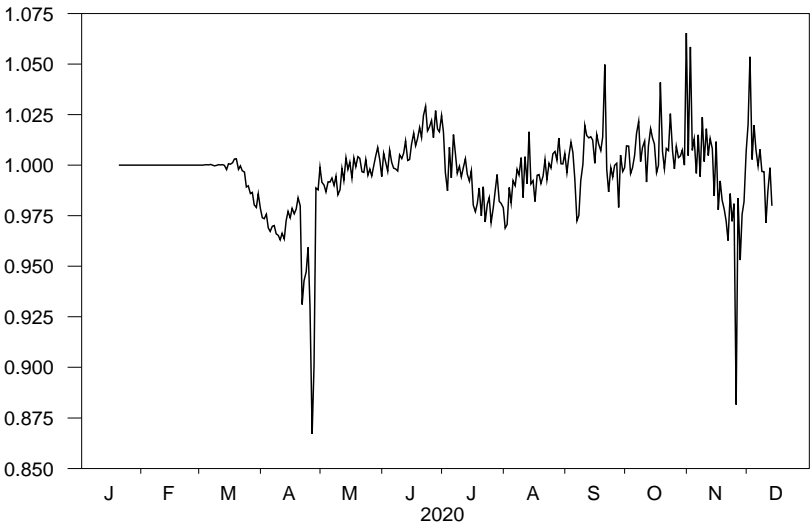


Figure 3.14: Irregular Factors from Combined Model

Example 3.1 State-Space Model with Fixed Beta

This is example `covid.fixedbeta.rpf`, which estimates the SEIR dynamic model using new cases as observable with fixed beta. This is covered in detail in Section 3.1.

Data are from <https://covid.ourworldindata.org/data/owid-covid-data.xlsx>

```
open data "uscovid.xlsx"
calendar(7) 2020:1:21
data(format=xlsx,org=columns,left=4) 2020:01:21 * total_cases $
  new_cases total_deaths new_deaths total_cases_per_million $
  new_cases_per_million total_deaths_per_million $
  new_deaths_per_million total_tests new_tests $
  total_tests_per_thousand new_tests_per_thousand
*
compute pop=330e+6
*
* sigma and gamma come from the settings using in the MIT model:
* https://www.covidanalytics.io/projections_documentation
*
compute sigma=log(2)/5.0 ;* Exponential distribution with 5 day median
compute gamma=log(2)/10.0 ;* Exponential distribution with 10 day median
*
* Starting guess for beta (based upon early estimates of reproduction rate)
*
compute rawR0=3.5
compute beta=rawR0*gamma
*
dec series se ei ir ee ii
*
* Set this up for solving out for the states-space representation (since
* there is simultaneity in the construction). Note that this works only
* if the parameters (like beta) are *not* time-varying.
*
* Write equations as FRML's
*
frml eieq ei = sigma*ee{1}
frml seeq se = beta*ii{1}
frml ireq ir = gamma*ii{1}
frml(identity) eedef ee = ee{1}+se{0}-ei{0}
frml(identity) iidef ii = ii{1}+ei{0}-ir{0}
*
group seirdata eieq seeq ireq eedef iidef
*
* This will solve out the model for a proper state space model (with
* only lags in the definitions).
*
function SolveOut
dsge(model=seirdata,a=a,f=f,cutoff=%na,roots=myroots) ei se ir ee ii
end
*
```

```

* Feasible estimates to use in the variances
*
clear eetilde iitilde
*
* Variances implied by model. (All are truncated below).
*
frml sweifrml = p=sigma,%max(1.0,eetilde*p*(1-p))
frml swifrml = p=gamma,%max(1.0,iitilde*p*(1-p))
frml swsefrml = p=beta*iitilde/pop,%max(10000.0,pop*p*(1-p))
*
* Use our best guess for E and I given data through t-1
*
function MakeTildes x sx
type vector *x
type symm *sx
compute eetilde(t)=x(4)
compute iitilde(t)=x(5)
end
*
function FixSigns x sx
type vector *x
type symm *sx
*
compute x=%plus(x)
end
*
compute [vect] c=%unitv(5,1)
*
* Allow for unknown number of pre-sample exposed and infectious
*
compute [vect] x0=%zeros(5,1)
compute rhoeeii=.9
compute eevar=10000.0,iivar=10000.0
compute [symm] sx0=%diag(||0.0,0.0,0.0||)~\$
%corrto cv(||rhoeeii||,||eevar,iivar||)
*
compute beta=rawR0*gamma
dlm(y=new_cases,start=SolveOut(),fpre=MakeTildes,fpost=FixSigns,$
a=a,f=f,c=c,sw=%diag(||sweifrml,swsefrml,swifrml||),$
x0=x0,sx0=sx0,type=filter,yhat=yhat,svhat=svhat,$
save=seirdlm,likelihoods=ll) * * xstates vstates
*
set ee = xstates(t)(4)
set ii = xstates(t)(5)
graph(footer="Estimate of Exposed")
# ee
graph(footer="Estimate of Infectious")
# ii
*
set yhatx = yhat(t)(1)
graph(footer="Actual and Predicted New Cases",$
key=upleft,klabls=||"Actual","Predicted"||) 2
# new_cases
# yhatx

```

```

*
set svhatx = sqrt(svhat(t)(1,1))
graph(footer="Standard error of prediction")
# svhatx
*
set lldiff = ll-ll{1}
graph(overlay=line,vlabel="Log Likelihood Element",$
      ovlabel="New Cases") 2
# lldiff
# new_cases

```

Example 3.2 State-Space Model with Deterministic Time-Varying Beta

This is example `covid.tvbeta.rpf`, which estimates the SEIR dynamic model using new cases as observable with time-varying deterministic beta. This is described in Section 3.1.

```

open data "uscovid.xlsx"
calendar(7) 2020:1:21
data(format=xlsx,org=columns,left=4) 2020:01:21 * total_cases new_cases $
total_deaths new_deaths total_cases_per_million new_cases_per_million $
total_deaths_per_million new_deaths_per_million total_tests $
new_tests total_tests_per_thousand new_tests_per_thousand
*
compute pop=330e+6
*
* sigma and gamma come from the settings using in the MIT model:
* https://www.covidanalytics.io/projections_documentation
*
compute sigma=log(2)/5.0 ;* Exponential distribution with 5 day median
compute gamma=log(2)/10.0 ;* Exponential distribution with 10 day median
*
* Starting guess for beta (based upon early estimates of reproduction rate)
*
compute rawR0=3.5
compute beta=rawR0*gamma
*
compute d0=21.0,t0=2020:03:23,maxg=.30
set betat = beta*(1+(maxg-1)*(1-exp(-(t-t0)/d0)^2))
*
* Feasible estimates to use in the variances
*
clear eetilde iitilde
*
* Variances implied by model. (All are truncated below at 1)
*
frml sweifrml = p=sigma,%max(1.0,eetilde*p*(1-p))
frml swifrml = p=gamma,%max(1.0,iitilde*p*(1-p))
frml swsefrml = p=betat{0}*iitilde/pop,%max(1.0,pop*p*(1-p))
*

```



```

function MakeTildes x sx
type vector *x
type symm *sx
compute eetilde(t)=x(4)
compute iitilde(t)=x(5)
end
*
function FixSigns x sx
type vector *x
type symm *sx
*
compute x=%plus(x)
end
*
* States are (in order) EI, SE, IR, EE and II.
*
dec rect a(5,5) f(5,3)
compute a=%zeros(5,5)
input f
  1  0  0
  0  1  0
  0  0  1
-1  1  0
  1  0 -1
*
function AMatrix t
type integer t
type rect AMatrix
*
compute a(1,4)=sigma
compute a(2,5)=betat(t)
compute a(3,5)=gamma
compute a(4,4)=1-sigma
compute a(4,5)=betat(t)
compute a(5,4)=sigma
compute a(5,5)=1-gamma
compute AMatrix=A
end
*
compute [vect] c=%unitv(5,1)
*
* Allow for unknown number of pre-sample exposed and infectious
*
compute [vect] x0=%zeros(5,1)
compute rhoeeii=.9
compute eevar=10000.0,iivar=10000.0
compute [symm] sx0=%diag(||0.0,0.0,0.0||)~\$
               %corrto cv(||rhoeeii||,||eevar,iivar||)
*
dlim(y=new_cases,fpre=MakeTildes,fpost=FixSigns,a=AMatrix(t),\$
      f=f,c=c,sw=%diag(||sweifrml,swsefrml,swirfrml||),\$
      x0=x0,sx0=sx0,type=filter,yhat=yhat,svhat=svhat,\$
      likelihoods=ll) * * xstates vstates
*

```

```

set ee = xstates(t) (4)
set ii = xstates(t) (5)
graph(footer="Estimate of Exposed")
# ee
graph(footer="Estimate of Infectious")
# ii
set yhatx = yhat(t) (1)
graph(footer="Actual and Predicted New Cases with Time-Varying Beta", $
    key=upleft, klabels=|| "Actual", "Predicted" ||) 2
# new_cases
# yhatx
set svhatx = sqrt(svhat(t) (1,1))
graph(footer="Standard error of prediction")
# svhatx
graph(footer="Assumed Beta")
# betat

```

Example 3.3 State-Space Model with Endogenous Time-Varying Beta

This is example `covid_endogbeta.rpf`, which estimates the SEIR dynamic model using new cases as observable with an endogenous time-varying beta. The details are in Section 3.3

```

open data "uscovid.xlsx"
calendar(7) 2020:1:21
data(format=xlsx, org=columns, left=4) 2020:01:21 * total_cases new_cases $
    total_deaths new_deaths total_cases_per_million new_cases_per_million $
    total_deaths_per_million new_deaths_per_million total_tests $
    new_tests total_tests_per_thousand new_tests_per_thousand
*
compute pop=330e+6
*
* sigma and gamma come from the settings using in the MIT model:
* https://www.covidanalytics.io/projections_documentation
*
compute sigma=log(2)/5.0 ;* Exponential distribution with 5 day median
compute gamma=log(2)/10.0 ;* Exponential distribution with 10 day median
*
compute rawR0=3.5
*
* Starting guess for beta (based upon early estimates of reproduction rate)
*
compute beta0=rawR0*gamma
*
* The states enter multiplicatively in the EI (and hence E) equations,
* so beta(t-1)I(t-1) has to be linearized. This requires a Z term for
* correcting the value.
*
* States are (in order) EI, SE, IR, BETA, EE and II.
*

```

```

* There are four shocks in this model (since it includes a shock to beta).
*
dec rect a(6,6)
dec vect z(6)
dec rect f(6,4)
*
compute a=%zeros(6,6)
compute z=%zeros(6,1)
*
input f
  1  0  0  0
  0  1  0  0
  0  0  1  0
  0  0  0  1
-1  1  0  0
  1  0 -1  0
*
* Loadings on shocks to states
*
compute c=%unitv(6,1)
*
* Expansion points
*
clear(zeros) iitilde eetilde eetilde betatilde
*
* Poke values into A
*
function AMatrix t
type rect      AMatrix
type integer t
*
compute a(1,5)=sigma
compute a(2,4)=iitilde(t)
compute a(2,6)=betatilde(t)
compute a(3,6)=gamma
compute a(4,4)=1.0
compute a(5,4)=iitilde(t)
compute a(5,5)=1-sigma
compute a(5,6)=betatilde(t)
compute a(6,5)=sigma
compute a(6,6)=1-gamma
compute AMatrix=A
end
*****
function ZVector t
type vector ZVector
type integer t
compute z(2)=-betatilde(t)*iitilde(t)
compute z(5)=z(2)
compute ZVector=z
end
*****
function MakeTildes x sx
type vector *x

```

```

type symm    *sx
compute eetilde(t)=x(1)
compute betatilde(t)=x(4)
compute eetilde(t)=x(5)
compute iitilde(t)=x(6)
end
*****
function FixSigns x sx
type vector *x
type symm    *sx
*
compute x=%plus(x)
end
*
* All states have a presample mean of zero except beta, which uses the
* assumed pre-sample value. beta has a standard deviation of .1, while
* the covariance matrix of pre-sample exposed and infectious is as in
* the simpler examples.
*
compute x0=%zeros(6,1)+beta0*%unitv(6,4)
compute rhoeeii=.9
compute eevar=10000.0,iivar=10000.0
compute [symm] sx0=%diag(||0.0,0.0,0.0||)~\0.01~\$
               %corrto cv(||rhoeeii||,||eevar,iivar||)
*
compute swbeta=1.e-8
*
* Variances implied by model. (All are truncated below at 1)
*
frml sweifrml = p=sigma,%max(1.0,eetilde*p*(1-p))
frml swifrml = p=gamma,%max(1.0,iitilde*p*(1-p))
frml swsefrml = p=betatilde*iitilde/pop,%max(1.0,pop*p*(1-p))
*
* Without adjustment for daily effects
*
dlim(y=new_cases,fpre=MakeTildes,fpost=FixSigns,a=AMatrix(t),f=f,$
     c=c,z=ZVector(t),sw=%diag(||sweifrml,swsefrml,swifrml,swbeta||),$
     x0=x0,sx0=sx0,type=filter,yhat=yhat,svhat=svhat) $
     * * xstates vstates
*
set ee = xstates(t)(5)
set ii = xstates(t)(6)
set betax = xstates(t)(4)
graph(footer="Estimate of Exposed")
# ee
graph(footer="Estimate of Infectious")
# ii
set r0 = betax/gamma
graph(footer="Estimate of R0")
# r0
graph(footer="Estimate of R0")
# r0 2020:04:20 *
set yhatx = yhat(t)(1)
graph(footer="Actual and Predicted New Cases", $

```

```

    key=upleft, klabels=|| "Actual", "Predicted" ||) 2
# new_cases
# yhatx
*
set svhatx = sqrt(svhat(t)(1,1))
graph(footer="Standard error of prediction")
# svhatx

```

Example 3.4 Model with Endogenous Beta and Day-of-Week Effects

This is example `covid_endogwdaily.rpf`, which estimates the SEIR dynamic model using new cases as observable with an endogenous time-varying beta and a multiplicative day-of-week cycle. The details are in Section 3.4.

```

open data "uscovid.xlsx"
calendar(7) 2020:1:21
data(format=xlsx, org=columns, left=4) 2020:01:21 * total_cases new_cases $
    total_deaths new_deaths total_cases_per_million new_cases_per_million $
    total_deaths_per_million new_deaths_per_million total_tests $
    new_tests total_tests_per_thousand new_tests_per_thousand
*
compute pop=330e+6
*
* sigma and gamma come from the settings using in the MIT model:
* https://www.covidanalytics.io/projections_documentation
*
compute sigma=log(2)/5.0 ;* Exponential distribution with 5 day median
compute gamma=log(2)/10.0 ;* Exponential distribution with 10 day median
*
compute rawR0=3.5
*
* Starting guess for beta (based upon early estimates of reproduction rate)
*
compute beta0=rawR0*gamma
*
* The states enter multiplicatively in the EI (and hence E) equations,
* so beta(t-1)I(t-1) has to be linearized. This requires a Z term for
* correcting the value.
*
* States are (in order) EI, SE, IR, BETA, EE and II.
*
* There are four shocks in this model (since it includes a shock to beta).
*
dec rect a(6,6)
dec vect z(6)
dec rect f(6,4)
*
compute a=%zeros(6,6)
compute z=%zeros(6,1)
*

```

```

input f
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
-1 1 0 0
1 0 -1 0
*
* Loadings on shocks to states
*
compute c=%unitv(6,1)
*
* Expansion points
*
clear(zeros) iitilde eetilde eetilde betatilde
*
* Poke values into A
*
function AMatrix t
type rect      AMatrix
type integer t
*
compute a(1,5)=sigma
compute a(2,4)=iitilde(t)
compute a(2,6)=betatilde(t)
compute a(3,6)=gamma
compute a(4,4)=1.0
compute a(5,4)=iitilde(t)
compute a(5,5)=1-sigma
compute a(5,6)=betatilde(t)
compute a(6,5)=sigma
compute a(6,6)=1-gamma
compute AMatrix=A
end
*****
function ZVector t
type vector ZVector
type integer t
compute z(2)=-betatilde(t)*iitilde(t)
compute z(5)=z(2)
compute ZVector=z
end
*****
function FixSigns x sx
type vector *x
type symm    *sx
*
compute x=%plus(x)
end
*
* All states have a presample mean of zero except beta, which uses the
* assumed pre-sample value. The variance of beta is high (4.0), while
* the covariance matrix of pre-sample exposed and infectious is as in
* the simpler examples.

```

```

*
compute x0=%zeros(6,1)+beta0*%unitv(6,4)
compute rhoeeii=.9
compute eevar=10000.0,iivar=10000.0
compute [symm] sx0=%diag(||0.0,0.0,0.0||)~\.01~\$
      %corrto cv(||rhoeeii||,||eevar,iivar||)
*
* Variances implied by model. (All are truncated below at 1)
*
frml sweifrml = p=sigma,%max(1.0,eetilnde*p*(1-p))
frml swifrml = p=gamma,%max(1.0,iitilde*p*(1-p))
frml swsefrml = p=betatilde*iitilde/pop,%max(1.0,pop*p*(1-p))
*
* With multiplicative seasonal and irregular (in measurement equation)
*
set dtilde = 1.0
set utilde = 1.0
*
@SeasonalDLM(span=7,type=additive,a=adaily,c=cdaily,f=fdaily)
compute [vector] zdaily=7.0*%unitv(%rows(adaily),1)
*
compute [rect] airreg=||0.0||
compute [vect] zirreg=||1.0||
compute [rect] firreg=||1.0||
compute [rect] cirreg=||1.0||
*
dec frml[vect] cfs
frml cfs = c*dtilde*utilde~~cdaily*etilnde*utilde~~cirreg*etilnde*dtilde
dec frml mufs
frml mufs = -2.0*dtilde*etilnde*utilde
*
compute nbase=%rows(f)
compute dpos=nbase+1
compute ipos=dpos+6
*
function MakeTildesS x sx
type vect *x
type symm *sx
*
* These need to be the filtered value at the end of t-1
*
compute betatilde(t)=x(4)
compute eetilde(t)=x(5)
compute iitilde(t)=x(6)
*
* These need to be the predicted value at t given t-1
*
compute eetilde(t)=sigma*x(5)
compute dtilde(t)=7-%sum(%xsubvec(x,dpos,dpos+5))
compute utilde(t)=1.0
end
*
compute swbeta=1.e-8
compute scalefac=.04

```

```

*
compute swdaily=.005^2*scalefac
frml swdailyf = swdaily
*
compute swirreg=.01^2*scalefac
frml swirregf = %if(t>=2020:04:22.and.t<=2020:04:28,9.0*swirreg,swirreg)
*
compute [vect] x0daily=%ones(6,1)
compute [symm] sx0daily=%diag(%fill(6,1,.05^2*scalefac))
*
compute [vect] x0irreg=%ones(1,1)
compute [symm] sx0irreg=%diag(%fill(1,1,0))
*
dlim(y=new_cases, fpre=MakeTildesS, fpost=FixSigns, $
  a=AMatrix(t)~\adaily~\airreg, z=ZVector(t)~\zdaily~\zirreg, c=cfs, $
  mu=mufs, f=f~\fdaily~\firreg, x0=x0~\x0daily~\x0irreg, $
  sx0=sx0~\sx0daily~\sx0irreg, $
  sw=%diag(||sweifrml, swseifrml, swifrml, swbeta, swdailyf, swirregf||), $
  type=filter, yhat=yhat, svhat=svhat) * * xstates vstates
*
set ee = xstates(t)(5)
set ii = xstates(t)(6)
set betax = xstates(t)(4)
graph(footer="Estimate of Exposed")
# ee
graph(footer="Estimate of Infectious")
# ii
set r0 = betax/gamma
graph(footer="Estimate of R0")
# r0
graph(footer="Estimate of R0")
# r0 2020:04:20 *
set yhatx = yhat(t)(1)
graph(footer="Actual and Predicted New Cases", $
  key=upleft, klabels=||"Actual", "Predicted"||) 2
# new_cases
# yhatx
*
set eihat = xstates(t)(1)
graph(footer="Actual New Cases and Model New Infections") 2
# new_cases
# eihat
*
set svhatx = sqrt(svhat(t)(1,1))
graph(footer="Standard error of prediction")
# svhatx
*
set dx = xstates(t)(dpos)
set ux = xstates(t)(ipos)
graph(footer="Seasonal Factors from Combined Model\\Mondays highlighted", $
  grid=%weekday(t)==1)
# dx
graph(footer="Irregular Factor from Combined Model")
# ux

```

Index

covid_endogbeta.rpf, 55
covid_endogwdaily.rpf, 58
covid_fixedbeta.rpf, 51
covid_ssmcycles.rpf, 27
covid_tvbeta.rpf, 53
covid_weeklycycles.rpf, 27

DLM instruction

FPOST option, 32
FPRE option, 23
START option, 33

ESMOOTH instruction, 18

seir_with_errors.rpf, 13
seir_with_tvbeta.rpf, 15
seir_wo_errors.rpf, 12