A set of MATLAB-CUDA programs to find low-energy defibrillating signals by adjoint optimization

Alejandro Garzón

April 14, 2025

Abstract

This document explains the compilation and operation of a set of MATLAB-CUDA programs for finding low-energy defibrillating signals in electrophysiological models. The programs implement the procedure explained in Refs. [1, 2], based on the minimization of a functional of the signal, defined over the model solution. The optimization is performed by the gradient descent approach, with the functional gradient computed efficiently by the adjoint method.

1 Method description

References [1, 2] present a method for finding low-energy electric field signals $E(t)$ capable of producing defibrillation in electrophysiological models of two-dimensional cardiac tissue. The method is based on the minimization of a functional $\mathcal{L}$ of $E(t)$,

$$\mathcal{L} = \frac{1}{2}\mathcal{M} + \frac{\alpha}{2}\mathcal{N}, \tag{1}$$

that penalizes the energy $\mathcal{N}$ of $E(t)$ and the spatial variation $\mathcal{M}$ of the model state variables at a given time T. $\mathcal{L}[E(t)]$ is minimized by the gradient descent approach with the functional gradient $\mathcal{G}(t)$ computed efficiently by the adjoint method. To explain the installation and operation of the software that computes the optimal $E(t)$, a sketchy presentation of the method equations follows (the full details can be found in Ref. [1]).

After space discretization, the model's partial differential equations become a system of ordinary differential equations (ODEs), with a huge number of state variables, in the order of tens of thousands. If the state variables are gathered in a vector $\mathbf{w}$, the ODE system takes the form

$$\dot{\mathbf{w}} = L\mathbf{w} + F(\mathbf{w}) + E(t)\mathbf{b}. \tag{2}$$

To compute the solution $\mathbf{w}(t)$ quickly, the numerical algorithms were programmed in CUDA C++ language for parallel execution on general-purpose graphics processing units (GPUs). For ease of use, the CUDA program was then wrapped in a MATLAB mex-function.

The functional gradient $\mathcal{G}(t)$ is computed as

$$\mathcal{G}(t) = \alpha E(t) + \mathbf{b}^{\mathsf{T}}\boldsymbol{\lambda}(t), \tag{3}$$

where $\boldsymbol{\lambda}$ is a Lagrange multiplier that obeys the *adjoint equations*

$$-\dot{\boldsymbol{\lambda}} = (L + J_F)^{\mathsf{T}}\boldsymbol{\lambda}, \tag{4a}$$

$$\boldsymbol{\lambda}(T) = R\mathbf{w}(T). \tag{4b}$$

The ODE system (4a) has as many variables as the system (2). Hence, for efficiency, the numerical solution of (4a) was programmed in CUDA C++ language and wrapped in a MATLAB mex-function too.

Given a signal $E_s(t)$, the gradient descent method utilizes the functional gradient to produce an improved signal $E_{s+\Delta s}(t)$ using

$$E_{s+\Delta s}(t) = E_s(t) - \Delta s\, \mathcal{G}(t)|_{E_s(t)}. \tag{5}$$

More exactly, for small enough Δs, the new signal $E_{s+\Delta s}(t)$ reduces $\mathcal{L}$,

$$\mathcal{L}[E_{s+\Delta s}(t)] < \mathcal{L}[E_s(t)].$$

Therefore, iteration of (5) generates a sequence of signals $\{E_s(t), E_{s'}(t), E_{s''}(t), \ldots\}$ that converges to one of many local minima of $\mathcal{L}$. Some of such minima correspond to low-energy defibrillating signals $E(t)$.

The next section describes the installation and operation of the programs that solve Eq. (2) and perform the gradient descent iteration (5).

2 Software compilation and operation

CORRECT THIS: The following instructions are to be executed in a Linux operating system. They include Linux commands, identified with the prompt `$`, and MATLAB commands, starting with `>>`. Copy the file `adjoint_optimization.tar.gz` to a directory in your machine where you would like to keep the programs. Expand the file with

```
$ tar -xzvf adjoint_optimization.tar.gz
```

This creates the directory `adjoint_optimization` containing the MATLAB scripts (extension `.m`) and the source code of the mex-functions.

2.1 Forward evolution equation

The script `save_fk_forward_Et.m` computes the solution $\mathbf{w}(t)$ of the forward evolution equation (2) and saves it to a file. This program calls the mex-function `fk_forward_Et_mex`, which must be generated by compilation of the source files, using

```
>>  mexcuda -output fk_forward_Et_mex fk_forward_Et_param_mex.cpp ↘
→fk_forward_Et_mono_PARAMETERS.cu
```

The function `mexcuda` executes the CUDA compiler `nvcc` from either the CUDA toolkit installed with MATLAB or a system-wide CUDA toolkit.

`save_fk_forward_Et.m` reads input data from the file `infile_forward`. An example of this file in displayed in Listing 1. It contains MATLAB-style assignments and accepts MATLAB-style comments, starting with `%`. Table 1 explains the meaning of the input variables. The example input files referred to in Listing 1 are provided in directory `adjoint_optimization/data`.

Listing 1: Input file `infile_forward` for `save_fk_forward_Et.m`

```
Et_file = 'Et_pulse_N_1_E0_5.00_t0_20.0_300ms.mat';
final_time = 200; % ignored, unless Et_file is the empty string
nstep = 100;
icondfile = 'spirals.mat';
outfile = ''; % make empty to enable use of outfile_prefix
outfile_prefix = 'ys_mex_';
mex_function = @fk_forward_Et_mex;
grid_file = 'lap_Cfile_2nd_order_grad_dx_0p035_N_256_hole_dens_16.mat';
in_path = './data/';
out_path = './data/';
```

The script is executed by simply typing its name (without the extension) at the MATLAB prompt

```
>> save_fk_forward_Et
```

which, for the given input variables, creates the output file `ys_mex_Et_pulse_N_1_E0_5.00_t0_20.0_300ms.mat` containing the solution $\mathbf{w}(t)$. This solution is plotted by the script `plot_ys.m`, described in Sec. 2.3.

2.2 Gradient descent

The script `simple_grad_desc.m` performs the gradient descent iteration (5) to generate the sequence of electric field signals $\{E_s(t), E_{s'}(t), E_{s''}(t), \ldots\}$. To compute the functional gradient, the program calls the mex-function `fk_grad_Et_mex`, which solves the systems of equations (2) and (4a). This mex-function is generated by compilation of the source files using

Variable	Description
`Et_file`	File with $E(t)$ in dV/cm. The script also reads from this file the final time T. If set to the empty string, the script assumes $E(t) = 0$.
`final_time`	Final time T in milliseconds. Ignored, unless `Et_file` is the empty string.
`nstep`	Number of time steps at which $\mathbf{w}(t)$ is saved.
`icondfile`	File with initial tissue state $\mathbf{w}_0$, $\mathbf{w}(0) = \mathbf{w}_0$.
`outfile`	File where $\mathbf{w}(t)$ will be saved. Set to the empty string to enable the use of `outfile_prefix`.
`outfile_prefix`	If `outfile` is the empty string, the name of the file where $\mathbf{w}(t)$ will be saved is composed as the concatenation of `outfile_prefix` and `Et_file`.
`mex_function`	mex-function that computes $\mathbf{w}(t)$.
`grid_file`	File with distribution of non-conducting patches in the tissue.
`in_path`	Directory where input files are read from.
`out_path`	Directory where output files are saved to.

Table 1: Meaning of variables in Listing 1.

```
>> mexcuda -output fk_grad_Et_mex fk_grad_Et_param_mex.cpp ↘
→fk_grad_Et_mono_PARAMETERS.cu
```

Every time a batch of certain number of iterations (set through the input variable `nmod`) is completed, the program saves to a file the values of $\mathcal{L}$, $\mathcal{N}$, and $\mathcal{M}$ for each iteration in the batch and $E_s(t)$ for the final iteration. The output filename is created as a concatenation of the following: (i) the string `simple_grad_desc_step_`, (ii) a string with the date at execution time, (iii) the string in the input variable `jobid` (which could be the job ID, if the script is executed in batch mode), (iv) the number of gradient descent iterations so far, and (v) the extension `.mat`. For instance `simple_grad_desc_step_2024-nov-02-17-32-50_000001_50.mat`.

`simple_grad_desc.m` reads input data from the file `infile_grad`. An example of this file in displayed in Listing 2. Table 2 explains the meaning of the input variables.

Listing 2: Example input file `infile_grad` for `simple_grad_desc.m`

```
alpha=0.5; % weight of electric field integral
ds=1e-3; % pseudo-time step duration
ns=200; % total number of pseudo-time steps
final_time=300;
gamma = [1, 0.2, 0.2]; % weights of gradients of variables
nmod = 50;
Et_seed_file = 'Et_pulse_N_1_E0_5.00_t0_20.0_300ms.mat';
icondfile = 'spirals.mat';
grid_file = 'lap_Cfile_2nd_order_grad_dx_0p035_N_256_hole_dens_16.mat';
S_file = 'S_1o4_N_256_hole_dens_16.mat';
mex_function = @fk_grad_Et_mex;
in_path = './data/';
out_path = './data/';
jobid = '000001';
```

The script is executed by typing its name at the MATLAB prompt,

```
>> simple_grad_desc
```

The script `plot_LMN.m`, described in Sec. 2.3, plots the data of some example output files.

Variable	Description
`alpha`	Value of parameter α in Eq. (1) in units of $\mathrm{cm}^2/(\mathrm{dV}^2\,\mathrm{ms})$
`ds`	Value of Δs in Eq. (5) in units of $(\mathrm{dV/cm})^2$
`ns`	Number of iterations of the recurrence relation in Eq. (5)
`final_time`	Final time T in milliseconds.
`gamma`	Vector $(\gamma_1, \gamma_2, \gamma_3)$ used in the computation of $\mathcal{M}$ in Eq. (1)
`nmod`	Number of iterations of Eq. (5) at which $\mathcal{L}$,$\mathcal{N}$,$\mathcal{M}$ and $E_s(t)$ are saved to a file
`Et_seed_file`	File with initial electric field signal $E_0(t)$ in dV/cm.
`icondfile`	File with initial tissue state $\mathbf{w}_0$, $\mathbf{w}(0) = \mathbf{w}_0$.
`grid_file`	File with distribution of non-conducting patches in the tissue.
`S_file`	File with matrices used in the computation of $\mathcal{M}$.
`mex_function`	mex-function that computes the functional gradient $\mathcal{G}(t)$.
`in_path`	Directory where input files are read from.
`out_path`	Directory where output files are saved to.
`jobid`	String included in the name of the output files.

Table 2: Meaning of the variables in Listing 2.

2.3 Plotting scripts

The following scripts plot the results of `save_fk_forward_Et.m` and `simple_grad_desc.m`.

- `plot_ys.m`: reads $\mathbf{w}(t)$ from a file written by `save_fk_forward_Et.m` and plots the voltage variable on the two-dimensional domain for several times. The file read by the script must be created by execution of `save_fk_forward_Et.m` with the provided input data.
- `plot_LMN.m`: reads a group of files written by `simple_grad_desc.m` and plots $\mathcal{L}$,$\mathcal{N}$ and $\mathcal{M}$ as functions of s and the final electric field signal $E_s(t)$. The example files read by the script are provided in directory `adjoint_optimization/data`.

3 Test

This software has been tested with MATLAB Release R2021a using the `nvcc` compiler installed with MATLAB, on an NVIDIA Tesla V100 PCIe 16 GB GPU.

References

[1] Alejandro Garzon and Roman O Grigoriev. Ultra-low-energy defibrillation through adjoint optimization. *arXiv preprint arXiv:2407.05115*, 2024. URL: https://arxiv.org/abs/2407.05115.

[2] Alejandro Garzón and Roman O Grigoriev. Ultra-low-energy defibrillation through adjoint optimization. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 34(11):113110, 2024. doi:10.1063/5.0222247.