This software in CalciumBufferingTLS.2 implements the stochastic gradient descent algorithm to maximize the likelihood function in the Total Least Squares sense in order to to fit Ca2+ fluorescence data to a Ca2+ buffering model and provide estimates of parameters. This document describes a user interface for the algorithm implemented in the statistical programming environment R. To use the code provided, the user must have an installation of R. The R base system may be downloaded from http://www.r-project.org/. The R base system comes with a simple graphical user interface (RGui) and requires the integrated development environment RStudio, which can be downloaded from http://www.rstudio.com/). As of the writing of this documentation, the code has been tested on R version 3.1.1, and may not work correctly with either previous or later releases.

The algorithms make use of functions provided by two packages which are not included in the base distribution: minpack.lm and mosaic. No special action is required on the part of the user, if the packages fail to load then the program will attempt to install them. If the packages cannot be installed, an error condition will occur. The most likely causes of a failure to install these packages are using the incorrect version of R or lacking an internet connection.

The algorithms are provided in the distributed R source files Run.Ca.Buffering.TLS.R and Ca.Buffering.TLS.Sub.R. The files can be conceptually divided into three sections. The first section (Run.Ca.Buffering.TLS.R) contains user defined parameters which must be specified by modifying the source code. The function library and executable script are contained in Ca.Buffering.TLS.Sub.R. These sections will generally not need to be modified.

**Getting Started**

After installing R and RStudio, open RStudio. RStudio contains 4 window panes. The default layout includes a source code editor in the top left, R console on the bottom left, an environment pane in the top right, and file selection pane on the bottom right.

From the top menu, select File -> open project and select “CalciumBufferingTLS.RProj” from the source directory. Then select File -> open file and open “Setup.Ca.Buffering.TLS.R”, then select Code -> Source to run the script. This script will download and install required R packages.

The process may try to update R packages that are already loaded, in which case the message “Error in install.packages : updating loaded packages” will be written to the R console, and the user will be prompted to restart R to continue the installation. If this occurs, select yes to restart R. If all required packages are successfully downloaded and installed, the message “Packages were successfully installed and loaded,” will be printed to the R console.

The source directory also includes a sample data set. Users are encouraged to fit the test data set to orient themselves with the interface and parameter options. The data sets provide a template for users to apply to their own data. To fit the model, select file -> open file and open Run.Ca.Buffering.TLS.R. The file specifies parameter values controlling the fitting routines which should be modified by the user for their specific requirements. These parameters are discussed in the section “User defined parameters” of this document. The default values are set to be suitable for the sample data set accompanying the source code.

To start the analysis, execute Run.Ca.Buffering.TLS.R by selecting code -> source from the top menu. The user will be prompted to select the required data files. A brief description for each file is presented in the R console. The example data set contains fluorescence measurements, fluorescence measurement standard errors, calcium increment measurements and calcium increment measurement standard errors in separate csv files: f.csv, f.sem.csv, ca.csv, and ca.sem.csv, respectively. Select the appropriate files when prompted. The fitting process will then execute as described in the Fitting Procedure section of this manual.

**User defined parameters**

As is typical with model fitting procedures, the user must specify a number of parameters to define and constrain the model. The parameters defined in Section I are as follows:

*F.MIN and F.MAX*

Model fitting procedures uniformly involve varying a set of model parameters to minimize some objective or loss function. The total least squares procedure involves an additional step of estimating the measurement errors to minimize the total squared error. The data for the Ca2+ buffering model consist of two fluorescence measurements, a fluorescence measurement before the Ca2+ increment or stimulus (finitial) and a fluorescence measurement after the stimulus (ffinal), and an independent determination of the change in total Ca2+ concentration. For convenience of notation we will denote the vector of fluorescence measurements (finitial, ffinal) as the vector **f** = (fi, ff), and the change in total Ca2+ concentration as y.

The total least squares estimation procedure involved estimating measurement errors for each measurement. Let us call this error **df**, such that the weighted distance between the observed data and the surface defined by the model is minimized. To avoid spurious results, we must ensure that **f** + **df** is a physically reasonable measurement. That is, we should require at least that **f** + **df** is greater than zero and less than the maximal fluorescence. Consider the equation (Eq. 1 from the paper) for converting normalized fluorescence F = (f/fmax) into [Ca2+ ]Free

Where RF is the dynamic range of the indicator dye. For F + dF < 1/RF or F + dF > 1, [Ca2+]Free is negative, which is physically impossible; for F = 1, [Ca2+]Free is undefined. We exclude these possibilities by requiring that F.MIN < **f** + **df** < F.MAX, for some user specified values F.MIN and F.MAX.

***Parameters, parameter bounds, and fixed parameters***

The full model contains nine total parameters:

Dynamic range RF, rf

The kd of the indicator dye, kd.dye

Total concentration of the indicator dye, bt.dye

The kd of endogenous buffers to be modeled, kd.1 and kd.2

The total concentration of the corresponding buffers, bt.1 and bt.2

a nonsaturable Ca2+ buffer, kappa.nonsaturable

This represents a buffer with sufficiently low affinity that its buffering capacity does not change appreciably over the range of free Ca2+ values investigated,

Ca2+ inaccessible fraction of intracellular volume VE , accessible.volume

All of the parameters are specified in a single vector, beta

The user must specify starting parameters and upper and lower limits for a particular run. The user must also designate which parameters are to be fixed at their starting values, and which parameters are varied to perform the fit. The logical vector parameter.is.fixed specifies which parameters are to be considered fixed. Values set to "TRUE" are treated as fixed parameters. Note that R is case sensitive, the logical values TRUE and FALSE must be in all capital letters.

Although the user must specify starting values for the parameters, the stochastic gradient descent algorithm is designed to escape local "false" optima in the parameter space that can be attributed to measurement noise. Therefore, it is not especially sensitive to starting estimates provided that a unique optimum exists. Extremely poor choices for starting parameters, however, may cause the algorithm to behave poorly.

The algorithm requires lower and upper bounds on parameters, and poor performance is almost guaranteed by improper values. Simple physical considerations provide useful guidelines for choosing good boundaries. All kd's must be positive, and excessively high kd values would prevent saturation over the observable range of [Ca2+]Free. Therefore, the upper bounds on the endogenous buffer kd's may be taken to be some multiple of the dye kd. Extremely low affinity buffers can be modeled as nonsaturable buffers with kappa.nonsaturable; this parameter must be nonnegative and an upper limit should be estimated, possible by considerations of the preparation. accessible.volume, VE, must be between zero and one. However, some allowance should be made for under estimation of the total volume. A reasonable range for VE may be estimated from knowledge of the preparation. It is unlikely that cells have a fraction of accessible volume much less than about 0.5.

*max.iterations*

max.iterations is the number of iterations the stochastic gradient descent algorithm will perform before terminating. Because of the stochastic nature of the algorithm, it is difficult to develop a well-defined convergence criteria for termination. We have decided, therefore, to terminate the fitting procedure by reaching a pre-defined number of iterations. The quality of the fit and convergence of the algorithm can be investigated by means of bootstrap resampling.

***Parameters controlling the learning rate***

For the gradient descent algorithm, a vector of parameters j, the jth approximate solution, is used to calculate the next approximation as (equation S-11 from the supplement):

Where Q is the total least squares objective function and the sum is over the set of N observations. For the stochastic variant, rather than summing over all N observations we select one or more observations at random for each update step. The parameter  usually referred to as the learning rate, is a scaling parameter which controls the rate of convergence. If  is set too small, then the algorithm will require many iterations to converge. Conversely, if  is set too large, the update step will overshoot and the algorithm will diverge. It is often desirable to vary  so that larger updates are taken during earlier iterations, and step size decaying with each iteration. For this implementation the behavior is controlled by two parameters:

*num.steps*

The initial value  for the first iteration is set by  = (beta.upper – beta.lower)/num.steps, where beta.upper and beta.lower are the parameter bounds and num.steps is a user defined parameter. As num.steps increases the initial updates will be smaller. As the size of the parameter space defined by the limits increases the initial steps will decrease in magnitude.

*decay.constant*

The decay.constant parameter controls the rate at which decreases. Specifically, on each iteration  decay.constant.

***Parameters for bootstrap estimation of confidence intervals***

*do.bootstrap.estimate and bootstrap.replicates*

Standard errors for the parameters are estimated in the usual manner, that is, based on a linear approximation at the solution. In our experience, the estimates provided by the linear approximation are often inaccurate. Wherever we have reported standard errors, the estimates were provided by bootstrap resampling, and we would encourage users to do the same. Previously, this software did not return any estimates of parameter standard errors, save for bootstrap estimates, because of the errors introduced by the linear approximation. They are returned now as a convenience to the user. Although they do not provide a very accurate estimate, the linear approximation may be used as a guide to decide for which data sets to estimate bootstrap standard errors.

Bootstrap estimates may be provided by setting do.bootstrap.estimate = TRUE (as noted above, R is case sensitive). If error estimates are desired, then the number of bootstrap replicates is specified by the value of n.bootstrap.replicates. Fits to bootstrapped data sets are performed in parallel, with the number of parallel process controlled by n.threads. A significant shortcoming of TLS is the computational time required. This difficulty is significantly exacerbated by performing bootstrap estimates of confidence intervals.

This can be partially ameliorated by relaxing some of the controls on the SGD algorithm. The default settings are useful when there is no reliable a priori estimate for the affinities and concentrations of the calcium buffers. For fitting to bootstrap replicates, the parameters from the original data set provide good starting estimates, which may allow for tighter upper and lower parameter bounds or using fewer iterations. Even with these modifications, confidence intervals or standard error estimates will be time consuming. Therefore, this step is best saved for selected data sets.

***Advanced settings***

The ‘Advanced settings’ section controls a range of disparate behaviors, the unifying theme is that the parameters in this section control heuristic methods for improving the quality of fit or decreasing the run time. These heuristic mechanisms do not have sound theoretical support, but were instead developed by trial and error on simulated data sets.

*Parameters controlling behavior when the solution lies on the boundary*

If the current parameters beta lie too close to the parameter boundaries, beta.upper and beta.lower, then the algorithm may become stuck at the boundary. Each update step in the stochastic gradient descent algorithm is essentially random, and in practice the solution often approaches the boundary at some point. The algorithm employs some heuristic mechanisms to ensure that the algorithm does not get trapped at the boundary when it inevitably approaches.

This ‘escape’ behavior is controlled by several parameters under the ‘Advanced features’ heading of Run.Ca.Buffering.TLS.R: parameter.boundary.margin, p.restart, p.restart.decay, p.restart.grow.

parameter.boundary.margin: Controls how close to the boundary the solution is allowed to be before restarting. If at any point we have abs(beta - beta.upper) < abs(beta.upper - beta.lower) \* parameter.boundary.margin or abs(beta - beta.upper ) < abs(beta.upper - beta.lower) \* parameter.boundary.margin, the beta is reset either randomly or the best parameters so far.

p.restart: If the parameters are "too close to the boundary" as defined above, then the reset behavior described above is executed with probability p.restart

p.restart.decay: If the parameters are reset, then p.restart = p.restart \* p.restart.decay, this option is included to prevent the algorithm from restarting at every iteration, which may happen if the parameter limits are poorly chosen or on certain data sets.

p.restart.grow: If the parameters are on the boundary and a restart action is not performed, then p.restart = p.restart \* p.restart.grow.

Setting p.start, p.start.grow or p.start.decay to 0 will cause the program to never perform a parameter reset from lying on the boundary. This can significantly reduce the running time for the algorithm, but may significantly reduce the accuracy of the estimation procedure.

*p.check.obj*

Evaluating the TLS objective function is computationally expensive and this prohibits us from performing a full evaluation on each update. The value of the objective function is calculated with probability p.check.obj on each iteration. A higher probability of p.check.obj will cause more objective function evaluations to execute, increase the probability of finding the minimum and increase the run time.

*p.goto.best.par*

Each iteration, with probability p.goto.best.par, set the current value of beta to the best value observed so far.

**Running in batch mode**

The data files may be specified in one of two ways. If interactive.file.chooser = TRUE then when the script is ran the user will select the files interactively through a file chooser dialog box. If interactive.file.chooser = FALSE then the file names and data directory path must be specefied.

***Data files and formats***

The user must provide four separate data files, one containing the fluorescence, one containing total Ca2+ increment measurements, and because TLS requires estimates for the weights of the residuals for each measurement, the user must provide two separate files containing the standard errors for each measurement. All data files must be in csv format. It is assumed that the first row of each file contains variable names for each column, and that the data starts on line 2. Each file must contain the same number of rows. When the script is run, the user will be presented with a dialog box and prompted to select each file. The format and content of each file are briefly described below, and a sample data set has been included with the source code for reference.

***Fluorescence measurements***

The fluorescence data must be normalized, as the algorithm expects the files to contain F = f/fmax. It is also good practice to remove data points very close to one (f close to fmax) as these points will provide very little information and may be safely removed without degrading the results. Our simulations have shown that near a solution the weighted residuals from measurements with F near one are typically very small. This arises from of the nonlinearity in the equation for [Ca2+]Free. As we approach F=1, infinitesimally small changes in the F will provide arbitrarily good agreement with the measured Ca2+ increment. Hence, they provide no information and may be safely discarded.

The first prompt and dialog box will ask the user to provide a csv file containing the fluorescence measurements. The file must contain two columns, the first containing the fluorescence before the stimulus or Ca2+ increment, and the second containing fluorescence after the Ca2+ increment. An example is provided in the file named fluorescence.csv.

***Fluorescence measurement errors***

Fluorescence measurement errors are taken as random variables from normal distributions, and the correct weights for the fits are the inverse of their variance. The analysis presented in the paper includes only data sets where no or minimal fluorescence decay is observed between measurements. In this case, the fluorescence before the stimulus is taken to be the average value of fluorescence over n time points preceding the stimulus, and the measurement errors are taken to be the standard error of the mean.

In principle, the measurement errors for different data points may be heterogeneous, i.e. the data may be heteroskedastic. The fitting algorithms supplied also accommodate for this possibility. However, this facility should be used with some care. Taking fluorescence measurements as an example again, we know from analyzing our experimental data that fluorescence measurement errors are approximately independent of the fluorescence values as long as the photomultiplier tubes are sufficiently below saturation, i.e. the errors are homoscedastic. As described above, we also have independent estimates for the measurement uncertainties for each point in our data sets. Due to sampling errors, these estimates have a finite variance. Therefore, naively using the independent standard error estimates for each measurement in fact introduces an additional source of noise, and one may be better served to use their average for the entire data set. This is the method employed for the analysis in the paper and was used to provide the estimated errors in the accompanying file fluorescence.sem.csv.

The second prompt and dialog box will ask the user to provide a csv file containing the fluorescence measurement errors. The file must contain two columns, with the first column containing the errors corresponding to fluorescence measurements before the stimulus or Ca2+ increment, and the second column containing fluorescence measurement errors after the Ca2+ increment.

**Changes in total Ca2+**

The third prompt and dialog box will ask the user to provide a csv file containing the estimated total Ca2+ increments. The file must contain one column containing the total Ca2+ increment. The accompanying file delta.ca.total.csv illustrates the required format and sample contents.

**Ca2+ increment measurement errors**

The fourth and final prompt and dialog box will ask the user to provide a csv file containing the measurement errors corresponding to the total calcium increments. The file must contain one column with the estimated standard errors. The user is again allowed to provide independent estimates for the measurement error for each data point, and the warnings provided with respect to error estimation for fluorescence still apply. The accompanying file delta.ca.total.sem.csv illustrates the required format and sample contents.

**Fitting progress**

After the user responds to the following series of prompts, the program will start the fitting procedure. The procedure proceeds conceptually as follows:

The first step in the fitting procedure is a deterministic fit of the model using the Levenberg-Marquadt algorithm, the total squared error and parameter values after this initial fit will be output to the R console. These parameter values are used to start the stochastic gradient descent, and both the parameter values and total squared error are saved. While the stochastic gradient descent is running, status updates are periodically provided. The iteration number will be printed every nth iteration, where n is the number of rows in the csv files. Whenever the stochastic gradient descent update leads to a solution with a lower total squared error than the saved values, the Levenberg-Marquadt algorithm will be rerun using the new parameters as starting parameters. The resulting solution is guaranteed to be the best solution found so far, and the new parameter values and total squared error are output to the R console and saved. If at any point, the current test parameters lie on the upper or lower bounds specified, a new set of test parameters is randomly chosen between the upper and lower bounds and the Levenberg-Marquadt algorithm is run using the randomly chosen parameters. The stochastic gradient descent algorithm will continue from the result. The process will continue until the maximum number of iterations has been reached. Due to the stochastic nature of the process, monitoring convergence would be a difficult and error prone task. The stochastic gradient descent algorithm is guaranteed to converge under certain conditions, e.g. an infinitesimal learning rate and an arbitrarily large number of iterations.

In lieu of achieving guaranteed convergence by running an infinite number of iterations, we would recommend that users calculate bootstrap confidence intervals of the parameters as a convergence test. The fitting procedure is very time consuming. With our computer approximately half an hour was required for a single run on the data set provided. For estimating standard errors, multiply this by the number of bootstrapped data sets to be fit. Thus we may expect a run time of approximately two days to estimate standard errors using 100 bootstrapped data sets. The run time can be reduced by running the program on a multicore processor, the number of parallel process to run is controlled by n.threads. The bootstrap estimating procedure is always performed “in parallel,” even when n.threads is set to 1. This is significant because R’s parallel computing interface does not allow parallel processes to perform input/output operations. Therefore printing of status updates is suppressed.