# JoSIM - Superconducting Circuit Simulator

Developers Manual
v1.2

**Johannes A. Delport**

Stellenbosch University
South Africa
July 12, 2018

# Contents

# 1 Introduction and setup

## 1.1 Introduction

JoSIM was developed under IARPA contract SuperTools(via the U.S. Army Research Office grant W911NF-17-1-0120). JoSIM is a analogue circuit simulator with SPICE syntax input that has inherent support for the superconducting Josephson junction element.

JoSIM is meant to function as a replacement to the aging simulator JSIM[1]. JoSIM is written in modern C++ and is fully customizable and extendable to offer support for improved superconducting elements as well better approximations to the Josephson effect in superconducting materials.

A *.cir* or *.js* file containing a SPICE syntax circuit netlist is provided as input. The circuit netlist, given appropriate input excitations can then be simulated through transient analysis. Results of this simulation can be either plotted for quick reference or saved as either a space delimited (*.dat*) or a comma separated value (*.csv*) file.

Fig.1 shows an overview of what JoSIM aims to accomplish. This is much like any other SPICE deck simulator with the exception that it incorporates native handling of the Josephson junction.

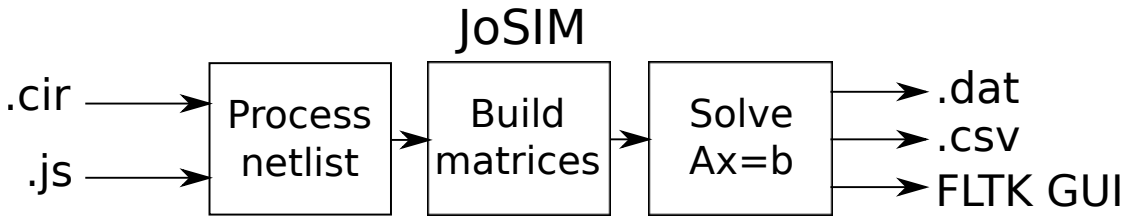


Figure 1: A macro overview of JoSIM

## 1.2 Initial setup

JoSIM can be found at JoSIM.git from where it can be cloned and compiled for either UNIX or Windows. Within this repository there will be a **CMakeLists.txt** which is a recipe used to compile JoSIM using CMake.

To compile the source a working C++ compiler with support for C++14 is required. Additionally SuiteSparse linear algebra libraries are required but are provided in the repository. Git version control software is recommended but is not required to compile JoSIM. Furthermore, the user has the option to enable additional features for plotting. These features are disabled by default, but would require either FLTK or Matplotlib (through Python) and can be enabled/disabled during compilation.

A single executable binary is generated using the CMake recipe and can be placed anywhere on the system as well as freely redistributed.

## 1.3 License

JoSIM is governed by the MIT license which is a very permissive license that allows anyone to redistribute it as well as commercialize it without repercussions. The MIT license allows use of this software within proprietary software as long as all copies of the licensed software includes a copy of the MIT license as well as the copyright notice.

## 1.4 Building from source

### 1.4.1 UNIX

These instructions were executed on a vanilla install of CentOS 7 Minimal to reduce oversight in the compilation instructions created by previous package installs. For other distributions please use the package manager relevant to the distribution of choice.

#### 1.4.1.1 CentOS 7 Minimal Enviroment Set-up

A working internet connection is required, this might be enabled by default in other distributions however from Minimal the user would be required to run:

```
$ sudo ifup <network interface>
```

Where the network interface can be identified using the *ip addr show* command.
Once the internet connection is up and running, install git version control software using the command:

```
$ sudo yum install git
```

Thereafter, navigate to a directory where compilation will take place and execute:

```
$ git clone https://github.com/JoeyDelp/JoSIM.git
```

For the minimal install the user would also need to install standard development tools by executing:

```
$ sudo yum groupinstall "Development Tools"
```

This will install the latest version of *gcc* and *make* tools available to CentOS 7. Additionally, CMake 3 will be required to compile this tool properly and would require the activation of the *epel-release* repository.

```
$ sudo yum install epel-release
$ sudo yum install cmake3
```

The basic version without graphical plotting, requires only a single external library SuiteSparse. Install it using the command:

```
$ sudo yum install suitesparse suitesparse-devel
```

Navigate to the newly cloned/extracted JoSIM directory then run the following commands:

```
$ mkdir build
$ cd build
$ cmake3 ..
$ make
```

This will generate a JoSIM executable in the *build* directory.

#### 1.4.1.2 CentOS 7 Graphical Environment Set-up

JoSIM offers the ability to do visual plotting of the results through either the FLTK graphical library or the Python based Matplotlib library. The former will require the installation of the FLTK libraries using the command:

```
$ sudo yum install mesa-libGL-devel fltk fltk-fluid fltk-
   devel
```

FLTK plotting can be enabled using the cmake flag during the build step in the previous section:

```
$ cmake3 -DUSING_FLTK=1 ..
```

The latter Matplotlib library requires that the Python pip, devel, numpy and matplotlib be installed using:

```
$ sudo yum install python-pip python-devel tkinter tk tk-
   devel
$ sudo pip install numpy matplotlib
```

Matplotlib plotting can be enabled using the cmake flag during the build step in the previous section:

```
$ cmake3 -DUSING_MATPLOTLIB=1 ..
```

#### 1.4.1.3 Apple macOS

Apple macOS is very similar to most Unix systems and therefore follows much the same procedure. The user would clone the repository and install CMake as well as the necessary libraries as indicated in previous sections. These libraries can be installed using either Homebrew, Macports or compiled from source using the standard macOS compilers. Much effort is made in the CMake file for JoSIM to attempt to identify the location of the required libraries. If the libraries cannot be found then alterations to the CMake file would need to be made to suit the users unique circumstances.

### 1.4.2 Windows

A Microsoft Visual Studio solution is provided and can be found in the *src* folder. This is by far the easiest way to compile the software under a Microsoft Windows environment. Simply open the solution and click build (F6) to build either Debug or Release targets for the software.

There are several configurations that allow for each of the graphical engines as well as the one without. FLTK static libraries for Windows are distributed with the source of JoSIM and will allow for the seamless compilation for this configuration.

Python on the other hand will require additional setup by the user. The easiest method of installing Python and the only one tested for this software is Anaconda Python `https://www.anaconda.com/download/#windows`. Either of the two versions 3.6 or 2.7 will work as the software has been tested to work on both. Version 3.6 is recommended as it boasts improvements to the UI elements of Matplotlib, which allow additional functionality.

To execute the Python version of this tool please ensure that the Python executable location is the the Windows PATH environment variable.

The executables once compiled can be found under *bin/win* followed by the chosen configuration.

# 2 Technical discussion

## 2.1 Modified nodal analysis

There are many ways to set-up a set of linear equations to solve the voltage or currents in a circuit. One of the more well known ways is to use nodal analysis which creates an equation for each node defined in the circuit netlist. This method is the basis on which the original Berkeley SPICE[2] was built. This method however only calculates the voltages of every node which makes it difficult to handle components that are voltage dependent such as inductors and junctions.

This drawback lead to the creation of the modified nodal analysis which is an extension to the prior with the ability of calculating some of the branch currents in the circuit.[3] We therefore make use of the MNA to build the set of linear equations within JoSIM due to the large use of inductors as well as Josephson junctions in superconductivity.

Another useful feature of MNA is the way that every component can be represented as a sub-matrix we call a stamp. The summation of all the stamps provide us with the A, x as well as b matrices that are required to solve the linear equations. These stamps will be discussed further in the following subsection.

## 2.2 Trapezoidal integration

Much like the nodal analysis mentioned before, there are multiple methods of solving differential equations with minimal error in a digital system. The most basic method is the forward Euler method which is a first-order approximation method where the error is proportional to the step size.[4]. Solving differential equations however produce a local truncation error which is the difference between the numerical solution and the exact solution after one step. It can be shown that the local truncation error for the forward Euler method is proportional to the square of the time step taken which makes this method less accurate compared to higher order methods.

We focus rather on using the trapezoidal rule for solving the linear equations as this method becomes increasingly more accurate as the time steps become smaller. The trapezoidal rule is a second-order method for solving differential equations. We can express the trapezoidal method as:

$$\left(\frac{dx}{dt}\right)_n = \frac{2}{h_n}\left(x_n - x_{n-1}\right) - \left(\frac{dx}{dt}\right)_{n-1} \tag{1}$$

In this case the $n$ is the current time step and $n-1$ refers to the previous time step. By using this method to solve differential equations we are able to create generic stamps for each component that JoSIM can handle.

To demonstrate this method and how a stamp is formed we will show an example of an

inductor. The inductor in Fig.2 has a general equation to determine the voltage across

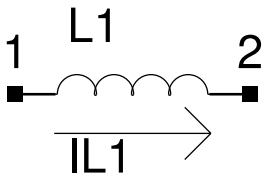


Figure 2: A basic inductor with current flowing through it

it as:

$$v_{L1}(t) = L1\frac{di}{dt} \tag{2}$$

When we apply the trapezoidal rule we find (2) can be rewritten as:

$$\begin{aligned} v_n &= L1\left[\frac{2}{h_n}\left(I_n - I_{n-1}\right) - \left(\frac{di}{dt}\right)_{n-1}\right] && (3)\\ &= \frac{2L1}{h_n}\left(I_n - I_{n-1}\right) - L1\left(\frac{di}{dt}\right)_{n-1} \\ &= \frac{2L1}{h_n}\left(I_n - I_{n-1}\right) - v_{n-1} \\ \therefore I_n &= \frac{h_n}{2L1}\left(V_n + V_{n-1}\right) + I_{n-1} && (4)\\ \therefore \frac{h_n}{2L1}V_n - I_n &= -\frac{h_n}{2L1}V_{n-1} + I_{n-1} && (5) \end{aligned}$$

Where (5) is the current step voltage and current as a function of the previous step values. We further expand the (5) by stating that the voltage is the potential across the two nodes: $v = v_1 - v_2$.

$$\frac{h_n}{2L1}(V_1)_n - \frac{h_n}{2L1}(V_2)_n - I_n = -\left(\frac{h_n}{2L1}V(V_1)_{n-1} - \frac{h_n}{2L1}(V_2)_{n-1}\right) - I_{n-1} \tag{6}$$

Which we can then write in matrix form as:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & \frac{-2L1}{h_n} \end{bmatrix}\begin{bmatrix} V_1 \\ V_2 \\ I_{L1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\frac{2L1}{h_n}(I_{L1})_{n-1} - ((V_1)_{n-1} - (V_2)_{n-1}) \end{bmatrix} \tag{7}$$

(7) is a generic stamp that we can place into the A matrix which describes the inductor L1.

We can do the same for a resistor, capacitor, current source, voltage source, Josephson junction and a transmission line. The stamps for each of these components can be found in Appendix

## 2.3 LU decomposition

When the A matrix has been set up as detailed in the previous section all that is left to do is to solve the $Ax = b$ problem using some form of iterative method. We choose KLU from the SuiteSparse[5] library to accomplish this task.

This requires the A matrix to be in compressed row storage (CRS) format which is a data structure of 3 vectors. The first of these vectors contains all the non-zero elements in the A matrix. The second contains first a 0 followed the total number of non-zero elements after each row such that the final entry in the vector is the total number of non-zero elements. The third vector contains the column index of each non-zero element. As an example, the following sparse matrix of $5x5$

$$\begin{bmatrix} 1 & 0 & 0 & 4 & 0 \\ 0 & 3 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 4 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

would yield a CSR format data structure of

$$\begin{aligned} nnz &= [1, 4, 3, 2, 1, 4, 5, 1] \\ rowptr &= [0, 2, 4, 5, 7, 8] \\ colind &= [0, 3, 1, 2, 2, 1, 3, 4] \end{aligned}$$

which has a total of 21 elements compared to the 25 required for the original A matrix. This difference of course becomes much larger the larger the A matrix becomes as well as the sparser the matrix becomes. Which is almost always the case for electrical type simulations.

Once in this format we can proceed with the KLU factorization. Due to the MNA forcing only the RHS to change upon every time step we can easily do the LU decomposition for the A matrix only once at the beginning of the time loop. Where after the system is simply solved using with a new RHS upon each iteration.

## 2.4 Data structures & speed considerations

JoSIM relies heavily on the use of the C++ unordered map data structure which creates a hash table for quick lookup. This immensely simplifies code legibility as well as component identification within later stages of the process.

Initially a standard map was used, however this very negatively impacted the speed of execution of the program and thus alternatives were sought. Ideally the use of unordered maps are not perfect if speed was the only consideration, however they do provide a good balance between implementation, speed and debugging.

There is still large room for improvement with regards to parallel processing of specific stages within the execution, however the largest part of the process (the time loop) is very loop dependent and therefore cannot be parallel processed. This also limits the speed improvement to relatively large circuits as the overhead required for thread delegation slows down the execution on smaller problems.

### 2.5 FLTK

JSIM lacked the ability to provide direct feedback on the results of the simulation. Unless the user had some script to plot the data file that it produced or some third-party application that could plot the results no clear feedback was received.

With JoSIM we introduce the implementation of the cross-platform C++ GUI library FLTK[6] which allows the results of the simulation to be instantly plotted in a data graph window. Though the full extent of the FLTK library has not been explored. The current implementation acts as a proof-of-concept design.

### 2.6 Matplotlib

With the complexity of FLTK yet to be explored an additional option for plotting the results has been provided. This method utilizes a C++ header only that interfaces with Python to display the results in a Matplotlib window. This method of plotting instantly allows the user to resize, pan and save the results. File formats include SVG, PNG, EPS and many more.

Though easy to implement and interpret, the method requires a lot of additional setup for each individual system and becomes very hard to debug if errors occur. This method is also limited to only the functionality provided by Matplotlib and alternatives will be looked at in the future.

# 3 Input files

Input files follow a SPICE syntax similar to JSIM. This allows for a direct use of the files that are already executable using JSIM as well as tools that write output files for use with JSIM.

Each SPICE file (*.cir* or *.js*) can be broken into subsections such as comments, subcircuits, main circuit and control sections. Each individual line is categorized by the first character. This is key to the efficiency of the entire tool as that single first character defines what the program does with that line as the file is read in line by line.

An example of a input file is given in the example section of this document.

# 4 Command line arguments/switches

A full range of the available command switches and their arguments is provided to the user upon specification of the *-h* switch.

```
[$ ./JoSIM_mac_Debug -h

JoSIM: Josephson Junction Superconductive SPICE Circuit Simulator
Copyright (C) 2018 by Johannes Delport (jdelport@sun.ac.za)
v1.2 compiled on Jul 12 2018 at 15:33:10
Plotting engine: MATPLOTLIB
Parallelization is DISABLED

JoSIM help interface
====================
-c(onvention)|  Sets the subcircuit convention to left(0) or right(1).
             |  Default is left. WRSpice (normal SPICE uses right)
             |  Eg. X01 SUBCKT 1 2 3     vs.     X01 1 2 3 SUBCKT
-g(raph)     |  Plot the requested results using a plotting library
             |  If this is enabled with verbose mode then all traces are plotted
-h(elp)      |  Displays this help menu
-o(utput)    |  Specify output file for simulation results (.csv)
             |  Default will be output.csv if no file is specified
-m(atlab)    |  [Legacy] JSIM_N output file format (.dat)
             |  Default will be output.dat if no file is specified
-v(erbose)   |  Runs JoSIM in verbose mode
--v(ersion)  |  Displays the JoSIM version info only

Example command: josim -g -o ./output.csv test.cir
```

Figure 3: JoSIM help command menu

Each help menu item is pretty self explanatory and should not require further documentation. These command switches can be placed in any order and can be concatenated so long as the final command is the input file.

# 5 Output

## 5.1 Comma seperated value

The output to a comma separated value format file was chosen due to the large support therefore and ease of import into third-party tools such as Microsoft Excel. This simplifies the movement, copying and removal of columns from the output file a lot simpler as well.

## 5.2 Space seperated value

This is the legacy format that JSIM uses to output it's results into. We opt to support this file format due to the large user base of JSIM and the amount of plotting/viewing tools written around this output format. This output format can also be easily read in and manipulated by third party tools and we shall therefore keep this option with the same command switch as JSIM.

## 5.3 GUI plotting window

The output of results directly into a plotting window that is user viewable improves the prototyping speed immensely as the user can immediately see whether the output is as desired or not. Though primitive at present with the lack of any axis or tooltips to show that the values of each graph are as well as the lack of the ability to resize the graphs within the window, we will continue to expand on this functionality to eventually have a user-friendly output window.

# 6 Examples

Below is an example file that Chains together a DCSFQ, 3 x JTLs and a SINK cell.

```
1  .SUBCKT JTL 4 5
2  B01          3            7            jj1     area=2.16
3  B02          6            8            jj1     area=2.16
4  IB01         0            1            pwl(0 0 5p 280u)
5  L01          4            3            2.031p
6  L02          3            2            2.425p
7  L03          2            6            2.425p
8  L04          6            5            2.031p
9  LP01         0            7            0.086p
10 LP02         0            8            0.086p
11 LPR01        2            1            0.278p
12 LRB01        7            9            1p
13 LRB02        8            10           1p
14 RB01         9            3            5.23
15 RB02         10           6            5.23
16 .model jj1 jj(rtype=1, vg=2.8mV, cap=0.07pF, r0=160, rn=16, icrit=0.1mA)
17 .ends JTL
18 .SUBCKT DCSFQ 2 17
19 B01          5            3            jj1     area=1.32
20 B02          5            6            jj1     area=1
21 B03          9            10           jj1     area=1.5
22 B04          13           14           jj1     area=1.96
23 B05          15           16           jj1     area=1.96
24 IB01         0            8            pwl(0 0 5p 162.5u)
25 IB02         0            12           pwl(0 0 5p 260u)
26 L01          2            1            0.848p
27 L02          0            1            7.712p
28 L03          1            3            1.778p
29 L04          5            7            0.543p
30 L05          7            9            3.149p
31 L06          9            11           1.323p
32 L07          11           13           1.095p
33 L08          13           15           2.951p
34 L09          15           17           1.63p
35 LP01         0            6            0.398p
36 LP02         0            10           0.211p
37 LP03         0            14           0.276p
38 LP04         0            16           0.224p
39 LPR01        7            8            0.915p
40 LPR02        11           12           0.307p
41 LRB01        4            5            1p
42 LRB02        18           6            1p
43 LRB03        19           10           1p
44 LRB04        20           14           1p
45 LRB05        21           16           1p
46 RB01         3            4            8.56
47 RB02         18           5            11.30
48 RB03         19           9            7.53
49 RB04         20           13           5.77
50 RB05         21           15           5.77
51 .model jj1 jj(rtype=1, vg=2.8mV, cap=0.07pF, r0=160, rn=16, icrit=0.1mA)
52 .ends DCSFQ
53 .SUBCKT SINK 2
54 B01          1            4            jj1     area=2.16
55 IB01         0            5            pwl(0 0 5p 280u)
56 L01          2            1            0.517p
57 L02          1            3            5.307p
58 LP01         0            4            0.086p
59 LPR01        1            5            0.265p
60 LRB01        4            6            1p
61 RB01         6            1            5.23
62 ROUT         0            3            4.02
63 .model jj1 jj(rtype=1, vg=2.8mV, cap=0.07pF, r0=160, rn=16, icrit=0.1mA)
64 .ends SINK
65 IA 0 1 pwl(0 0 170p 0 176p 600u 182p 0 370p 0 376p 600u 382p 0 600p 0 606p 600u 612p 0 700p 0
       706p 600u 712p 0)
66 X01          DCSFQ          1          2
67 X02          JTL            2          3
68 X03          JTL            3          4
69 X04          JTL            4          5
70 X05          SINK           5
71 .tran 0.25p 1000p 0 0.25p
72 .print nodev 1 0
73 .print nodev 3 0
74 .print nodev 5 0
```

This file can be found under *JoSIM/test/dcsfq_jlt_sink.js* and we will now demonstrate

this example using JoSIM.

```
[$ ./JoSIM_mac_Debug -g -o ../test/dcsfq_jtl_sink.js

 JoSIM: Josephson Junction Superconductive SPICE Circuit Simulator
 Copyright (C) 2018 by Johannes Delport (jdelport@sun.ac.za)
 v1.2 compiled on Jul 12 2018 at 15:33:10
 Plotting engine: MATPLOTLIB
 Parallelization is DISABLED

 Path specified for input file: ../test/dcsfq_jtl_sink.js
 Input file specified as: dcsfq_jtl_sink.js

 Circuit characteristics:
 Subcircuits:                    3

 SINK component count:           9
 SINK JJ count:                  1
 DCSFQ component count:          32
 DCSFQ JJ count:                 5
 JTL component count:            14
 JTL JJ count:                   2

 Main circuit component count:   84
 Main circuit JJ count:          12

 Simulating:
 100%[==============================]
```

Figure 4: JoSIM command line window output from execution of the dcsfq_jtl_sink.js

Fig 4 shows the default output from the JoSIM executable produces some statistics about the file provided as input and the reuse of subcircuits within the file. It also gives the user a count of the amount of Josephson junctions used within the circuit as this is usually taken as a measure of how large the circuit is.

The results of the *-g* command switch can be seen in Fig 5 and 6. Which we can clearly see from is the results expected given the type of circuit. The 3 graphs that are plotted are the node voltages at the input to the DCSFQ, the middle JTL and the input of the SINK.

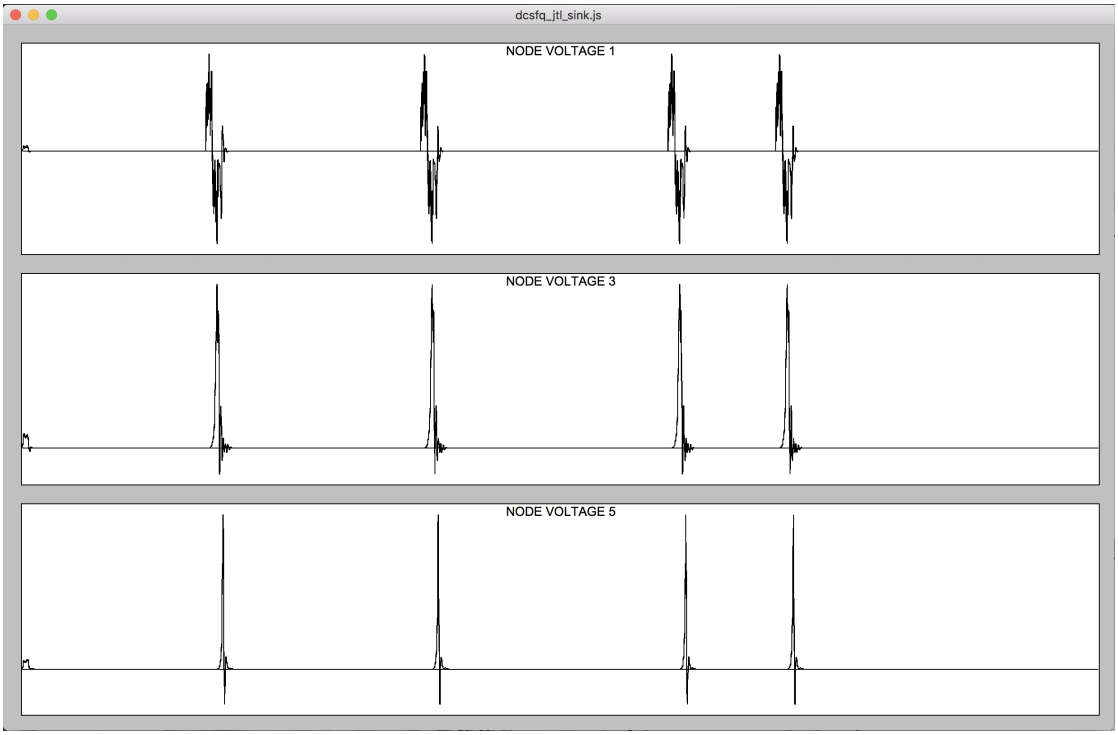


Figure 5: JoSIM FLTK results window from execution of the dcsfq_jtl_sink.js file

There are a few larger examples included in the test folder which can be executed to test the execution speed of JoSIM.

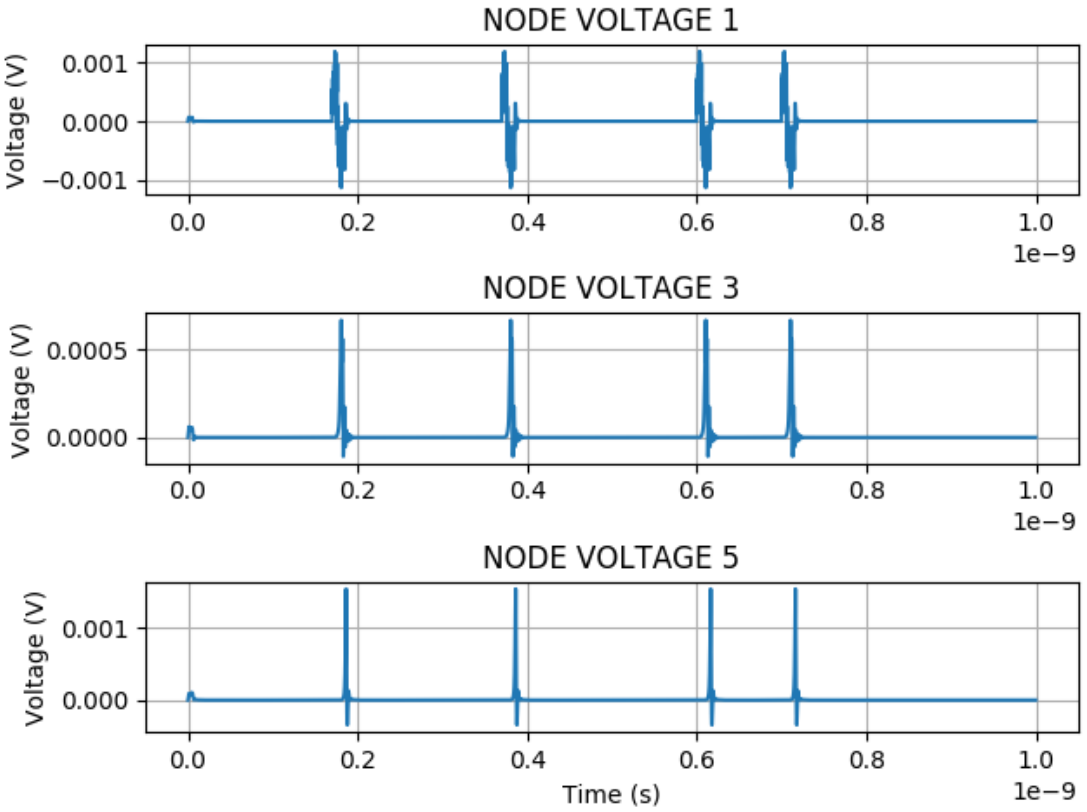


Figure 6: JoSIM Matplotlib results window from execution of the dcsfq_jtl_sink.js file

## 7 Error handling and exceptions

An attempt is made to handle as many of the exceptions thrown as possible and to provide the user with accurate and meaningful error messages in the case of an error. If any issues are not caught or bugs are picked up during the usage of JoSIM, please do not hesitate to report them on the git page under issues. The software has had very little exposure until now. It is a version 1 tool and there will be bugs, please use with caution and please report any issues.

## 8 Planned improvements

There are several planned improvements to JoSIM which include per device temperature dependence as well as global temperature to more accurately model the effects of superconductive materials.

Engine improvements will include new Josephson junction models that extend beyond the default RCSJ model, as well as the ability to do phase based simulation will usher in

the ability to simulate the Werthamer approximated model for the Josephson junction. Further engine improvements will also include parallel processing of certain stages within the execution as well as the ability to switch to different linear solvers.

# Appendices

## A Component Stamps

### A.1 Resistor

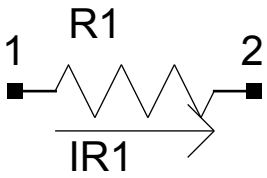


Figure 7: A basic resistor with current flowing through it

$$\begin{aligned} v_{R1}(t) &= i_{R1}(t)R1 \\ (V_1 - V_2)_n &= I_n R1 \\ \frac{1}{R1}(V_1)_n - \frac{1}{R1}(V_2)_n - I_n &= 0 \end{aligned} \tag{8}$$

We however do not need to calculate the current through the Resistor as it is not always needed, we therefore omit it in the matrix and calculate it when the user requests it.

$$\begin{bmatrix} \frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & \frac{1}{R} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

### A.2 Capacitor

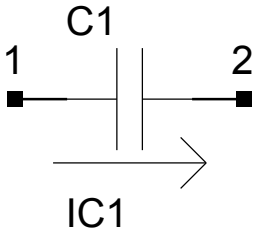


Figure 8: A basic capacitor with current flowing through it

$$\begin{aligned} i_n(t) &= C1\frac{dv}{dt} \\ i_n(t) &= \frac{2C1}{h_n}\left[(V_n - V_{n-1}) - \left(\frac{di}{dt}\right)_{n-1}\right] \\ \frac{2C1}{h_n}(V_1)_n - \frac{2C1}{h_n}(V_2)_n - I_(n) &= \frac{2C1}{h_n}(V_1 - V_2)_{n-1} + I_{n-1} \end{aligned} \tag{9}$$

Once again we omit the current for the capacitor as it will only complicate the calculation. We therefore only calculate the voltage output.

$$\begin{bmatrix} \frac{C}{h_n} & -\frac{C}{h_n} \\ -\frac{C}{h_n} & \frac{C}{h_n} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} \frac{C}{h_n} \\ -\frac{C}{h_n} \end{bmatrix}$$

### A.3 Voltage source

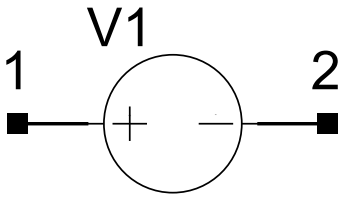


Figure 9: A basic voltage source

A voltage source simply adds the voltage as a new row and column to the matrix, similar to a branch current.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ V1 \end{bmatrix}$$

### A.4 Current source

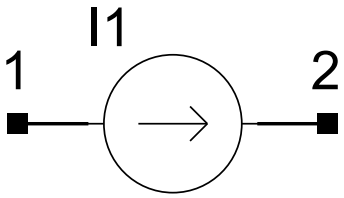


Figure 10: A basic capacitor with current flowing through it

A current source does not affect the A matrix in any way and simply adds or subtracts the current supplied at the respective nodes.

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} -I1 \\ I1 \end{bmatrix}$$

### A.5 Josephson junction

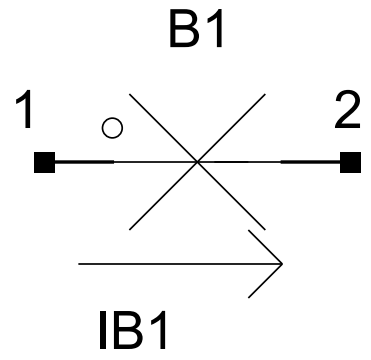


Figure 11: A basic Josephson junction

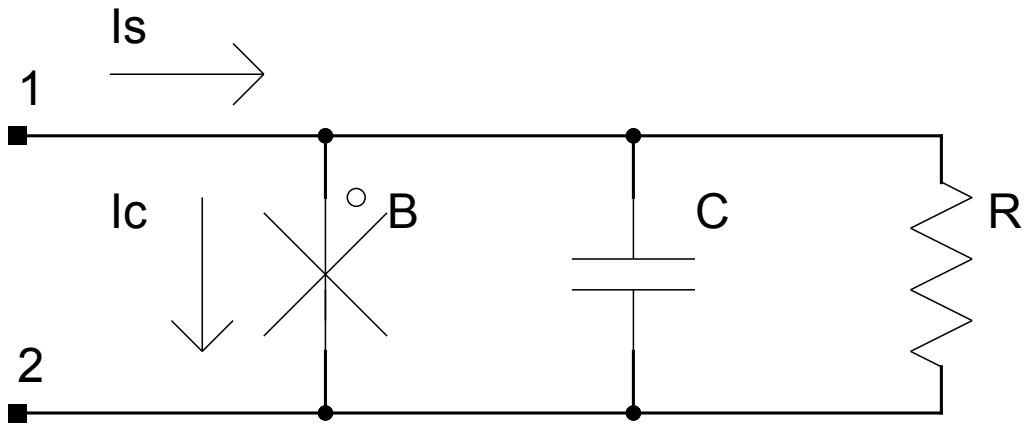


Figure 12: A basic Josephson junction in Resistively, Capacitively Shunted Junction form

$$I_s = -I_c \sin \phi_n^0 + \frac{2C}{h_n} v_{n-1} + C \left( \frac{dv}{dt} \right)_{n-1} \tag{10}$$

In this case $I_c$, $C$, $\phi_n^0$ are the junction critical current, capacitance and initial phase guess.

$$\phi_n^0 = \phi_{n-1} + \frac{h_n}{2} \frac{2e}{\bar{h}} (v_{n-1} + v_n^0) \tag{11}$$

$$
\begin{aligned}
v_n^0 &= v_{n-1} + h_n \left(\frac{dv}{dt}\right)_{n-1} \\
&= v_{n-1} + h_n \left[\frac{2}{h_n}\left(v_{n-1} - v_{n-2}\right) - \left(\frac{dv}{dt}\right)_{n-2}\right] \\
v_n^0 &= 3v_{n-1} - 2v_{n-2} - h_n \left(\frac{dv}{dt}\right)_{n-2} \qquad (12)
\end{aligned}
$$

When we apply 12 to 11 we can approximate an equation for the initial phase guess. To approximate this we would need to specify values for 3 time steps before the first time step. By assuming these values are always 0 we can then implement this as a stamp.

$$
\begin{bmatrix}
\frac{2C}{h_n} + \frac{1}{R} & -\frac{2C}{h_n} - \frac{1}{R} & 0 \\
-\frac{2C}{h_n} - \frac{1}{R} & \frac{2C}{h_n} + \frac{1}{R} & 0 \\
-\frac{h_n}{2}\frac{2e}{h} & \frac{h_n}{2}\frac{2e}{h} & 1
\end{bmatrix}
\begin{bmatrix} V_1 \\ V_2 \\ \phi \end{bmatrix} =
\begin{bmatrix} I_s \\ -I_s \\ \phi_{n-1} + \frac{h_n}{2}\frac{2e}{h} v_{n-1} \end{bmatrix}
$$

### A.6 Transmission line

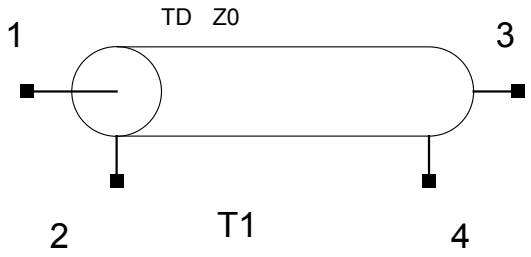


Figure 13: A basic lossless transmission line

The implementation of a lossless transmission line requires the everything on the right-hand side to be delayed by the $TD$ specified by the user. The $Z0$ is the line impedance and is a function of the transmission line length.

The voltage on between node 1 and 2 is delayed by $TD$ before appearing at node 3 and 4, where after a reflection of the result at node 3 and 4 is observed at node 1 and 2. This effect continues every $TD$ until the voltage is completely diminished by the line impedance.

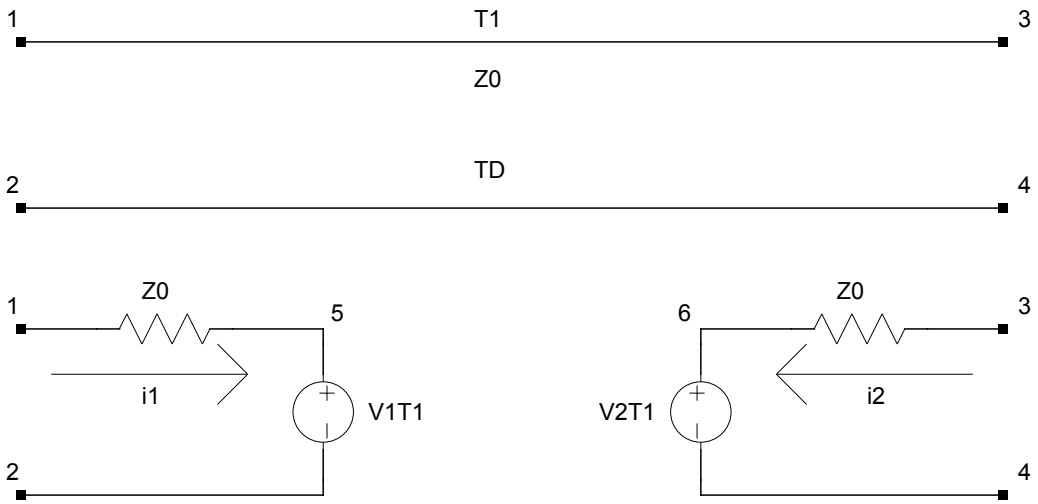


Figure 14: A basic lossless transmission line broken into components

The dependent voltage sources on either side of the transmission line can be described by

$$V1T1(t) = [V_3(t - TD) - V_4(t - TD)] + Z0 \cdot i_2(t - TD) \tag{13}$$
$$V2T1(t) = [V_1(t - TD) - V_2(t - TD)] + Z0 \cdot i_1(t - TD) \tag{14}$$

where

$$i_1 = \frac{V_1 - V_5}{Z0} \tag{15}$$
$$i_2 = \frac{V_3 - V_6}{Z0} \tag{16}$$

$$\begin{bmatrix} \frac{1}{Z0} & -\frac{1}{Z0} & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{Z0} & \frac{1}{Z0} & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{1}{Z0} & -\frac{1}{Z0} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{Z0} & \frac{1}{Z0} & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V1 \\ V5 \\ V2 \\ V3 \\ V6 \\ V4 \\ V1T1 \\ V2T1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ V1T1 \\ V2T2 \end{bmatrix}$$

## References


[1] E. S. Fang and T. Van Duzer. *A Josephson integrated circuit simulator (JSIM) for superconductive electronics application*. Extended Abstracts of 1989 International Superconductivity Electronics Conference, 407-410, 1989.

[2] Laurence W. Nagel and D.O. Pederson. *SPICE (Simulation Program with Integrated Circuit Emphasis)*. EECS Department, University of California, Berkeley, 1973.

[3] Ho, Ruehli, and Brennan. *The Modified Nodal Approach to Network Analysis*. Proc. 1974 Int. Symposium on Circuits and Systems, San Francisco. pp. 505509.

[4] Atkinson, Kendall A. *An Introduction to Numerical Analysis (2nd ed.)* New York: John Wiley & Sons, 1989

[5] Timothy A. Davis *Direct Methods for Sparse Linear Systems* SIAM, Philadelphia, Sept. 2006.

[6] F. Costantini, D. Gibson, M. Melcher, A. Schlosser, B. Spitzak and M. Sweet. *FLTK 1.4.0 Programming Manual*. 2018. [Online]. Available: `http://www.fltk.org/`. [Accessed: 29- Apr- 2018].