Documentation: standardModel.R

An example implementation

B. Stephens Hemingway

08/11/2020

Introduction

This document demonstrates the implementation of the standard fitness fatigue model using the R function file (standardModel.R)

The resource utilised in this notebook is:

A bespoke R function (standardModel.R) which applies expanding-window cross-validation, available here

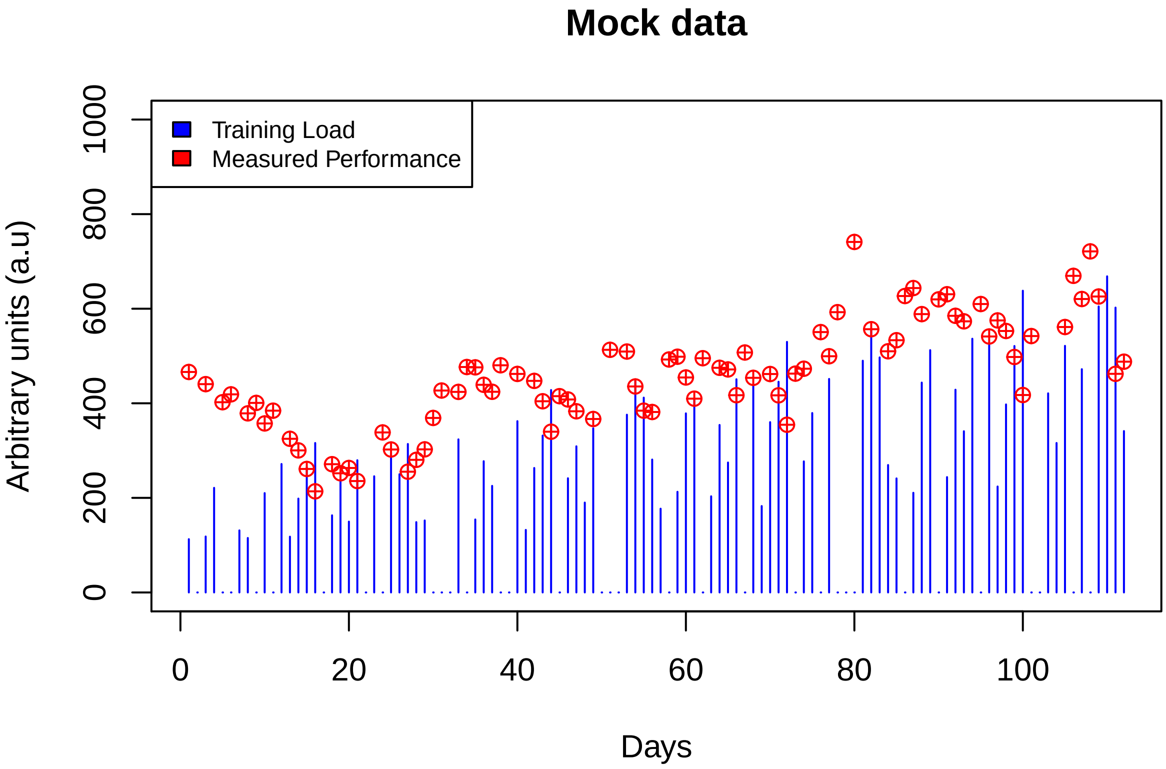
Construction of mock data for the demonstrations

A mock data-set has been constructed to facilitate demonstration of the fitting process using the resource described, and also highlights the format that researchers and practitioners should apply if replicating these methods with their own data sets. It is worth noting that NA values are **required** to fill gaps indicating an absence of a measured performance value on the associated day. Using zero values will lead to unexpected results.

```
performances <- c(466.2,NA,440.5,NA,402.3,418.9,NA,378.7,400.8,357.4,384.4,NA,324.9,300.4,261.1,213.8,N

loads <-  c(112.7, 0, 118.3, 221.2, 0, 0, 131.25, 115.2, 0, 210.1, 0, 271.6, 117.95, 198.45, 264.6, 316

mockData <- data.frame("days" = 1:112,
                       "performances" = performances,
                       "loads" = loads
                      )
rm(loads,performances)
head(mockData)
```

```
##   days performances loads
## 1    1        466.2 112.7
## 2    2           NA   0.0
## 3    3        440.5 118.3
## 4    4           NA 221.2
## 5    5        402.3   0.0
## 6    6        418.9   0.0
```

```
plot(mockData$days, mockData$loads, type = "h", ylim = c(0,1000),
       col = "blue",
       ylab = "Arbitrary units (a.u)", xlab = "Days",
       main = "Mock data")
points(mockData$days, mockData$performances, pch = 10, col = "red")
legend("topleft", c("Training Load","Measured Performance"),
       fill = c("blue","red"), cex = 0.75)
```



Applying the R function (standardModel.R)

Using the R script is extremely straightforward, and requires only your training/performance input in the format shown above and a few arguments passed to the function call.

Load the R function and set up function arguments

```
source("standardModel.R")

# Set up box constraints for the parameter estimates
boxConstraints <- data.frame("lower" = c(1,0.01,1,0.01,1),
                             "upper" = c(1000,10,50,10,50))

# Set some best guesses for the parameter estimates (p*,kg,Tg,kh,Th)
bestGuess <- c(400, 1, 25, 1.7, 10)
```

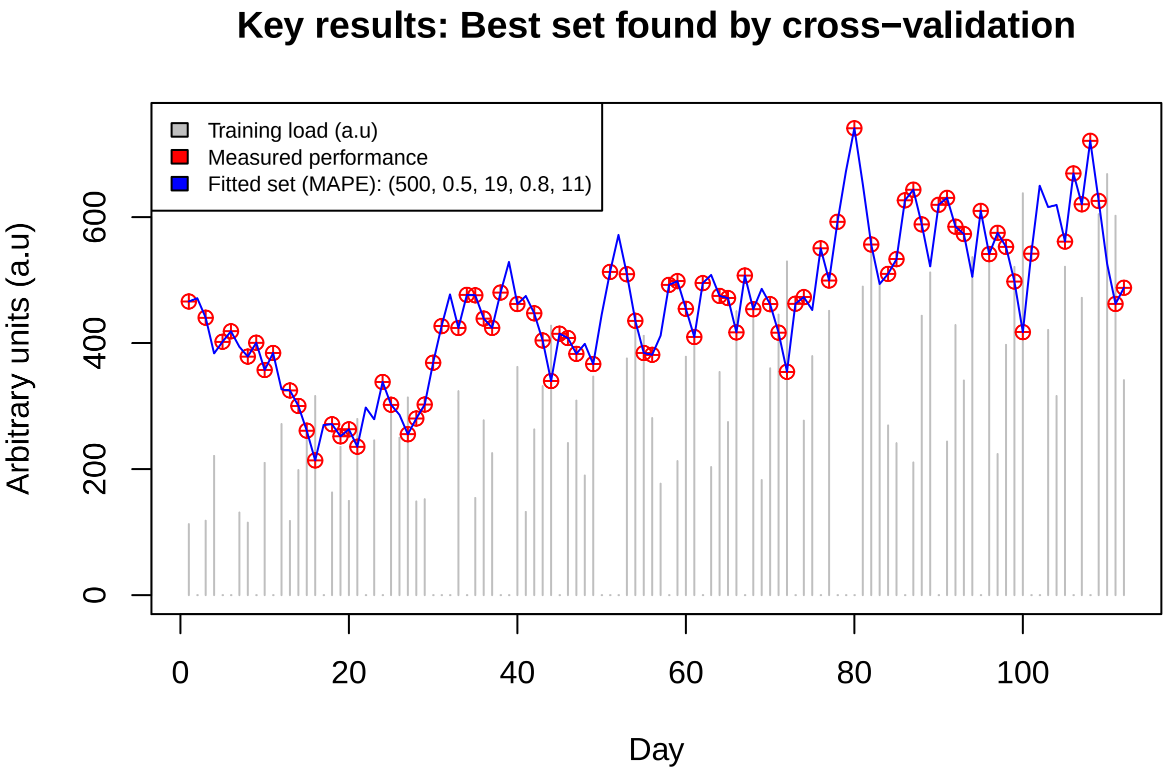
Function dependencies

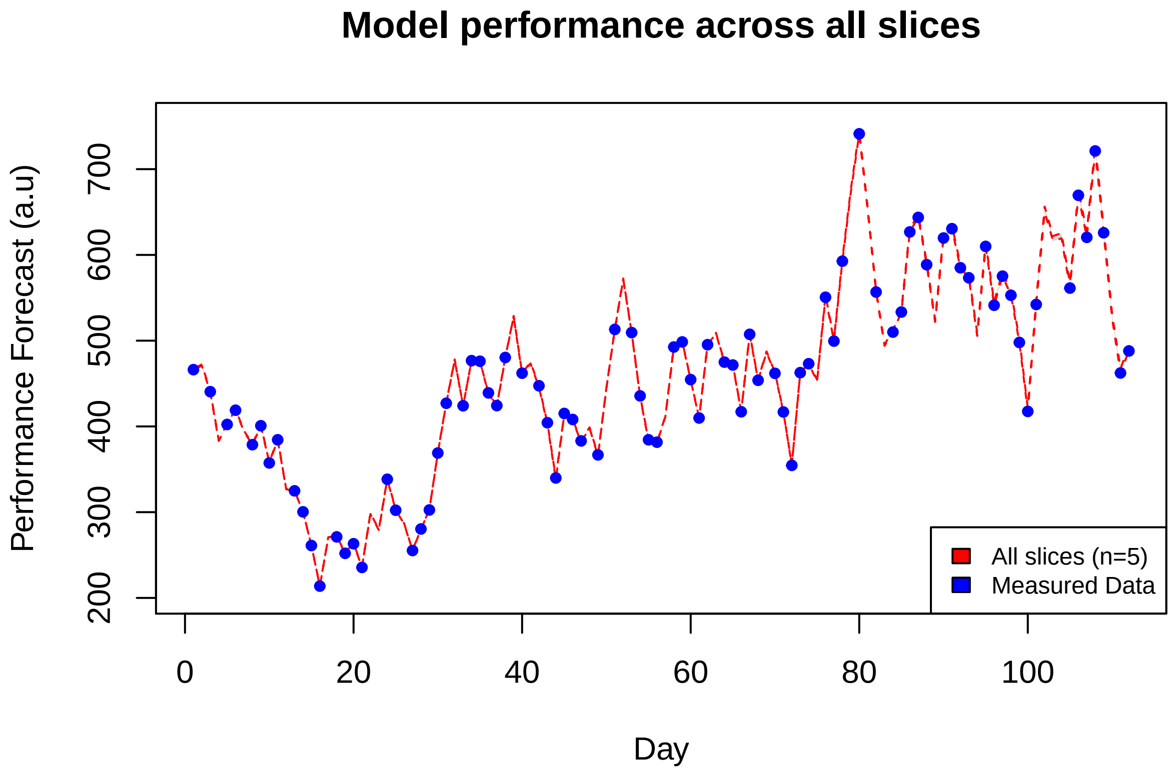
```
# Function dependencies

# Note: The function call itself will try to install any of the dependencies you do not already have. B

install.packages("caret","DEoptim","ModelMetrics","ggplot2", dependencies = TRUE)
```

Call the calibration function with default fitting method (L-BFGS-B)

```
example1 <- standardModel(inputData = mockData,
                          constraints = boxConstraints,
                          startValues = bestGuess)
```

```
## [1] No initialWindow argument supplied
## [1] Defaults used for initialWindow, testHorizon and expandRate
## [1] ------------------------------------------------------------
## [1] initialWindow = 67 days | (60%)
## [1] testHorizon = 22 days | (20%)
## [1] expandRate = 4 days | (4%)
## [1] ------------------------------------------------------------
## [1] Check these are appropriate for your implementation
## [1] COMPLETE: TABULATING MODEL
## [1] ----------------------------------------------------------------------
## [1] SUMMARY STATISTICS (ALL SLICES):
## [1]
##              p_0       k_g      T_g       k_h       T_h    MSE_fnval
## Min.    499.6122 0.3668864 19.23318 0.6683668  9.943799 0.0007177753
## 1st Qu. 499.7519 0.3673576 19.25568 0.6699358  9.975727 0.0097342327
## Median  499.9981 0.4478192 19.80953 0.7483702 10.447084 0.0610305789
## Mean    500.1816 0.4347250 20.10452 0.7355852 10.346576 0.2654872017
## 3rd Qu. 500.3040 0.4949138 21.06396 0.7946013 10.682016 0.5750331510
## Max.    501.2417 0.4966480 21.16024 0.7966521 10.684253 0.6809202708
## [1]
##         RSQ_Train RMSE_Train  RMSE_test   MAPE_test
## Min.       99.990 0.02679133 0.04119537 0.006106039
## 1st Qu.    99.991 0.09866221 0.18771825 0.031104076
## Median     99.999 0.24704368 0.66543188 0.109419623
## Mean       99.996 0.39119711 1.11045118 0.180938780
## 3rd Qu.   100.000 0.75830940 2.21987321 0.362457236
## Max.      100.000 0.82517893 2.43803717 0.395606925
## [1]
## [1] BEST PARAMETERS RECOVERED (TRAIN AND TEST):
## [1]
##         p0      k_g      T_g       k_h      T_h          MSE
## 1 499.9981 0.496648 19.23318 0.7966521 10.68425 0.0007177753
## [1]
##   RSQ RMSE_train  RMSE_test   MAPE_test
## 1 100 0.02679133 0.04119537 0.006106039
## [1]
## [1] ----------------------------------------------------------------------
## [1] PRINTING SUMMARY PLOTS: SEE CONSOLE
```


Key results: Best set found by cross−validation
Training load (a.u)
Measured performance
Fitted set (MAPE): (500, 0.5, 19, 0.8, 11)
Arbitrary units (a.u)
0
200
400
600
0
20
40
60
80
100
Day



We see that the object returned by the function and assigned to example 1 is a list of 4 elements

```
str(example1)
```

```
## List of 4
##  $ bestMem         :’data.frame’:    1 obs. of  10 variables:
##   ..$ p0        : num 500
##   ..$ k_g       : num 0.497
##   ..$ T_g       : num 19.2
##   ..$ k_h       : num 0.797
##   ..$ T_h       : num 10.7
##   ..$ MSE       : num 0.000718
##   ..$ RSQ       : num 100
##   ..$ RMSE_train: num 0.0268
##   ..$ RMSE_test : num 0.0412
##   ..$ MAPE_test : num 0.00611
##  $ summary         : num [1:6, 1:10] 500 500 500 500 500 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:6] "Min." "1st Qu." "Median" "Mean" ...
##   .. ..$ : chr [1:10] "p_0" "k_g" "T_g" "k_h" ...
##  $ allSlices       :’data.frame’:    10 obs. of  5 variables:
##   ..$ slice1: num [1:10] 501.242 0.367 21.16 0.67 9.976 ...
##   ..$ slice2: num [1:10] 500.304 0.448 19.81 0.748 10.447 ...
##   ..$ slice3: num [1:10] 499.612 0.367 21.064 0.668 9.944 ...
##   ..$ slice4: num [1:10] 499.998 0.497 19.233 0.797 10.684 ...
##   ..$ slice5: num [1:10] 499.752 0.495 19.256 0.795 10.682 ...
##  $ slicePerformance:'data.frame':    112 obs. of  5 variables:
##   ..$ slice1: num [1:112] 467 472 441 384 403 ...
##   ..$ slice2: num [1:112] 466 472 441 384 403 ...
##   ..$ slice3: num [1:112] 466 471 440 383 402 ...
##   ..$ slice4: num [1:112] 466 471 440 384 402 ...
##   ..$ slice5: num [1:112] 466 471 440 384 402 ...
```

To shine more clarity on the object returned:

- *bestmem* is the a data-frame containing the 'best' parameter set found from all the slices. This set is selected by the lowest associated MAPE value found from the various slices (for the test/validation set).
- *summary* is a data-frame comprising summary statistics for the various parameters and error measures across slices (e.g. the parameters themselves, the cost function value (MSE), R squared (RSQ) and Root-mean-squared-error (RMSE) on the training and test set, and finally mean-average-percentage-error (MAPE) on test set)
- *allSlices* is a data-frame that provides the raw data for each slice
- *slicePerformance* is a data-frame that corresponds to the final modeled performance values associated with each slice

Thats it! This bespoke function makes it easy to fit the standard FFM using expanding window cross-validation. The function also allows for a bit of deviation from the default process using arguments that can be passed to it at the call. In particular, you can fit the model using an evolutionary strategy (differential evolution, DE) instead of the standard BFGS quasi-Newton method. You are also able to specify the tuning parameters for the cross-validation algorithm. Finally, you can also choose to view the iterative trace provided by the optimiser at run-time. The following code block makes the available options clear:

```r
# Call the function using differential evolution (sans starting values)
standardModel(inputData = mockData,
              constraints = boxConstraints,
              # Note that startValues are not required when using method "de"
              method = "de")

# Call the function using differential evolution but request optim() trace
standardModel(inputData = mockData,
              constraints = boxConstraints,
              method = "de",
              doTrace = TRUE)

# Adjust the tuning parameters for the cross-validation algorithm and use the default gradient approach
standardModel(inputData = mockData,
              constraints = boxConstraints,
              method = "bfgs", # this is the default anyway
              startValues = bestGuess,
              initialWindow = 70, # First window size (train set)
              horizon = 10, # Look forward 10 days (test set size)
              expandRate = 5 # Expand the window by 5 days per new slice
              doTrace = TRUE,)
```

To recap in case that code block wasn't clear enough, your options to deviate away from the defaults within the function call are as follows:

1. To request a *trace* on the optimisation via **doTrace = TRUE**

2. To use differential evolution instead of L-BFGS-B via **method = “de”**. Note you do not then need to include the **startValues** argument, but if you do it will just be ignored anyway.
3. To specify values (in days) for **initialWindow**, **horizon** and **expandRate**