Fitting and Extending Exponential Smoothing Models with Stan

Slawek Smyl <slsmyl@microsoft.com>

Qinqin Zhang <qinzh@microsoft.com>

International Symposium on Forecasting
Riverside, California, USA, 2015

Fitting and Extending Exponential Smoothing Models with Stan

Slawek Smyl, Qinqin Zhang
Microsoft
{slsmyl, qinzh}@microsoft.com

Abstract. Time series forecasting is a fundamental challenge to various industries like retail and finance. Most of the classical forecasting models, including regression, ARIMA, exponential smoothing, assume that the characteristics of input data can be fully captured except for mean-zero Gaussian noise terms. However, this assumption is frequently invalid for highly volatile data with heavy-tail distributions. Analytical tractability is a big obstacle combating non-Gaussian noise terms. But with Stan (http://mc-stan.org), a probabilistic programming language implementing full Bayesian statistical inference, implementing innovative time series forecasting algorithms becomes easier. In this study, we discuss a few extensions to Holt-Winters Double Exponential Smoothing algorithm including
1) using a robust error distribution (Student)
2) allowing trend and sigma to grow non-linearly with time series data
3) incorporating prior business knowledge into the model
4) using skewed error distribution.

We demonstrate improvements brought by this framework in forecasting with irregular, real life data sampled from some Microsoft online services. We also discuss the situations under which we suggest to implement this framework instead of classical forecasting exponential smoothing models. Last but not least, we supplement the paper with sample codes we use to generate the examples in RStan.

1. Probabilistic Programming and Stan

Probabilistic Programming is a relatively new discipline that attempts to express probabilistic models, such as Probabilistic Graphical Models, using the power of high-level computer languages (Wikipedia n.d.) (PROBABILISTIC-PROGRAMMING.org n.d.). Applying probabilistic programming in time series analysis allows for the flexibility of defining models with arbitrary dynamics and estimating models numerically leveraging the power of computation.

Among various probabilistic programming languages, Stan (Stan Development Team 2015) is a relatively mature and comprehensive option. It is a flexible programming tool which implements full Bayesian statistical inference with Hamiltonian Markov Chain Monte Carlo. In Stan, models are specified in a language with syntax and some conventions similar to a mix of R and C++. Then the models are translated to C++ and compiled so that sampling can be executed.

Using Stan as a probabilistic programming tool has several benefits. First, with the interface package RStan (Stan Development Team 2015), users can easily develop Stan codes combined with R codes using an R-scripting environment. Second, Stan can run sampling on several chains in parallel. Third, Stan is an open source software implemented in C++, hence runs on all major platforms. Fourth, Stan has a good documentation and an active user discussion group.

2. Example: Non-linear Regression in Stan

Before diving into the details of applying probabilistic programming to improve forecasting, we first illustrate how a model estimation can be carried out in Stan by an example.

Assume we throw up a dense ball in a world without air friction. We measure the vertical location of the ball with an instrument whose measurement error is proportional to the absolute value of speed. The dynamics of this system could be captured as follows: $y_t = -gt^2/2 + v_0 t + \varepsilon_t$. Here y_t is the measured vertical position of the ball at time t, g stands for the gravitational constant, v_0 is the ball's initial velocity, and ε_t is the measurement error. We assume that ε_t obeys a normal distribution with mean zero and standard deviation $\sigma_t = \alpha_e|v_t| + \beta_e$, where v_t is the velocity of the ball at time t, i.e., $v_t = v_0 - gt$.

Consider the estimation task of recovering the initial velocity v_0 and the measurement error coefficients α_e and β_e with a series of measured locations $\{y_t\}$. Next we show how this estimation task is completed using regression with R and the package RStan.

The complete R code that generate the example data and conduct the regression is listed in Appendix A0. Figure 1 below shows the falling trajectory of the ball and the measured positions. One observation is that the measured positions are closer to the real positions when the speed of the ball is closer to zero. The parameters we used are v_0=15 m/s, α_e =0.15, and β_e =0.05.

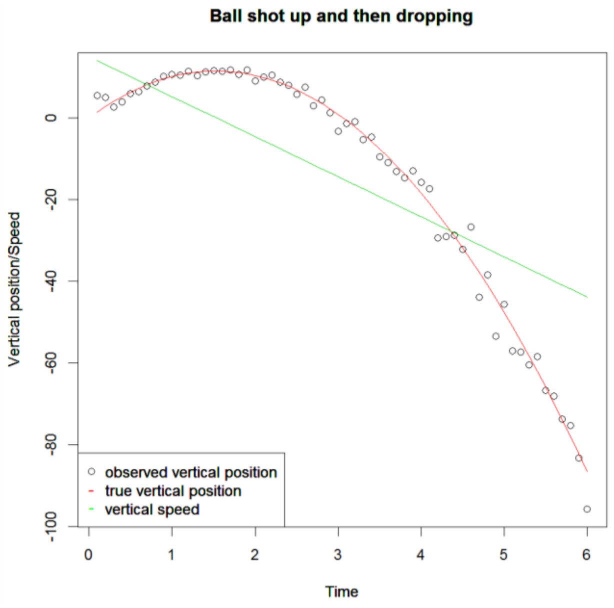

Figure 1. Falling Trajectory and Measured Positions

Assume we do not know the values of the three parameters, but need to estimate them based on a series of location measurements $\{y_t\}$. Next we describe the estimation process in Stan.

First, we need to specify prior distributions of the parameters. The following code define α_e and β_e to have an exponential prior and v_0 to have a uniform prior:

```
  alpha_e ~ exponential(1);
  beta_e ~ exponential(1);
  v0 ~ uniform(0 , 40);
```

Then the following code defines the system dynamics:

```
 for (it in 1:n) {
      t <- it*deltaT;
      v <- v0-g*t;
      y[it] ~ normal(-g*t^2/2+v0*t,
      alpha_e*fabs(v)+beta_e);
  }
```

Here y is a vector of length n recording the measured positions and deltaT is the time gap between two consecutive measurements. Stan fits parameters α_e, β_e, and v_0 through Hamiltonian Markov Chain Monte Carlo.

Please note that model iteration is defined with looping. Because Stan uses numerical methods for model fitting, using conjugate priors is not necessary nor even beneficial. In addition, the prior distributions of parameters can be defined in a very flexible way. For example, we can define a truncated normal prior with

```
        alpha_e ~ normal(0, 1) T[0,];
```

to ensure that α_e is always nonnegative.
Similarly, instead of assuming a uniform prior for v_0, we can apply a gamma prior with

```
         v0 ~ gamma(aGAMMA , bGAMMA);
```

where aGAMMA and bGAMMA would be constants selected by users matching their prior belief about v_0. Figure 2 gives an example of a suitable Gamma distribution.

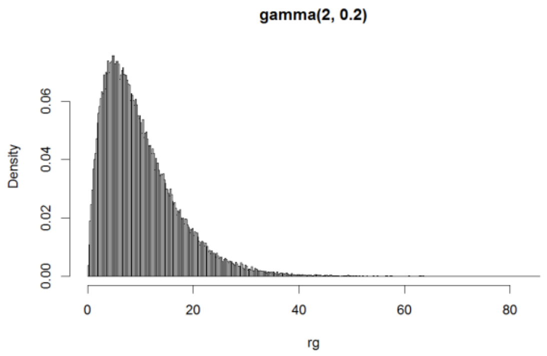

Figure 2. Probability Density Function of Gamma (2, 0.2)

Table 1 summarizes the estimated distributions of the parameters for a particular run of regression using Stan. Although the estimations do not match the parameters' true values exactly, they are very close to the true values used to generate $\{y_t\}$.

	Mean	2.50%	25%	50%	75%	97.5%
α_e	0.16	0.13	0.15	0.16	0.17	0.19
β_e	0.06	0	0.02	0.04	0.08	0.21
v_0	15.04	14.95	15.01	15.04	15.07	15.16

Table 1. Estimated Distribution of α_e, β_e, and v_0

The above example illustrates the basics of parameter estimation in Stan. Next, we discuss how to apply Stan in time series analysis to improve forecast performance. We only demonstrate our methodology in extending the classical exponential smoothing models. However, the principles can be easily extended to other popular time series analysis models in addition to exponential smoothing.

3. Exponential Smoothing

Exponential smoothing is one of the most popular forecasting approaches (Goodwin 2010), which includes a wide range of specific models. Many software implementations provide ready-to-use exponential smoothing components. For example, the popular Forecast package (Hyndman and Khandakar 2008) in R enables users to automatically select the best model from the ARIMA and the Exponential Soothing families. However, with more generally purposed probabilistic programming languages (e.g., Stan), we can easily extend existing exponential smoothing models without considering about analytical tractability of the models. We find that modeling flexibility is extremely helpful in forecasting, especially when real-life data suggests violations of assumptions in classical models, e.g., the normality assumptions of residuals.

Exponential smoothing models have many variations. Following the naming convention in Hyndman et al 2008, we name the models by three capitalized letters, e.g., AMN. The first letter describes the error, the second letter describes the trend, and the third letter describes the seasonality. Here A stands for additive, M stands for multiplicative, and N stands for none.

In the rest of this paper, we present a few extensions to Exponential Smoothing ANN and AAN models, the latter is also known as the Holt-Winters Double Exponential Smoothing Model. Their model formulations are listed below:

ANN: $\mu_t = l_{t-1}$
$l_t = l_{t-1} + \alpha\varepsilon_t$

AAN: $\mu_t = l_{t-1} + b_{t-1}$
$l_t = l_{t-1} + b_{t-1} + \alpha\varepsilon_t$
$b_t = b_{t-1} + \beta\varepsilon_t$

In Section 6 we are also referring to multiplicative error and local trend models so we provide their descriptions too:

MAN: $\mu_t = l_{t-1} + b_{t-1}$
$l_t = (l_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$
$b_t = b_{t-1} + \beta(l_{t-1} + b_{t-1})\ \varepsilon_t$

AMN: $\mu_t = l_{t-1} * b_{t-1}$
$l_t = l_{t-1} * b_{t-1} + \alpha\varepsilon_t$
$b_t = b_{t-1} + \beta * \varepsilon_t / l_{t-1}$

Here μ stands for expected value of the observation, l stands for level, b stands for trend, α is the level smoothing factor, β is the trend smoothing factor, and ε is a random error term with mean zero and variance σ^2.

Appendix A1 lists full Stan model specification for AAN (Holt-Winters Double Exponential Smoothing) and ANN models. The definition of exponential smoothing models in Stan is very similar to the example in Section 2. Once a model is fitted, the parameters, e.g., σ, can be extracted and used for forecasting by simulation in R. Based on the simulation results, we can get distributional information like various quantiles. Figure 3 is a snapshot of forecasts generated by the AAN model using Stan in R.

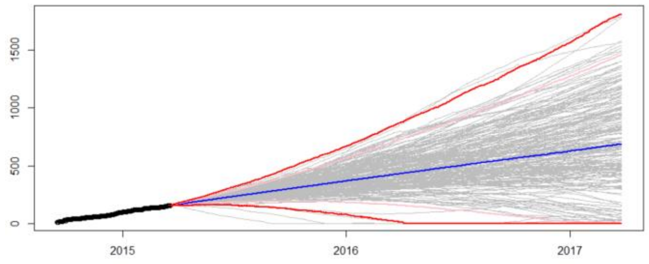

Figure 3. Snapshot of ANN forecasts

4. General characteristics of our time series

For most of the standard built-in exponential smoothing tools, the error term is generally assumed to have a mean zero Normal distribution with constant variance, i.e., no heteroscedasticity. However, in practice, data often causes deviations from the standard Normal distribution assumption. The flexibility of estimating and forecasting general structured models become the key to success in practical forecasting business.

Challenges occur frequently when forecasting for Microsoft online business. Here we first discuss some observations of our time series data that cannot be easily incorporated into existing built-in solutions. Then we explain the details of integrating these features by extending existing exponential smoothing models using probabilistic programming.

First, the variations among consecutive time series points are sometimes too large to be fully captured by a light-tailed distribution like the Normal Distribution. Second, the time series sometimes have growth rates faster than linear but slower than exponential. Third, the randomness of the time series often grows with values. Fourth, from the business side, expert subjective information should be integrated into forecasting. Last but not least, from the implementation side, forecasts should be generated in a scalable and automated fashion when hundreds or even thousands curves to be forecasted every day. The forecasting methodology should require a minimum amount of data scientists' attention when in production mode.

In the next few sections, we describe the details of the extensions motivated by the above observations.

5. Fat-tailed error distributions

In general, fitting of an exponential smoothing model can be performed without using any error distribution, e.g. by minimizing mean square error (MSE). However, maximizing log-likelihood is usually considered a better criterion, which requires assumptions on error distribution. For example, the default fitting function in R package forecast, ets, implements log-likelihood maximization. Error distribution is also a necessary part in the state-space formulation of exponential smoothing.

Once a model is fit, the point forecast can be obtained by applying an iterative formula (Hyndman et al 2008, p.17), again without referring to any error distribution. However, in order to obtain some confidence intervals, an error distribution needs to be assumed (unless one uses resampling of errors, which is only an approximation).

Typically Normal distribution is used for its analytical tractability. However, the time series encountered in Microsoft online services are often far from smooth, with huge sudden shifts caused by major customers' usage, price changes, new product launches, marketing activities, and etc. Figure 4 shows an example of volatile demand sampled from some Microsoft online service:

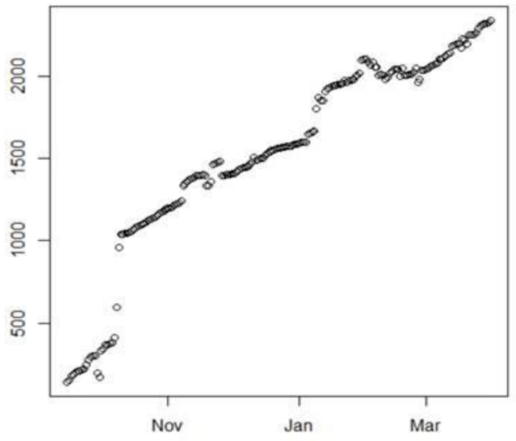

Figure 4. Demand Sampled From a Microsoft Online Service

One can of course attempt to "normalize" the time series by applying log or Box-Cox transform, but this approach tends to be manual, requiring attention to each curve. This manual approach is not scalable to be implemented on forecasting large number of time series curves every day. An alternative solution is to allow the error distribution to be either fat-tailed, e.g. Student, or a mixture of some distributions, e.g. Normal, with drastically different standard deviations. Both these approaches can be easily implemented in Stan by modifying the definition of the error distribution in source code. In our experimentation a single Student error distribution performed better than a mix of distributions.

6. Non-linear growth rates and Generalizing Additive and Multiplicative Trend Models

In our business, we often see fast growing time series, with growth pattern somewhere in-between linear and exponential.

In local trend additive models the formulation of expected value, $\mu_t = l_{t-1} + b_{t-1}$, leads to a linearly growing forecast; while in multiplicative ones the formulation, $\mu_t = l_{t-1} * b_{t-1}$, leads to an exponentially growing forecast. In the latter case, in order for a model to be reasonably stable, the trend b needs to be close to 1, so it can be further split into b=1+bb, where bb is small, which leads to: $\mu_t = l_{t-1} * b_{t-1} = l_{t-1} * (1 + bb_{t-1}) = l_{t-1} + l_{t-1} * bb_{t-1}$.

Both of the cases can be generalized by the following equation: $\mu_t = l_{t-1} + b_{t-1} * l_{t-1}^{powTrend}$, where powTrend is a continuous factor within [0, 1].

This generalization allows for a range of growing models, from additive ones, when powTrend is close to 0, to multiplicative model, when powTrend is close to 1. Therefore, using probabilistic programming one can fit automatically a model with non-linear growth rates without individual attention to every time series.

The Stan model would include the following statement:

```
y[t] ~ student_t(nu, l[t-1]+b[t-1]*l[t-1] ^powTrend,
   coefOfSigma*l[t-1]^powSigma).
```

In similar fashion global trend models could be generalized, the Stan code would look similar, except that instead of a local trend b_{t-1}, we would have a global parameter b:

```
y[t] ~ student_t(nu, l[t-1]+b*l[t-1] ^powTrend,
   coefOfSigma*l[t-1]^powSigma).
```

7. Heteroscedasticity and Generalizing Additive and Multiplicative Error Models

Applying a similar reasoning as in Section 6, we can generalize AAN and MAN into a generalized AAN model where the standard deviation of error σ grows non-linearly with level: coefOfSigma*l[t-1]^^{powSigma}. Here coefOfSigma stands for the coefficient of σ and powSigma is a power ratio varies between [0, 1]. When powSigma approaches 0, we get an Additive Error Model; and when it approaches 1, we get a Multiplicative Error Model. In our experiments powSigma tends to be between 0.5 and 0.7. We think it is reasonable because as the value of a time series grows, its variability also grows, but the speed of the growth of variability is typically lower than being proportional to the value itself.

To precisely understand how heteroscedasiticity is captured in the extended model, we provide the Stan's code for a global trend model discussed in this and the previous sections:

```
{…
parameters {
  real<lower=MIN_NU, upper=15> nu;
  real coefOfDrift;
  real <lower=MIN_SIGMA> coefOfSigma;
  real<lower=0,upper=1> powDrift;
  real<lower=0,upper=1> powSigma;
}
….
model {
  coefOfDrift ~ cauchy(0,1);
  coefOfSigma ~ cauchy(0,1) T[MIN_SIGMA,];
  for (t in 2:n) {
    y[t] ~ student_t(nu,    //degrees of freedom
        l[t-1]+coefOfDrift*l[t-1]^powDrift, //location
        coefOfSigma*l[t-1]^powSigma); //scale
  }
}
```

Here MIN_NU and MIN_SIGMA are constants close to 1 and 0 respectively, used to avoid numerical instabilities.

8. Incorporating business knowledge/beliefs

Stan is a Probabilistic Programming system implementing Bayesian inference. Therefore it is possible, and actually quite easy, to incorporate prior beliefs into the models, e.g. beliefs about the speed of growth. It may be useful when only a small amount of data is available, e.g. a few months, while the prediction horizon is large, e.g. 2 years. It is also important when the data contains an abnormal subset which does not really reflect the underlying patterns. For example, the usage of some online service may include more activity in June, as people are trying to use up their pre-paid quotas, or December slowdown in some professional-oriented systems. An example of the latter pattern is shown in Figure 5.

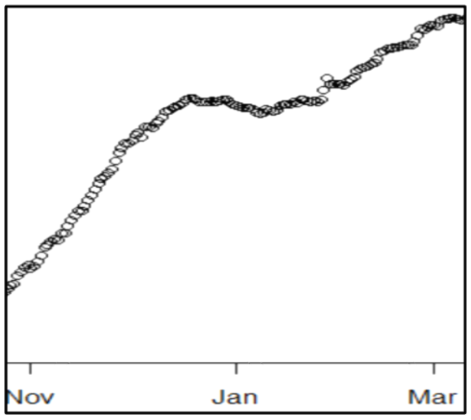

Figure 5. Slowing Down of Demand around Holiday Season

In this example shown in Figure 5, we really do not have much more historical data. Based on our experience with other online services of similar type, we suspect that the plateau is not really reflecting the underlying long-term trend. A solution is to extend the model described in Section 7 by specifying prior on coefficient powDrift. Instead of using a Uniform distribution which assigns weights evenly, we can use a Beta distribution to assign more weights to higher values of powDrift. Figure 6 shows a Beta distribution with parameters α = 5 and β = 1:

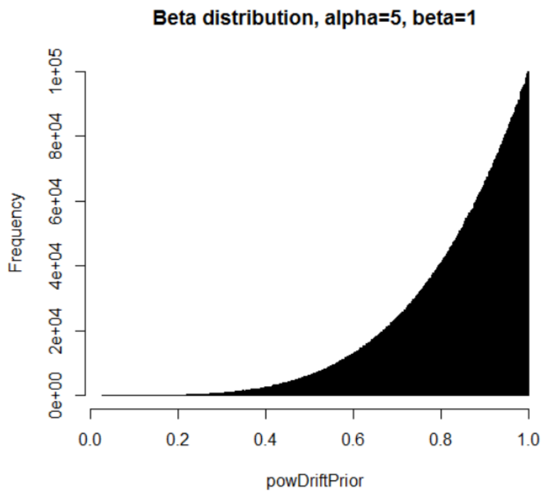

Figure 6. Demand Sampled From Some Microsoft Online Service

In the resulting model, powDrift is fitted as a compromise between what data speaks and what experience suggests. Below there is an example code; it's complete version is listed in Appendix 2.

```
model {
  coefOfDrift ~ cauchy(0,1);
  powDrift ~ beta(ALPHA_DRIFT, BETA_DRIFT);
  coefOfSigma ~ cauchy(0,1) T[MIN_SIGMA,];
  powSigma ~ beta(ALPHA_SIGMA, BETA_SIGMA);

  for (t in 2:n) {
    y[t] ~ student_t(nu,
       l[t-1]+coefOfDrift*l[t-1]^powDrift, //mu
       coefOfSigma*l[t-1]^powSigma);   //sigma
  }
}
```

Here ALPHA_DRIFT, BETA_DRIFT, ALPHA_SIGMA, BETA_SIGMA are constants passed to Stan.

9. Plateaus and skewed error distribution

Models with drifts are affected strongly by plateaus because the changes of drifts parameters can cause forecast to change significantly. At the same time, plateau is a relatively frequent feature of our curves - apart from holiday seasons they are also caused by occasional capacity constraints. As soon as the constraint is gone, the curve tends to grow with a similar or even faster speed than before. Therefore a preferred model should not be affected much by a plateau. A suitable class of models is one without explicit drift, but where drift or trend arises out of skewedness of the error distribution. This can be achieved by using symmetric error distribution during fitting, then collecting the error (residuals) distribution and using it to skew the generated errors during the future simulation phase (forecasting).

Alternatively, and preferably, a skewed error distribution could be used during fitting. Stan has only skewed Normal distribution, but skewed Student density function can be added relatively easily. There are several competing definitions of skewed Student distribution, we use one defined by (Azzalini 2003)

$$\frac{2}{\omega} t\left(\frac{x-\xi}{\omega};\nu\right) T\left(\alpha\,\frac{x-\xi}{\omega}\sqrt{\frac{\nu+1}{\nu+Q_x}};\nu+1\right), \qquad x \in \mathbb{R}$$

where t(·;ν) is the Student's t density with ν > 0 degrees of freedom, T(·;ν + 1) is the t distribution function for ν + 1 degrees of freedom and Qx = ω−2(x − ξ)2, and α is the skew coefficient. (Adelchi Azzalini 2012)

This definition can be encoded in Stan as follows:

```
real skew_student_t_log(real y, real nu, real mu,
   real sigma, real skew) {
   real z; real zc;
   if (sigma <= 0) reject("Scale has to be positive.
         Found sigma=", sigma);

     z<- (y-mu)/sigma;
     zc<- skew*z*sqrt((nu+1)/(nu+square(z)));
     return -log(sigma) + student_t_log(z, nu, 0, 1) +
       student_t_cdf_log(zc, nu+1, 0, 1);
   }
```

…and used in the modelling section:

```
model {
   sigma ~ normal(0,1) T[MIN_SIGMA,];
   tskew ~ normal(0.5,1);

   for (t in 2:n) {
     y[t] ~ skew_student_t(nu, l[t-1],
       coefOfSigma*l[t-1]^powSigma, tskew);
   }
}
```

This simpler model is capable of expressing a range of growth patterns: for powSigma close to 0, we would have homoscedastic model which, together with the skewness, would produce linear growth pattern. Similarly, for powSimga close to 1 (and together with the skew) we would get a close to exponential growth pattern. Anything in-between would produce a faster

than linear but slower than exponential growth. By rescaling powSigma (e.g. PowSigma=2*powSigma-1) one would also allow for the damped models.
Importantly, it is less affected by plateaus, as illustrated below.

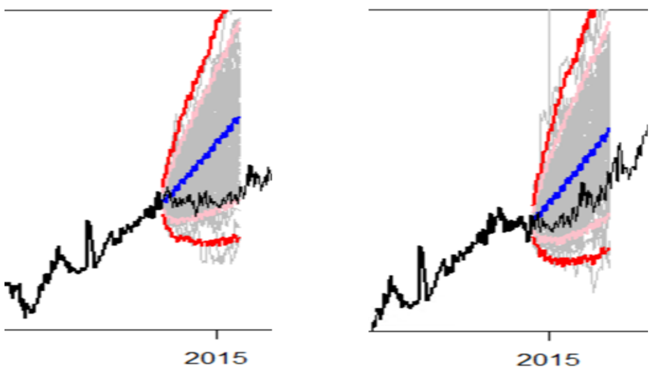

Figure 7.
Two forecasts, the left done at the beginning, and the right done a few weeks later, during a plateau. The forecast (e.g. blue line signifying the median forecast) is not affected much.

10. Summary

Motivated by our unusual time series (high volatility, continually growing with occasional plateaus) we extended and generalized Exponential Smoothing models using Probabilistic Programming approach and Stan.
As Probabilistic Programming is Bayesian in its core, incorporating business beliefs is simple.
Probabilistic Programming offers the ability to easily experiment and create new models that are likely to be analytically non-tractable.

REFERENCES

Adelchi Azzalini, Reinaldo B. Arellan-Valle. 2012. "Maximum penalized likelihood estimation for skew-normal and skew-t distributions." *Journal of Statistical Planning and Inference* 419–433.

Azzalini, A. and A. Capitanio. 2003. "Distributions generated by pertubation of symmetry with emphasis on a multivariate skew t distribution." *Journal of the Royal Statistical Society B* 579–602.

Gelman, A., Jakulin, A., Pittau, M. G., and Su, Y.-S. 2008. "A weakly informative default prior distribution for logistic and other regression models." *Annals of Applied Statistics* 2 (4): 1360-1383.

Goodwin, Paul. 2010. "The Holt-Winters Approach to Exponential Smoothing: 50 Years Old and Going Strong." *Foresight* (19): 30-33.

Hyndman, Rob J, and Yeasmin Khandakar. 2008. "Automatic time series forecasting: the forecast package for R." *Journal of Statistical Software* 26 (3): 1-22. http://ideas.repec.org/a/jss/jstsof/27i03.html.

Hyndman, Rob J. at al. 2008. *Forecasting with Exponential Smoothing.* Springer.

PROBABILISTIC-PROGRAMMING.org. n.d. *PROBABILISTIC-PROGRAMMING.org.* http://probabilistic-programming.org/wiki/Home.

Stan Development Team. 2015. *RStan: the R interface to Stan, Version 2.6.0.* http://mc-stan.org/rstan.html.

—. 2015. "Stan Modeling Language Users Guide and Reference Manual, Version 2.6.2." http://mc-stan.org/.

—. 2015. *Stan: A C++ Library for Probability and Sampling, Version 2.6.0.* http://mc-stan.org/.

Taylor, James W. 2012. "Short-Term Load Forecasting with Exponentially Weighted Methods." *IEEE Transactions on Power Systems* 27: 458-464.

Wikipedia. n.d. *Probabilistic programming language.* Accessed 05 05, 2015. http://en.wikipedia.org/wiki/Probabilistic_programming_language.

Appendix 0

R code that generates the "ball measurements" and then conducts regression with Stan

```
library(rstan)
library(doParallel)

CLUSTER_SIZE=4
stopCluster(cl)
Sys.sleep(2)
CLUST_OUT_PATH="c:/temp/rclust.txt"
file.remove(CLUST_OUT_PATH)
cl <- makeCluster(CLUSTER_SIZE, outfile=CLUST_OUT_PATH)
registerDoParallel(cl)


#### Example data generation
deltaT=0.1; period=6; g=9.81;
ae=0.15; be=0.05; v0=15 # assumed parameter values

numOfPoints=as.integer(period/deltaT)
t_a=rep(0,numOfPoints); v_a=rep(0,numOfPoints)
yt_a=rep(0,numOfPoints); y_a=rep(0,numOfPoints)
for (it in 1:numOfPoints) {
      t=it*deltaT; t_a[it]=t
      v=v0-g*t;    v_a[it]=v
      yt=-g*t^2/2+v0*t;   yt_a[it]=yt
      error=rnorm(1, mean=0,sd=ae*abs(v)+be)
      y=yt+error;  y_a[it]=y
      print(paste(t,v,yt,y))
}

plot(t_a,y_a, main='Ball shot up and then dropping', ylab='Vertical position/Speed',
xlab='Time')
lines(t_a, yt_a, col=2, type='l')
lines(t_a,v_a, col=3)
legend("bottomleft",
      c("observed vertical position", "true vertical position", "vertical speed"),
      col=c("black", "red", "green"),
      pch=c(1,45,45));


### Regression
parametersX <- c("alpha_e", "beta_e", "v0")
modelG <- "
      data {
            int<lower=1> n;
            vector[n] y;
            real <lower=0> deltaT;
            real <lower=0> aGAMMA;
            real <lower=0> bGAMMA;
      }
      transformed data {
            real g;
            g<-9.81;
      }
      parameters {
```

```
                real <lower=0> v0;
                real <lower=0> alpha_e;
                real <lower=0> beta_e;
        }
        model {
                real t; real v;

                alpha_e ~ exponential(1);
                beta_e ~ exponential(1);
                v0 ~ gamma(aGAMMA , bGAMMA);

                for (it in 1:n) {
                        t <- it*deltaT;
                        v <- v0-g*t;
                        y[it] ~ normal(-g*t^2/2+v0*t, alpha_e*fabs(v)+beta_e);
                }
        }
"
stanModelG <- stan_model(model_code=modelG) #compile Stan model

aGAMMA=2; bGAMMA=0.2
data <- list(y=y_a, n=length(y_a), deltaT=deltaT, aGAMMA=aGAMMA, bGAMMA=bGAMMA)

ret<- foreach(i=1:CLUSTER_SIZE, .packages=c("rstan")) %dopar% {#run sampling in parallel
        initializations <- list(list(
                alpha_e=rexp(1,rate=1), beta_e=rexp(1,rate=1),
                v0=rgamma(1, shape=aGAMMA, rate = bGAMMA)))

        sampling(stanModelG,
                data=data,
                init=initializations,
                pars=parametersX,
                iter=10000,
                chains=1,
                refresh = -1)
}
samples <- sflist2stanfit(ret)
print(samples)

alpha_e=mean(extract(samples)$alpha_e)
beta_e=mean(extract(samples)$beta_e)
v0=mean(extract(samples)$v0)
print(paste("alpha_e=",round(alpha_e,2),"  beta_e=",round(beta_e,2), "  v0=",round(v0,2)))
```

Appendix 1

An implementation of AAN model in Stan. Remove or comment out highlighted lines for ANN model.

```
data {
  int<lower=1> n;
  vector<lower=0>[n] y;
}
parameters {
  real<lower=0> sigma;
  real <lower=0,upper=1>bSm;  // specifying range creates a uniform prior
  real <lower=0,upper=1>levSm;
}
transformed parameters {
  vector[n] b;
  vector<lower=0>[n] l;

  l[1] <-y[1];
  b[1] <- 0;

  for (t in 2:n) {
    l[t]  <- levSm*y[t] + (1-levSm)*(l[t-1]+b[t-1]) ;
    b[t] <-  bSm*(l[t]-l[t-1]) + (1-bSm)*b[t-1] ;
  }
}
model {
  sigma ~ Cauchy(0,2) T[0,];   // positive side of Cauchy distribution as prior

  for (t in 2:n) {
    y[t] ~ normal(l[t-1]+b[t-1], sigma);
  }
}
```

Appendix 2

An implementation of our extended AAN model in Stan.

```
functions {
  real skew_student_t_log(real y, real nu, real mu, real sigma, real skew) {
    real z; real zc;
    if (sigma <= 0)
      reject("Scale has to be positive.  Found sigma=", sigma);

    z<- (y-mu)/sigma;
    zc<- skew*z*sqrt((nu+1)/(nu+square(z)));
    return -log(sigma) + student_t_log(z, nu, 0, 1) + student_t_cdf_log(zc, nu+1, 0, 1);
  }
}
data {
  int<lower=1> n;
  vector<lower=0.001>[n] y;
  real<lower=1> minNu;
}
```

```
transformed data {
  real MIN_SIGMA;
  MIN_SIGMA<-0.0001;//to void some numerical instabilities
}
parameters {
  real<lower=minNu,upper=15> nu;
  real<lower=MIN_SIGMA> coefOfSigma; //to void some numerical instabilities
  real <lower=0,upper=1>levSm;
  real <lower=0,upper=1>powSigma;
  real tskew;
}
transformed parameters {
  vector[n] l;

  l[1] <-y[1];
  for (t in 2:n) {
    l[t]  <- levSm*y[t] + (1-levSm)*l[t-1] ;
  }
}
model {
  coefOfSigma ~ normal(0,1) T[MIN_SIGMA,];
  //powSigma ~ beta(alphaBETA, betaBETA);
  tskew ~ normal(0.5,1);

  for (t in 2:n) {
    y[t] ~ skew_student_t(nu, l[t-1], coefOfSigma*fabs(l[t-1])^powSigma, tskew);
  }
}
```

Appendix 3

Forecasting by simulation in R.

```
#prevLevel, powSigmaM, levSmM, nuM, sigmaM are means obtained from samples provided by Stan
MIN_VAL=1;
yf<- foreach(i=1:CLUSTER_SIZE, .packages=c("sn"), .combine=rbind) %dopar% {
   yf=matrix(lastLevelM,nrow=NUM_OF_TRIALS, ncol=MAX_FORECAST_HORIZON)
   for (irun in 1:NUM_OF_TRIALS) {
      for (t in 1:MAX_FORECAST_HORIZON) {
         error=rst(n=1, xi=0, omega=sigmaM*(prevLevel)^powSigmaM, alpha=tskewM, nu=nuM)
         yf[irun,t]=max(MIN_VAL,prevLevel+error)
         currLevel=levSmM*yf[irun,t] + (1-levSmM)*prevLevel ;

         if (currLevel>MIN_VAL) {
            prevLevel=currLevel
         }
      }
   }
   yf
}
avgYfs=apply(yf,2,quantile,probs=c(0.05,0.5,0.95,0.99,0.01,0.1,0.9))
```