

U.S. Fleet Life Cycle Assessment and Material-Flow Estimation (FLAME-US)

User Manual

**Current developer**: Alexandre Milovanoff (alexandre.milovanoff@mail.utoronto.ca)

**Last update**: 4/30/2021

Table of Contents

[1. Introduction 3](#_Toc70701107)

[2. Installation of FLAME 3](#_Toc70701108)

[2.1. Prerequisites 3](#_Toc70701109)

[2.2. Set-up FLAME 3](#_Toc70701110)

[3. Structure of FLAME 4](#_Toc70701111)

[3.1. <architecture> 4](#_Toc70701112)

[Files to update manually 4](#_Toc70701113)

[Scripts 4](#_Toc70701114)

[Files automatically generated 5](#_Toc70701115)

[3.2. <functions> 5](#_Toc70701116)

[Function script description 5](#_Toc70701117)

[Class 6](#_Toc70701118)

[3.3. <inputs> 6](#_Toc70701119)

[3.4. <outputs> 7](#_Toc70701120)

[Files to define and run the simulations 7](#_Toc70701121)

[3.5. <utils> 9](#_Toc70701122)

[Script descriptions 9](#_Toc70701123)

[4. Environments in FLAME 9](#_Toc70701124)

[The function attribute environment 10](#_Toc70701125)

[The data input environment 10](#_Toc70701126)

[The function results environment 10](#_Toc70701127)

[5. Running and simulating FLAME 10](#_Toc70701128)

[5.1. Running 10](#_Toc70701129)

[5.2. Creating simulations, saving simulation results and reading simulation results 11](#_Toc70701130)

[6. Developing FLAME 11](#_Toc70701131)

[6.1. Adding input data 11](#_Toc70701132)

[6.2. Creating new functions 12](#_Toc70701133)

[7. References 13](#_Toc70701134)

# Introduction

The Fleet Life Cycle Assessment and Material-Flow Estimation (FLAME) model aims to estimate the historical and prospective greenhouse gas emissions (GHG) of light-duty vehicle (LDV) fleets, and the GHG emission impacts of technological or policy changes. This user manual pertains to the U.S. version of the FLAME model, that has been developed in Milovanoff et al. (2019) and updated in Milovanoff et al. (2020).

The FLAME model comprises four modules (vehicle, fleet, automotive material flow and LCA modules). The vehicle module establishes the historical and projected vehicle characteristics (i.e., curb weight, material composition, fuel consumption, utility factor, range) by vehicle technology, size and model year of the LDVs. Then, the fleet module simulates the fleet turnover with annual new sales, outgoing and on-road stocks of LDVs, as well as kilometers traveled and fuel use. The automotive material flow module calculates the material flow of new and scrapped vehicles. Finally, the LCA module calculates the annual life cycle GHG emissions of the U.S. LDV fleet from 2015 to 2050.

The code of the FLAME model uses the R programming language. The code uses both object-oriented and functional programming. It means that some objects are created with data, code and methods. Then, these objects are used in functions with attributes. All functions are defined and created separately and are used linearly or recursively. Specific outcomes of the model (e.g., fleet fuel use, GHG emissions) are provided by some specific functions.

# Installation of FLAME

## Prerequisites

* R version 4.0.3
* R libraries: readxl 1.3.1 ; reshape2 1.4.4 ; tidyr 1.1.2 ; stringr 1.4.0 ; scales 1.1.1 ; DiagrammeR 1.0.6.1 ; DiagrammeRsvg 0.1 ; rsvg 2.1.1
* Developed and tested with RStudio (Version 1.2.5042)

## Set-up FLAME

* Download the model from the GitHub repository on your computer.
* The model was developed using R and RStudio. Install [RStudio](https://www.rstudio.com/products/rstudio/). The RStudio Desktop Open Source License (free) has been used for the development of the model.
* Once R and Rstudio are operational, open the repository on your computer and open 'project\_setup.RProj' in the main folder. It opens RStudio and sets up the working directory to the repository.
* Open and run "architecture/library\_dependencies.R". It installs all the packages used at some point in the model (i.e., in the functions or to create the plots). The model is now operational.

# Structure of FLAME

The FLAME code comprises 5 folders.

* architecture: Contains files that describe the workflow and structure of the FLAME model, such as the list of functions, the list of attributes, the list of inputs.
* functions: Contains the R scripts of the different functions of the model. Each function has a specific goal (e.g., estimate prospective vehicle fuel consumption, simulate the stock of LDVs by model year, size and technology).
* inputs: Contains the exogenous inputs of the functions and some R scripts to format the input data from raw inputs data.
* utils: Contains additional functions to run the model. These functions are operating functions, meaning that they do not effect the model's results but help load and run the functions (e.g., loading the inputs data in the environment, loading the function attribute values, running the simulations and saving the results).

## <architecture>

This folder contains the files that describe the workflow of the model and the values of the attributes to be used in the simulations. The folder contains files that are created automatically by the model, files that should be updated manually and scripts that generate the files.

### Files to update manually

* attribute\_value.csv: This file contains the list of all exogenous function's attributes with their abbreviated name (column “Attribut”e), their type (column “Type”, i.e., 'cha' means character, 'num' means numeric), their value to be used in the current simulations (colum “Value”), their default value (column “Default”), all working values (column “All”) and their description (column “Description”). This file is both updated by the model itself and should also be updated by the user. See 5.2 Creating new functions for more details.
* list\_of\_internal\_attributes.csv: This file contains the list of function’s attributes that should be considered as internal attributes in the model. Internal attributes are defined endogenously in the model and are not defined by the user. Examples of internal attributes are the vehicle size and technology for functions that perform specific tasks given the size and technology.

### Scripts

* model\_framework\_setup.R: This script updates all the files contained in the architecture folder. It creates the list of functions included in the model, their associated attributes, their inputs (data inputs but also interdependencies between functions). This script also saves all the data inputs into the data input environment. This script should be used every time the structure of the FLAME model is altered.
* library\_dependencies.R: List and install the set of libraries used at some point in the model.

### Files automatically generated

* function\_attributes.csv: Contains the list of model's functions and their attributes by type ('Internal' for endogenous attributes; 'External' for exogenous attributes).
* function\_attributes\_matrix.csv: Contains the same information than function\_attributes.csv but only for external attributes and in a matrix format (with the rows being the functions and the columns the external attributes).
* function\_explicit\_attributes.csv: Contains the list of model's functions with the external attributes.
* function\_inputs.csv: Contains the list of model's functions and their inputs by type (.csv, .xlsx, .R).
* function\_inputs\_matrix.csv: Contains the same information than function\_inputs.csv but in matrix format (with the rows being the inputs and the columns the model's functions).
* function\_sources\_matrix.csv: Contains the same information than function\_inputs\_matrix.csv but only for model's function as inputs. This matrix shows the interdependencies of the functions and is used by some "utils" functions to fasten the simulations.
* functions\_descriptions.csv: Contains the list of model's functions and their functional descriptions.

## <functions>

This folder contains the scripts of the functions. Note that the functions contained in the 'utils' folder are operating functions, i.e., they help to run the model but do not directly contribute to the model's results. Details regarding function descriptions and the organization of the functions can be found in the “architecture” folder (“flame\_workflow\_chart.png” for the workflow chart, “function\_descriptions.csv” for the function descriptions).

There are two types of functions included in the “functions folder”: Internal and result functions.

Internal functions perform specific tasks to a set of internal and external attributes. For example, survival\_rate\_f is an internal function that returns the survival rate of a vehicle of a specific age and size (internal attributes) based on a survival rate model defined by the user (external attribute). Outputs of internal functions are not supposed to be used for analytical purposes by the user, they perform specific tasks for the other functions.

Result functions return outputs that can be used for analytical purposes, such as fleet\_fuel\_u\_f that returns the fuel used by the LDVs over the simulation period. The attributes if result functions are external attributes. Ultimately, result functions are simulated by the user for analytical purposes.

### Function script description

* Each script is a function. The names have to be the same between the script and the function. Otherwise, the model does not work properly.
* The description of the function within the script is preceded by "#' Function"
* The functions have attributes:
  + The internal attributes are attributes that are not defined by the user and they do no have default values.
  + External attributes are defined by the user in 'architecture/attribute\_value.csv' and are marked with the NA default value in the function.
* When a function is called in R, a local environment specific to the function is created. To initiate the values of the external attributes in this local environment, the first line of the function runs the operating function 'attribute\_f' with the name of the model's function. This function stores the external attribute with their specific values in the local environment. The values of the external attributes derive from 'architecture/attribute\_value.csv'.
* If a result function uses outputs from another result function, this latter one should be called using the following line  
  "X\_f\_res <- do.call(fun\_res\_f,list(fun\_name="X\_f"))".
* If a function uses an internal function, a simple “do.call(X\_f,list(internal\_attribute1, …))” works properly.
* Note that an output of a result function is a list of elements (preferably data frames).

### Class

Classes can be defined and used in FLAME for object-oriented programming. The "class.R" script defines the classes.

Two classes are created: vehicleClass and fleetClass.

An object belonging to the vehicleClass represents all model year vehicles associated with a specified technology and size. Data stored in a vehicleClass object are fuel consumption, utility factor, specifications (horsepower, acceleration time), battery type, material composition and component composition for all model year vehicles. This class is used in the result function vehicle\_module\_f and the internal functions that use vehicleClass objects as internal attributes are: vehicle\_initialize\_f, vehicle\_fc\_changes\_efficiency\_f, vehicle\_weight\_changes\_f, vehicle\_fc\_changes\_weight\_f, vehicle\_utility\_factor\_f.

An object belonging to the fleetClass represents the LDV fleet with all the vehicle technology, size, and model years over the simulation period. Data stored in a fleetClass object are lists of vintaged vehicle stocks and of scrapped vehicles by technology, size and age, technology market shares, total sales and stock data. The fleetClass is used in the result function fleet\_vint\_stock\_f and the internal functions that use fleetClass objects as internal attributes are: fleet\_initialize\_f, fleet\_vint\_stock\_initialization\_f, fleet\_vint\_stock\_update\_with\_sales\_f, fleet\_ldv\_stock\_update\_gcam\_f, fleet\_vint\_stock\_update\_with\_stock\_f, fleet\_ldv\_stock\_update\_constant\_f, and fleet\_vint\_stock\_update\_with\_stock\_f.

## <inputs>

The “inputs” folder contains all exogenous inputs data that are used by the functions. Data are categorized into three groups: data, model and user.

Files in the “inputs/data” folder derive directly from primary sources or are created from reports and other papers.

Files in the “inputs/model” folder are files formatted by scripts contained in the “inputs/scripts” folder. Usually, these files are formatted copies of files contained in the “inputs/data” folder or tables created from API.

Files in the "inputs/user" folder are inputs created by the user, such as tables to match different names from different tables.

Input data are preloaded into an Rdata file in FLAME (see 4.Environments in FLAME for more details) and the file can be found in the “inputs” folder (“input\_data\_environment.RDATA”). Managing the pre-loaded input data is possible with the “data\_input\_management.csv” file. This file contains the list of input data to save in the Rdata file. The “Variable\_name” column refers to the name of the data frame associated with the input data, the “File” column refers to the path of the input data, the “Sheet\_name” refers to the name of the spreadsheet if the input data is an .xlsx file, the “File\_format” is the format of the file (e.g., .csv, .xlsx) and finally “Rownames” refer to the column that contains the row names in the input data. If “Rownames”is empty, the data frame will not have row names.

## <outputs>

The “outputs” folder contains the saved results of the FLAME model and the files that define the simulations. This folder can be kept outside of the main folder of the FLAME model to separate the simulation results and the code of the model.

The “outputs/results” folder contain the saved results in .Rdata format. The saved results are saved using the operating functions contained in the “utils/write\_f.R” script. Then, these results are read using the operating functions contained in the “utils/read\_f.R” script.

### Files to define and run the simulations

In FLAME, outputs of result functions can be saved according to three types of simulations.

The first and most important type is the “scenario” simulation type. It is also used by the other simulation types. A “scenario” simulation runs the specified result function across multiple sets of attributes-values. For example, a “scenario” simulation is defined with two external attributes (e.g., att1, att2) and three sets of attributes-values (e.g., first set is [att1 = low, att2 = constant], second set is [att1 = high, att2 = constant] and third set is [att1 = high, att2 = high]). In that “scenario” simulation, the result function and all upstream functions are run three times, once per set of attribute-values. Note that all other external attributes use their default values as specified in “architecture/attribute\_value.csv”. The file that defines the “scenario” simulations is:

* scenarios.xlsx: Excel spreadsheets that define “scenario” simulation.
  + 'Scenario' column contains the name of the scenario ;
  + 'Attribute' column contains the name of the attributes ;
  + 'Type' column contains the type of the attributes (e.g., num or cha) ;
  + The other columns define the sets of attributes-values with with the first row being the name of the runs and the other rows being the attribute's values.

The second type of simulation is called “simulation”. A specified result function is run across multiple sets of attributes-values. The difference with the “scenario” simulation type is that the “simulation” type uses the set defined in a “scenario” type then adds new sets of attributes-values. For example, a “simulation” simulation is defined with three external attributes (att3, att4, att5) and two sets of attributes-values ([att3 = 1, att4 = REF, att5 = def] and [att3 = 2, att4 = HOP, att5 = low]). If the “scenario” simulation of reference is the scenario previously described (with two external attributes and three sets of attributes-values), the result function is run six times, or two times for the two sets of attributes-values defined in the “simulation” type and the first set of attributes-values defined in the “scenario” type, then two times for the two sets of the “simulation” type with the second set of the “scenario” type, and finally two times associated with the last set of the “scenario” type. Note that the “simulation” simulations can be defined as discrete simulations or continuous simulations. In a discrete simulation, all sets of attributes-values are manually defined. In a continuous simulation, the sets of attributes-values are defined from numerical ranges for the attribute values. Therefore, continuous simulations require at least one numerical external attribute. A “simulation” simulation type can be used across multiple result functions and multiple “scenario” type. Their utility is to assess, for example, the influence of a set of attributes (e.g., methane leakage rates and electricity decarbonization) on other attributes (e.g., deployment levels of electric vehicles) without having to define six sets of attributes-values initially. Additionally, one “scenario” simulation (e.g., deployment levels of electric vehicles) can be run through multiple “simulation” simulations. The file used to define “simulation” simulations is:

* simulations.xlsx: Excel spreadsheets that define the sets of attributes-values associated with “simulation” simulation types. Each spreadsheet represents a function with the following syntax. Each column in the simulation represents a specific run with defined attributes. All attributes not specified have default values.
  + 'Simulation' column contains the simulation name.
  + 'Attribute' column contains 'name' as the second row of the simulations then the list of attributes to be udpated
  + 'Type' column contains the type of attribute:
    - 'attr' is a normal attribute (not to keep in the simulation output).
    - 'attr\_tk' is an attribute to keep in the simulation output.
  + The other columns define the values of the attributes in the simulation. The first row of the simulation defined the name of the run, then the other rows are assoociated with the attributes and define the attribute values to use.

The last type of simulations is “sens\_analysis”. In this type, a list of external attributes is provided by the user and the result function is run across a selected “scenario” simulation with all the values attached to all external attributes. To continue the example defined before, if the “sens\_analysis” is defined as three external attributes, each with three potential values, and if the reference “scenario” simulation is the one previously defined (two attributes with three sets of attributes-values), then the specified result function is run twenty-seven times, nine times (three attributes each with three potential values) for each set of attributes-values defined in the “scenario” simulation. Note that the “sens\_analysis” simulations can be defined as singe factor analysis (as previously defined) or as multi factor analysis. In multi factor analysis, a function to evaluate is defined as well as conditions. Then the values of the external attributes defined by a single factor sensitivity analysis are evaluated on the specified function and the values that return the low and the high values of the function to evaluate with the specified conditions are returned.

The file associate with the “sens\_analysis” type is:

* sens\_analysis.xlsx: Excel spreadsheets that contain the list of attributes to simulate by sensitivity analysis. For a given sensitivity analysis, the specified function is run with all the values of the specified attributes (from 'architecture/attribute\_value.csv', column 'All') considering default values for other attributes, except the attributes defined in the “scenario” simulation. The “single\_factor” spreadsheet is:
  + 'Sensitivity' column contains the number of the sensitivity analysis.
  + 'Sensitivity name' column contains the name of the sensitivity.
  + 'Attributes' column contains the list of attributes to perform the sensitivity on. The attributes are separated by “;”.

A script template (“results\_script.R”) to run and save the simulation results is provided in the “outputs” folder. This script template first loads the FLAME model in the global environment. Then, the path of the “outputs” folder is specified (if the user wants to keep the outputs folder in the main FLAME folder, then the path is “outputs”).

## <utils>

The “utils” folder contains the functions that are necessary to operate the model but do not affect the model's results.

### Script descriptions

* data\_f.R: Contains functions to format and process data frames.
* data\_processing\_f.R: Contains functions to rename strings.
* model\_builder.R: Script that reads all functions contained in the model and extract attributes, inputs and descriptions. The results are written in the "architecture" folder.
* plot\_f.R: Functions that help plot graphs.
* read\_f.R: Functions that read simulation results and convert them into data frames.
* utils\_f.R: Diverse functions that are used by the model.
* write\_f.R: Functions that simulate the model and write the simulation results into R files.

# Environments in FLAME

An environment in R is a virtual space that stores variables (symbols and objects). Without any specification, the tasks performed by the R code add entries in the global environment. For example, when a data frame “dt” is created, this data frame can be found on the right side of RStudio in the “Data” category of the global environment. In order to fasten the simulations and to use consistent external attributes throughout the simulation, FLAME creates three local environments that are stored withing the global environment: “fun.att.env”, “fun.datainput.env” and “fun.res.env”.

### The function attribute environment

“fun.att.env” is a local environment that contains two data frames: attribute\_value and function\_attributes. Attribute\_value is a copy of the csv file “architecture/attribute\_value.csv”. Therefore, it contains the list of external attributes and their associated values. With this environment, when a function is called, the model will search the values of the associated external attributes and will load them in the local environments created by the functions. This environment is generated with the operating function “utils/utils\_f/load\_attribute\_value()”. This operating function is called in the “model\_setup.R” script.

### The data input environment

“fun.datainput.env” is a local environment that contains a copy of the all the data frames associated with the input data. When a function calls an input data, the model will load the associated data frames in the local environment of the function. This environment is created from the operating function “utils/utils\_f/load\_attribute\_value()”. The environment is created by loading the “inputs/input\_data\_environment.Rdata” file. This Rdata file is created by the operating functions “utils/utils\_f/save\_input\_data\_f()” and is managed by the “inputs/data\_input\_management.csv” file. The functions load input data with the operating function “get\_input\_f(input\_name = “NAME\_OF\_THE\_INPUT)”.

### The function results environment

“fun.res.env” is a local environment that contains the results of the already called result functions. Initially, this local environment does not exist. When a result function is called by another function using the operating function “utils/utils\_f/fun\_res\_f(fun\_name)”, the model will first verify if the called function is not already present in the “fun.res.env” environment. If the function results are already in the environment, the model loads the results in the local environment of the second function. If the results function is not present, the mode calls it, then saves it in the “fun.res.env” environment and then loads it in the second function.

# Running and simulating FLAME

There are two main ways to run the FLAME. The first one is simply by calling the functions of interests. This solution is particularly interesting when developing but does not save the results externally and requires the model to run every time. The second solution is to create simulations, to save the results of the simulations and to read them. Note that only result function can be called without specifications. As previously mentioned, internal functions require internal attributes as they perform specific tasks within the functions. They can therefore be called if the required internal attributes are provided.

## Running

To simply run the model:

* Once the model is set up, you can run the functions of the model individually by opening "project\_setup.RProj" in the main folder. It opens RStudio and sets up the working directory.
* Then run the "model\_setup.R" script. It loads the functions in the R environment, the function attributes and the data inputs.
* You can now run the functions individually by calling their names (e.g., fleet\_lci\_f()). You can also call the functions by using the operating function “fun\_res\_f(fun\_name)” (e.g., fun\_name = “fleet\_lci\_f”). This second solution ensures that the results of the function is saved in the “fun.res.env”. Therefore, if you call “fun\_res\_f(fun\_name)” again, the model extracts the results directly from the environment and is faster.
* Note that this simple method to run the model returns the results associated with the set of attributes-values contained in the “architecture/attribute\_value.csv” file.

## Creating simulations, saving simulation results and reading simulation results

The second solution creates simulations, stores them and return them.

* Once the model is set up, you can create simulations and run the selected functions across the specified simulations. As a reminder, there are three types of simulations: “scenario”, “simulation”, and “sens\_analysis”. The “simulation” and “sens\_analysis” still require the definition of a “scenario”.
* The simulations can be created in the outputs folder, more precisely in the “scenario.xlsx”, “simulations.xlsx”, “sens\_analysis.xlsx” file. First, start by creating the “scenario” type of interest. Then, if needed, you can create the “simulation” or “sens\_analysis” simulations.
* To save the results of the simulations, open the “results\_script.R” file in Rstudio, load the “model\_setup.R” script to load the functions, operating functions, input data and default attributes. Then specific the path of the “outputs” folder (you can keep “outputs\_path <- "outputs"” if you keep the outputs folder in the main FLAME folder). Then you can simulate the “scenario” type with the operating function “write\_scenario\_f()”, the “simulation” type with the operating function “write\_simulation\_f()” and the “sens\_analysis” type with the operating function “write\_sens\_analysis\_f()”.
* One the simulations have been saved, you can read them and extract a formatted version of the outputs by using the operating functions “read\_scenario\_f()”, “read\_simulation\_f()”, “read\_sens\_analysis\_f()”

# Developing FLAME

## Adding input data

Adding input data in FLAME is possible and requires the developer to follow a strict procedure:

* The input data can be added in the “inputs/data” folder. Preferably a .csv file, but .xlsx files also work.
* Then, a row needs to be added to the “inputs/data\_input\_management.csv” file. Adds the name of the input data to be considered in the model, the path of the file, the spreadsheet name if the file is a .xslx, the format of the file and the column to be considered as row names (no row names if left empty).
* The pre-loaded data frames of the input data contained in the “inputs/input\_data\_environment.RDATA” needs to be updated with the new input data. Otherwise, the FLAME model does not include the new input data. To do so, open the “architecture/model\_framework\_setup.R” script and load the two last lines (“source("utils/utils\_f.R")” and “save\_input\_data\_f()”). The new input data is not pre-loaded along with the other input data.
* Finally, to call the input data in a function, uses the operating function “get\_input\_f(input\_name = “NAME\_OF\_THE\_INPUT)”.

## Creating new functions

Creating new functions is possible in FLAME and requires following a strict procedure:

* Create a new script in the “functions” folder. Name this script with the name of function to create (e.g., “new\_function\_f”). Note that the function name has to finish by “\_f”, otherwise the model will not recognize it.
* To add a description to this function, adds a line in the first lines of the script starting with “#' Function:”.
* Create the function by using the normal procedure in R   
  (e.g., “new\_function\_f <- function(external\_att1=NA, external\_att2=NA){ … }”)
* If the function is an internal function, then adds the internal attribute in the list of function attributes. If the internal attribute is a new attribute in the model, don’t forget to add it to the list of internal attributes found in the “architecture/list\_of\_internal\_attributes.csv” file. Otherwise, the model will not consider this attribute as internal and will consider it as external, requiring the user to specify a value.
* The function can use any attributes already created. If the function requires a new external attribute (e.g., “new\_function\_att1”), this attribute can be added to the list of function attribute and it will be updated in the model framework in one of the following steps.
* To write the function, always start with the line calling the operating function that loads in the local environment of the function the external attributes (“attribute\_f()”)  
  (e.g.,: “attribute\_f(fun\_name = "new\_function\_f")”)
* To call input data in the function, always uses the operating function “get\_input\_f()”.  
  (e.g., “input\_data <- get\_input\_f(input\_name = “input\_data\_name”)”)
* To call the results of another result function, always uses the operating function “fun\_res\_f()”.  
  (e.g.,: “fleet\_lci\_f\_res <- do.call(fun\_res\_f,list(fun\_name=’fleet\_lci\_f’))”)
* Outputs of a result function should be a list of elements, ideally data frames, with specified names.  
  (e.g., “return(list(new\_function\_dt1= new\_function\_dt1, new\_function\_dt2= new\_function\_dt2))”)

# References

Milovanoff, A., Kim, H.C., De Kleine, R., Wallington, T.J., Posen, I.D., MacLean, H.L., 2019. A Dynamic Fleet Model of U.S Light-Duty Vehicle Lightweighting and Associated Greenhouse Gas Emissions from 2016 to 2050. Environ. Sci. Technol. 53, 2199–2208. https://doi.org/10.1021/acs.est.8b04249

Milovanoff, A., Posen, I.D., MacLean, H.L., 2020. Electrification of light-duty vehicle fleet alone will not meet mitigation targets. Nat. Clim. Chang. 10, 1102–1107. https://doi.org/10.1038/s41558-020-00921-7