CurveFit Tool for Covid-19 Model Estimation

Abstract

This Appendix gives details for the mathematical and statistical models and fitting procedure used to obtain the estimates for the Covid-19 death rates. The general model is summarized in the introduction, followed by additional details and specifications. The modeling framework is by design extendable, as more data and tools become available.

1. Introduction

This appendix describes the curve-fitting package `CurveFit` developed to estimate death rates from all available locations in order to allow forecasts and estimates to support planning and healthcare.

`CurveFit` supports customized parametrized curves that can be fit to data. We focused on parametrized forms (in contrast to nonparametric inference, e.g. splines [3]) for several reasons:

- Parametric functions capture key signals from noisy data due to simple parametrization.
- Parameters are interpretable, and can be modeled using covariates in a transparent way.
- Parametric forms allow for more stable inversion approaches, for current and future work.
- Parametric functions impose rigid assumptions that make forecasting more stable.

For the Covid model, we considered sigmoidal shapes, described in Section 2. The basic curve lays the foundation for the analysis, while assumptions on noise and relationships between locations are specified through the statistical model, discussed in Section 3. Assumptions and expert knowledge can be communicated to the model through priors and constraints, detailed in Section 4. All estimation is carried out using an optimization procedure, described in Section 5. Finally, posterior uncertainty is estimated from the fits, as described in Section 6. Current settings used to obtain fits are summarized in Section 7.

2. Functional Form for Covid-19

We considered several functional forms to model the death rate of the Covid-19 virus. Based on both currently available data, the log rate starts slowly, increases quickly, and then flattens out again as either social distancing or saturation goes into effect. This is the classic sigmoid shape. We first tried building the analysis using the sigmoidal function

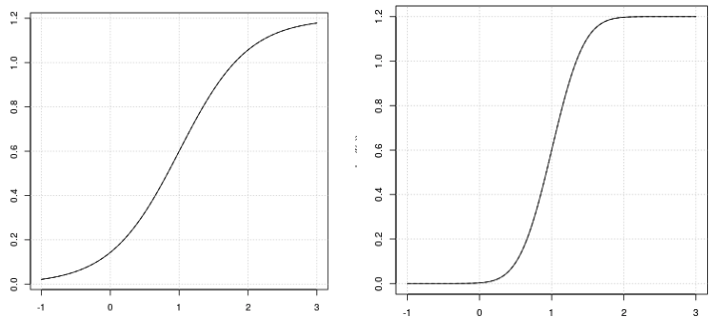

Figure 1. Expit function $\widetilde{D}$ (left) and ERF function D (right). The ERF function fits the available Covid-19 data better than Expit.

$$\widetilde{D}(t;\alpha,\beta,p) = \frac{p}{1+\exp(-\alpha(t-\beta))}$$

where p controls the level, β the shift, and α the growth. We quickly discovered that the ERF error function provided a better fit to the data:

$$D(t;\alpha,\beta,p) = \frac{p}{2}\left(\Psi(\alpha(t-\beta)\right) = \frac{p}{2}\left(1+\frac{2}{\sqrt{\pi}}\int_0^{\alpha(t-\beta)}\exp\left(-\tau^2\right)d\tau\right)$$

`CurveFit` allows the user to specify an arbitrary functional form, so that other models could be considered as more data becomes available. The tool also allows fitting in four spaces:

- Log space: log (data) vs. $\log(D)$
- Linear space: data vs. D
- Derivative of log space: increments of log data vs. derivative of $\log(D)$
- Derivative space linear space: increments of data vs. derivative of D.

For the current model we fit in log space, that is, log of the observations to the log of D. For the D functional form, the three main parameters are:

- Level: p controls the maximum asymptotic level that the rate can reach
- Slope: α controls the speed of the infection
- Inflection: β is the time at which the rate of change of D is maximal.

Since data are available across multiple locations, we have to consider the relationships of these parameters by location and covariates. These specifications are discussed in the next section.

3. Statistical Specification

Statistical assumptions link parameters together across locations. `CurveFit` allows any parameter to be specified using both a link function and covariates using the generalized linear modeling framework [2]:

$$\text{parameter} = \text{link}(\text{covariate} * \text{multiplier} + \text{random effect}).$$

For the Covid-19 model, the only link functions that are used are the identity (for β) and the exponential function to ensures the slope α and the level p are non-negative.

The ability to parametrize by covariates is a key functionality of the model. For example, the only covariate used in the death rate model is the duration between a threshold of the rate and social distancing policy, and this covariate drives the inference of the covariate multiplier for the inflection point or the level in the models we consider for the analysis. As more data becomes available, `CurveFit` can be used to incorporate additional understanding to further link the covariates.

To finish the specification, we give to examples of the models we consider. For the first example, we let the covariate link the inflection points β_i across locations:

$$\begin{aligned} \log(y_i^t) &= \log(D(t; \text{AlphaLink}_i, \text{BetaLink}_i, \text{PLink}_i)) + \epsilon_i^t \\ \text{AlphaLink}_i &= \exp(\text{intercept} + \text{random effect}) \\ \text{BetaLink}_i &= \gamma\text{Covariate} + \text{random effect} \\ \text{PLink}_i &= \exp(\text{intercept} + \text{random effect}) \end{aligned} \tag{1}$$

In the second example, we consider the level p linked through the covariate and the specification changes to

$$\begin{aligned} \log(y_i^t) &= \log(D(t; \text{AlphaLink}_i, \text{BetaLink}_i, \text{PLink}_i)) + \epsilon_i^t \\ \text{AlphaLink}_i &= \exp(\text{intercept} + \text{random effect}) \\ \text{BetaLink}_i &= \text{intercept} + \text{random effect} \\ \text{PLink}_i &= \gamma\text{Covariate} + \text{random effect} \end{aligned} \tag{2}$$

Estimating this model requires solving for the value of the γ multiplier along with intercepts and effects given statistics on errors.

4. Specifying Priors and Constraints

The `CurveFit` tool lets the user specify prior knowledge using two interfaces: Bayesian priors and constraints. Both types of information can be used to inform estimation of all parameters and covariate multipliers.

Constraints are assumed to be simple bound constraints, that is, we can specify

$$\text{lower bound} \leq \text{parameter} \leq \text{upper bound}$$

for any parameter we wish to infer. Since the functional form D is highly nonlinear, constraints are particularly useful to stabilize the numerical solution of the inference problem. Constraints can also be thought of as 'uniform priors', that is, they require parameters to stay in a certain range, but do not prescribe any particular value in that range.

At this point `CurveFit` assumes that prior distributions are Gaussian $N(\mu, \sigma^2)$ where the parameter μ encodes the prior belief, while σ^2 specifies confidence in this belief.

5. Optimization Procedure

The final optimization problem includes the GLM specifications such as (1) or (2), along with Gaussian priors and bound constraints. The fitting problem in the current version of `CurveFit` is thus a bound-constrained nonlinear least squares problem.

To solve this optimization problem, we use the L-BFGS-B algorithm [4], implemented in `SciPy`[1].

The L-BFGS-B algorithm requires derivatives of the objective function. We use numerical differentiation, implemented using the complex step method, to compute these derivatives for any user-specified functional form [1].

Since the curves are highly nonlinear, the nonlinear least squares problem is highly nonconvex, and therefore initialization is important. We initialize values of parameters to their location-specific fits, and then run the full optimization model as specified in Section 3 from this starting point.

6. Uncertainty Quantification

The `CurveFit` tool provides draws for individual locations used in the model estimation. Location-specific samples then inform aggregate uncertainty of downstream estimates.

We partition the uncertainty as coming from two sources: fixed effects and random effects. Fixed effects in the model are averages of parameters across locations, and covariate multipliers. Random effects are specific to location.

6.1. Fixed Effects

To estimate the fixed effect uncertainty, we use asymptotic statistics.

For any estimator obtained by solving a nonlinear least squares problem

$$\hat{\theta} = \arg\min_{\theta} := \frac{1}{2\sigma^2}\|y - f(\theta; X)\|^2_{\Sigma^{-1}}$$

we can approximate posterior covariance using the inverse of the Fisher information matrix:

$$\mathcal{I}(\theta) = V[\nabla\mathcal{M}(\theta)] = V[J_\theta^T\Sigma^{-1}(f(\theta; X) - y)] = J_\theta^T\Sigma^{-1}J_\theta$$

where

$$J_{\hat{\theta}} := \nabla_\theta f(\theta; X)|_{\theta=\hat{\theta}} \tag{3}$$

is the Jacobian of $f(\theta)$ evaluated at the computed estimate $\hat{\theta}$. We therefore get

$$V(\hat{\theta}) = \mathcal{I}(\hat{\theta})^{-1} = (J_{\hat{\theta}}^T\Sigma^{-1}J_{\hat{\theta}})^{-1} \tag{4}$$

6.2. Random Effects

To estimate the variance at each location, we first obtain an empirical variance-covariance matrix using the random effect fits by location, denoted by V_0.

Given a location with no observations, its uncertainty will be driven by V_0, which captures the variation across location. However, if a location has data, we can obtain a location-specific fit and uncertainty estimates using the same technique as in Section 6.1 but applied to this location. That is, with the prior V_0, the likelihood changes to

$$\hat{\theta}_i = \arg\min_{\theta} := \frac{1}{2}\theta^T V_0^{-1}\theta + \frac{1}{2\sigma^2}\|y_i - f_i(\theta; X_i)\|^2_{\Sigma_i^{-1}}$$

and then we have

$$V_i(\hat{\theta}) = ((J_i)_{\hat{\theta}}^T\Sigma_i^{-1}(J_i)_{\hat{\theta}} + V_0^{-1})^{-1}. \tag{5}$$

7. Settings Used for Results

To develop the initial results in the current manuscript, we used four main models fit with the `CurveFit` tool.

- Short-range, with covariate on β_j (1)
- Long-range, with covariate on β_j (1)
- Short-range, with covariate on p_j (2)

[1] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-lbfgsb.html

- Long-range, with covariate on p_j (2)

In all models, standard errors were chosen to decrease linearly for later observations compared to earlier observations, since they reflected cumulative rates and were more accurate.

Prior information was input to the model mainly through the covariate multiplier γ. At a high level, for short-range models the multipliers for different locations were allowed to deviate from mean value (estimated using Wuhan data), while for long-range models the Wuhan value was a main driver for the forecasts. Specifically, the variance of the location-specific multipliers was 10 times as high for the short-term model as for the long-term. All random effects corresponding to location specific estimates were modeled as Gaussian. For all models, constraints were placed on inflection point values to occur between $t = 20$ and $t = 80$ across all locations in the training dataset to stabilize the fit. For both short-range and long-range models, draws are created using ensembles for the model types (1) and (2).

To create the final draws, we used simple linear interpolation in log increment space:

$$\boxed{\text{increment of log D} = \lambda(t)[\text{increment of log D (long)}] + (1 - \lambda(t))[\text{increment of log D (short)}]}$$

where

$$\lambda(t) = \min\left(1, \max\left(0, \frac{t - t_s}{t_e - t_s}\right)\right).$$

where t_s and t_e are start and end times for the period. The resulting draw is then constructed by aggregating the joint increments over (t_s, t_e).

Moving forward, the most up-to-date source for model specifications, settings, and options in the tool will be found in the documentation of `CurveFit` tool.

References

[1] J. R. Martins, P. Sturdza, and J. J. Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software (TOMS)*, 29(3):245–262, 2003.

[2] J. A. Nelder and R. W. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384, 1972.

[3] P. Zheng, A. Y. Aravkin, R. Barber, R. J. Sorensen, and C. J. Murray. Trimmed constrained mixed effects models: Formulations and algorithms. *arXiv preprint arXiv:1909.10700*, 2019.

[4] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.