Supplementary Text

**Boolink, a Graphical Interface for Open Access Boolean Network Simulations and Use in Guard Cell CO2 Signaling**

Aravind Karanam1,#, David He1,#, Po-Kai Hsu2, Sebastian Schulze2, Guillaume Dubeaux2, Richa Karmakar 1, Julian I. Schroeder2, Wouter-Jan Rappel1\*

1Physics Department, University of California, San Diego, La Jolla, CA 92093, USA.

2Division of Biological Sciences, Section of Cell and Developmental Biology, University of California, San Diego, La Jolla, CA 92093-0116, USA.

The following text contains instructions to download and run Boolink on a personal computer like an office laptop. Boolink is free to run and distribute, and uses other open-source software packages like Python and C++. In addition, it describes the Boolean equations introduced in Stomata 2.1. Boolink can be freely downloaded from the GitHub repository (https://github.com/dyhe-2000/Boolink-GUI or https://github.com/Rappel-lab/Boolink-GUI). This text contains detailed installation instructions for machines with Windows operating systems while the GitHub repository contains instructions for the Linux (Ubuntu) and Mac operating systems.

The structure of this document is indicated in the list below. **Section 1** contains a primer on Boolean algebra. **Section 2** contains an extended discussion of the changes made to Stomata 2.0 to arrive at Stomata 2.1. **Section 3 and 4** relate to the Windows operating system. Specifically, **section 3** describes the installation instructions for C++ and Python required for a direct installation of Boolink, **section 4** contains instructions to download Boolink from the code-sharing website GitHub. **Section 5** describes how to run a few example Boolean networks using Boolink. **Section 6** describes the process of designing your own Boolean network. It is advisable to try out the “simple network” (details in the main text and in Section 5) before running the ABA network or the updated versions Stomata 2.0 and Stomata 2.1. Creating a new Boolean network requires an elementary knowledge of Boolean algebra besides empirical data on the system one wishes to model. Finally, in **section 7**, we list some common errors encountered during installation and running of Boolink and ways to get around them.

The installation instructions are dependent on the operating system of the computer, so make sure that you are referring to the right document. Refer to the GitHub repository ([https://github.com/dyhe-2000/Boolink-GUI.git](https://github.com/dyhe-2000/BoolSim-GUI.git) or [https://github.com/Rappel-lab/Boolink-GUI](https://github.com/Rappel-lab/BoolSim-GUI)) for instruction manuals for all the operating systems at one place.

# 1. A primer on Boolean Logic

To specify relationships between variables using Boolean logic, Boolean variables, and functions are used. Boolean variables, like the states of a switch in an electrical circuit, can only take one of two values: OFF (0) and ON (1). In a reaction pathway, the ON state of a variable denotes high activity or concentration while the OFF state denotes low activity or concentration. The relationships between nodes can be described by three basic functions:

* **NOT,** or negation gate: Denoted by NOT or ~, This operation takes one Boolean variable as input (a unary operator) and returns the *negated*, or complementary state as output, i.e., NOT(1) = 0 and NOT(0) = 1. This operation can also be written compactly as ~, i.e., NOT(1)=~1=0.
* **AND,** or conjunction gate: This operation takes two Boolean variables as input (a binary operator) and returns 1 only if both the states are equal to 1. This operation is also called product because the outcome is identical to multiplying the Boolean variables. The compact way of writing this gate is using the symbol &. The truth table, which lists all possible inputs and their respective outputs, for an AND gate is as follows:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A AND B** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* **OR,** or disjunction gate: This operation takes two Boolean variables as input (a binary operator) and returns 1 if at least one of the variables equals 1. This operation is also called sum because the outcome is identical to adding the boolean variables. The short way of writing this gate is using the symbol |. The truth table, which lists all possible inputs and their respective outputs, for the OR gate is as follows:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A OR B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

An arbitrarily complex Boolean function can be composed using the above three operations. A widely used and intuitive way to express boolean equations is the so-called Sum of Products (SoP) form, which is a sum (i.e., series of terms linked by OR logic) of products (i.e., series of terms linked by AND logic and perhaps NOT logic). Consider an example from the ABA-induced stomatal closure Boolean network:

ABI2 = (NOT(ROS) AND(ROP11)) or (NOT(ROS) & NOT(RCARs))

which can be compactly written as

ABI2 = ~ROS & ROP11 | ~ROS & ~RCARs

This equation is made of two product terms linked by an OR operator. The simplicity of the SoP form lies in the fact that the equation evaluates to ON (1) if and only if at least one of the product terms evaluates to ON (1). The above equation may also be reported as

ABI2 = ~[ROS | ~ROP11 & RCARs]

Such a form can be expanded into the SoP form without parentheses using the following identities:

* De Morgan’s laws

~(A & B) = ~A | ~B

~(A | B) = ~A & ~B

* Distributive law (product over sum)

A & (B | C) = A&B | A&C

## 1.1. Deriving Boolean Equations from Empirical Relationships

Boolean equations may be derived from empirical relationships established from experiments, exhaustively measuring the outcomes of all possibilities in a truth table. In the following, we illustrate how to derive boolean equations in the SoP form from a truth table. Consider a node X in a Boolean network, and three nodes A, B, and C upstream of it. Let us say X follows the update rule: X is ON if at least two of its upstream nodes are ON. If we list all possibilities of A, B, and C, and the outcome of X in each case, the truth table looks as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **X** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**Table 1**: Truth table for X and its three upstream nodes A, B, and C. X is ON if at least three of its upstream nodes are ON.

To derive the Boolean update equation for X in SoP form, we gather all the products (connected by ANDs) and add them together (connected by ORs):

X = ~A&B&C + A&~B&C +A&B&~C + A&B&C

This is the correct Boolean update equation since it considers each case where the outcome is ON separately and exactly once. However, by allowing some redundancy and using basic Boolean logic, the equations in some cases can be simplified and made to look more intuitive. For our example, we can repeat an existing term in the sum without changing the outcome. We may thus write X=(~A+A)&B&C + A&(~B+B)&C +A&B&(~C+C), where we have added the term A&B&C twice. Next, we use the ~A+A=1, ~B+B=1, and ~C+C=1 to obtain X=B&C + A&C +A&B. Finally, replacing each sum with OR, we get

X = A&B | B&C | C&A

## 1.2. Analysis of a simple boolean network

Section 2 of the main text described a simple Boolean network with three intermediate nodes and input and output nodes. The network in the main text was compactly written in terms of the symbols &, |, and ~. In terms of the logical operations AND, OR, and NOT, the equations take on the form

IN = IN

X = Y and (not IN)

Y = X

Z = not X

OUT = (not Y) and Z

This illustrative network has the interesting property that while using the asynchronous update scheme and when the input (IN) is absent, the output (OUT) is active 50% of the time. When the input is present, the output is always active. In the former case, the system has an equal chance of reaching one of the two attractors of the system, only one of which leads to an active output node. In the latter case, the system always reaches the (only) attractor which leads to an active output node.

The evolution of the states to the attractor state for this network can be described as follows. In the case without input, (left panel, figure 4 in the main text) there are two attractor states (0,0,1) and (1,1,0); there is no escape from these states. States (0,0,0) and (0,0,1) directly lead to (0,0,1) while (1,1,0) and (1,1,1) directly lead to (1,1,0). Each of the remaining four states on the lattice can reach either attractor with 50% probability. We illustrate this with an example calculation: consider the probability of the state (1,0,1) reaching (0,0,1). There is a direct connection to (0,0,1) which it can take with a probability ⅓. The other path is through (1,0,0) and (0,0,0). (1,0,0) has two exits, along x and y directions. Choosing the x-direction leads to (0,0,0) and hence to (0,0,1). Therefore the total probability to reach (0,0,1) from (1,0,1) is ⅓ + ⅓\*½ = ½. Having calculated the probability of reaching each attractor starting from each node, it is easy to see that the probability of reaching (0,0,1) starting from a random node is ½. Note that in an asynchronous update scheme, a node can be updated a second time only after all the nodes are updated once, albeit in any order.

In the case with the input, (right panel, figure 4 in the main text) there is only one attractor (0,0,1), and every state reaches (0,0,1) in a few time steps.

Finally, we contrast the trajectories of states obtained above with those by using synchronous update scheme. In synchronous update scheme, all the nodes are updated together based on the previous state of the system. For example, consider the synchronous update of (1,0,1) in the absence of input (IN = 0) which was examined above for asynchronous update. In one step (1,0,1) evolves to (0,1,0), and then the system oscillates between the two states. Evidently, the system trajectory in synchronous update is not ‘continuous’ in the state space.

For the network at hand, the network has only one attractor (0,0,1) in the presence of input and it reaches it in the utmost two updates. In the absence of output, two of the eight states settle to (0,0,1) in one step, two settles to (1,1,0) in one step, and the remaining four reach the (1,0,1) ⇿ (0,1,0) oscillation in the utmost two steps.

# 2. Stomata 2.1

As detailed in the main text, our initial attempt Stomata 2.0 to include CO2 signaling into the ABA network involved a branch upstream of GHR1. However, the experimental results are inconsistent with the results of Stomata 2.0, which motivated us to seek ways to improve the network. Our experimental results showed that the steady-state conductance level in the absence of ABA is higher for low CO2 than for high CO2. In the Boolean simulations, we vary the input nodes ABA and CO2 between 0 and 1, corresponding to low and high values, respectively. The output is the node Closure, which can be related to stomatal conductance through Conductance=1-Closure. Thus, we want the model to reflect the following experimental observations:

1. Leaves equilibrated at low CO2 concentrations have a higher conductance than those equilibrated at high CO2 concentrations *prior to* the application of ABA. This should correspond, in the model, to higher conductance for (CO2=0, ABA=0) than for (CO2=1, ABA=0). Our experimental data suggest that the stomatal conductance for low CO2 concentrations is approximately twice as large as for high CO2 concentrations.
2. The steady state conductance of the leaves *after* the application of ABA is greater for the lower CO2 concentrations than for higher CO2 concentrations. This should correspond, in the model, to higher conductance for (CO2=0, ABA=1) than for (CO2=1, ABA=1). Our experimental data suggest that the stomatal conductance for low CO2 is roughly identical to the stomatal conductance for high CO2 before the application of ABA.

To translate these experimental findings into a quantitative Boolean state of the output node, we require the model to reproduce the following observations:

ABA=0, CO2=0: Closure=0 (Conductance=1)

ABA=0, CO=1: Closure=0.5 (Conductance=0.5)

ABA=1, CO2=0: Closure=0.5 (Conductance=0.5)

ABA=1, CO2=1: Closure=1 (Conductance=0)

The Closure node in the ABA network model is affected by two nodes, Microtubule and H2OEfflux, through the equation Closure = H2OEfflux AND Microtubule. To achieve an intermediate level of closure, required for the conditions ABA=0, CO2=1 and ABA=1, CO2=0, we need one of the two nodes at 100% activity and the other at 50% activity (fluctuating between 0 and 1). For full closure, we need both nodes at 100% while for full conductance we need both nodes at 0. In Stomata 2.1, this is achieved through the following modifications:

1. Ca2c = ~Ca2ATPase & (CIS | CaIM) | (ABA&CO2)
2. CaIM = ~ABH1 & (NtSyp121 | MRP5) | ~ERA1 | Actin | CO2
3. Microtubule = TCTP | Microtubule & ABA
4. H2OEfflux = AnionEM & PIP21 & KEfflux & ~Malate | CO2

where the modifications are shown in blue. As a reminder, the symbol & represents AND logic and the symbol | represents OR logic.

**Motivation for these modifications**

1. In the original Albert version of the model, Ca2c (cytosolic calcium) has 50% activity due to oscillations between Ca2c and Ca2ATPase, if and only if ABA=1. Since the original network has no CO2 input, this is independent of the state of CO2. We added input from CO2 so that Ca2c has 100% activity if both ABA=1 and CO2=1. Adding CO2 to calcium was motivated by evidence that cytosolic calcium is involved in CO2-induced closure.
2. CO2 is added to CaIM through an OR gate to ensure that Microtubule=0.5 when CO2=1, even in the absence of ABA.
3. In the original ABA network model, Microtubule was always either 0 (if ABA=0) or 1 (if ABA=1), even though Ca2c=0.5. This is because of the feedback loop from Microtubule onto itself. To achieve Microtubule=0.5, we make the feedback loop dependent on ABA. As a result, Microtubule=0.5 if ABA=0 and CO2=1 but Microtubule=1 when both ABA and CO2 are 1.
4. In the original ABA network model, H2OEfflux is 0 if ABA is absent, independent of the state of CO2. With this modification, H2OEfflux=1 when ABA=0 and CO2=1.

As a result of these modifications, the network is able to reproduce the experimental results (See Fig. 5 in the main text). It is straightforward to simulate and visualize the network using BoolSim, which can be used to analyze the response:

When ABA=1 and CO2=1, H2OEfflux = 1 as all the terms are 1. Furthermore, Ca2c oscillations are superseded by a sustained activation because of the ABA&CO2 term in modification #1. Hence Microtubule is maintained at 1 as well and Closure=1.

When ABA=1 and CO2=0, H2OEfflux is at 50% since AnionEM is at 50%, and PIP21, KEfflux, and ~Malate are at 100% activation. Ca2c is at 50% (due to oscillations) in the absence of CO2 but the positive feedback of Microtubule is activated in the presence of ABA, making it 100% active. Thus, Microtubule=1, H2OEfflux=0.5, and Closure=0.5.

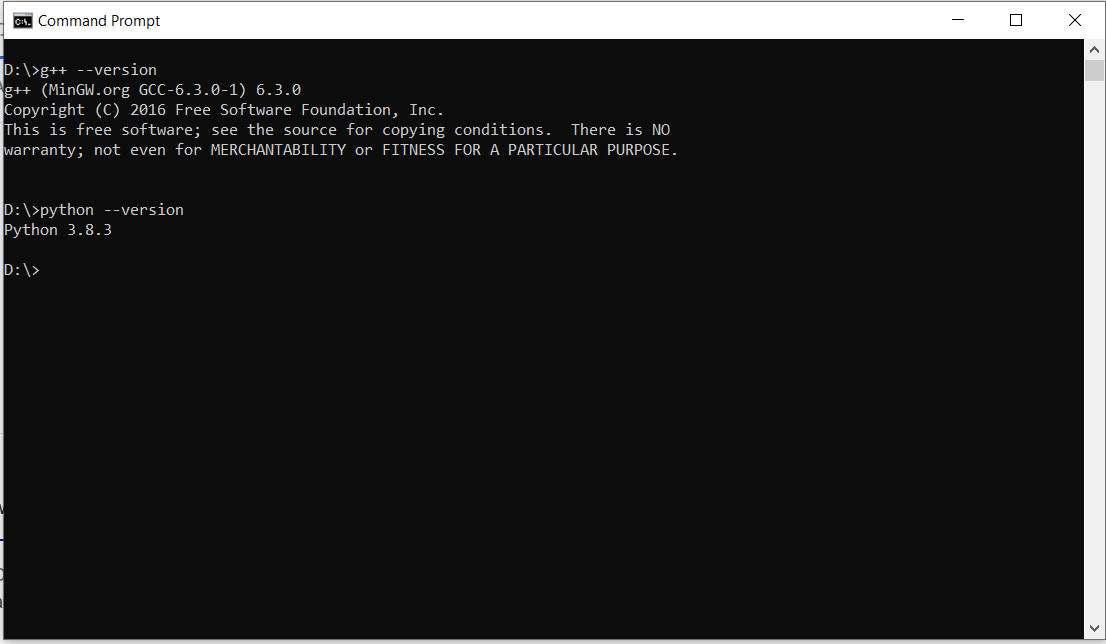
When ABA=0 and CO2=1, H2OEfflux=1 due to the CO2 branch implemented in Stomata 2.0 and also present in Stomata 2.1. Similarly, the addition of the CO2 branch to CaIM causes Ca2c oscillations even in the absence of ABA. Thus, Ca2c is at 50% activity and Microtubule=0.5, as the positive feedback is shut down in the absence of ABA. Thus, Microtubule=0.5, H2OEfflux=1, and Closure=0.5.

When ABA=0 and CO2=0, both H2OEfflux and Microtubule are shut down, and there will be no activity of closure. In other words, Microtubule=0, H2OEfflux=0, and Closure=0.

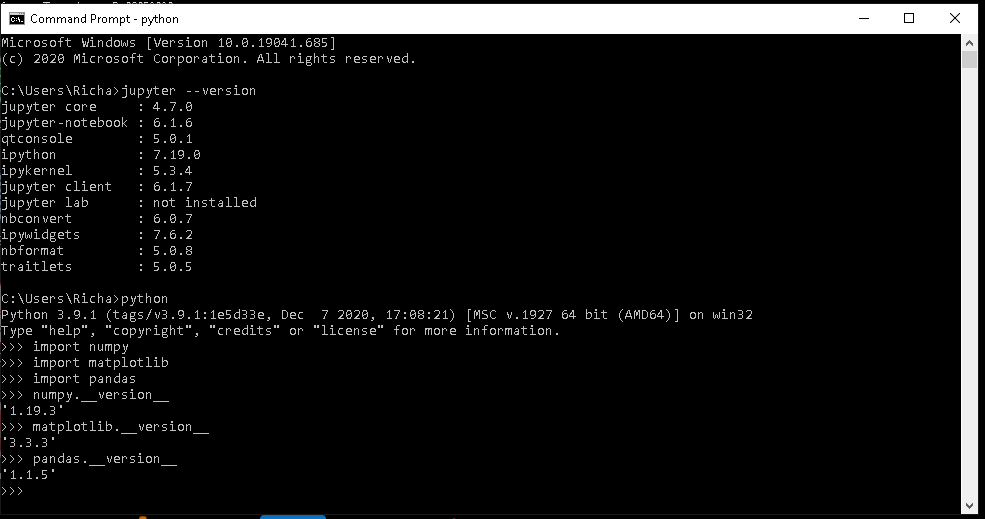
# 3. Installing BoolSim - Software Requirements (Windows)

BoolSim requires that the GNU C++ compiler (called g++) and Python be installed on the user’s computer. Both the tools are freely available. The following steps run through the installation of the two programs.

1. Firstly, check if g++ is already installed on your computer. Open the command prompt by right-clicking on the start button and click on “command prompt”; alternatively, search for “command prompt” in the search bar in the start menu and open the application from the search results. Type ‘g++ --version’ without the quotes and hit Enter (see screenshot 1).
   1. BoolSim requires the g++ version to be 6 or higher. If g++ is properly installed, the command prompt displays the version of g++ followed by a message. (See screenshot 2). Otherwise, it shows an error (see Section 6 for troubleshooting).
   2. Once g++ is installed, we come back to this step to verify that the installation is proper and complete. If g++ is installed, please go to step #5.
2. To install g++ we first download and install a program called MinGW. This program installs the necessary compilers for us in a matter of a few clicks.
3. This video runs through g++ installation using MinGW ( <https://youtu.be/sXW2VLrQ3Bs> ). Follow the steps as shown in the video. Here are some tips you might find useful while working through the video:
   1. URL to download MinGW: <https://sourceforge.net/projects/mingw/>
   2. The installation of MinGW requires changing environment variables. The video explains one way of accomplishing this. A different way to change environment variables on your system is to open System Settings (which you can open by searching for it in the search bar in the start menu) and type ‘environment variables’ in the search bar of the system settings window.
4. After the g++ installation is complete, go to step 1. If successful, your computer can now compile and run c++ programs.
5. Next, we install Python. To check if Python is already installed, type ‘python --version’ in the command prompt and hit Enter.
   1. As before, if Python is properly installed, the command prompt shows the version number and a message. If it is not installed, it displays an error (see Section 6 for troubleshooting).
   2. BoolSim requires python3. Once Python is installed, we come back to this step to verify that the installation is proper and complete. If installed, go to step #7.
   3. If python3 is installed on your computer, the above command shows a version 2.x. In that case, continue with the next step and install python3. You might also need to substitute python3 for python while running commands that use python, such as “python3 --version”, “python3 BoolSim\_Windows.py”.
6. To install Python on your Windows machine, follow the steps in this video: <https://youtu.be/4Rx_JRkwAjY> You may stop at 4:00 in this video once Python is installed. Close the installer and go to step 5. You should be able to see python’s version (3.8 or higher) when you run “python --version” on cmd (as shown in the screenshot 1).
7. Next, we install a few required Python packages: numpy, matplotlib, pandas, and jupyter. Run these commands in the command prompt after Python is installed:
   1. pip install numpy==1.19.3
   2. pip install matplotlib
   3. pip install pandas
   4. pip install jupyter
8. After the above packages are installed, you may check their version by trying the commands given in screenshot 2 below.
9. Good to go!

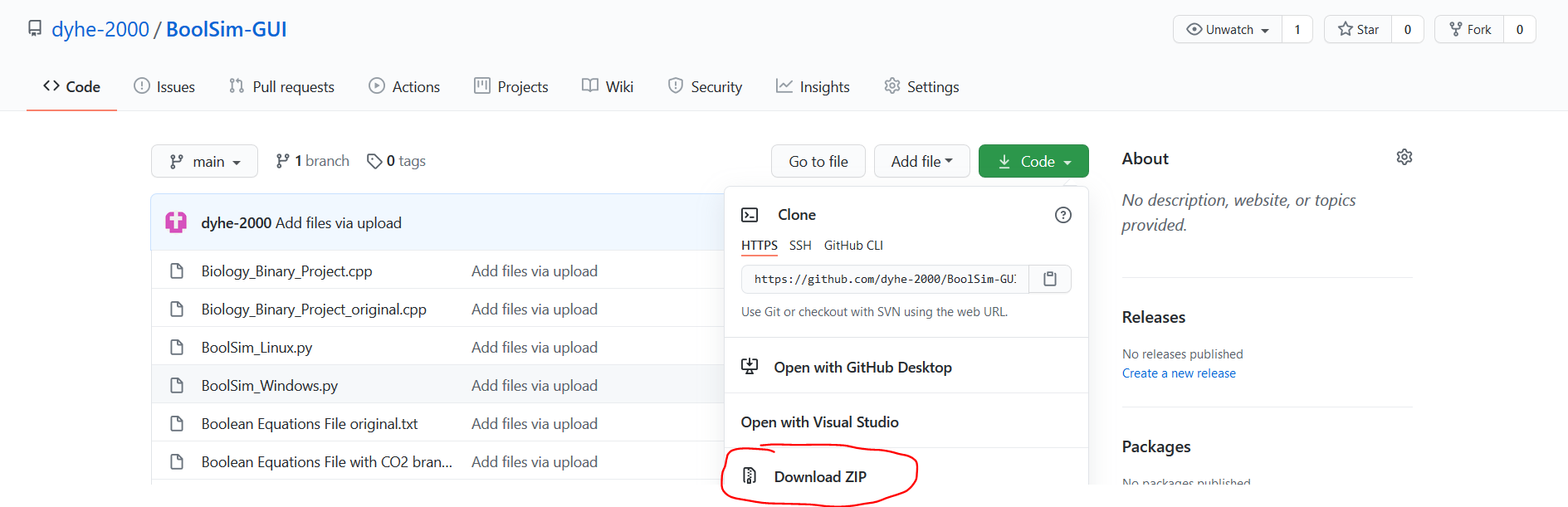


Screenshot 1. The version you see on your command prompt might be different.



Screenshot 2. The version you see on your command prompt might be different.

# 4. Downloading BoolSim (Windows)

1. Create a new folder on your computer where you want to run BoolSim. For example, D:\BoolSim\_simulation\
2. Visit <https://github.com/dyhe-2000/BoolSim-GUI.git> or <https://github.com/Rappel-lab/BoolSim-GUI> for downloading the package. Click on the green button named Code and then click on Download ZIP in the dropdown. See Screenshot 3. 

Screenshot 3: Download page of BoolSim from GitHub

1. Save the zip file in the new folder you’ve just created. If the zip file is saved to the Downloads folder by default, cut and paste the zip file into the new folder.
2. Then open the new folder you created in File Explorer. Right-click on the zip file and extract it at the current location. A new folder by the name “BoolSim-GUI-main” is created. Open that folder. It contains all the files needed to run BoolSim.
3. To run BoolSim, we open the command prompt at the current location. To do that, select the address bar in the File Explorer by pressing Alt+D or by right-clicking on the address bar. Once the address is selected (in our example D:\BoolSim\_simulation\BoolSim-GUI-main\), replace the address by the command ‘cmd’ in the address bar and hit Enter. A command prompt pops open.
4. In the command prompt, type ‘python BoolSim\_Windows.py’ (“python3 BoolSim\_Windows.py” to avoid a conflict if python2 is already installed) and hit Enter. A new window will pop up.
5. Click on ‘Agree’ to enter the start page of BoolSim. This is the **Home Page** of BoolSim. You are now ready to use BoolSim! (See Screenshot 4 and the locations of Menu Bar and Home Page options. We will be referring to these often in the following steps).

Screenshot 4. Home page of BoolSim

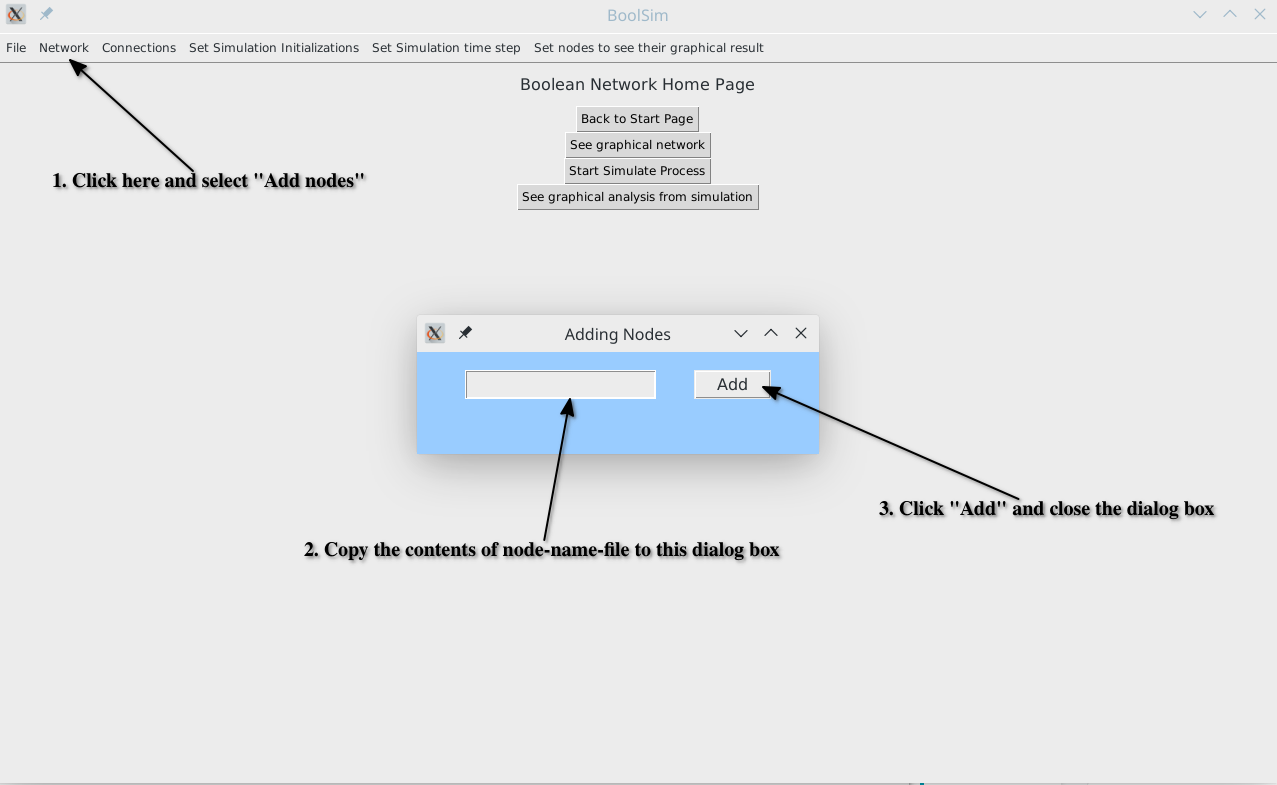
# 5. Running BoolSim

To simulate a Boolean network using BoolSim, you need information on the network. In BoolSim, that information is organized into two files - the first containing the names and initial states of the nodes (if the latter are pre-determined) which we call **node-name-file**, and the second containing the Boolean update equations for each of the nodes defined in node-name-file, which we call **equation-file**. Sections 6.1 and 6.2 describe how to create these files from empirical data. BoolSim comes the files corresponding to three example networks: a “simple network”, ABA-induced stomatal closure network, and ABA-CO2-induced stomatal closure network (named Stomata 2.0 and Stomata 2.1 for two versions). Refer to section 6 to locate these files in the current folder.

It is recommended that you try out the simple network first before simulating more complicated networks. The following steps assume that you have located the node-name-file and equation-file for the network you wish to simulate.

To run BoolSim using the files, please proceed with the following steps.

1. **To load the nodes of the network into BoolSim**,
   1. Click Network ➝ Add Node (in the menu bar).
   2. Copy-paste the contents of the node-name-file into the text box in the pop-up menu titled ‘Adding Nodes’, and click on the ‘Add’ button.
   3. Close the pop-up menu. See screenshot 5.

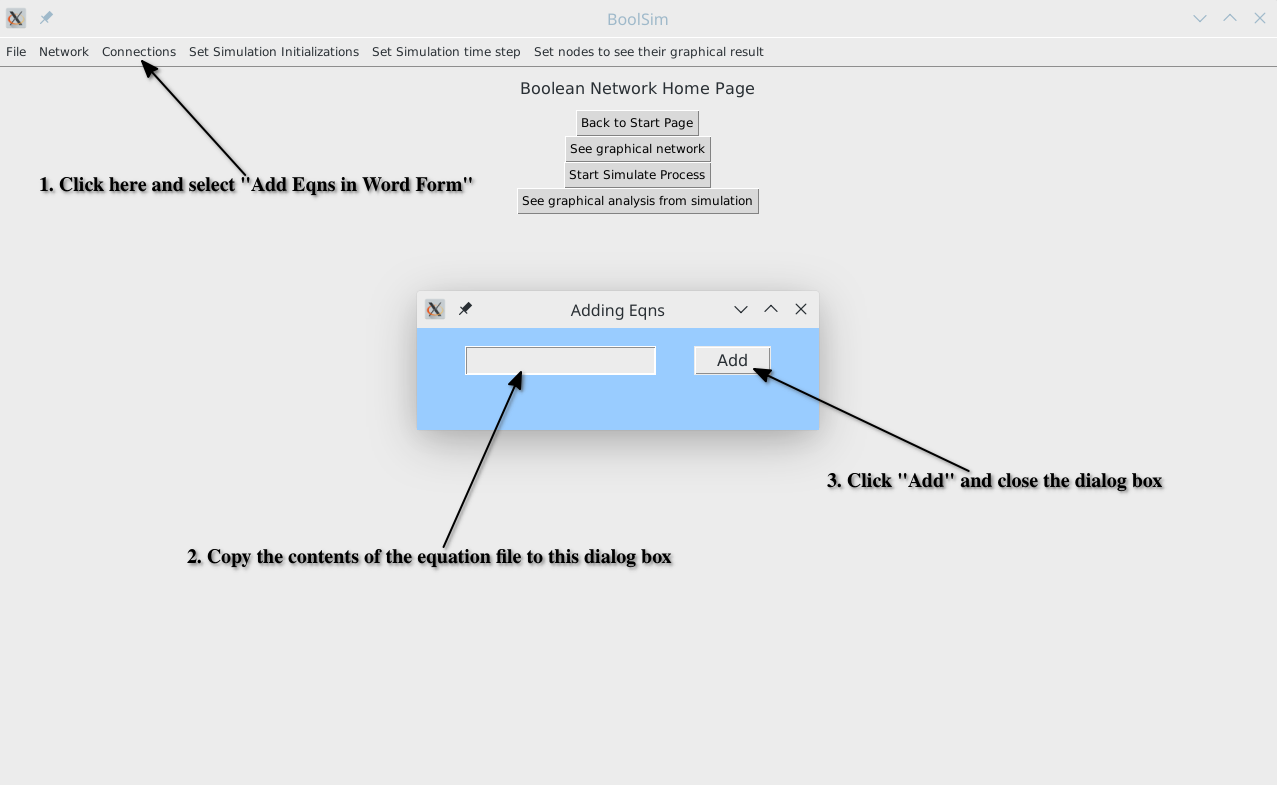


Screenshot 5: Loading node names and their initial states into BoolSim.

A specific example of a node-name-file can be found in the folder Booksim\BoolSim-GUI-main\ sample\_data\_files \ simple\_network\_data\_files. The node-name-file is called simple\_net\_node\_names.txt and represents the simple network (see also Section 4).

Next, we load the Boolean update equations into BoolSim. BoolSim admits equations written in two formats, word format and index format. The former is the natural way of writing the equations, i.e., with the names of the nodes and as shown in the main text. The latter is the internal representation of the network in BoolSim. Since it is easy to get confused while working with the index format, we do not recommend using it. Nevertheless, we provided the data files in that format.

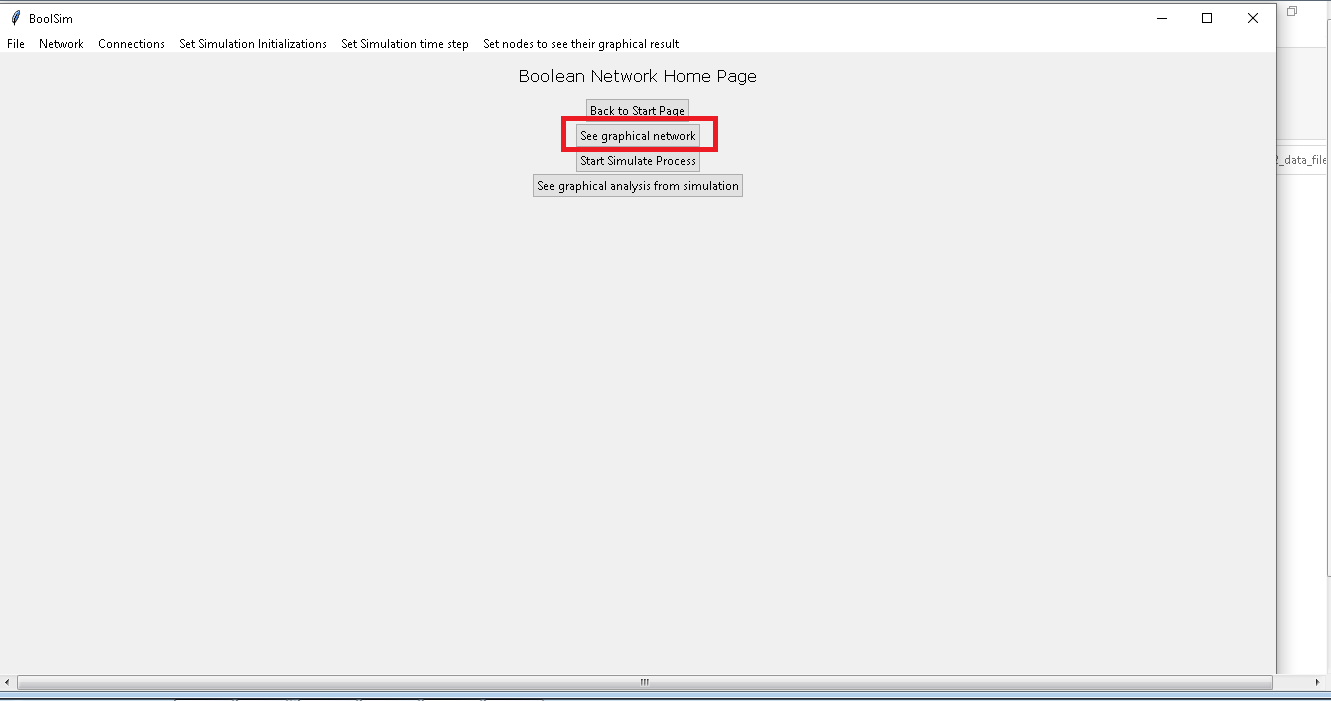
1. **To load the Boolean equations of the network into BoolSim**, you may use the equations in the word format or the index format. We do not recommend using the Index format.
   1. To add equations in the word format, click on Connections ➝ Add Eqns in Word Form. Copy-paste the contents of the equations-file (in word form) into the text box in the pop-up menu titled ‘Adding Eqns’, and click on the ‘Add’ button. Then close the pop-up menu. See screenshot 6.
   2. To add equations in the index format, click on Connections ➝Add Eqns in Index Form. Copy-paste the contents of the equations-file (in index form) into the text box in the pop-up menu titled ‘Adding Eqns’, and click on the ‘Add’ button. Then close the pop-up menu.



Screenshot 6: Loading equations in word format into BoolSim.

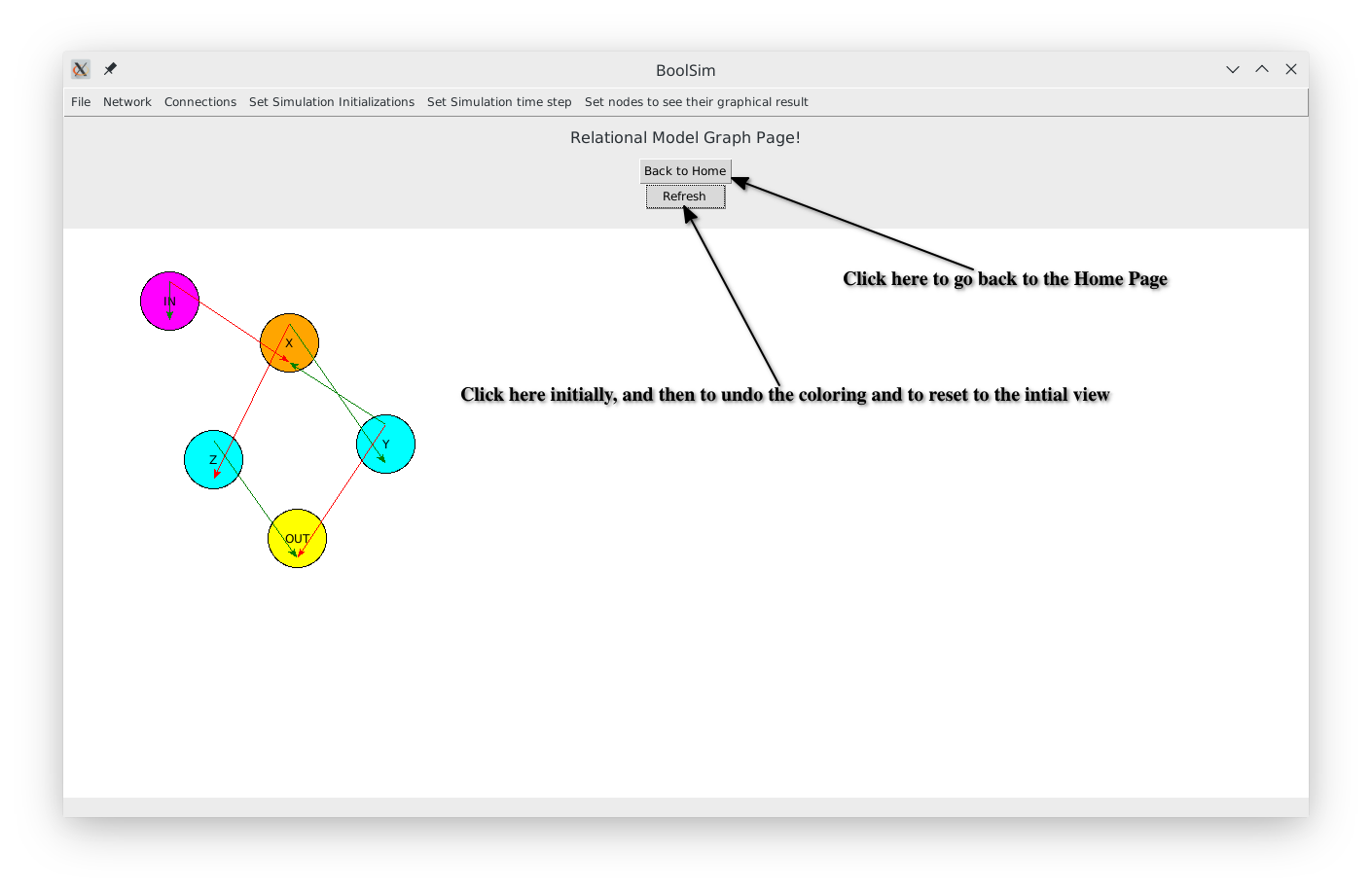
A specific example of a node-name-file can be found in the folder Booksim\BoolSim-GUI-main\ sample\_data\_files \ simple\_network\_data\_files. The node-name-file is called simple\_net\_word\_eqns.txt and represents the simple network (see also Section 4).

1. To visualize the network, click on the **See Graphical Network** button on the homepage. See screenshot 7. The circles representing the nodes can be moved around by dragging for ease of visualization. Green arrows denote activation while red arrows denote inhibition. Self-regulation, which can be either positive or negative, is shown by an arrow that is entirely within the node. An arrow begins from the top of the source node to the bottom of the target node. The arrowhead of each arrow is at the target node. Click “Refresh” to reset to the initial view, and click “Back to Home” to return to the homepage. See screenshot 8.



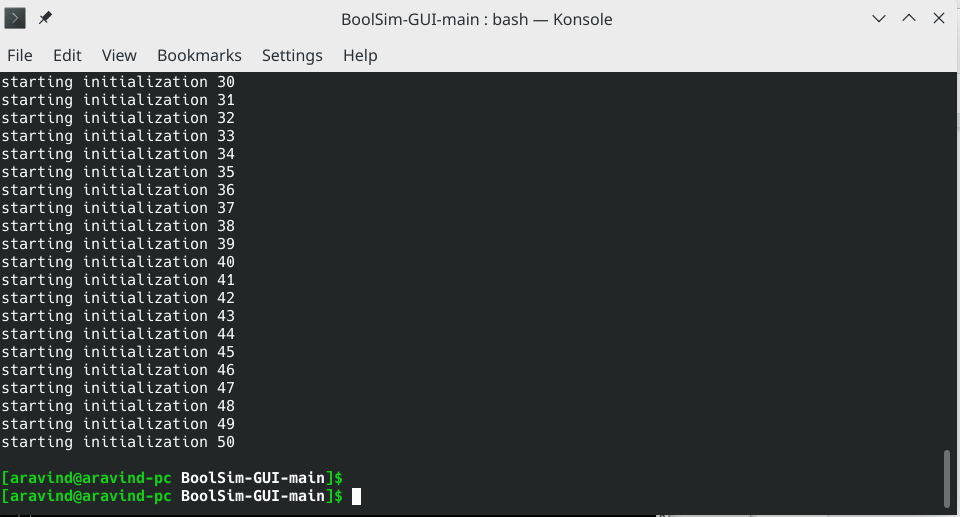
Screenshot 7: Click on “See graphical network” to visualize the Boolean network

1. **To** **search for upstream and downstream nodes** of the selected node, double-click on the node. The node itself is colored in orange, its upstream nodes are colored in magenta, and the downstream nodes in cyan. See screenshot 8. Right-clicking on or dragging any of the highlighted nodes reverts it to the default yellow.



Screenshot 8: Visualizing the “simple” Boolean network

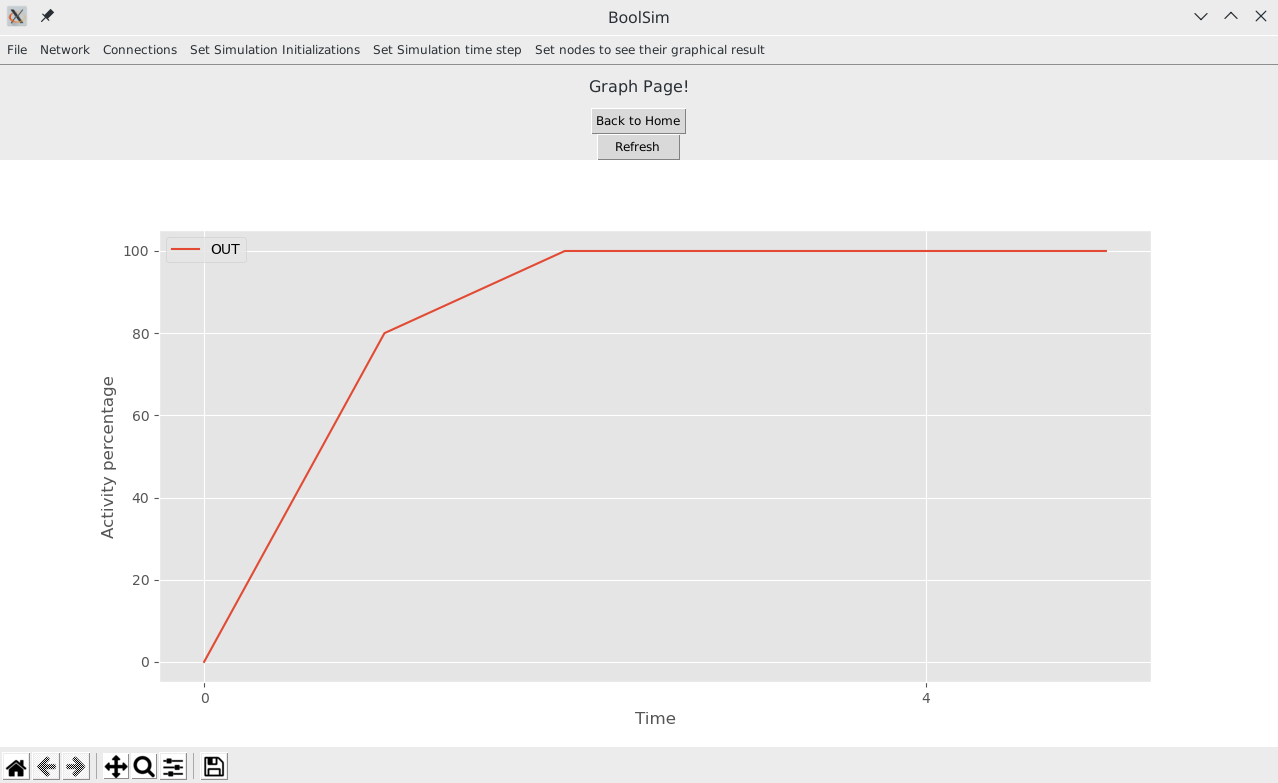
1. **To** **set the number of initial conditions** before simulating the network, click on the ‘Set Simulation Initializations’ option on the menu bar. You may choose one of the options in the dropdown menu by clicking on them or click on ‘Input arbitrary initializations’ to enter a (different) number into the dialog box. Click on the ‘Add’ button of the dialog box before closing it.
2. **To set the number of time steps,** click on the ‘Set Simulation Time Step’ option on the menu bar. You may choose one of the options in the dropdown menu by clicking on them or click on ‘Input arbitrary time steps’ to enter a (different) number into the dialog box. Click on the ‘Add’ button of the dialog box before closing it.
3. If not at the home page, click “Back to Home”. Otherwise, good to go!
4. **To** **run the simulation,** click on the ‘Start Simulate Process’ button on the home page. Depending on the complexity of the network and the number of time steps, this may take a few minutes. The button remains inactive while the simulation is running and becomes active once the simulation has been completed. Monitor the progress of the simulation on the command prompt window. At the beginning of each initialization, a message is displayed on the command prompt. See screenshot 9.



Screenshot 9: Monitor the command prompt while the simulation is running.

After the completion of the simulation, the results are stored in a text file result.txt in the same program folder. Caution: for complex networks, this file can be very large.

1. **To visualize the average activity of a particular node**, click ‘Set nodes to see their graphical result’➝ ‘set node’ on the menu bar and enter the name of the node. If you want to visualize more than one node, a separate procedure is required. For this, you need to open the text file ‘scratch paper.txt’ in the main folder and type each node that you wish to visualize on a new line. Copy all lines and paste it into the ‘set node’ box. For example, if you are simulating the simple network, you may type ‘IN’, ‘X’, ‘Z’ (without quotes) in separate lines in the file ‘scratch paper.txt’. Copy-paste these lines to the text box of the pop-up menu. Click on the ‘Add’ button on the pop-up menu and close it. Click on the ‘See graphical analysis from simulation’ button on the homepage. You will now see a plot and a warning dialog box. Close the warning dialog box and hit Refresh when the plot appears. This last step ensures that the plot only displays the nodes entered in the file. Screenshot 10 shows the activity of the OUT node in the simple network.



Screenshot 10: Activity of OUT node in the simple network.

1. Click on the ‘Back to Home’ button to get back to the home page of BoolSim in case you want to run a different simulation.

# 6. Preparing the Network

BoolSim comes packaged with text files containing node names and the update equations for a simple network and stomatal closure networks mediated by ABA and ABA+CO2. You may first simulate the ‘simple network’ to make sure everything is working fine. To simulate any boolean network on BoolSim, you need a file containing the names and initial states of the nodes (called node-name-file) and a file containing the update equations of the nodes (called equation-file). The node-name-file and equation-file for the predefined boolean networks are located in the sub-folders as listed below:

1. Folder containing the simple network: sample\_data\_files \ simple\_network\_data\_files
   1. Node-name-file: sample\_data\_files \ simple\_network\_data\_files \ simple\_net\_node\_names.txt
   2. Equation-file: sample\_data\_files \ simple\_network\_data\_files \ simple\_net\_word\_eqns.txt
2. Folder containing the ABA network: sample\_data\_files \ ABA\_data\_files
   1. Node-name-file: Node Name and their initial state with ABA.txt
   2. Equation-file (you may use one of the two, the word format is recommended):
      1. File with equations written in words: Boolean Equations in words with ABA.txt
      2. File with equations written in indices: Boolean Equations in index with ABA.txt
3. Folder containing the ABA-CO2 network (Stomata 2.0): sample\_data\_files \ ABA\_CO2\_data\_files \ stomata2-0
   1. Node-name-file: Node Name and their initial state Stomata2-0.txt
   2. Equation-file (you may use one of the two, the word format is recommended):
      1. File with equations written in words: Boolean Equations in words Stomata2-0.txt
      2. File with equations written in indices: Boolean Equations in index Stomata2-0.txt
4. For Stomata 2.1, refer to sample\_data\_files \ ABA\_CO2\_data\_files \ stomata2-1. The files are named similarly as in Stomata 2.0

To define your own network, follow the steps below. You may refer to the above data files as templates for the files you create.

## 6.1. Creating a node-name-file

1. First, create a folder inside sample\_data\_files to store the files pertaining to your new network. For example, create a folder ‘my\_bool\_network’ inside sample\_data\_files.
2. Open Notepad (or your favorite text editor) and save a new file in this directory. For example, save the new file as ‘node-name-file.txt’ in the ‘my\_bool\_network’ folder.
3. In your node-name-file, type the name of each node of your network in a separate line.
4. If the initial state of a node is to be fixed, instead of being assigned randomly, type the initial state (0 or 1) after a space following the name.
5. In the visualization of the network, if you want a node to be colored in a different color (yellow is the default color), type the color at the end of the line for that node. Available colors are purple, blue, green, yellow, orange, and red.

For example, when a node named SLAC1 needs to be colored orange and if its initial state should be 0, you should type

SLAC1 0 orange

in the line containing SLAC1.

When a node named ROS is randomly initialised but needs to be colored blue in the visualization, you should type

ROS blue

in a new line.

1. The contents of this file will be copied into a dialog box. Save this file before closing.
2. Refer to the node-name-files of the pre-defined networks for examples.

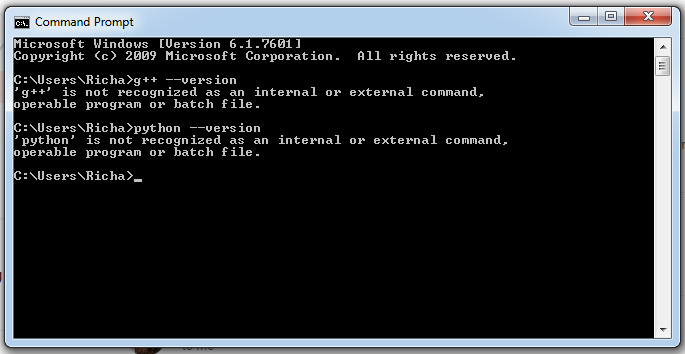
## 6.2. Creating an equation-file

1. Create a new text file in Notepad (or your favorite text editor) and save it in the same directory as you did the node-name-file. For example, save this file as ‘equation\_file.txt’ in the ‘my\_bool\_network’ folder.
2. Word format is recommended for the equation-file. The instructions given here are for the word format.
3. Each line in the equation-file contains an update equation for a node. It is recommended to type out the equations in the same order as they appear in node-name-file, though it is not compulsory. The equations should be in the format as described below. There is also a scratch paper.txt for formatting the input to the program. Also, check the existing examples.
4. On the left hand side (LHS), type the name of the node being updated. The names are case-sensitive, so they should be identical to the names in the file for node names.
5. The right hand side (RHS) should be written in the Sum of Products (SoP) form as described in the next section ‘A primer on Boolean Logic’
6. Look at the existing equation-files in word format for examples.

# 7. Troubleshooting

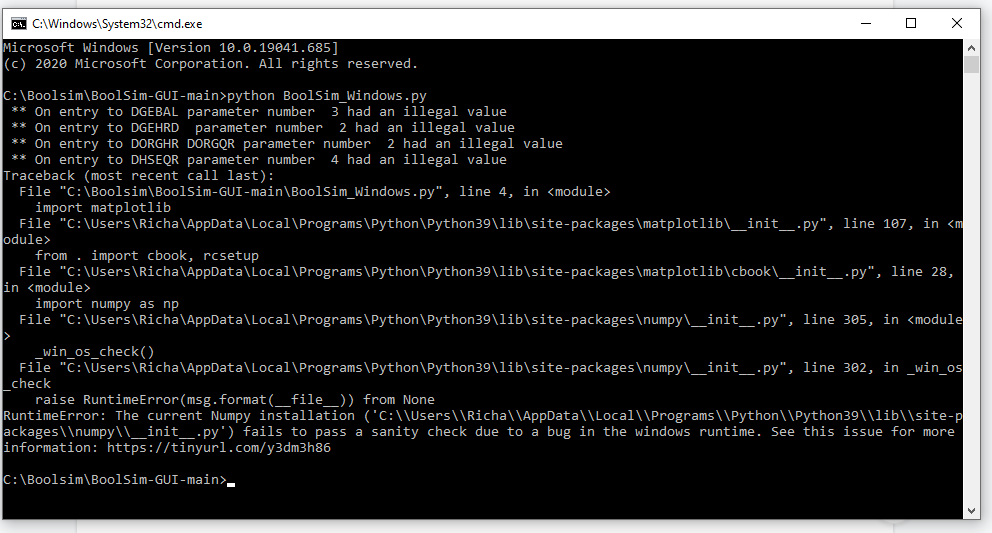
Here we list some common errors - and their fixes - that could be encountered while installing various packages and running BoolSim.

1. **Python or g++ are unrecognized**. This happens when g++ and/or python are not installed on the computer yet. See screenshot 11.
   1. You may encounter this error even if you have Python installed in some other way, say with Spyder. It is advisable to do a fresh and complete installation of Python as described in the video referenced in Section 1.



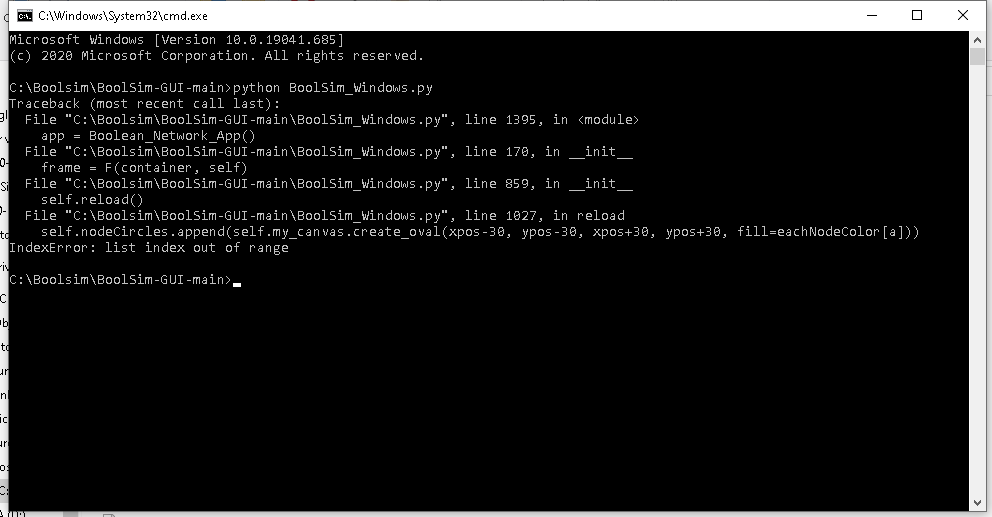
Screenshot 11: g++ and python are not recognized as commands on the command prompt before they are installed on the computer.

1. **Numpy installation fails to pass a sanity check.** This seems to be a common problem with installing Numpy on Windows systems, especially if you tried running the command “pip install numpy”. See screenshot 12. The error would not occur if you run “pip install numpy==1.19.3”, specifying the version number too, as mentioned in the instructions in Section 1.

****

Screenshot 12: This error can be overcome by specifying the version of Numpy during its installation

1. **List out of range.** This error might be encountered when you try to run “python BoolSim\_Windows.py”. See Screenshot 13. In this case please check ‘Boolean Equations File’, ‘Word Boolean Equations File’, ‘Node Name and their initial state’ in the folder ‘BoolSim-GUI-main’. These files should be empty. If something is written in these files, then please empty and save them, and try again.



Screenshot 13: This error generally shows up if the data files of the network are not empty or compatible with each other while starting BoolSim.