Analysis of simulated GISP data

Scott Olesen

The simulated data are in the `data/` folder. The columns show:

- the clinic (of 40 total) that the isolate was collected from
- the year of isolation
- the month of isolation (0 = January, 11 = December)
- the 2-fold dilution `y`
- the months since the start of the dataset `t`
- a combined clinic/year label

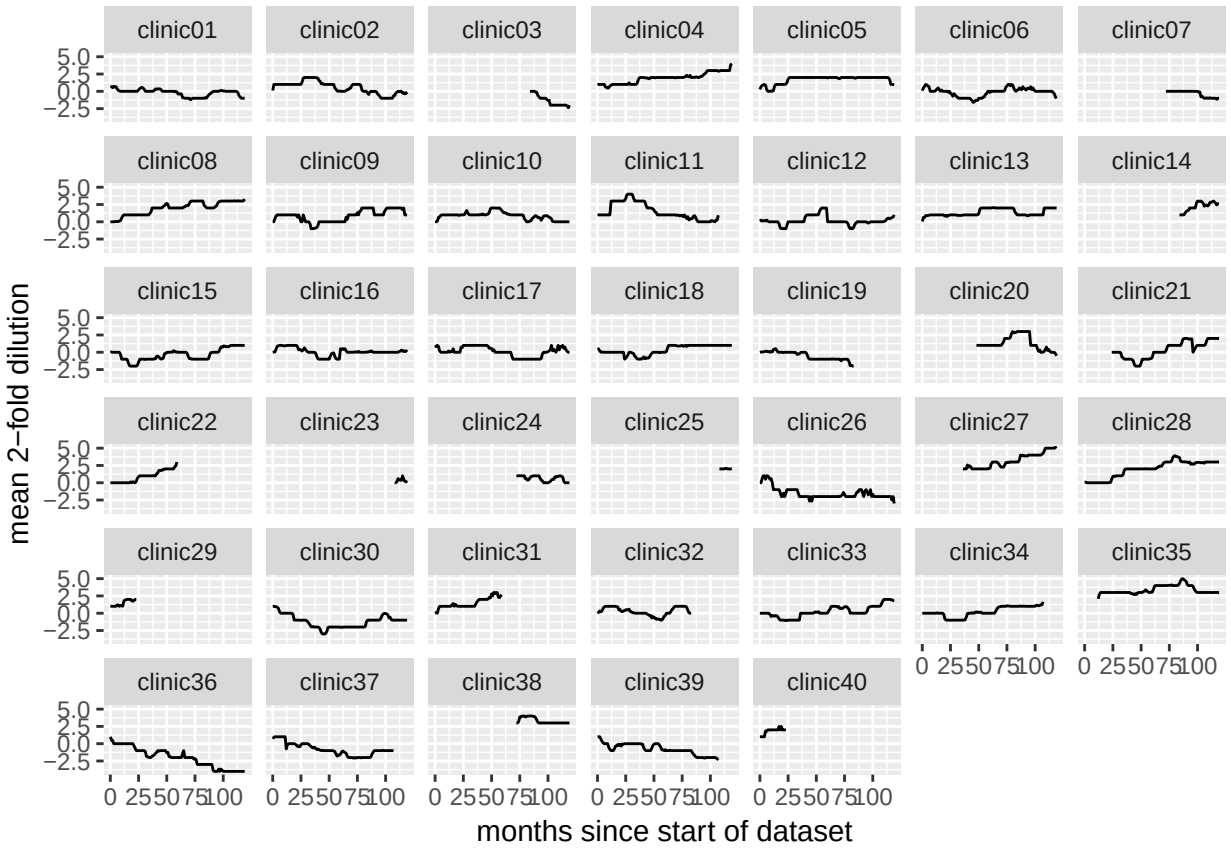
```
dat = read_tsv('data/simulated_gisp_data.tsv') %>%
  mutate(t = (year - min(year)) * 12 + month,
         clinic_year = str_c(clinic, '_', year))

dat
```

```
## # A tibble: 56,047 x 7
##    isolate_id clinic    year month     y     t clinic_year
##         <int> <chr>    <int> <int> <int> <dbl> <chr>
##  1          1 clinic01     1     0     1    0. clinic01_1
##  2          2 clinic01     1     0     0    0. clinic01_1
##  3          3 clinic01     1     0     0    0. clinic01_1
##  4          4 clinic01     1     0     1    0. clinic01_1
##  5          5 clinic01     1     0     1    0. clinic01_1
##  6          6 clinic01     1     0     1    0. clinic01_1
##  7          7 clinic01     1     0     1    0. clinic01_1
##  8          8 clinic01     1     0     1    0. clinic01_1
##  9          9 clinic01     1     0     1    0. clinic01_1
## 10         10 clinic01     1     0     1    0. clinic01_1
## # ... with 56,037 more rows
```

The data were simulated to reproduce some of the features of the original GISP dataset. Notably, there are secular trends in the data, which differ between clinics, and the seasonal pattern is not obvious at all to the eye. The plot shows the monthly average 2-fold dilutions in each clinic and month.

```
dat %>%
  group_by(clinic, t) %>%
  summarize(y = mean(y)) %>%
  ggplot(aes(t, y)) +
  facet_wrap(~ clinic) +
  geom_line() +
  xlab('months since start of dataset') +
  ylab('mean 2-fold dilution')
```



Note that these data are *not* intended to reflect the actual values in the GISP data. They are only for demonstration of the methods; *not* for showing how the precise results of the paper came about.

The nonlinear model uses R's `nls` function. There are many clinic/year combinations, so rather than enumerating them all in a formula passed to nls, we use `model.matrix` and matrix multiplication `%*%`, storing the slope and intercept estimates in vectors, rather than as individual numbers.

The slope values correspond to the $B_{c(i)}$ and the intercepts to the $C_{c(i)}$ terms in the paper.

```
# number of clinic/year combinations in the data
n_cys = length(unique(dat$clinic_year))

# create a model matrix: rows represent data rows, columns represent each of
# the clinic/years. Most entries are 0; 1 means that this data row is from the
# corresponding clinic/year.
model_matrix = model.matrix(~ 0 + clinic_year, dat) %>%
  set_colnames(str_replace(colnames(.), '^clinic_year', ''))

stopifnot(dim(model_matrix) == c(nrow(dat), n_cys))

# as a first guess for clinic/year intercepts, use the mean values
start_intercepts = dat %>%
  group_by(clinic_year) %>%
  summarize(y = mean(y)) %>%
  arrange(clinic_year) %T>%
  # check that the order of these values matches the model matrix columns
  { stopifnot(all(.$clinic_year == colnames(model_matrix))) } %>%
  pull(y)
```

```r
# as a first guess for clinic/year slopes, just use zero
start_slopes = rep(0, n_cys)

# sinusoidal + linear fit function
omega = 2 * pi / 12
fit_f = function(month, A, phase, slope, intercept) {
  intercept_term = drop(model_matrix %*% intercept)
  slope_term = drop(model_matrix %*% slope) * month
  A * sin(omega * (month - phase)) + slope_term + intercept_term
}

# check if the model has been run previously. if it has, don't bother running
# it again, since it takes a few minutes.
model_fn = 'gisp_model.rds'
if (file.exists(model_fn)) {
  model = read_rds(model_fn)
} else {
  # fit the model
  model = nls(y ~ fit_f(month, A, phase, slope, intercept),
              start = list(A = 0.5, phase = 0.0, intercept = start_intercepts, slope = start_slopes),
              data = dat)

  # then save it
  write_rds(model, model_fn)
}
```

We interrogate the `model` object to get the point estimates, standard errors, and confidence intervals for the amplitude A and phase terms. The data were generated with amplitude $A = 0.1$ and phase 1.

```r
summary(model)$coefficients[1:2, ]
```

```
##         Estimate Std. Error  t value      Pr(>|t|)
## A     0.09384033 0.00213666 43.91917  0.000000e+00
## phase 0.95352188 0.04288257 22.23565 4.667022e-109
```

```r
confint.default(model)[1:2, ]
```

```
##            2.5 %    97.5 %
## A     0.08965255 0.0980281
## phase 0.86947358 1.0375702
```