Kristofer Björnson

Tight-Binding ToolKit

Quick start

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide essential information needed to get started with TBTK.

1.2 Audience

The target audience is physicists interested in using TBTK to perform numerical calculations involving bilinear Hamiltonians. While TBTK is designed to be handled from the terminal, prior experience with the terminal is not assumed.

1.3 Overview

1.3.1 Working in the terminal

To enable users without experience of working in the terminal, the most important terminal commands are introduced in chapter 2. The experienced user can skip this chapter.

1.3.2 TBTK

Chapter 3 begins with a section listing prerequisites. Most supercomputers should come configured such that the required prerequisits are satisfied by default, but configuration may be necessary, especially if working on a personal computer. Users with proper system administration experience can use this section to ensure that their system is correctly configured before installing TBTK. If the user does not feel confident setting this up, or lack the adequate system administration rights, the section is instead intended to provide enough information to allow for effective communication with the system administrator.

Once the prerequisites are satisfied, information regarding how to install and update TBTK is provided, as well as information regarding initialization required each time a terminal window is opened. The chapter ends with a section that describes how to create a new project.

1.4 Feedback

Feedback on this document is very appreciated, so do not hesitate to send an email to **kristofer.bjornson@physics.uu.se** if anything is unclear, missing, not working, etc. The code has so far only been tested on a few different types of machines, with a limited number of compiler versions etc. Information regarding the success or failure with using the code on machines with a different configuration is therefore greatly appreciated.

2. Working in the terminal

2.1 Basic terminal commands

Type	Command	Action
Symbol	**.**	Current directory
	..	Directory one level up in the directory tree
	~	Home directory
Command	**ls**	List files in current directory
	cd directory	Move into directory
	cd ..	Move up one level in the directory tree
	cp file newFile	Copy file
	cp -r folder newFolder	Copy a folder and its content
	rm file	Remove file
	mkdir newDirectory	Create directory
	rmdir directory	Remove empty directory
	rm -r directory	Remove directory and its content
	man mkdir	Manual page for a command (here mkdir)

2.2 Edit a text file

There are many different text editors, each with their own strengths and weaknesses. Each line below would open the text file in a different editor. Nano is a very simple to use editor, while vi, vim, and emacs have a lot more useful features, but can initially be very difficult to work with.

```
nano textfile
vi textfile
vim textfile
emacs textfile
```

2.3 Login to supercomputer (or other remote machine)

```
ssh username@supercomputer.location.se
password: ...
```

2.4 Copy files to and from supercomputer

Copy from personal computer to home folder on supercomputer

```
scp someFile username@supercomputer.location.se:~/some/path/
```

Copy from home folder on supercomputer to personal computer

```
scp username@supercomputer.location.se:~/some/path/ .
```

Note that ∼ is short hand for the home folder, and an absolute path can be specified by replacing ∼**/some/path/** with **/some/absolute/path/**. Similarly **.** refers to the current folder, and it is possible to copy to another folder by replacing **.** by **some/path/on/the/personal/computer**. Finally, it is possible to rename the file at the same time as it is copied by replacing ∼**/some/path/** with ∼**/some/path/newFileName**.

2.5 Packing and compressing files

Pack and compress **file1, file2, file3, folder1**, and **folder2** into **archiveName.tgz**

```
tar -cvzf archiveName.tgz file1 file2 folder1 file3 folder2
```

Uncompress and unpack

```
tar -xvzf archiveName.tgz
```

3. Setting up TBTK

3.1 Prerequisites

Summary of operating systems that TBTK has been verified to work with

Operating system	Distribution	version
Linux	Scientific Linux Ubuntu	? 16.04 LTS
Mac OS	?	?
Windows	?	?

Summary of libraries and softwares that are required to use TBTK, as well as optional libraries that allows for more features to be used.

Type	Name	Required	Used for
Library	**Blas**	**Yes**	Linear algebra
	Lapack	**Yes**	Linear algebra
	FFTW3	No	Fourier transform
	Arpack	No	ArnoldiSolver
	SuperLU v5.2.1	No	ArnoldiSolver
	OGRE	No	3D visulaization
	OpenCV	No	TBTKImageToModel
Software	**GCC** (min v4.9)	**Yes**	Compiling TBTK
	NVCC	No	Compiling with GPU support
	Python	No	Plotting

While **blas** and **lapack** usually are installed on supercomputers and are available by default, the other libraries are not necessarily so. Similarly, multiple versions of the **gcc** compiler can be expected to be installed on most supercomputers. However, to check which particular compiler that is used by default, type

```
gcc --version
```

If the version number is not 4.9 or higher, it is most often possible to switch the compiler by loading the appropriate **module**. Type

```
module avail
```

to see what modules are available on the system, and find the module name which indicates that it contains gcc v4.9. The exact name can differ from system to system, but assuming that the above command shows that a module with for example the name **gcc/4.9** is available, then load this module using

```
module load gcc/4.9
```

Verify that the correct compiler is loaded by once again typing

```
gcc --version
```

Similarly the CUDA compiler **nvcc** usually has to be loaded using a similar statement. Assume that the **module avail** command above also shows that a module named for example **cuda/v9.0** is available. Then type

```
module load cuda/v9.0
```

To check that the compiler is available, type

```
nvcc --version
```

3.2 Install TBTK

Download source code

```
git clone https://github.com/dafer45/TBTK
```

Enter TBTK folder

```
cd TBTK
```

Choose specific TBTK version (here v0.9). This is optional and should only be done if you know that you want to use a specific version.

```
git checkout tags/v0.9
```

Setup environment

```
source init_session.sh
```

Install TBTK. Execute the first line if the computer does not have CUDA support (GPU) in the form of a nvcc compiler. Execute the second if it has.

```
./install.sh
./install.sh -CUDA
```

Perform a basic test by compiling all the templates

```
./test.sh
```

3.3 Update TBTK

Setup environment. Only do this if this has not been done since the terminal window was opened.

```
source init_session.sh
```

Download the lates version.

```
git pull
```

Reinstall the updated library. Execute the first line if the computer does not have a CUDA support (GPU) in the form of a nvcc compiler. Execute the second if it has.

```
./update.sh
./update.sh -CUDA
```

3.4 Each time a new terminal window is opened

Whenever a new terminal window is opened. First make sure that the correct **gcc** compiler is loaded according to Section 3.1, and that if needed, so also **nvcc**. Nect, **cd** into the TBTK folder and execute

```
source init_session.sh
```

3.5 Setup a new project

It is recommended to create a separate folder called **TBTKProjects** and for each project create a subfolder according to

```
TBTKProjects/project1/
TBTKProjects/project2/
```

When starting a new project, it is recommended to copy one of the Template-projects in the directory **TBTK/Templates/** into **TBTKProjects/**, and to then modify the code of these templates. TBTK provides a command called **TBTKCreateProject** for simplifying this procedure. For example, assuming the current directory is the **TBTKProjects** folder, use the following line to create a new project that builds on top of the **BasicDiagonalization** template

```
TBTKCreateProject NewProject BasicDiagonalization
```

Enter into the new project folder

```
cd NewProject/
```

Modify the source code

```
nano src/main.cpp
```

Compile (create executeable file from the source code). The resulting binary will be located in **build/a.out**.

```
make
```

Run the program

```
./build/a.out
```