MiTMoJCo 1.2

Dmitry R. Gulevich
ITMO University, St. Petersburg, Russia

September 20, 2018

Contents

1 Introduction

MiTMoJCo (**Mi**croscopic **T**unneling **Mo**del for **J**osephson **Co**ntacts) represents a C library and collection of Python tools designed to assist modeling superconducting Josephson junctions within the formalism of microscopic tunneling theory. The purpose of the C code is to offer implementation of a computationally demanding part of this calculation which is evaluation of the superconducting pair and quasiparticle tunnel currents. The tunnel currents calculated by MiTMoJCo are offered to user's disposal to be employed in a specialized ODE/PDE solver or within a finite difference or finite element scheme in a custom C code. The source code contains a collection of Python tools to work with tunnel current amplitudes which characterize specific superconducting materials constituting a tunnel junction and examples of modeling some common cases of Josephson contacts.

MiTMoJCo is written by Dmitry R. Gulevich (ITMO University, St Petersburg, Russia) and is made publicly available under the GNU General Public License v3.0. This implies that you may freely copy, distribute, or modify the sources, but the copyright for the original code remains with the author and the ITMO University. The code can be downloaded from the github online repository https://github.com/drgulevich/mitmojco. If you find the code useful in your research, we kindly ask you to include a reference to the original paper [1] in all studies that use MiTMoJCo.

2 New in version 1.2

2.1 Changes between v1.0 and v1.1

1. MiTMoJCo 1.1 is compiled and installed via the CMake build process to a shared library libmitmojco.so (default location `/usr/local/lib`). MiTMoJCo header files are placed to the standard paths (by default `/usr/local/include/mitmojco`). To include them in your code use

```
#include <mitmojco/mitmojco.h> // MiTMoJCo main header file
#include <mitmojco/opt_filter.h> // MiTMoJCo optimum filtration
```

2. The library functions are now callable from C++. The header files are wrapped by

```
#ifdef __cplusplus
extern "C" {
#endif
...
#ifdef __cplusplus
}
#endif
```

to enable the calls from C++ code.

2.2 Changes between v1.1 and v1.2

1. Python module `mitmojco` for calculating tunnel current amplitudes (TCAs) and creation of custom TCAs fits.

2. Easy to use Jupyter Python notebook `amplitudes.ipynb` demonstrating the features of Python `mitmojco` module:
- calculation of bare BCS TCAs
- smoothing Riedel peaks in bare BCS TCAs
- creation of fits by a sum of complex exponentials.
- exporting fitted TCAs to file understood by MiTMoJCo C library functions.
- loading fitted TCAs from file.

3. The user is provided by more control over the tunnel current object by moving some of the private member variables of the class `TunnelCurrentType` to public. Memory of the previous evolution can be directly accessed via the member structure `MemState` of the tunnel current object.

4. An example of using MiTMoJCo in conjunction with `deal.II` finite element library [2, 3] for modeling 2D Josephson junctions.

5. Portability improved: requirement for CMake 3.5.1 in CMakeLists.txt reverted to CMake v2.6 or higher.

3 Basic Usage

3.1 Theory behind MiTMoJCo

The goal of MiTMoJCo is to aid evaluation of the tunnel current density in a Josephson tunnel junction,

$$j(\mathbf{r},t) = \alpha_N \frac{\partial \varphi}{\partial t} + \bar{j}(\mathbf{r},t) \tag{1}$$

where

$$\begin{aligned}\bar{j}(\mathbf{r},t) = \frac{k}{\operatorname{Re}\tilde{j}_p(0)} \int_0^\infty \Big\{ & j_p(kt') \sin\left[\frac{\varphi(\mathbf{r},t)+\varphi(\mathbf{r},t-t')}{2}\right] \\ & + \bar{j}_{qp}(kt') \sin\left[\frac{\varphi(\mathbf{r},t)-\varphi(\mathbf{r},t-t')}{2}\right] \Big\}\, dt',\end{aligned} \tag{2}$$

is the *reduced* tunnel current density which depends on the history of evolution of the superconducting phase difference $\varphi(\mathbf{r},t)$ through the convolution with pair and quasiparticle time-domain kernels $j_p(\tau)$ and $\bar{j}_{qp}(\tau)$. The tunnel current density (1) enters the integro-differential equation describing dynamics of superconducting phase difference in a Josephson tunnel junction,

$$\frac{\partial^2 \varphi}{\partial t^2} - \left(1 + \beta \frac{\partial}{\partial t}\right) \nabla^2 \varphi + \alpha_N \frac{\partial \varphi}{\partial t} + \bar{j}(\mathbf{r},t) = 0 \tag{3}$$

$$\mathbf{n}\cdot\left(1+\beta\frac{\partial}{\partial t}\right)\nabla\varphi=\mathbf{e}_z\cdot[\mathbf{n}\times\mathbf{h}]$$

where $\mathbf{n}$ is the in-plane outward normal, $\mathbf{h}$ is the normalized magnetic field in units $j_c\lambda_J$.

The bar over the reduced tunnel current density $\bar{j}(\mathbf{r},t)$ and a reduced quasiparticle time-domain kernel $\bar{j}_{qp}(\tau)$ signifies that the normal resistance contribution has been subtracted: it enters to the full tunnel current (1) and the equation (3) explicitly as a damping term $\alpha_N\partial\varphi/\partial t$. This is done for computational reasons to avoid the singularity at $\tau=0$ and obtain a regular behaviour of the quasiparticle time-domain kernel as a function of time. Furthermore, this allows constructing convenient semi-implicit numerical schemes where part of the tunnel current (the term $\alpha_N\partial\varphi/\partial t$) is computed implicitly. The time-domain kernels are defined by Fourier transforms of the *tunnel current amplitudes* which can be calculated theoretically from the Bardeen–Cooper–Schrieffer (BCS) theory, or, found experimentally.

In Eqs. (3) and (2) time is measured in units of the inverse angular Josephson frequency ω_J^{-1}, the spatial coordinates are expressed in units of the Josephson penetration length λ_J, the reduced current density $\bar{j}(\mathbf{r},t)$ is normalized to V_g/AR_N, where V_g is the gap voltage, A is the total area of the junction and R_N is the normal resistance.

3.2 Installation (UNIX-based OS)

MiTMoJCo 1.2 source code contains:

`include/`: folder with headers `mitmojco/mitmojco.h` and `mitmojco/opt_filter.h`;

`src/`: folder with the C source `mitmojco.c` and a supplementary optimum filtration routine `opt_filter.c`;

`module.py`: supplementary Python `mitmojco` module for working with tunnel current amplitudes;

`amplitudes.ipynb`: Jupyter notebook demonstrating the use of Python `mitmojco` module;

`amplitudes/`: folder containing a library of pre-calculated fits of tunnel current amplitudes for common types of Nb and NbN junctions;

documentation folder `doc/`;

`examples/`: folder containing examples of using MiTMoJCo C library;

`CMakeLists.txt`: an input to CMake used at installation.

`LICENSE`: GNU General Public License.

`README.md`: general information.

The latest development version can be downloaded by typing in the terminal

```
$ git clone https://github.com/drgulevich/mitmojco
```

Modify `CMakeLists.txt` to suit your needs. Default installation paths are set in `CMakeLists.txt` to `/usr/local/lib` for the shared library `libmitmojco` and `/usr/local`

`/include/mitmojco` for the headers. CMake version 2.6 or higher is required to run the CMake build procedure.

In the mitmojco directory create and enter the build/ folder. Then execute cmake, make and the installation procedure:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
```

Check the installation by running one of the provided examples in the `examples` folder. To compile the first example type

```
$ make example-1
```

Use similar command to compile other examples or type

```
$ make all
```

to compile all examples at once.

3.3 MiTMoJCo interface

The interface for MiTMoJCo is implemented in the spirit of object-oriented programming. To use MiTMoJCo in your C code include the header file,

```
#include <mitmojco/mitmojco.h>
```

and create a tunnel current object (`TunnelCurrentType` pointer) by calling the constructor declared in the header file as

```
TunnelCurrentType* mitmojco_create(
    char *filename, double a_supp, double kgap, double dt,
    int Ntotal, double *phi, int Nskip, int *skipinds );
```

Arguments of the constructor are:

`char *filename`: tunnel current amplitudes file,
`double a_supp`: pair current suppression parameter α_{supp},
`double kgap`: normalized gap frequency k,
`double dt`: integration time step,
`int Ntotal`: size of `phi` array,
`double *phi`: pointer to the superconducting difference array,
`int Nskip`: number of shadow nodes to skip,
`int *skipinds`: pointer to the array storing indices of the shadow nodes.

Often in numerical schemes one introduces *shadow nodes* used for treating the boundary conditions. However, the tunnel current only needs to be evaluated for the *physical*

nodes and, therefore, its evaluation at the shadow nodes can be skipped. The last two arguments accepted by the constructor are the number of the shadow nodes and the pointer to the array with their indices in the `phi` array.

The type `TunnelCurrentType` is defined in MiTMoJCo header file as a structure,

```
typedef struct {
    const char *filename;
    double a_supp;
    double kgap;
    double dt;
    int Ntotal;
    double *phi;
    int Nskip;
    int *skipinds;
    int Nnodes;
    int Nexps;
    MemState memstate;
    double Rejptilde0;
    double alphaN;
    double *jbar;
    void *self;
    bool error;
} TunnelCurrentType;
```

which has member variables:

`const char *filename`: pointer to tunnel current amplitude (TCA) file name (more on the tunnel current amplitudes in sec. 3.5).

`double a_supp`: pair current suppression parameter.

`double kgap`: normalized gap frequency.

`double dt`: integration time step.

`int Ntotal`: size of the array phi.

`double *phi`: pointer to the superconducting phase difference.

`int Nskip`: number of shadow nodes to skip. Often in numerical schemes one introduces shadow nodes used for treating the boundary conditions. However, the tunnel current only needs to be evaluated for the *physical nodes* and, therefore, its evaluation at the shadow nodes can be skipped.

`int *skipinds`: pointer to the array of node indices to skip.

`int Nnodes`: number of active nodes, Nnodes = Ntotal-Nskip.

`int Nexps`: number of exponentials used in fitting.

`MemState memstate`: struct containing previous evolution information.

`double Rejptilde0`: normalized critical current $\mathrm{Re}\,\tilde{j}_p(0)$.

`double alphaN`: damping due to the normal resistance α_N.

`double *jbar`: pointer to the reduced current density array $\bar{j}(\mathbf{r}, t)$.

`void *self` is used internally to access private member variables by the MiTMoJCo methods and is not intended for a regular user.

`bool error`: handler to check that no errors occurred during the object creation (`false` if no errors occurred an `true` is error occurred such as e.g. missing amplitudes file or incorrect file format).

Object member variables are accessed via the arrow operator `->`. For example, to access value of the error flag use `tunnel_current->error`.

To illustrate a particular example, a code implementing 1D model long Josephson junction discretized into 100 nodes (98 physical and 2 shadow nodes to treat the boundary conditions), may start with

```
// Superconducting phase difference array (100 nodes)
double *phi = malloc( 100 * sizeof(double) );

// Indices of the shadow nodes
int *skipinds={0,99};

// Create tunnel current object
TunnelCurrentType *tunnel_current = mitmojco_create(
    "../amplitudes/NbNb_4K2_008.fit", 0.7, 3.3, 0.01,
    100, phi, 2, skipinds );

// Check that no errors occurred during the object creation
if( tunnel_current->error )
    return;
```

The first two calls define superconducting phase difference array `phi` and array of shadow nodes indices `skipinds`. Then, the tunnel current object is created with parameters: tunnel current amplitudes from file `NbNb_4K2_008.fit` will be used, pair current suppression $\alpha_{\text{supp}} = 0.7$, normalized gap frequency $k = 3.3$ and time step 0.01. Two shadow nodes with indices 0 and 99 will be skipped in the evaluation of the tunnel current.

Upon creating the tunnel current object the three methods available to the user are, as declared in the header file,

```
extern void mitmojco_init( TunnelCurrentType *object );
extern void mitmojco_update( TunnelCurrentType *object );
extern void mitmojco_free( TunnelCurrentType *object );
```

Below we will discuss each of these methods in more detail.

Method `mitmojco_init` initializes the state of the memory integral, assuming no dynamics in the past (that is, the supplied state is assumed to be stationary in which the system existed for an infinite time). Use

```
mitmojco_init( tunnel_current );
```

to initialize the tunnel current object based on the initial value of `phi`. Note that that `mitmojco_init` should be called after the array `phi` is initialized as its values will be accessed by the address supplied to the object constructor.

Method `mitmojco_update` updates values of the tunnel current at the physical nodes, based on the updated values of the superconducting phase difference `phi` obtained within the numerical scheme (recall that the addresses of the physical nodes in `phi` are already known to MiTMoJCo at the constructor call during the object creation). Call

```
mitmojco_update( tunnel_current );
```

to calculate values of the reduced tunnel current density $\bar{j}(\mathbf{r}, t)$. These can be later accessed via the object pointer, `tunnel_current->jbar`.

Finally, `mitmojco_free` is used to empty the memory allocated for the tunnel current object,

```
mitmojco_free( tunnel_current );
```

It is possible to deal with several tunnel current objects simultaneously, e.g. FFO and a SIS, or, an array of Josephson junctions by creating independent object. As an example,

```
double phi_sis;

TunnelCurrentType *sis_tunnel_current = mitmojco_create(
    "../amplitudes/NbNb_4K2_008.fit", 0.7, 4.1, dt,
    1, &phi_sis, 0, NULL );

double *phi_ffo = malloc( 1000 * sizeof(double) );
int *skipinds_ffo={0,999};

TunnelCurrentType *ffo_tunnel_current = mitmojco_create(
    "../amplitudes/NbNb_4K2_016.fit", 0.8, 3.3, dt,
    1000, phi_ffo, 2, skipinds_ffo );
```

creates independent objects for a flux flow oscillator and SIS junction, each with its own parameters and different tunnel current amplitudes.

3.4 MiTMoJCo Python module for creation of tunnel current amplitudes

MiTMoJCo is provided with an easy-to-use module `mitmojco` written in Python 3 for creation of custom fits of TCAs at arbitrary temperature, superconducting gaps of

the materials and degree of Riedel peak smoothing. The supplied Jupyter notebook `amplitudes.ipynb` illustrates the use of `mitmojco` module by creating a fit of TCAs.

The work with `mitmojco` starts with the import statement

```
import mitmojco
```

upon which the following functions of the `mitmojco` module become available:

```
tca_bcs(T, Delta1, Delta2)
tca_smbcs(T, Delta1, Delta2, dsm)
new_fit(x, Jpair_data, Jqp_data, maxNterms, thr)
parsave(pAB,filename)
load_fit_parameters(filename)
tca_fit(pAB)
load_fit(x, filename)
```

Function `tca_bcs` returns bare (without smoothing) BCS TCAs evaluated from the Larkin and Ovchinnikov expressions [7] which are summarized in Ref. [1]; `tca_smbcs` returns smoothed TCAs obtained by smoothing bare BCS TCAs using the smoothing procedure of Ref. [8]; `new_fit` is used to calculate fits of exact TCAs and export them as a `.fit` file in the format suitable for the use by MiTMoJCo C library; `parsave` is used to save fit parameters to file; `load_fit_parameters` loads fit parameters from file; `tca_fit` returns TCA functions for the specified fit parameters; `load_fit` loads fit parameters from file and evaluates TCA functions at the frequencies `x`.

3.5 Pre-calculated Tunnel Current Amplitudes

The current amplitude files have extension `.fit`. User is supplied with a set of tunnel current amplitudes which are pre-calculated for a symmetric junction with $\Delta_1 = \Delta_2 = 1.4$ meV at temperature T=4.2 K and different degrees of smoothing, see Table 1. In the calculations of the current voltage characteristics (IVC) of Josephson flux flow oscillator in Ref. [1] value $\delta = 0.008$ was used as it fitted best the experimental IVC of a small $Nb/AlO_x/Nb$ junction. Note, that the tunnel current amplitudes are supplied without the account of the pair suppression $\alpha_{\rm supp}$ which is controlled separately within the function call (see the subsection 3.3 below).

The fits are found by calculating parameters which minimize the cost function

$$\sum_X \int_0^2 D(X^{\rm fit}, X^{\rm exact})^2 \, d\xi, \tag{4}$$

where

$$D(X^{\rm fit}, X^{\rm exact}) \equiv \frac{|X^{\rm fit} - X^{\rm exact}|}{\max(\tau_a/\tau_r, |X^{\rm exact}|)}, \tag{5}$$

is the relative difference between the fitted and exact functions $X = \mathrm{Re}\,\tilde{j}_p(\xi)$, $\mathrm{Im}\,\tilde{j}_p(\xi)$, $\mathrm{Re}\,\tilde{j}_{qp}(\xi)$, $\mathrm{Im}\,\tilde{j}_{qp}(\xi)$, and $\tau_{a,r}$ are absolute and relative tolerances, respectively. Three

of the fits `NbNb_4K2_001.fit`, `NbNb_4K2_008.fit` and `NbNb_4K2_064.fit` are shown in Fig. 1 along with the exact theoretical results and their relative difference defined by (5) ($\tau_a/\tau_r = 0.2$ in calculating the fits). The plots and the error information about the other fits from the Table 1 can be found in the `amplitudes\` folder.

Filename	T(K)	Δ_1(meV)	Δ_2(meV)	δ	N	τ_r	τ_a
`NbNb_4K2_001.fit`	4.2	1.40	1.40	0.001	10	0.005	0.001
`NbNb_4K2_002.fit`	4.2	1.40	1.40	0.002	9	0.005	0.001
`NbNb_4K2_004.fit`	4.2	1.40	1.40	0.004	9	0.004	0.0008
`NbNb_4K2_008.fit`	4.2	1.40	1.40	0.008	8	0.005	0.001
`NbNb_4K2_016.fit`	4.2	1.40	1.40	0.016	8	0.005	0.001
`NbNb_4K2_032.fit`	4.2	1.40	1.40	0.032	8	0.004	0.0008
`NbNb_4K2_064.fit`	4.2	1.40	1.40	0.064	8	0.005	0.001
`NbNbN_4K2_008.fit`	4.2	1.40	2.30	0.008	8	0.010	0.002
`NbNbN_4K2_015.fit`	4.2	1.40	2.30	0.015	8	0.004	0.0008

Table 1: Library of pre-calculated fits of tunnel current amplitudes (TCAs) supplied with MiTMoJCo. Tunnel current amplitudes are calculated from the BCS theory for Nb-AlO_x-Nb and Nb-AlN-NbN junctions and smoothed using different values of the phenomenological smoothing parameter δ. Number of the fitting exponentials N, relative and absolute tolerances of the fit in the frequency region $|\xi| \leq 2$ are also shown.

For historical reasons we also provide fits given in the Refs. [4] and [5]. However, due to their extremely bad performance in the subgap region, we discourage from using them for other purposes than debugging your code and/or reproducing results of Refs. [4] and [5, 6].

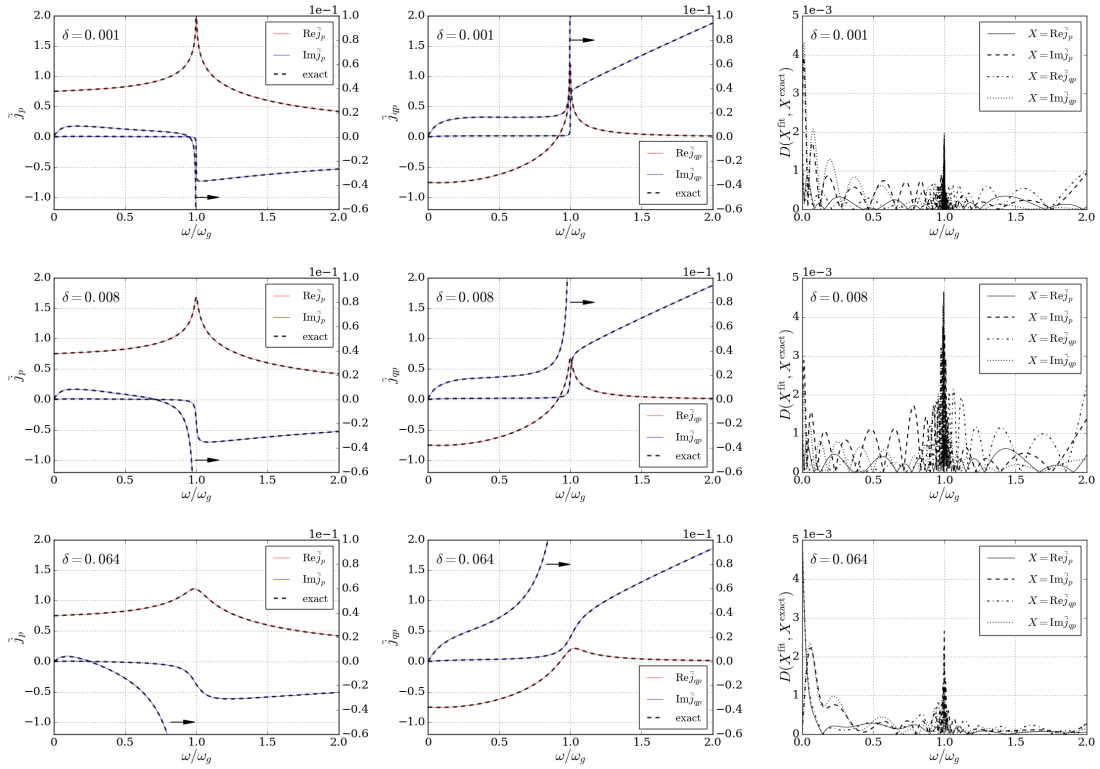

Figure 1: Fits of tunnel current amplitudes: `NbNb_4K2_001.fit` (top), `NbNb_4K2_008.fit` (middle) and `NbNb_4K2_064.fit` (bottom). Solid lines are the fit and dashed lines are the exact tunnel current amplitudes calculated from the BCS. To illustrate the behaviour of the tunnel current amplitudes in the subgap region, 20x zoom of the imaginary parts of the tunnel current amplitudes is shown. Relative difference of the fitted and exact amplitudes defined by Eq. (5) is shown on the rightmost figures.

Filename	N	Source
`OSZ_Table_1.fit`	4	Table 1 in Ref. [4]
`OSZ_Table_2.fit`	5	Table 2 in Ref. [4]
`GJHS_Table_1.fit`	4	Table 1 in Ref. [5]
`GJHS_Table_2.fit`	5	Table 2 in Ref. [5]

Table 2: Fits of tunnel current amplitudes of Refs. [4] and [5] . The fits are given for historical reasons only and should not be used for production unless reproducing the results of Refs. [4] and [5, 6] is your direct purpose. In all other cases use the tunnel current amplitude files listed in the Table 1

3.6 Optimum filtration of a sinusoidal signal

Optimum filtration routine for efficient calculation of a constant component of a sinusoidal signal is implemented in a supplementary code `opt_filter.c` and `opt_filter.h`. The code implements the algorithm outlined in Ref. [4]. To access the optimum filtration routine include the header file,

```
#include <mitmojco/opt_filter.h>
```

and begin with declaring the filter object, for example,

```
OptFilterType *voltage_filter = opt_filter_create(5);
```

where the argument of the constructor controls shape of the filtering window function: parameter n in Ref. [4]. n is integer and, in practice, takes values between 1 and 5, where 1 correspond to the arithmetic mean of the recorded values, see Ref. [4] for details. The type `OptFilterType` which emulates the class is defined in `opt_filter.h` as a structure,

```
typedef struct {
    int n;
    double a;
    double *y;
    void *self; // pointer to private struct
} OptFilterType;
```

The four methods accessible by the user are

```
extern void opt_filter_init( OptFilterType *object );
extern void opt_filter_update( OptFilterType *object, double signal );
extern double opt_filter_result( const OptFilterType *object );
extern void opt_filter_free( OptFilterType *object );
```

Method `opt_filter_init` initializes the filter object,

```
opt_filter_init(voltage_filter);
```

Method `opt_filter_update` makes a record of the signal value,

```
opt_filter_update(voltage_filter, voltage );
```

Method `opt_filter_result` returns value of the calculated dc component once the calculation is finished,

```
Vdc = opt_filter_result(voltage_filter);
```

Finally, `opt_filter_free` is used to clear the memory allocated to the filter object,

```
opt_filter_free(voltage_filter);
```

4 Examples

Examples of how MiTMoJCo can be used in modeling several common cases of Josephson junctions are provided.

To compile a particular example, type

```
$ make example-#
```

where `#` is the example number, or, type

```
$ make all
```

or, simply,

```
$ make
```

to compile all the provided examples.

4.1 Example 1: Current-biased SIS junction

Current-biased Josephson junction is described by

$$\ddot{\varphi} + \alpha_N \dot{\varphi} + \bar{j}(t) - \gamma = 0,$$

$$\bar{j}(t) = \frac{k}{\operatorname{Re} \tilde{j}_p(0)} \int_0^\infty \left\{ j_p(kt') \sin\left[\frac{\varphi(t) + \varphi(t-t')}{2}\right] + \bar{j}_{qp}(kt') \sin\left[\frac{\varphi(t) - \varphi(t-t')}{2}\right] \right\} dt',$$

where γ is the applied biasing current. To compile the first example, type in the terminal

```
$ make example-1
```

which produces executable `example-1` in the current directory. Executing

```
$ ./example-1
```

without the command line arguments produces an output

```
#================================================
#---- Example 1: Current-biased SIS Junction -----
#================================================
# Incorrect number of arguments.
# Please, supply 1 or 3 arguments in the following order:
# 1. Value of bias current (gamma_start)
# or
# 1. Starting value of bias current (gamma_start)
# 2. Final value of bias current (gamma_finish)
# 3. Bias current step (gamma_step)
```

Run

```
$ ./example-1 1.1
```

to calculate the normalized dc voltage at a specified bias current (e.g., $\gamma = 1.1$ in this example), or,

```
$ ./example-1 1.1 0.0 0.01
```

for a range of values. In this case, γ takes 1.1 as initial value (just above the critical current) and decreases down to 0.0 with step 0.01.

4.2 Example 2: Voltage-biased SIS junction under ac drive

Current through a small voltage-biased Josephson junction is given by the Eq. (1). In this case, one does not need to solve the differential equation, rather, the superconducting phase difference $\varphi(t)$ can be easily found from the fundamental Josephson relation if the time-dependence of the applied voltage is known.

All one needs to do in MiTMoJCo is to update φ and call

```
mitmojco_update( sis_tunnel_current );
```

at each time step to get the reduced tunnel current. Adding the normal resistance part $\alpha_N \dot{\varphi}$ will get us the full tunnel current.

In this example we assume a harmonic ac drive

$$V(t) = V_{dc} + V_{ac} \cos(\omega t)$$

and use the optimum filtration routine to get the resulting dc current component. Compile the code with

```
$ make example-2
```

and run

```
$ ./example-2
```

to see the command line arguments needed to run the simulation,

```
#================================================
#---- Example 2: Voltage-biased SIS Junction -----
#================================================
# Incorrect number of arguments.
# Please, supply 5 arguments in the following order:
# 1. Vac
# 2. omega
# 3. Vdc_start
# 4. Vdc_finish
# 5. Vdc_step
```

The arguments should be supplied in normalized units (units of $\hbar\omega_J/e$ for voltage and units of ω_J for the angular frequency). As an example, the command

```
$ ./example-2 1.5 1.0 0.0 6.0 0.02
```

calculates the SIS IVC for $V_{ac} = 1.5$, $\omega = 1.0$ in normalized units. The resulting SIS IVC obtained for different values of the driving frequency ω is shown in Fig. 3. In the simplest case of a harmonic drive discussed here, MiTMoJCo results coincide with those given by the SIS mixer theory where the explicit expressions in terms of the Bessel functions are known [9].

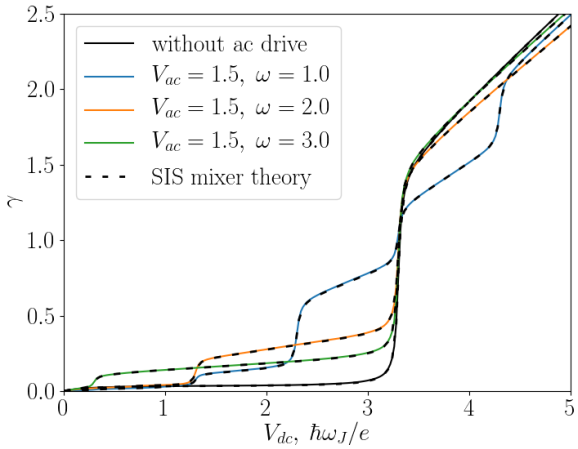

Figure 2: IVC of voltage-biased SIS junction under ac drive calculated by MiTMoJCo for in presence and absence of the ac drive. Parameters of the calculation are: tunnel current amplitudes file `NbNb_4K2_008.fit`, pair current suppression $\alpha_{\text{supp}} = 0.7$, normalized gap frequency $k = 3.3$. For verification of MiTMoJCo the known theoretical result from the SIS mixed theory [9] is also shown.

4.3 Example 3: Sine-Gordon breather in long Josephson junction

In this example we consider a 1D model of a long Josephson junction,

$$\varphi_{tt} - \left(1 + \beta\frac{\partial}{\partial t}\right)\varphi_{xx} + \alpha_N\varphi_t + \bar{j}(x,t) = 0$$

with open boundary conditions,

$$\varphi_x(\pm L/2, t) = 0.$$

Running the executable

```
$ ./example-3
```

produces the file `breather.dat` where dynamics of the superconducting phase difference is written during the time evolution of the breather. Use the supplied Python routine `breather.py` to animate. Start the Python 3 interactive mode,

```
$ ipython
```

Within the Python interactive mode execute

```
In [1]: run breather
```

to see animation of the breather dynamics.

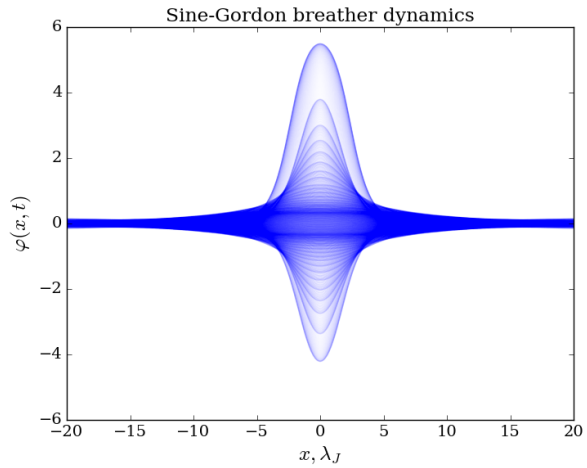

Figure 3: Time evolution of sine-Gordon breather. A large number of time frames were super-imposed to illustrate dynamics of the superconducting phase difference. Parameters used in the calculation are: Josephson junction normalized length $L = 40$, tunnel current amplitudes file `NbNb_4K2_008.fit`, pair current suppression $\alpha_{\text{supp}} = 0.7$, normalized gap frequency $k = 3.3$.

4.4 Example 4: Fluxon in an Annular Josephson junction

Compile

```
$ make example-4
```

running the executable

```
$ ./example-4
```

displays the information about the input parameters,

```
#=========================================================
#---- Example 4: Fluxon in Annular Josephson Junction ----
#=========================================================
# Incorrect number of arguments.
# Please, supply 1 or 3 arguments in the following order:
# 1. gamma_start
# 2. gamma_finish
# 3. gamma_step
```

Running the executable with the parameters

```
$ ./example-4 0.0 0.05 0.001
```

executes the calculation of the current-velocity curve for a fluxon displayed in Fig. 4. Note that the slope of the curve is larger for small velocities compared and than for moderate velocities. This is in agreement with the theoretical result of Ref. [6] (cf. their Fig.3).

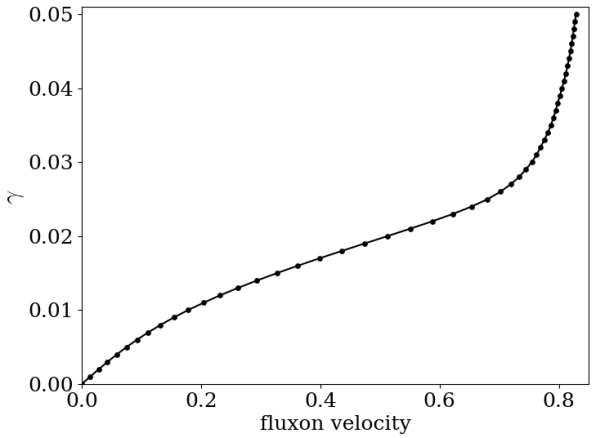

Figure 4: Current-velocity dependence for a single fluxon trapped in an annular Josephson junction of length $L = 40$. The parameters used in the calculations are: tunnel current amplitudes file `NbNb_4K2_008.fit`, pair current suppression $\alpha_{\text{supp}} = 0.7$, normalized gap frequency $k = 3.3$.

4.5 Example 5: Flux Flow Oscillator

Example-5 is based on the model of a flux flow oscillator introduced in Ref. [1]. For simplicity, the model without coupling to load is presented here.

As with the previous examples, use

```
$ make example-5
```

to compile the code

```
$ ./example-5
```

to display the info,

```
#================================================
#------- Example 5: Flux Flow Oscillator -------
#================================================
# Incorrect number of arguments.
# Please, supply 2 or 4 arguments in the following order:
# 1. Magnetic field (hext)
# 2. Value of bias current (gamma_start)
# or
# 1. Magnetic field (hext)
# 2. Starting value of bias current (gamma_start)
# 3. Final value of bias current (gamma_finish)
# 4. Bias current step (gamma_step)
```

Executing

```
$ ./example-5 3.5 0.0 0.2 0.002
```

starts the calculation of FFO IVC at a fixed magnetic field 3.5 and values of γ varied in the range from 0.0 to 0.2 with step 0.002. The typical output of the program run for FFO IVC calculation is as follows,

```
#================================================
#------- Example 5: Flux Flow Oscillator -------
#================================================
# Magnetic field (hext): 3.5000
# Bias current (gamma): (0.0000, 0.2000, 0.0020)
# OpenMP threads: 4
# Tapered ends FFO geometry:
#  L0: 400.000 micron
#  W0: 16.000 micron
#  S0: 40.000 micron
#  Wend: 1.000 micron
#  Josephson penetration (LAMBDA_J): 5.50 micron
# Surface damping (BETA): 0.0200
# Settling time (SETTLING_TIME): 200.00
# Integration time (TMAX): 1000.00
```

```
# FFO area: 191.74 LAMBDA_J^2
#------------------- MiTMoJCo --------------------
# Amplitudes loaded from file "amplitudes/NbNb_4K2_008.fit"
# MiTMoJCo started with parameters:
#  Normalized gap frequency (kgap): 3.300
#  Pair current suppression (a_supp): 0.70
#  Rejp~(0): 0.52799
#  alphaN: 0.28697
#  Number of fitting exponents (Nexps): 8
# Number of physical nodes (Nnodes): 800
# dx: 0.09091
# dt: 0.04545
#----------- Starting the calculation ------------
# Column 1: gamma
# Column 2: Vdc
0.0000 -0.000000
0.0020 0.014012
0.0040 0.029413
....
0.1420 1.725801
0.1440 1.733849
0.1460 1.742871
0.1480 3.204688
#-------------------------------------------------
# CPU time:        565.274841
# Wall clock time: 141.343814
```

As in the previous examples, the first part of the output displays information about the model: magnetic field, bias current, geometrical parameters, surface damping and numerical scheme details. The second part is the MiTMoJCo internal output when the starting routine is invoked: normalized gap frequency k (`KGAP`), pair current suppression α_{supp} (`A_SUPP`), normalized critical current ($\mathrm{Re}\,\tilde{j}_p(0)$), damping due to a normal resistance (α_N) and number of fitting exponents (`Nexps`). Finally, the calculation output is started. The output can be fed to gnuplot or Python's `numpy.loadtxt` routine for processing (the both treat the rows starting with a hash symbol `#` as a commentary). The program terminates as soon as the voltage jump to the gap region is detected.

4.6 Example 6: Modeling 2D Josephson junctions with deal.II+MiTMoJCo

In combination with `deal.II` the finite element library [2, 3] MiTMoJCo becomes a powerful tool for modeling of realistic 2D Josephson junctions. MiTMoJCo contains an example of modeling T-junction Terahertz chaotic oscillator [10] using the `deal.II` library implemented in C++. The code presented in folder `examples/dealii-mitmojco`

is tested with deal.II v8.5 [11].

5 Disclaimer

MiTMoJCo comes without any warranty nor guarantees to produce correct results. The author accepts no responsibility whatsoever for source code bugs, failures, crashes, expected or unexpected behavior.

This is user's responsibility to ensure that the time step `dt` and the spatial discretization `dx` suits his needs. In the examples 3-5 the ratio `dt/dx` is controlled by a parameter `DTREL` which we set to 0.5. We recommend not to exceed this value if the semi-implicit scheme like the one presented here is used as the larger values of `DTREL` may lead to instability.

In the example 4, when the current is applied fluxon shrinks in size due to the Lorentz contraction. Ensure that spatial discretization is sufficient to resolve the contracted fluxon. Ignoring this may lead to unphysical results as soon as the fluxon size becomes of the order of `dx`.

References

[1] D. R. Gulevich, V. P. Koshelets, and F. V. Kusmartsev, Phys. Rev. B 96, 024215 (2017).

[2] W. Bangerth, R. Hartmann and G. Kanschat, "deal.II — a general-purpose object-oriented finite element library", ACM Trans. Math. Softw. 33, 24 (2007).

[3] Open source finite element library `deal.II` http://www.dealii.org.

[4] A. A. Odintsov, V. K. Semenov and A. B. Zorin, IEEE Trans. Magn. 23, 763 (1987).

[5] N. Grønbech-Jensen, S. A. Hattel and M. R. Samuelsen, Phys. Rev. B 45, 12457 (1992).

[6] S. A. Hattel, N. Grønbech-Jensen and M. R. Samuelsen, Phys. Lett. A 178, 150 (1993).

[7] A. I. Larkin and Yu. N. Ovchinnikov, Sov. Phys. JETP 24, 1035 (1967).

[8] A. B. Zorin, I. O. Kulik, K. K. Likharev and J. R. Schrieffer, Sov. J. Low Temp. Phys. 5, 537 (1979).

[9] J. R. Tucker and M. J. Feldman, Rev. Mod. Phys. 57, 1055 (1985).

[10] D. R. Gulevich, V. P. Koshelets, and F. V. Kusmartsev, arXiv:1709.04052.

[11] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells, J. Numer. Math. 25, 137 (2017).