Experiments for fitting stochastic COVID model

John M. Drake

3/28/2020

Create pomp object with data

Not sure if all initial states have to be set to 1 or greater so do thatfor testing purposes. REMEMBER TO UNDO.

```
#data <- read.csv('https://raw.githubusercontent.com/CEIDatUGA/COVID-19-DATA/master/GA_daily_status_rep
data = read.csv('GA_daily_status_report_GDPH.csv')
data$date <- as.Date(data$date, format='%m/%d/%y')

covid_ga_data <- data %>% dplyr::select(date, cases_cumulative, fatalities_cumulative) %>%
                          tidyr::replace_na(list(cases_cumulative = 0, fatalities_cumulative = 0)) %>%
                          dplyr::mutate(days = 1:nrow(data)) %>%
                          dplyr::mutate(cases = cases_cumulative) %>%
                          dplyr::select(days, cases)

#extend data frame holding data by a month so that pomp runs simulations for that long
future = data.frame(days = max(covid_ga_data$days):(max(covid_ga_data$days)+31), cases = NA )
covid_ga_data = rbind(covid_ga_data,future)
covid_ga_data$cases <- covid_ga_data$cases+1   # Add one so there are no zeros when fitting
```

Code model. Notice we change the parameterizatio of beta factor so `beta_redfactor`> 0 rather than `beta_red_factor` > 1. Assume negative binomial observation model with fit parameter. NEED TO SWITCH TO EULER MULTINOMIAL MODEL AT LINE 129

```
## measurement model C snippets
rmeas <- "
  cases = rnbinom_mu(theta, rho * C);
"

dmeas <- "
  lik = dnbinom_mu(cases, theta, rho * C, give_log);
"

## initializer
rinit <- "
  S = 10600000;
  E1 = 35;
  E2 = 35;
  E3 = 35;
  E4 = 35;
  E5 = 35;
  E6 = 35;
  I1 = 14;
  I2 = 14;
  I3 = 14;
  I4 = 14;
  Iu1 = 111;
  Iu2 = 111;
  Iu3 = 111;
```

```
  Iu4 = 111;
  C = 1;
  Ru = 1;
"

## rprocess
covid_step_C <- "
  double trans[17]; //C indexes at 0, I hate that so I'm making things 1 bigger and start with index 1,
  double Epresymptom;
  double Idetected;
  double Iundetected;
  double foi; //force of infection
  double gamma; // rate of transition through I compartments
  double detect_frac; //fraction of those that get eventually diagnosed

  Epresymptom = E1+E2+E3+E4+E5+E6;  //all pre-symptomatic
  Idetected = I1+I2+I3+I4;          //all symptomatic that wil be detected
  Iundetected = Iu1+Iu2+Iu3+Iu4;    //all symptomatic/asymptomatic that won't be detected

  //force of infection
  //time dependent transmission, multiplied by different groups
  //each group can have its own transmission rate
  //t_int1 days after simulation start, an intervention reduces transmission rate by some factor
  //t_int1 is new, not in original code. There it was assumed to be the same as t_int2

  if (t<=t_int1)
        foi = beta_d*Idetected + beta_u*Iundetected + beta_e*Epresymptom;
  else
        foi = (beta_red_factor+1)*(beta_d*Idetected + beta_u*Iundetected + beta_e*Epresymptom);


  //time-dependent rate of movement through infected and detected classes
  //t_int2 days after simulation start, the time at which individuals are diagnosed and thus the time sp
  //t_int2 is caled z in original code

  if (t<t_int2) //if time is less then intervention time, duration spent in I is given by 1/gamma_u, otl
      gamma = gamma_u;
  else
      gamma = gamma_d;

  //time dependent fraction of those that move into detected category at the end of the E phase
  //t_int3 days after simulation start, the fraction detected (those that move into I instead of Iu afte
  //note that both higher fraction detected and faster rate of detection speed up arrival of individuals
  //t_int3 is called w in original code, detect_frac is called q/q0/q1 in the original code

  if (t<t_int3)
    detect_frac = detect_frac_0;
  else
    detect_frac = detect_frac_1;

  // define all transmission rates
  trans[1] = rbinom(S,1-exp(-foi*dt));              //transition from S to E
  trans[2] = rbinom(E1,1-exp(-sigma*dt));           // transition between E compartments 1/2
```

```
  trans[3] = rbinom(E2,1-exp(-sigma*dt));           // transition between E compartments
  trans[4] = rbinom(E3,1-exp(-sigma*dt));           // transition between E compartments
  trans[5] = rbinom(E4,1-exp(-sigma*dt));           // transition between E compartments 4/5
  trans[6] = rbinom(E5,1-exp(-sigma*dt));           // transition between E compartments 5/6

  trans[7] = rbinom(E6,(1-exp(-sigma*dt))*detect_frac);           // transition between E6 compartment a
  trans[8] = rbinom(E6,(1-exp(-sigma*dt))*(1-detect_frac));           // transition between E6 compartme
  trans[9] = rbinom(I1,1-exp(-gamma*dt));           // transition between I compartments 1/2
  trans[10] = rbinom(I2,1-exp(-gamma*dt));           // transition between I compartments 2/3
  trans[11] = rbinom(I3,1-exp(-gamma*dt));           // transition between I compartments 3/4
  trans[12] = rbinom(I4,1-exp(-gamma*dt));          // transition between I compartments and C

  trans[13] = rbinom(Iu1,1-exp(-gamma_u*dt));           // transition between Iu compartments 1/2
  trans[14] = rbinom(Iu2,1-exp(-gamma_u*dt));           // transition between Iu compartments 2/3
  trans[15] = rbinom(Iu3,1-exp(-gamma_u*dt));           // transition between Iu compartments 3/4
  trans[16] = rbinom(Iu4,1-exp(-gamma_u*dt));           // transition between Iu compartments and Ru

  // define all transmissions for each compartment
  S -= trans[1];

  E1 += trans[1] - trans[2];
  E2 += trans[2] - trans[3];
  E3 += trans[3] - trans[4];
  E4 += trans[4] - trans[5];
  E5 += trans[5] - trans[6];
  E6 += trans[6] - trans[7] - trans[8];

  I1 += trans[7] - trans[9];
  I2 += trans[9] - trans[10];
  I3 += trans[10] - trans[11];
  I4 += trans[11] - trans[12];

  Iu1 += trans[8] - trans[13];
  Iu2 += trans[13] - trans[14];
  Iu3 += trans[14] - trans[15];
  Iu4 += trans[15] - trans[16];

  C += trans[12]; //detected cases, assumed to be isolated and not further contribute to transmission
  Ru += trans[16]; //undetected cases that recover, assumed to not further contribute to transmission
"

varnames = c("S", "E1", "E2", "E3", "E4", "E5", "E6", "I1", "I2", "I3", "I4", "Iu1", "Iu2", "Iu3", "Iu4"
#parameter and variable names
parnames1 = c("beta_d", "beta_u", "beta_e", "beta_red_factor", "t_int1", "t_int2", "t_int3", "gamma_u",
#initial conditions of state variables are also parameters
parnames2 = c("E1_0", "E2_0", "E3_0", "E4_0", "E5_0", "E6_0", "I1_0", "I2_0", "I3_0", "I4_0", "Iu1_0",
parnames = c(parnames1,parnames2)
inivals = c(S_0 = 10600000, E1_0 = 35, E2_0 = 35, E3_0 = 35, E4_0 = 35, E5_0 =35, E6_0 =35, I1_0 = 14,
Ntot = sum(inivals)

#values for parameters. beta is scaled by population size here instead of inside the simulation functio
parvals = c(beta_d = 5e-7, beta_u = 0.25/Ntot, beta_e = 0.1/Ntot, beta_red_factor = 0.5, t_int1 = 12, t
```

```
pomp(
  data= covid_ga_data[1:25,],    # have to remove the NA to fit model, i.e. can only fit to the current
  times=seq(1,25,by=1), t0=0,
 # covar=covariate_table(birthdat,times="time"),  # save this as a model for including covariates in th
  dmeasure = Csnippet(dmeas),
  rmeasure = Csnippet(rmeas),
  rinit=Csnippet(rinit),
  rprocess = euler(
    step.fun = Csnippet(covid_step_C),
    delta.t = 1/20
  ),
  partrans=parameter_trans(
    log=c("beta_d", "beta_u", "beta_e", "beta_red_factor", "t_int1", "t_int2", "t_int3", "gamma_u", "ga
    logit=c("detect_frac_0", "detect_frac_1")
  ),
  statenames = varnames,
  obsnames=c("cases"),
  paramnames = parnames,
  accumvars = c("C"),
  params = c(parvals, inivals)
) -> covid_ga_model
```

Set up parallelization

```
#ncores <- detectCores()
registerDoParallel()
```

Fit with MIF.

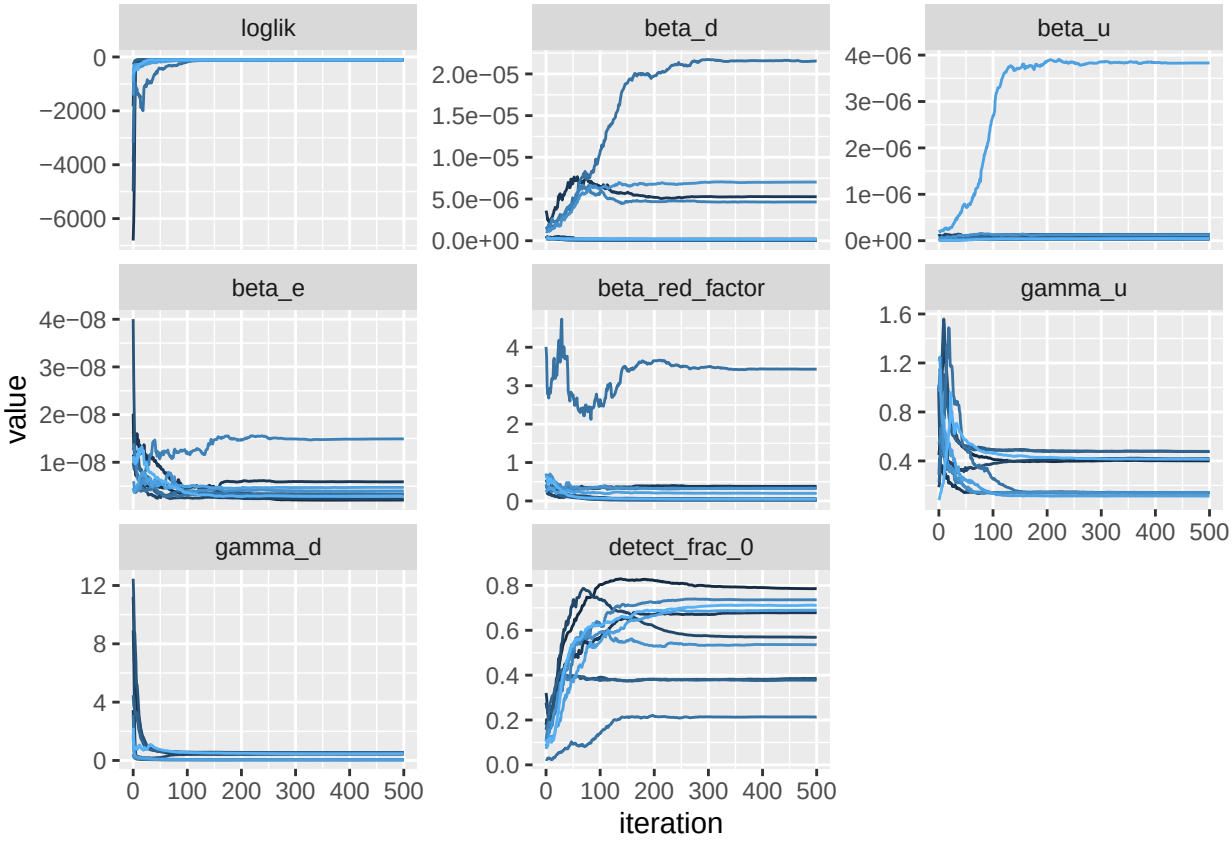
```
registerDoRNG(03292020)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
theta.guess <- theta.true <- coef(covid_ga_model)
estpars <- c("beta_d", "beta_u",  "beta_e", "beta_red_factor", "gamma_u", "gamma_d", "detect_frac_0")

foreach (i = 1:10, .combine = c) %dopar% {   #Inspect from multiple, randomly chosen starting points
  theta.guess <- theta.true
  theta.guess[estpars] <- rlnorm(n = length(estpars),
     meanlog = log(theta.guess[estpars]), sdlog = 1)
  mif2(covid_ga_model, Nmif = 500, params = theta.guess,
  Np = 2000, cooling.fraction = 0.5,
  rw.sd = rw.sd(beta_d = 0.02, beta_u = 0.02, beta_e = 0.02, beta_red_factor = 0.02, gamma_u = 0.02, ga
} -> mifs

# Use particle filter to get the likelihood at the end of MIF run
pf1 <- foreach(mf = mifs, .combine = c) %dopar% {
  pf <- replicate(n = 10, logLik(pfilter(mf, Np = 10000)))
  logmeanexp(pf)
}

# Pick the best parameter set
mf1 <- mifs[[which.max(pf1)]]
theta.mif <- coef(mf1)
```

Plotting

```
mifs %>%
  traces() %>%
  melt() %>%
  filter(variable=="loglik" | variable=="beta_d" | variable=="beta_u" | variable=="beta_e" | variable==
  ggplot(aes(x=iteration,y=value,group=L1,color=L1))+
  geom_line()+
  facet_wrap(~variable,scales="free_y")+
  guides(color=FALSE)
```



Here we set up Bayesian analysis. The first code chunk records various attempts to set up the priors. None of these worked, but not really sure why – leaving here for the time being in case I want to come back.

```
# hyperparams <- list(min = theta.mif[1:31]*0.99, max = theta.mif[1:31]*1.01)
# hyperparams$min[15:31] <- ceiling(hyperparams$min[15:31] )  # integerize state variables
# hyperparams_est <- list(min=hyperparams$min[estpars], max=hyperparams$min[estpars])  # just the param
#
# covid.dprior <- function (params, ..., log) {
#   f <- sum(dunif(c(parvals, inivals), min = hyperparams_est$min, max = hyperparams_est$max, log = TRU
#   if (log) f else exp(f)
# }

# covid.dprior <- function(params, ..., log=TRUE){
#     f <- dlnorm(params["beta_d"], meanlog=log(5.828e-7), sdlog=1, log=TRUE) +
#       dunif(params["beta_red_factor"], 0, 1, log=TRUE) +
#       dgamma(params["gamma_u"], scale=0.25, shape=8, log=TRUE) +
#       dunif(params["detect_frac_0"], min=0, max=0.3, log=TRUE)
#     if (log) f else exp(f)
```

```
# }
#
# covid_ga_model <- pomp(covid_ga_model, dprior = covid.dprior, paramnames=c("beta_d","beta_red_factor"
```

Here we set up a pMCMC closely following the tutorial at https://kingaa.github.io/pomp/vignettes/oaxaca.html#particle_mcmc. Note: I tried to set the a gamma ditributed prior for γ_u but received errors using unnamed arguments, so switched to lognormal. As written, `dmeas` returning illegal values, perhaps pecause euler multnomial misspectified at lines 129 and 130.

```
registerDoRNG(03292020)

priorDens <- "
  lik = dlnorm(beta_d,log(2e-6),1,1)+
    dlnorm(beta_u,log(5e-8),1,1)+
    dlnorm(beta_e,log(5e-8),1,1)+
    dunif(beta_red_factor,0.01,1,1)+
    dlnorm(gamma_u, log(0.5), 1, 1)+
    dlnorm(gamma_u, log(0.5), 1, 1)+
    dunif(detect_frac_0, 0.01, 0.6, 1);
  if (!give_log) lik = exp(lik);
"

tic <- Sys.time()   # set a timer

# for the mcmc proposal subroutine
rw.sd <- rep(0.01, length(estpars))    # set noise level in parameter random walk; used only when propo
names(rw.sd) <- estpars

n <- 10   # number of mcmc chains

foreach(i = 1:n, .combine = c) %dopar% {
  pmcmc(pomp(mf1, dprior = Csnippet(priorDens),
             paramnames=c("beta_d", "beta_u", "beta_e", "beta_red_factor", "gamma_u", "gamma_d", "detec
        Nmcmc = 500, Np = 1000,
        #proposal = mvn.rw.adaptive(rw.sd, scale.start=1, shape.start=1))        # alternate noise mode
        proposal = mvn.diag.rw(rw.sd))      # alternate noise model
} -> pmcmc1

toc <- Sys.time()
print(toc-tic)

plot(pmcmc1)
```

The following is a placeholder for fiting via ABC. CODE NOT WORKING.

```
foreach (i = 1:10, .combine = c) %dopar% {
  theta.guess <- theta.true
  theta.guess[estpars] <- rlnorm(n = length(estpars),
     meanlog = log(theta.guess[estpars]), sdlog = 1)

  # the following lines need to be rewritten for abc
  mif2(covid_ga_model, Nmif = 50, params = theta.guess,
  Np = 2000, cooling.fraction = 0.7,
  rw.sd = rw.sd(beta_d = 0.02))
```

```
  abc(pomp)

} -> abc
```

Here is a fully worked example from pomp that demonstrates MIF, synthetic likelihood (a la Wood 2010), and pMCMC.

```
loc <- url("https://kingaa.github.io/pomp/vignettes/parus.csv")
dat <- read.csv(loc)
head(dat)
parus <- pomp(dat,times="year",t0=1959)

stochStep <- Csnippet("
  N = r*N*exp(-c*N+rnorm(0,sigma));
")

pomp(
  parus,
  rprocess=discrete_time(step.fun=stochStep,delta.t=1),
  rinit=Csnippet("N = N_0;"),
  paramnames=c("r","c","sigma","N_0"),
  statenames=c("N")
) -> parus

rmeas <- Csnippet("pop = rpois(phi*N);")
dmeas <- Csnippet("lik = dpois(pop,phi*N,give_log);")

pomp(parus,
     rmeasure=rmeas,
     dmeasure=dmeas,
     statenames=c("N"),
     paramnames=c("phi")
) -> parus

coef(parus) <- c(N_0=2,r=20,c=1,sigma=0.1,phi=200)

mif2(parus, Nmif=30, Np=1000,
     cooling.fraction.50=0.8,cooling.type="geometric",
     rw.sd=rw.sd(r=0.02,sigma=0.02,phi=0.02,N_0=ivp(0.1))
) -> mf

plot(mf)

pf <- replicate(5, pfilter(mf,Np=1000))
ll <- sapply(pf,logLik)
logmeanexp(ll,se=TRUE)

probe(mf, nsim=200,
      probes=list(
        mean=probe.mean("pop"),
        sd=probe.sd("pop"),
        probe.acf("pop",transform=sqrt,lags=c(1,2)),
        probe.quantile("pop",prob=c(0.2,0.8))
      )) -> pb
```

```
plot(pb)

probe_objfun(pb, nsim=200, est = c("N_0","r"),
  seed = 669237763L) -> pm
subplex(par=coef(pm,c("N_0","r")),fn=pm) -> fit
pm(fit$par)
summary(pm)

priorDens <- "
  lik = dnorm(sigma,0.2,1,1)+
    dnorm(phi,200,100,1)+
    dexp(r,0.1,1);
  if (!give_log) lik = exp(lik);
"

pmcmc(pomp(mf, dprior=Csnippet(priorDens),
           paramnames=c("sigma","phi","r")),
      Nmcmc = 500, Np = 1000,
      proposal = mvn.diag.rw(
        rw.sd=c(N_0=0.1, sigma=0.02, r=0.02, phi=0.02)
      )) -> pmh

plot(pmh,pars=c("loglik","log.prior","N_0","sigma","r","phi"))
```