

Nonlinear Regression on Stochastic Data

Robert Gorczyca

The problem of efficient nonlinear least-squares regression of a sinusoidal curve to equity price data evades standard off-the-shelf curve fitting software. Off-the-shelf regression algorithms like Matlab `lsqcurvefit.m` are extremely sensitive to the starting parameters and perform quite poorly without accurate initialization (see Figure 1). The task here undertaken therefore was to develop an efficient method for accurately estimating the parameters of the best-fit curve for initialization of this optimization software. And though my motivation was at first simply the adaptation of extant software to this special purpose, the algorithm I designed routinely outperforms the optimization software initialized by the most reliable and computation-intensive method.

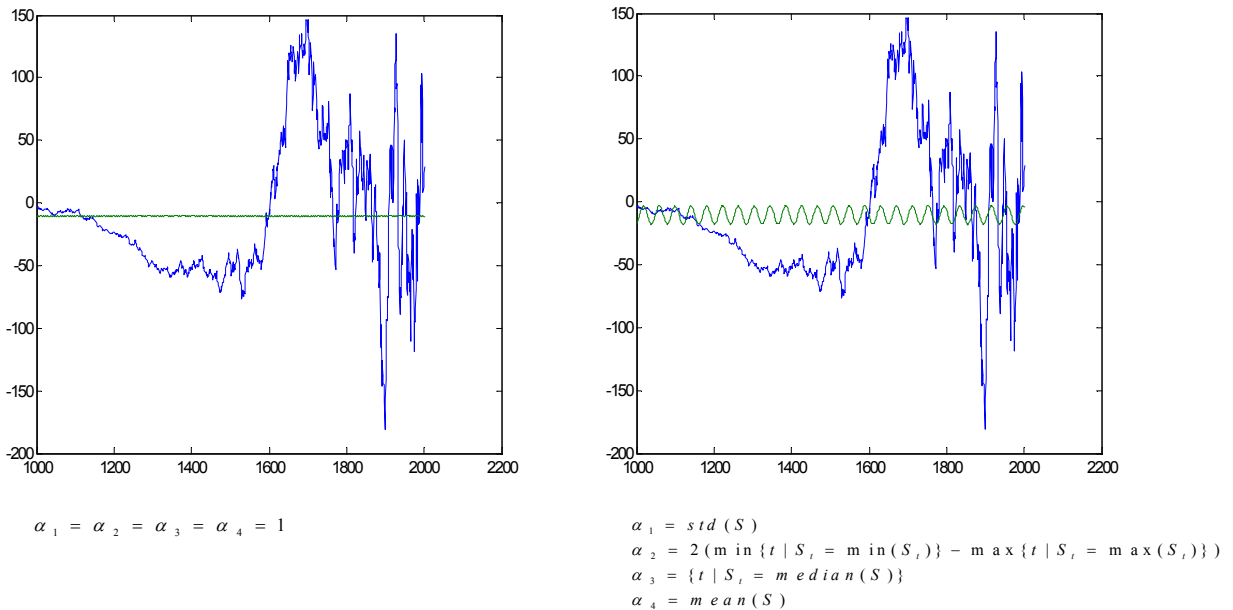


Figure 1: Model fit by MATLAB: `lsqcurvefit.m` poorly initialized

We consider nonlinear regression of the curve

$$y = \alpha_1 \sin(\alpha_2(x + \alpha_3)) + \alpha_4 \quad (1)$$

to a one-dimensional geometric Brownian motion S_t with drift μ and volatility σ given by

$$dS = \mu S dt + \sigma S dW_t$$

$$S_t = S_0 e^{(\mu - \sigma^2/2)t + \sigma dW_t}$$

We generate a random path according to this continuous, continuous-time process to represent an equity price time series observed at points $t = [t_1 \ t_2 \ \dots \ t_i \ \dots \ t_N]^T$. We will utilize the

notational convention $S_t = S_i$. Ex post this construction, we discard the fact that the series is lognormally distributed in favor of the assumption that our price process is governed by exponential growth attributable to appreciation of the underlying asset and by a collection of overlapping oscillatory market forces (external to the firm) of various frequency and intensity. We might interpret high intensity, low frequency forces as business cycles and low intensity, high frequency oscillations as the push and pull buyers and sellers exert daily on prices in the marketplace. We subtract the exponential trend from the data and then use sine curves iteratively to model these periodic forces. For simplicity let us assume that amplitude correlates directly with frequency and that the frequencies of these curves relate multiplicatively. That is to say, the period of each force is k times the period of the next smaller force and $1/k$ times the period of the next larger force for some scaling factor $k > 1$. Furthermore, we will assume that the period of the largest force is proportional by some factor p to the length of the time series. Accordingly, we can estimate the parameters of the sine curve representing the i^{th} largest force by iterating the following algorithm i times:

1. Regress (1) on S for $(\hat{\alpha}_1, \hat{\alpha}_2, \hat{\alpha}_3, \hat{\alpha}_4)$.
2. Subtract $y = \hat{\alpha}_1 \sin(\hat{\alpha}_2(x + \hat{\alpha}_3)) + \hat{\alpha}_4$ from S. Redefine S as this remainder.
3. Partition S into k intervals. Redefine S as the last of these intervals.

In this way we can fit a sum of nonlinear sine curves (a model of many parameters) to an interval of relatively few observations without overfitting. Since the parameters of the larger curves are estimated from regressions on supersets of the current S, they dominate our model of the current S and influence our local regression on the current S, but are only weakly dependent upon our current S. Since our primary purpose for point estimation will be price forecasting, we keep the most recent subset of the data each time we reduce S. By adding the point estimates that our exponential curve and each subsequent sine curve yield on the most recent time interval we obtain a very close fit to S on this interval that has low generalization error to points in the immediate future.

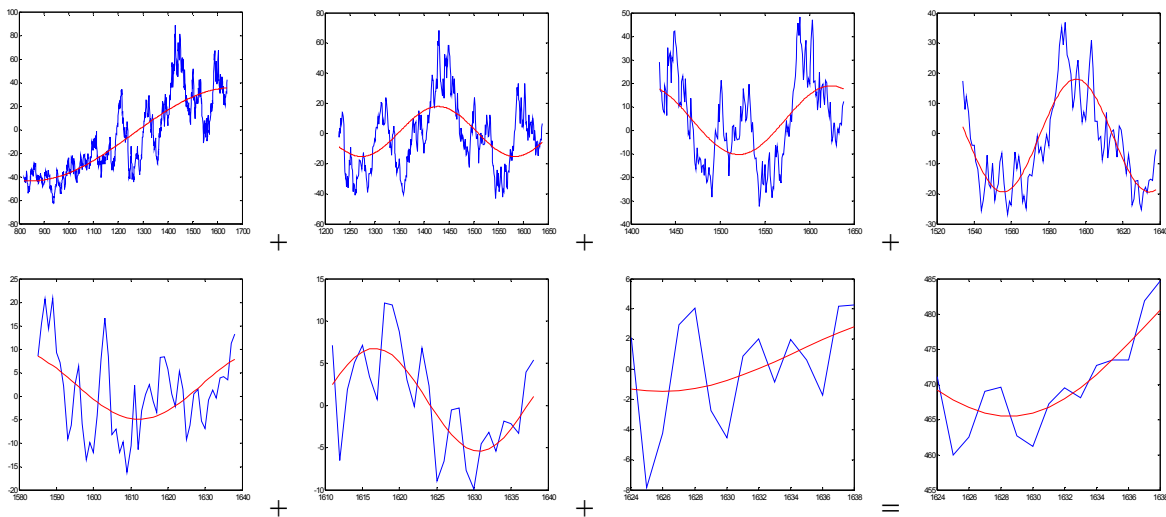


Figure 2: We fit each curve to a subset of the residuals of the previous regression. The sum of the curves models the most recent data closely without overfitting.

The current literature offers no closed-form or universal algorithmic solution to the general nonlinear least-squares problem that encompasses regression of (1) on S . Most specialized curve-fitting techniques rely heavily on initial visual interpretation of data, and more versatile optimization applications often require such accurate guidance from the user that they undermine their own efficacy. When confronted with the task of approximating parameters of a nonlinear sine curve with which to initialize regression software, Stanford professor of Statistics Art Owen suggested a simple iteration over all possible values within a given range for each predictor in search of the RSS minimizer. Specifically for each parameter α_i , $i = 1, 2, 3, 4$, we specify a lower bound l_i , upper bound u_i , and distance between estimates $\frac{(u_i - l_i)}{m_i}$ and test all

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in [l_1, u_1] \times [l_2, u_2] \times [l_3, u_3] \times \{mean(S)\} \in \mathbb{R}^4$$

such that $\alpha_i = \frac{j(u_i - l_i)}{m_i} + l_i$ for some $j = 0, 1, \dots, m_i$.

This approach, which we will call the naïve method (without in any way intending to suggest that professor Owen is naïve), achieves accuracy through brute computational force rather than intelligent design and completely ignores efficiency, where lies the entire difficulty of the problem. We will measure our algorithm for parameter estimation against the naïve method: we will value our algorithm above the naïve method iff in the same computational time our algorithm achieves greater accuracy (i.e. lower RSS), which is to say, iff our algorithm achieves the same accuracy as the naïve method in less computational time.

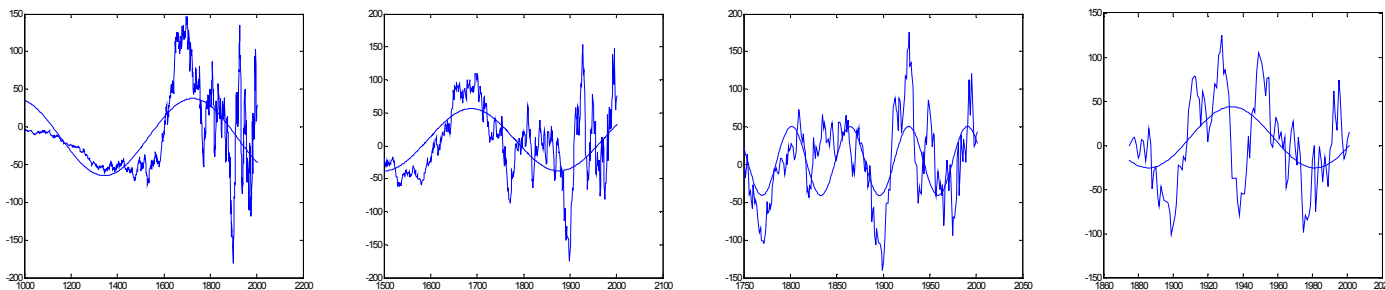


Figure 3: The naïve method performs satisfactorily when used to select initial parameters for a nonlinear least squares regression algorithm out of a small set of possible values. This method pays computation time for increased accuracy, however. In particular, the naïve method alone can estimate the true parameters of our curve (1) to any degree of accuracy (i.e. without initializing another optimization algorithm) but the computation time necessary to guarantee this accuracy increases exponentially with the degree. A request for Matlab to iterate over 100 possible values for each of $\alpha_1, \alpha_2, \alpha_3$ (10^6 combinations) took several minutes. Iteration over 1000 possible values each (10^9 combinations) overwhelmed my PC.

Consider instead the following procedure for estimating the parameters.

Let $\hat{\alpha}_4 = \frac{1}{N} \sum_{i=1}^N S_i$ and $\hat{\alpha}_3 = \min \{t_i \mid S_i = \text{median}(S)\}$.

(If S is even, we will define the median as the lesser of the two middle points.)

Smooth S with an $\lfloor n/2 \rfloor$ moving average to eliminate noise (say $n=1/100 \cdot \text{length of time series}$).

That is, compute $S_j^* = \frac{1}{n} \sum_{i=1}^n S_{j-\lfloor n/2 \rfloor+i}$, $j = \lfloor n/2 \rfloor, \dots, N-n+\lfloor n/2 \rfloor$.

Let the (N-n)-vector V represent the estimated slope of S at each point in S^* . For $i = 1, \dots, N-n+1$,

regress $\begin{bmatrix} S_i \\ \vdots \\ S_{i+n-1} \end{bmatrix} = \beta_i \begin{bmatrix} t_i \\ \vdots \\ t_{i+n-1} \end{bmatrix} + \beta_0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} + \begin{bmatrix} \varepsilon_i \\ \vdots \\ \varepsilon_{i+n-1} \end{bmatrix}$ and set $V_{i+\lfloor n/2 \rfloor} = \hat{\beta}_i$. That is, we perform

a moving linear regression on n points and assign the slope coefficient to the midpoint of the

interval. Let the (N-n)-vector C represent the estimated curvature of S at each point in S^* .

For each $i = 1, \dots, N-n+1$, let $C_i = \frac{V_{i+1} - V_{i-1}}{2}$. Since $\frac{d^2y}{dx^2} = -\alpha_2^2 \alpha_1 \sin(\alpha_2(x - \alpha_3))$, the curvature of our path ought to be a linear function of S^* centered at the mean of S. So regress

$C = \beta_1 Z + \beta_0 1 + \varepsilon$ where $Z = S^* - \hat{\alpha}_4 = S^* - \text{mean}(S)$ and let $\hat{\alpha}_2 = \sqrt{|\hat{\beta}_1|}$.

Now let $t^* = \{t_j : j = \lfloor n/2 \rfloor, \dots, N-n+\lfloor n/2 \rfloor\}$ be the t_i corresponding to each S_i^* .

Let $Y = \sin(\alpha_2(t^* - \alpha_3)) + \alpha_4$ and regress $S^* = \beta_1 Y + \beta_0 1 + \varepsilon$. Let $\hat{\alpha}_1 = \hat{\beta}_1$. Finally,

update $\hat{\alpha}_1 := \hat{\alpha}_1 \text{std}(S^*) / \text{std}(\hat{\alpha}_1 Y)$ to ensure that our curve has the same variance as our data.

The fit determined by these parameters proves to be amazing good. These estimates routinely yield R^2 values nearly equal to those `lsqcurvefit.m` generates when initialized by the naïve method iterated over 10^6 combinations, while requiring a lower order of computations. When used to initialize `lsqcurvefit.m`, this procedure often outperforms the naïve method. We thus have significantly reduced the computation time necessary to accurately estimate the least squares fit without even requiring an optimization algorithm. We lack any guarantee, however, that our estimates are unbiased for the least squares fit or that our estimates converge to the least squares parameters as we optimize n or iterate this method. By reducing the nonlinear least squares problem to a series of linear least squares problems, our method should achieve a fit that fit that is good in the least squares sense. And we see visually that it does. We can show that our method yields reliable estimates of *the* least squares fit only empirically, however, by visual and statistical comparison with the curve fit by the naïve method, which we know must converge to the absolute least squares minimum.

We therefore optimize our method by a gradient descent algorithm to ensure that our parameters converge to the least squares solution. Rather than attempt to solve the problem

$$\arg \min_{\alpha} \|S - \alpha_1 \sin(\alpha_2(t - \alpha_3)) + \alpha_4\| = \arg \min_{\alpha} \sum_{i=1}^n (S_i - \alpha_1 \sin(\alpha_2(t_i - \alpha_3)) + \alpha_4)^2$$

we address the similar problem

$$\arg \min_{\alpha} \left\| S^* - \alpha_1 \sin(\alpha_2(t^* - \alpha_3)) + \alpha_4 \right\| = \arg \min_{\alpha} \sum_{i=1}^n (S_i^* - \alpha_1 \sin(\alpha_2(t_i^* - \alpha_3)) + \alpha_4)^2 \quad (2)$$

$\frac{d^2 y}{dx^2} = -\alpha_2^2 \alpha_1 \sin(\alpha_2(x - \alpha_3))$ implies that the distance of our curve from the data at a point should vary inversely with the difference between the curvature of the data and the curvature of our curve at this point. Accordingly, we can fit our curve to the data by fitting our curve to the curvature of the data. So we can solve (2) by instead solving

$$\arg \min_{\alpha} \left\| C + \alpha_2^2 \alpha_1 \sin(\alpha_2(t^* - \alpha_3)) \right\| = \arg \min_{\alpha} \sum_{i=1}^n (C + \alpha_2^2 \alpha_1 \sin(\alpha_2(t_i^* - \alpha_3)) + \alpha_4)^2 \quad (3)$$

We iterate our estimation method until convergence by introducing gradient descent optimization for each parameter into the stage at which the method estimates the parameter. That is, at each stage, we first optimize a parameter according to gradient descent, using the derivative of (3) with respect to the parameter, and then input this optimized value into the estimation and optimization of the other parameters. An optimized estimate of α_3 allows our method to provide a better estimate of α_2 , which we can then optimize to better estimate α_1 , which we then optimize. We need not adjust α_4 . This algorithm yields the following curves, and guarantees that the estimated parameters approach the least squares fit.

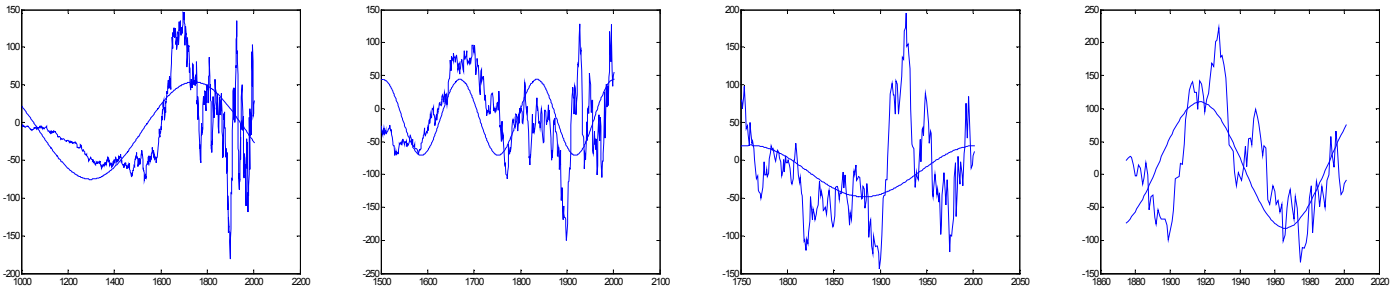


Figure 4: least squares curves fit by gradient descent on our estimation method

Our estimation method equipped with gradient descent optimization still proves computationally more efficient than the naïve method equipped with `lsqcurvefit.m`, for the same accuracy. I thus advocate this approach over the naïve method.