The Journal of Open Source Software

UnfoldSim.jl: A toolbox for simulating continuous event-based time series data for EEG and beyond

Judith Schepers [1], **Luis Lips** [1], **Maanik Marathe** [1], **and Benedikt V. Ehinger** [1,2]

1 Institute for Visualisation and Interactive Systems, University of Stuttgart, Germany **2** Stuttgart Center for Simulation Science, University of Stuttgart, Germany

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 19 February 2024
Published: unpublished

Summary

`UnfoldSim.jl` is a Julia package used to simulate multivariate time series, with a focus on EEG, especially event-related potentials (ERPs). The user provides four ingredients: 1) an experimental design, with both categorical and continuous variables, 2) event basis functions specified via linear or hierarchical models, 3) an inter-event onset distribution, and 4) a noise specification. `UnfoldSim.jl` then simulates continuous EEG signals with potentially overlapping events. Multi-channel support via EEG-forward models is available as well. `UnfoldSim.jl` is modular, providing intuitive entrance points for individual customizations. The user can implement custom designs, components, onset distributions or noise types to tailor the toolbox to their needs. This allows support even for other modalities, e.g. single-voxel fMRI or pupil dilation signals.

Statement of Need

In our work (e.g. Ehinger & Dimigen (2019), Dimigen & Ehinger (2021)), we often analyze data containing (temporally) overlapping events (e.g. stimulus onset and button press, or consecutive eye-fixations), non-linear effects, and complex experimental designs. For a multitude of reasons, we need to simulate such kind of data: Simulated EEG data is necessary to test preprocessing and analysis tools, validate statistical methods, illustrate conceptual issues, test toolbox functionalities, and find limitations of traditional analysis workflows. For instance, such simulation tools allow for testing the assumptions of new analysis algorithms and testing their robustness against any violation of these assumptions.

While other EEG simulation toolboxes exist, they each have limitations: they are dominantly MATLAB-based, they do not simulate continuous EEG, and they offer little support for designs more complex than two conditions or with non-linear effects.

Functionality

The toolbox provides four abstract types: `AbstractDesign`, `AbstractComponent`, `AbstractOnset` and `AbstractNoise`. In the following, we present the concrete types that are currently implemented. In addition, users can also implement their own concrete types fitting their individual needs.

Experimental designs

Currently, we support a single and a multi-subject design. They are used to generate an experimental design containing the conditions and levels of all predictors. The multi-subject design uses the `MixedModelsSim.jl` toolbox (Alday et al., 2024) and allows a flexible specification of the random-effects structure by indicating which predictors are within- or between-subject (or item). Tailored randomisation is possible via a user-specified function, which is applied after design generation. Designs can be encapsulated, for instance, the `RepeatDesign` type which repeats the generated event table multiple times, thus generating new trials. Currently, only balanced designs are implemented, i.e. all possible combinations of predictor levels have the same number of trials. However, a tutorial on how to implement a new design for imbalanced datasets is provided.

Event basis functions (Components)

`UnfoldSim.jl` provides a `LinearModelComponent` and a `MixedModelComponent` for single- and multi-subject simulation respectively. These components determine the shape of the response to an event. They consist of a basis function which is weighted by the user-defined regression model. The user specifies a basis function for the component by either providing a custom vector or choosing one of the prespecified bases. For example, the toolbox provides simplified versions of typical EEG components e.g. N170 which are implemented as temporally shifted Hanning windows. Further, in the components' model formulae, fixed-effects (βs) and random effects (`MultiSubjectDesigns` only) need to be specified.

Each component can be nested in a `MultichannelComponent`, which, using a forward head-model, projects the simulated source component to the multi-channel electrode space. Using `Artifacts.jl` we provide on-demand access to the HArtMuT (Harmening et al., 2022) model.

To generate complex activations, it is possible to specify a vector of `<:AbstractComponents`.

Inter-onset distributions

The inter-onset distribution defines the distance between events in the case of a continuous EEG. Currently, `UniformOnset` and `LogNormalOnset` are implemented. By specifying the parameters of the inter-onset distribution, one indirectly controls the amount of overlap between two or more event-related responses. Figure 1 illustrates the parameterization of the two implemented onset distributions.

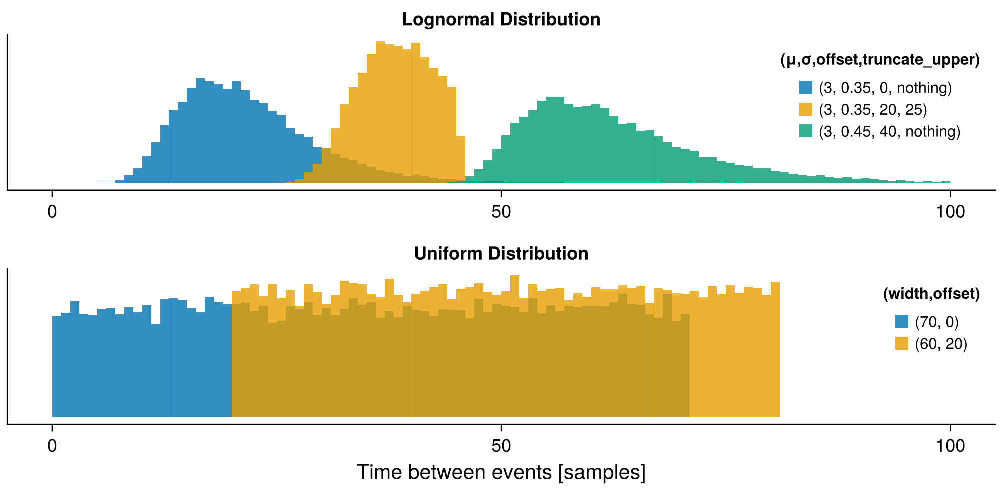

Figure 1: Illustration of the inter-onset distributions. The colour indicates different sets of parameter values.

Noise types

UnfoldSim.jl offers different noise types: `WhiteNoise`, `RedNoise`, `PinkNoise` and exponentially decaying autoregressive noise (`ExponentialNoise`) (see Figure 2). In the future, we will add simple autoregressive noise and noise based on actual EEG data.

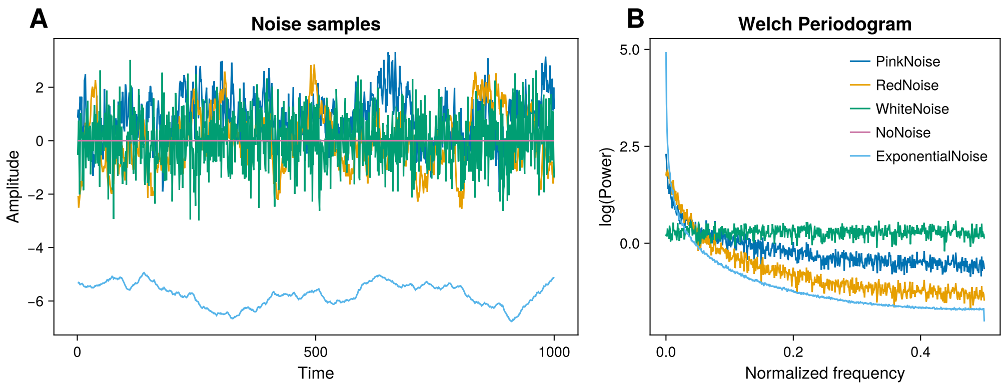

Figure 2: Illustration of the different noise types (indicated by colour). Panel **A** shows the noise over time. Panel **B** displays its $log_{10}(power)$ at normalized frequencies.

Simulation example

In this section, one can find an example of how to use `UnfoldSim.jl` to simulate continuous EEG data. Additional examples can be found in the `UnfoldSim.jl` documentation. Moreover, to get started, the `UnfoldSim.jl` toolbox offers the function `predef_eeg` which, depending on the input, simulates continuous EEG data either for a single subject or multiple subjects.

In the following, we will first provide examples for the four simulation "ingredients" mentioned above which will then be used to simulate data.

1. We specify an **experimental design** with one subject in two experimental conditions including a continuous variable with 10 levels. To mimic randomization in an experiment, we shuffle the trials using the `event_order_function` argument. To generate more trials we repeat the design 100 times which results in 2000 trials in total.

```
design =
    SingleSubjectDesign(;
        conditions = Dict(
            :condition => ["car", "face"],
            :continuous => range(0, 5, length = 10),
        ),
        event_order_function = x -> shuffle(deepcopy(StableRNG(1)), x),
    ) |> x -> RepeatDesign(x, 100);
```

Table 1 shows the first rows of the events data frame resulting from the experimental design that we specified.

Table 1: First five rows extracted from the events data frame representing the experimental design. Each row corresponds to one event. The columns *continuous* and *condition* display the levels of the predictor variables for the specific event and the *latency* column denotes the event onset (in samples).

continuous	condition	latency
2.22222	face	200

continuous	condition	latency
4.44444	car	400
3.88889	car	600
1.11111	car	800
0.555556	car	1000

2. Next, we create a signal consisting of two different **components**. For the first component, we use the prespecified N170 base and include a condition effect of the "face/car" condition i.e. faces will have a more negative signal than cars. For the second component, we use the prespecified P300 base and include a linear and a quadratic effect of the continuous variable: the larger the value of the continuous variable, the larger the simulated potential.

```
n1 = LinearModelComponent(;
    basis = n170(),
    formula = @formula(0 ~ 1 + condition),
    β = [5, 3],
);

p3 = LinearModelComponent(;
    basis = p300(),
    formula = @formula(0 ~ 1 + continuous + continuous^2),
    β = [5, 1, 0.2],
);

components = [n1, p3]
```

3. In the next step, we specify an **inter-onset distribution**, in this case, a uniform distribution with width = 0 and offset = 200 which means that the inter-event distance will be exactly 200 samples.

```
onset = UniformOnset(; width = 0, offset = 200)
```

4. As the last ingredient, we specify the **noise**, in this case, Pink noise.

```
noise = PinkNoise(; noiselevel = 2)
```

Finally, we combine all the ingredients and simulate data (see Figure 3). To make the simulation reproducible, one can specify a random generator.

```
eeg_data, events_df = simulate(StableRNG(1), design, components, onset, noise);
```

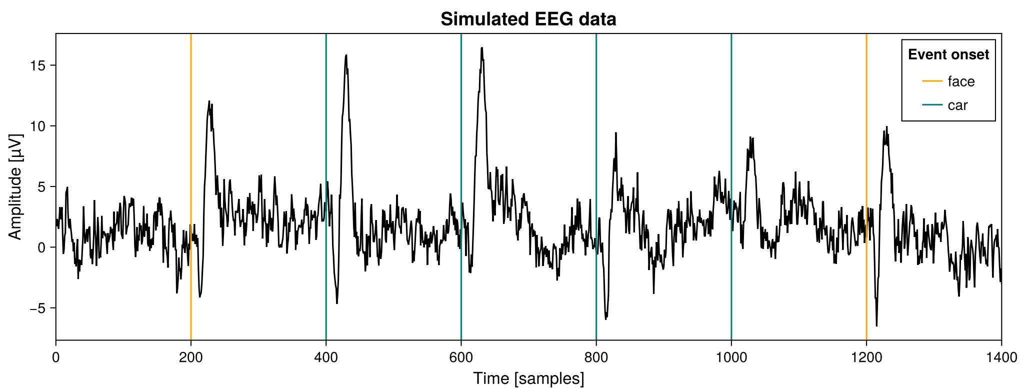

Figure 3: First 1400 samples from the simulated continuous EEG data. The vertical lines denote the event onsets and their colour represents the respective condition i.e. car or face.

To validate the simulation results, we use the `Unfold.jl` toolbox (Ehinger & Dimigen, 2019) to fit a regression model to the simulated data and examine the estimated regression parameters and marginal effects. For the formula, we include a categorical predictor for *condition* and a non-linear predictor (based on splines) for *continuous*.

```
m = fit(
    UnfoldModel,
    Dict( Any => (
            @formula(0 ~ 1 + condition + spl(continuous, 4)),
            firbasis(τ = [-0.1, 1], sfreq = 100, name = "basis"),
        )),
    events_df,
    eeg_data,
);
```

In subplot A of Figure 4, one can see the model estimates for the different coefficients and as intended there is a condition effect in the first negative component and an effect of the continuous variable on the second (positive) component. The relation between the levels of the continuous variable and the scaling of the second component is even clearer visible in subplot B of Figure 4 which depicts the estimated marginal effects of the predictors. Instead of showing the regression coefficients, we can evaluate the estimated function at specific values of the continuous variable.

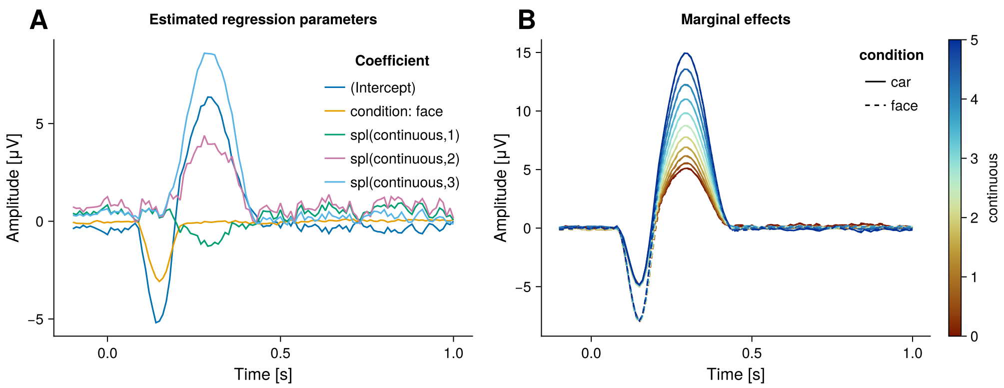

Figure 4: Regression results for the simulated data. Panel **A** displays the estimated regression coefficients over time. Panel **B** shows the estimated marginal effects i.e. the estimated event-related potential at different predictor levels.

As shown in this example, `UnfoldSim.jl` and `Unfold.jl` can be easily combined to investigate the effects of certain features, e.g. the type of noise or its intensity on the analysis result and thereby assess the robustness of the analysis.

Related tools

Not many toolboxes for simulating EEG data exist. Nearly all toolboxes we are aware of have been developed in proprietary MATLAB, and most have not received any updates in the last years or updates at all, and have very specific applications (e.g. `EEGg` (Vaziri et al., 2023), `SimMEEG` (Herdman, 2021), `SEED-G` (Anzolin et al., 2021), `EEGSourceSim` (Barzegaran et al., 2019), `simBCI` (Lindgren et al., 2018)).

In the following, we highlight two actively developed MATLAB-based tools: `Brainstorm` (Tadel et al., 2011) and `SEREEGA` (Krol et al., 2018). Both toolboxes are based in MATLAB and provide forward-simulation of EEG signals. `Brainstorm` especially excels at the visualization of the forward model, and provides interesting capabilities to generate ERPs based on phase-aligned oscillations. `SEREEGA` provides the most complete simulation capabilities with a greater focus on ERP-component simulation, tools for benchmarking like signal-to-noise specification, and more realistic noise simulation (e.g. via random sources).

In Python, `MNE-Python` (Gramfort et al., 2013) provides some tutorials to simulate EEG data, but the functionality is very basic. `HNN-Core` (Jas et al., 2023) can simulate realistic EEG data, but as it is based on neurocortical column models and dynamics, its usage is very detailed, realistic and involved.

In contrast to these tools, `UnfoldSim.jl` has a higher-level perspective, uniquely focusing on the regression-ERP aspect. `UnfoldSim.jl` provides functions to simulate multi-condition experiments, uniquely allows for modeling hierarchical, that is, multi-subject EEG datasets, and offers support to model continuous EEG data with overlapping events. Further, the implementation in Julia offers a platform that is free, that actively encourages research software engineering methods, that makes it easy to add custom expansions via the `AbstractTypes`, and finally, if one is not convinced about the elegancy and speed of Julia, it allows for easy and transparent access from Python and R.

Acknowledgements

Funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany´s Excellence Strategy – EXC 2075 – 390740016. The authors further thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Judith Schepers. Moreover, the authors would like to thank Tanja Bien for her valuable feedback on the paper manuscript.

Package references

Please note that we only mention the main dependencies of the toolbox here, but the dependencies of the dependencies can be found in the respective `Manifest.toml` files. Furthermore, please note that we only list rather than cite the packages for which we could not find any citation file or instruction.

Julia (Bezanson et al., 2017)
DataFrames.jl (Bouchet-Valat & Kamiński, 2023)
Distributions.jl (Besançon et al., 2021; Lin et al., 2019)
Documenter.jl (Hatherly et al., n.d.)
DSP.jl (Kornblith et al., 2023)
FileIO.jl
Glob.jl
HDF5.jl
HypothesisTests.jl
ImageFiltering.jl Literate.jl
LiveServer.jl
Makie.jl (Danisch & Krumbiegel, 2021)
MixedModels.jl (Bates et al., 2023)
MixedModelsSim.jl (Alday et al., 2024)
Parameters.jl
PrettyTables.jl (Chagas et al., 2023)
ProjectRoot.jl
SignalAnalysis.jl
Statistics.jl
StableRNGs.jl
StatsBase.jl

StatsModels.jl
TimerOutputs.jl
ToeplitzMatrices.jl
Unfold.jl (Ehinger & Dimigen, 2019)
UnfoldMakie.jl (Mikheev et al., 2023)

References

Alday, P., Bates, D., DeBruine, L., Ehinger, B., Calderón, J. B. S., Schepers, J., Choe, J., & Schwetlick, L. (2024). *RePsychLing/MixedModelsSim.jl: v0.2.9* (Version v0.2.9). Zenodo. https://doi.org/10.5281/zenodo.10669002

Anzolin, A., Toppi, J., Petti, M., Cincotti, F., & Astolfi, L. (2021). SEED-g: Simulated EEG data generator for testing connectivity algorithms. *Sensors*, *21*(11), 3632. https://doi.org/10.3390/s21113632

Barzegaran, E., Bosse, S., Kohler, P. J., & Norcia, A. M. (2019). EEGSourceSim: A framework for realistic simulation of EEG scalp data using MRI-based forward models and biologically plausible signals and noise. *Journal of Neuroscience Methods*, *328*, 108377. https://doi.org/10.1016/j.jneumeth.2019.108377

Bates, D., Alday, P., Kleinschmidt, D., Calderón, J. B. S., Zhan, L., Noack, A., Bouchet-Valat, M., Arslan, A., Kelman, T., Baldassari, A., Ehinger, B., Karrasch, D., Saba, E., Quinn, J., Hatherly, M., Piibeleht, M., Mogensen, P. K., Babayan, S., Holy, T., … Nazarathy, Y. (2023). *JuliaStats/MixedModels.jl: v4.22.3* (Version v4.22.3). Zenodo. https://doi.org/10.5281/zenodo.10268806

Besançon, M., Papamarkou, T., Anthoff, D., Arslan, A., Byrne, S., Lin, D., & Pearson, J. (2021). Distributions.jl: Definition and modeling of probability distributions in the JuliaStats ecosystem. *Journal of Statistical Software*, *98*(16), 1–30. https://doi.org/10.18637/jss.v098.i16

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Bouchet-Valat, M., & Kamiński, B. (2023). DataFrames.jl: Flexible and fast tabular data in julia. *Journal of Statistical Software*, *107*(4), 1–32. https://doi.org/10.18637/jss.v107.i04

Chagas, R. A. J., Baumgold, B., Hertz, G., Ranocha, H., Wells, M., Ritchie, N., Christensen, Z. P., gustaphe, pdeffebach, Pasquier, B., Kawczynski, C., Sambrani, D., Vanguelov, D., Hanson, E., Butterworth, I., Peters, J., Kraak, J., Storopoli, J., TagBot, J., … lorenzoh. (2023). *Ronisbr/PrettyTables.jl: v2.3.1* (Version v2.3.1). Zenodo. https://doi.org/10.5281/zenodo.10214175

Danisch, S., & Krumbiegel, J. (2021). Makie.jl: Flexible high-performance data visualization for Julia. *Journal of Open Source Software*, *6*(65), 3349. https://doi.org/10.21105/joss.03349

Dimigen, O., & Ehinger, B. V. (2021). Regression-based analysis of combined EEG and eye-tracking data: Theory and applications. *Journal of Vision*, *21*(1), 3–3. https://doi.org/10.1167/jov.21.1.3

Ehinger, B. V., & Dimigen, O. (2019). Unfold: An integrated toolbox for overlap correction, non-linear modeling, and regression-based EEG analysis. *PeerJ*, *7*, e7838. https://doi.org/10.7717/peerj.7838

Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., Goj, R., Jas, M., Brooks, T., Parkkonen, L., & Hämäläinen, M. S. (2013). MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*, *7*(267), 1–13. https://doi.org/10.3389/fnins.2013.00267

Harmening, N., Klug, M., Gramann, K., & Miklody, D. (2022). HArtMuT—modeling eye and muscle contributors in neuroelectric imaging. *Journal of Neural Engineering*, *19*(6), 066041. https://doi.org/10.1088/1741-2552/aca8ce

Hatherly, M., Piibeleht, M., Ekre, F., & Contributors. (n.d.). *Documenter: A documentation generator for julia*. https://github.com/JuliaDocs/Documenter.jl

Herdman, A. T. (2021). SimMEEG software for simulating event-related MEG and EEG data with underlying functional connectivity. *Journal of Neuroscience Methods*, *350*, 109017. https://doi.org/10.1016/j.jneumeth.2020.109017

Jas, M., Thorpe, R., Tolley, N., Bailey, C., Brandt, S., Caldwell, B., Cheng, H., Daniels, D., Pujol, C. F., Khalil, M., Kanekar, S., Kohl, C., Kolozsvári, O., Lankinen, K., Loi, K., Neymotin, S., Partani, R., Pelah, M., Rockhill, A., … Jones, S. (2023). HNN-core: A python software for cellular and circuit-level interpretation of human MEG/EEG. *Journal of Open Source Software*, *8*(92), 5848. https://doi.org/10.21105/joss.05848

Kornblith, S., Lynch, G., Holters, M., Santos, J. F., Russell, S., Kickliter, J., Bezanson, J., Adalsteinsson, G., Arslan, A., Yamamoto, R., jordancluts, Pastell, M., Kelman, T., Arthur, B., Krauss, T., HDictus, El-Saawy, H., Kofron, J., Hanson, E., … Smith, J. (2023). *JuliaDSP/DSP.jl: v0.7.9* (Version v0.7.9). Zenodo. https://doi.org/10.5281/zenodo.8344531

Krol, L. R., Pawlitzki, J., Lotte, F., Gramann, K., & Zander, T. O. (2018). SEREEGA: Simulating event-related EEG activity. *Journal of Neuroscience Methods*, *309*, 13–24. https://doi.org/10.1016/j.jneumeth.2020.109017

Lin, D., White, J. M., Byrne, S., Bates, D., Noack, A., Pearson, J., Arslan, A., Squire, K., Anthoff, D., Papamarkou, T., Besançon, M., Drugowitsch, J., Schauer, M., & contributors, other. (2019). *JuliaStats/Distributions.jl: a Julia package for probability distributions and associated functions*. https://doi.org/10.5281/zenodo.2647458

Lindgren, J. T., Merlini, A., Lecuyer, A., & Andriulli, F. P. (2018). simBCI—a framework for studying BCI methods by simulated EEG. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *26*(11), 2096–2105. https://doi.org/10.1109/TNSRE.2018.2873061

Mikheev, V., Döring, S., Gärtner, N., Baumgartner, D., & Ehinger, B. (2023). *UnfoldMakie* (Version v0.4.0). Zenodo. https://doi.org/10.5281/zenodo.10235220

Tadel, F., Baillet, S., Mosher, J. C., Pantazis, D., & Leahy, R. M. (2011). Brainstorm: A user-friendly application for MEG/EEG analysis. *Computational Intelligence and Neuroscience*, *2011*, 1–13. https://doi.org/10.1155/2011/879716

Vaziri, A. Y., Makkiabadi, B., & Samadzadehaghdam, N. (2023). EEGg generating synthetic EEG signals in matlab environment. *Frontiers in Biomedical Technologies*, *10*(3), 370–381. https://doi.org/10.18502/fbt.v10i3.13165