ADAM is the next step in time series analysis and forecasting. Remember exponential smoothing and functions like `es()` and `ets()`? Remember ARIMA and functions like `arima()`, `ssarima()`, `msarima()` etc? Remember your favourite linear regression function, e.g. `lm()`, `glm()` or `alm()`? Well, now these three models are implemented in a unified framework. Now you can have exponential smoothing with ARIMA elements and explanatory variables in one box: `adam()`. You can do ETS components and ARIMA orders selection, together with explanatory variables selection in one go. You can estimate ETS / ARIMA / regression using either likelihood of a selected distribution or using conventional losses like MSE, or even using your own custom loss. You can tune parameters of optimiser and experiment with initialisation and estimation of the model. The function can deal with multiple seasonalities and with intermittent data in one place. In fact, there are so many features that it is just easier to list the major of them:

1. ETS;
2. ARIMA;
3. Regression;
4. TVP regression;
5. Combination of (1), (2) and either (3), or (4);
6. Automatic selection / combination of states for ETS;
7. Automatic orders selection for ARIMA;
8. Variables selection for regression part;
9. Normal and non-normal distributions;
10. Automatic selection of most suitable distributions;
11. Advanced and custom loss functions;
12. Multiple seasonality;
13. Occurrence part of the model to handle zeroes in data (intermittent demand);
14. Model diagnostics using plot() and other methods;
15. Confidence intervals for parameters of models;
16. Automatic outliers detection;
17. Handling missing data;
18. Fine tuning of persistence vector (smoothing parameters);
19. Fine tuning of initial values of the state vector (e.g. level / trend / seasonality / ARIMA components / regression parameters);
20. Two initialisation options (optimal / backcasting);
21. Provided ARMA parameters;
22. Fine tuning of optimiser (select algorithm and convergence criteria);
23. …

All of this is based on the Single Source of Error state space model, which makes ETS, ARIMA and regression directly comparable via information criteria and opens a variety of modelling and forecasting possibilities. In addition, the code is much more efficient than the code of already existing smooth functions, so hopefully this will be a convenient function to use. I do not promise that everything will work 100% efficiently from scratch, because this is a new function, which implies that inevitably there are bugs and there is a room for improvement. But I intent to continue working on it, improving it further, based on the provided feedback (you can submit an issue on github if you have ideas).

Keep in mind that starting from smooth v3.0.0 I will not be introducing new features in `es()`, `ssarima()` and other conventional functions for univariate variables in `smooth` – I will only fix bugs in them and possibly optimise some parts of the code, but there will be no innovations in

them, given that the main focus from now on will be on `adam()`. To that extent, I have removed some experimental and not fully developed parameters from those functions (e.g. occurrence, oesmodel, updateX, persistenceX and transitionX).

Now, I realise that ADAM is something completely new and contains just too much information to cover in one post. As a result, I have started the work on an online textbook. This is work in progress, missing some chapters, but it already covers many important elements of ADAM. If you find any mistakes in the text or formulae, please, use the "Open Review" functionality in the textbook to give me feedback or send me a message. This will be highly appreciated, because, working on this alone, I am sure that I have made plenty of mistakes and typos.

## Example in R

Finally, it would be boring just to announce things and leave it like that. So, I've decided to come up with an R experiments on M, M3 and tourism competitions data, similar to how I've done it in 2017, just to show how the function compares with the other conventional ones, measuring their accuracy and computational time:

Huge chunk of code in R

```
# Load the packages. If the packages are not available, install them
from CRAN
library(Mcomp)
library(Tcomp)
library(smooth)
library(forecast)

# Load the packages for parallel calculation
# This package is available for Linux and MacOS only
# Comment out this line if you work on Windows
library(doMC)

# Set up the cluster on all cores / threads.
## Note that the code that follows might take around 500Mb per thread,
## so the issue is not in the number of threads, but rather in the RAM
availability
## If you do not have enough RAM,
## you might need to reduce the number of threads manually.
## But this should not be greater than the number of threads your
processor can do.
registerDoMC(detectCores())

##### Alternatively, if you work on Windows (why?), uncomment and run
the following lines
# library(doParallel)
# cl <- detectCores()
# registerDoParallel(cl)
#####

# Create a small but neat function that will return a vector of error
measures
errorMeasuresFunction <- function(object, holdout, insample){
    return(c(measures(holdout, object$mean, insample),
```

```
            mean(holdout < object$upper & holdout > object$lower),
            mean(object$upper-object$lower)/mean(insample),
            pinball(holdout, object$upper, 0.975)/mean(insample),
            pinball(holdout, object$lower, 0.025)/mean(insample),
            sMIS(holdout, object$lower, object$upper,
mean(insample),0.95),
            object$timeElapsed))
}

# Create the list of datasets
datasets <- c(M1,M3,tourism)
datasetLength <- length(datasets)
# Give names to competing forecasting methods
methodsNames <- c("ADAM-ETS(ZZZ)","ADAM-ETS(ZXZ)","ADAM-ARIMA",
                  "ETS(ZXZ)","ETSHyndman","AutoSSARIMA","AutoARIMA");
methodsNumber <- length(methodsNames);
# Run adam on one of time series from the competitions to get names of
error measures
test <- adam(datasets[[125]]);
# The array with error measures for each method on each series.
## Here we calculate a lot of error measures, but we will use only few
of them
testResults <- array(NA,c(methodsNumber,datasetLength,length(test$
accuracy)+6),
                             dimnames=list(methodsNames, NULL,
                                           c(names(test$accuracy),
                                             "Coverage","Range",

"pinballUpper","pinballLower","sMIS",
                                             "Time")));

#### ADAM(ZZZ) ####
j <- 1;
result <- foreach(i=1:datasetLength, .combine="cbind",
.packages="smooth") %dopar% {
    startTime <- Sys.time()
    test <- adam(datasets[[i]],"ZZZ");
    testForecast <- forecast(test, h=datasets[[i]]$h, interval="pred");
    testForecast$timeElapsed <- Sys.time() - startTime;
    return(errorMeasuresFunction(testForecast, datasets[[i]]$xx,
datasets[[i]]$x));
}
testResults[j,,] <- t(result);

#### ADAM(ZXZ) ####
j <- 2;
result <- foreach(i=1:datasetLength, .combine="cbind",
.packages="smooth") %dopar% {
    startTime <- Sys.time()
    test <- adam(datasets[[i]],"ZXZ");
    testForecast <- forecast(test, h=datasets[[i]]$h, interval="pred");
    testForecast$timeElapsed <- Sys.time() - startTime;
```

```
    return(errorMeasuresFunction(testForecast, datasets[[i]]$xx,
datasets[[i]]$x));
}
testResults[j,,] <- t(result);

#### ADAMARIMA ####
j <- 3;
result <- foreach(i=1:datasetLength, .combine="cbind",
.packages="smooth") %dopar% {
    startTime <- Sys.time()
    test <- adam(datasets[[i]], "NNN",
                 order=list(ar=c(3,2),i=c(2,1),ma=c(3,2),select=TRUE));
    testForecast <- forecast(test, h=datasets[[i]]$h, interval="pred");
    testForecast$timeElapsed <- Sys.time() - startTime;
    return(errorMeasuresFunction(testForecast, datasets[[i]]$xx,
datasets[[i]]$x));
}
testResults[j,,] <- t(result);

#### ES(ZXZ) ####
j <- 4;
result <- foreach(i=1:datasetLength, .combine="cbind",
.packages="smooth") %dopar% {
    startTime <- Sys.time()
    test <- es(datasets[[i]],"ZXZ");
    testForecast <- forecast(test, h=datasets[[i]]$h,
interval="parametric");
    testForecast$timeElapsed <- Sys.time() - startTime;
    return(errorMeasuresFunction(testForecast, datasets[[i]]$xx,
datasets[[i]]$x));
}
testResults[j,,] <- t(result);

#### ETS from forecast package ####
j <- 5;
result <- foreach(i=1:datasetLength, .combine="cbind",
.packages="forecast") %dopar% {
    startTime <- Sys.time()
    test <- ets(datasets[[i]]$x);
    testForecast <- forecast(test, h=datasets[[i]]$h, level=95);
    testForecast$timeElapsed <- Sys.time() - startTime;
    return(errorMeasuresFunction(testForecast, datasets[[i]]$xx,
datasets[[i]]$x));
}
testResults[j,,] <- t(result);

#### AUTO SSARIMA ####
j <- 6;
result <- foreach(i=1:datasetLength, .combine="cbind",
.packages="smooth") %dopar% {
    startTime <- Sys.time()
    test <- auto.ssarima(datasets[[i]]);
```

```
    testForecast <- forecast(test, h=datasets[[i]]$h, interval=TRUE);
    testForecast$timeElapsed <- Sys.time() - startTime;
    return(errorMeasuresFunction(testForecast, datasets[[i]]$xx,
datasets[[i]]$x));
}
testResults[j,,] <- t(result);

#### AUTOARIMA ####
j <- 7;
result <- foreach(i=1:datasetLength, .combine="cbind",
.packages="forecast") %dopar% {
    startTime <- Sys.time()
    test <- auto.arima(datasets[[i]]$x);
    testForecast <- forecast(test, h=datasets[[i]]$h, level=95);
    testForecast$timeElapsed <- Sys.time() - startTime;
    return(errorMeasuresFunction(testForecast, datasets[[i]]$xx,
datasets[[i]]$x));
}
testResults[j,,] <- t(result);

# If you work on Windows, don't forget to shutdown the cluster via the
following command:
# stopCluster(cl)
```

After running this code, we will get the big array (7x5315x21), which would contain many different error measures for point forecasts and prediction intervals. We will not use all of them, but instead will extract MASE and RMSSE for point forecasts and Coverage, Range and sMIS for prediction intervals, together with computational time. Although it might be more informative to look at distributions of those variables, we will calculate mean and median values overall, just to get a feeling about the performance:

A much smaller chunk of code in R

```
round(apply(testResults[,,c("MASE","RMSSE","Coverage","
Range","sMIS","Time")],
            c(1,3),mean),3)
round(apply(testResults[,,c("MASE","RMSSE","Range","MIS","Time")],
            c(1,3),median),3)
```

This will result in the following two tables (boldface shows the best performing functions):

```
Means:
               MASE RMSSE Coverage Range  sMIS  Time
ADAM-ETS(ZZZ) 2.415 2.098    0.888 1.398 2.437 0.654
ADAM-ETS(ZXZ) 2.250 1.961    0.895 1.225 2.092 0.497
ADAM-ARIMA    2.551 2.203    0.862 0.968 3.098 5.990
ETS(ZXZ)      2.279 1.977    0.862 1.372 2.490 1.128
ETSHyndman    2.263 1.970    0.882 1.200 2.258 0.404
AutoSSARIMA   2.482 2.134    0.801 0.780 3.335 1.700
AutoARIMA     2.303 1.989    0.834 0.805 3.013 1.385

Medians:
               MASE RMSSE Range  sMIS  Time
```

```
ADAM-ETS(ZZZ) 1.362 1.215 0.671 0.917 0.396
ADAM-ETS(ZXZ) 1.327 1.184 0.675 0.909 0.310
ADAM-ARIMA    1.476 1.300 0.769 1.006 3.525
ETS(ZXZ)      1.335 1.198 0.616 0.931 0.551
ETSHyndman    1.323 1.181 0.653 0.925 0.164
AutoSSARIMA   1.419 1.271 0.577 0.988 0.909
AutoARIMA     1.310 1.182 0.609 0.881 0.322
```

Some things to note from this:

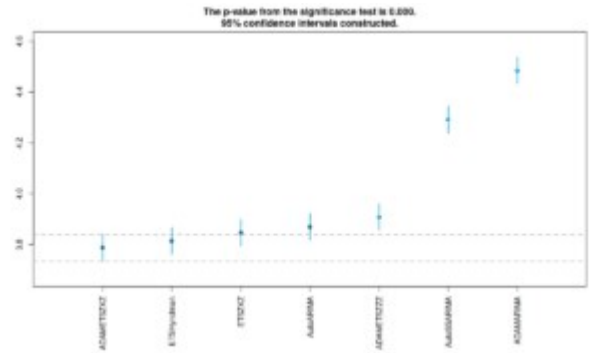
- ADAM ETS(ZXZ) is the most accurate model in terms of mean MASE and RMSSE, it has the coverage closest to 95% (although none of the models achieved the nominal value because of the fundamental underestimation of uncertainty) and has the lowest sMIS, implying that it did better than the other functions in terms of prediction intervals;
- The ETS(ZZZ) did worse than ETS(ZXZ) because the latter considers the multiplicative trend, which sometimes becomes unstable, producing exploding trajectories;
- ADAM ARIMA is not performing well yet, because of the implemented order selection algorithm and it was the slowest function of all. I plan to improve it in future releases of the function;
- While ADAM ETS(ZXZ) did not beat ETS from forecast package in terms of computational time, it was faster than the other functions;
- When it comes to medians, `auto.arima()`, `ets()` and `auto.ssarima()` seem to be doing better than ADAM, but not by a large margin.

In order to see if the performance of functions is statistically different, we run the RMCB test for MASE, RMSSE and MIS. Note that RMCB compares the median performance of functions. Here is the R code:

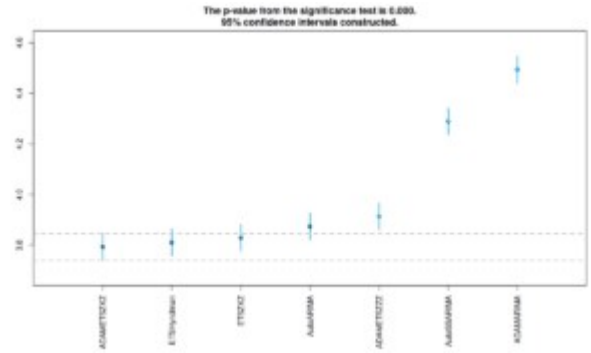
A smaller chunk of code in R for the MCB test

```
# Load the package with the function
library(greybox)
# Run it for each separate measure, automatically producing plots
rmcbResultMASE <- rmcb(t(testResults[,,"MASE"]))
rmcbResultRMSSE <- rmcb(t(testResults[,,"RMSSE"]))
rmcbResultsMIS <- rmcb(t(testResults[,,"sMIS"]))
```

And here are the figures that we get by running that code



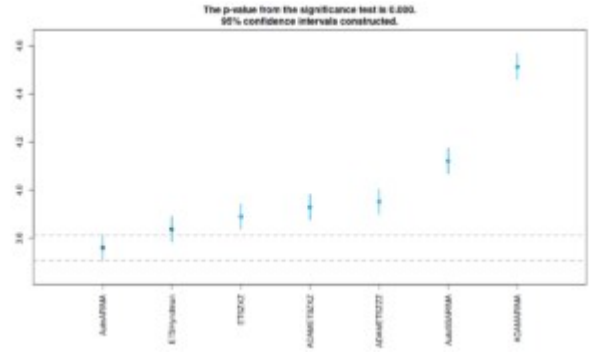
RMCB test for MASE




RMCB test for RMSSE

As we can see from the two figures above, ADAM-ETS(Z,X,Z) performs better than the other functions, although statistically not different than ETS implemented in `es()` and `ets()` functions. ADAM-ARIMA is the worst performing function for the moment, as we have already noticed in the previous analysis. The ranking is similar for both MASE and RMSSE.

And here is the sMIS plot:




RMCB test for sMIS

When it comes to sMIS, the leader in terms of medians is `auto.arima()`, doing quite similar to `ets()`, but this is mainly because they have lower ranges, incidentally resulting in lower than needed coverage (as seen from the summary performance above). ADAM-ETS does similar to `ets()` and `es()` in this aspect (the intervals of the three intersect).

Obviously, we could provide more detailed analysis of performance of functions on different types of data and see, how they compare in each category, but the aim of this post is just to demonstrate how the new function works, I do not have intent to investigate this in detail.