`tdasampling` User's Manual

Parker Edwards

March 8, 2018

Contents

This manual is for Linux distributions, which is the only OS for which `tdasampling` has been tested.

1 Installation

Installation is via pip. In this example, the current working directory is a virtual environment. It will install the scripts to the directory's bin folder rather than into the global bin folder.

```
$ pip install tdasampling
```

The pip command will install the Python dependencies automatically.

2 Installing requirements

```
$ /bin/bash
```

2.1 `libspatialindex` and `rtree`

2.1.1 Libspatialindex

If you have root access to the machine you're using, you can follow the standard installation instructions. First you need to install libspatialindex, with installation instructions at `https://libspatialindex.github.io/install.html`.

So, for instance, having changed your working directory to wherever you want to place the source files before installation:

```
$ curl http://download.osgeo.org/libspatialindex/spatialindex-src-1.8.5.tar.gz | tar xvz
$ cd spatialindex-src-1.8.5/
$ cmake .
$ make
$ sudo make install
```

If you don't have root access to the installation machine, you'll want to install libspatialindex into a non-standard location. For reasons that are currently unclear to me, Ctypes throws errors even if you install the libraries in a non-standard location. Instead, you can do the following. Note the lack of "make install".

```
$ curl http://download.osgeo.org/libspatialindex/spatialindex-src-1.8.5.tar.gz | tar xvz
$ cd spatialindex-src-1.8.5/
$ cmake -DCMAKE_INSTALL_PREFIX=/home/parker/test_dir/local .
$ make
```

2.1.2 Adjusting rtree if libspatialindex is installed without root access

The `Rtree` Python package provides a Python interface over `libspatialindex` using Ctypes. If you have installed the `libspatialindex` library non-globally (without root access), then it has a hard time locating the library files. The `Rtree` package provides an environment variable you can export in your shell to fix this, but you have to export it every time you want to use `tdasampling`. An example:

```
$ export SPATIALINDEX_C_LIBRARY=
"/home/parker/test_dir/spatialindex-src-1.8.5/bin/libspatialindex_c.so"
```

As an alternative, you can just hard code the non-standard location of your libspatialindex files via the following command. If you change the location of libspatial, you'll need to update it again. An example of the location of the source file is /home/parker/test_dir/lib/python2.7/site-packages/rtree/core.py. The line you need to change is line 122. It looks like this (with surrounding context):

```
if 'SPATIALINDEX_C_LIBRARY' in os.environ:
    lib_name = os.environ['SPATIALINDEX_C_LIBRARY']
else:
    lib_name = find_library('libspatialindex_c')
```

This should be changed to something that looks like this:

```
if 'SPATIALINDEX_C_LIBRARY' in os.environ:
    lib_name = os.environ['SPATIALINDEX_C_LIBRARY']
else:
    lib_name = '/home/parker/test_dir/spatialindex-src-1.8.5/bin/libspatialindex_c.so'
```

If you make this adjustment, you won't have to worry about exporting environmental variables every time you want to use `rtree`.

2.2 MPI

Follow the installation instructions for MPI at `http://www.mpich.org/downloads/` or another favorite MPI implementation.

2.3 Bertini

Bertini offers both binary and source downloads at `https://bertini.nd.edu/download.html`. If you were using a 64-bit Linux machine, you could download and unpack the binaries using:

```
$ curl https://bertini.nd.edu/BertiniLinux64_v1.5.1.tar.gz | tar xvz
```

Optionally, you can also add the binary folder to your PATH by modifying ~/.bashrc (replace ~/.bashrc with the appropriate file for your system). You can do the same for mpiexec.

3 Quick start tutorial

The quick start will be sampling a quartic variety in 3 variables and 1 equation:

$$4x^4 + 7y^4 + 3z^4 - 3 - 8x^3 + 2x^2y - 4x^2 - 8xy^2 - 5xy + 8x - 6y^3 + 8y^2 + 4y = 0.$$

It assumes you've gone through all the installation steps. For the sake of the example, we're using a virtualenv which is in our current working directory.

First step: Make a directory for the example.

```
$ mkdir quartic_example
```

Next we need to put a polynomial_system file in the directory specifying the system to be sampled. It looks like this:

```
x,y,z
4*x^4+7*y^4+3*z^4-3-8*x^3+2*x^2*y-4*x^2-8*x*y^2-5*x*y+8*x-6*y^3+8*y^2+4*y
```

The first line is a comma separated list of variables we used in the system. There is only one other line because there is only one equation in our polynomial system. Note that we needed to explicitly provide the operator * for multiplication.

The next step is to set up the Bertini parameter files using the setup script:

```
$ bin/sampling-setup --processors 8 quartic_example/
```

The –processors 8 option tells the setup script to use 8 processes for homotopy continuation. The setup script produced a directory quartic_example/minimizer containing all of the files.

Next we'll set up a parameters file. It is saved as quartic_example/parameters.json and looks like this:

```
{
        "bounds": [-3.0,3.0,-3.0,3.0,-3.0,3.0],
        "density_parameter": 5e-1,
        "number_of_functions": 1,
        "number_of_processors_for_bertini": 4,
        "number_of_parallel_instances": 1
}
```

This tells the algorithm that we are sampling the desired variety in the region $[-3.0, 3.0] \times [-3.0, 3.0] \times [-3.0, 3.0] \subseteq \mathbb{R}^3$ and that we want the sample to have density 0.5.

The number_of_processors_for_bertini tells the algorithm how many processors to use per instance of the algorithm. The number of instances of the algorithm is given by number_of_parallel_instances. These options are discussed in more depth in Section 4. The last step is to execute the sampling:

```
$ bin/tdasampling -p quartic_example
```

Output is placed in the file quratic_example/500e-3_sample.txt. There is some progress information displayed during execution.

4 Advanced settings

4.1 Parallelism

There are two levels of parallelism in the sampling algorithm:

1. At the top level of parallelism, we can split the box where we're looking for samples into smaller boxes and run the sampling algorithm in parallel on the samples individually

2. At the second level of parallelism, `bertini` is run repeatedly to solve systems produced by the algorithm. Putting more processors towards Bertini lowers the time cost of each of these loops. In general there will be (top-level)*(second-level) processes.

There is a tradeoff between these two levels of parallelism. Roughly speaking, doubling the number of processors doing either task will half the amount of time taken at that level (per iteration). To inform your decision, you can check the top line of the file minimizer/start_points. The number in the top line roughly conveys the maximum number of processes that Bertini will find useful.

`tdasampling` supports providing ssh hostnames for nodes on an ssh-accessible cluster. The hosts will be used for Bertini computations, **but not for top level computations**. The top level sampling executions will all run on the executing machine. The minimizer directory must be accessible from all nodes.

4.2 Options

The options in this section are the keys to use in parameters.json files. Command line support is currently untested.

4.2.1 bounds

Required. The region of Euclidean space in which to produce a sample. For a region $[x_1, y_1] \times \cdots \times [x_n, y_n]$ input should be in the form:

```
{
"bounds": [x_1,y_1,\dots,x_n,y_n]
}
```

4.2.2 density_parameter

Required. Requested density of sampling. If t is the float/double-valued density, input should be in the form:

```
{
"density_parameter": t
}
```

4.2.3 number_of_functions

Required. Number of functions n in polynomial system.

```
{
"number_of_functions": n
}
```

4.2.4 number_of_parallel_instances

Number k of top level instances of the sampling algorithm to run.

```
{
"number_of_parallel_instances": k
}
```

4.2.5 mpi_executable_location

If your mpiexec executable is not runnable via just "mpiexec", you can set this option to give a specific path to the executable.

```
{
"mpi_executable_location": /a/path/to/mpiexec
}
```

4.2.6 bertini_executable_location

If your bertini executable is not runnable via just "bertini", you can set this option to give a specific path to the executable.

```
{
"bertini_executable_location": /a/path/to/bertini
}
```

4.2.7 number_of_processors_for_bertini

Number of processors p to use for Bertini for each parallel instance of the algorithm.

```
{
"number_of_processors_for_bertini": p
}
```

4.2.8 hosts

List of SSH names for hosts to use during parallel Bertini runs. Processors are currently assigned to hosts on a cylcic bases in the order they appear in the hosts list. In particular, if the number of processors you request isn't a multiple of the number of hosts, the hosts earlier in the list will be assigned more processes. The following is an example list of hosts.

```
{
"hosts": "linhost1,linhost2,adifferenthost15,localhost"
}
```

4.2.9 skip_interval

One of the parameters controlling the heuristics for keeping the number of points in the final sample low. Should be a number $t \in (0, 1)$. Lower numbers correspond to fewer points, but more compute time. The default is recommended.

```
{
"skip_interval": t
}
```

4.2.10 rolling_average_length

One of the parameters controlling heuristics for keeping the number of points in the final sample low. Should be an integer k. In general, it may be useful to set this parameter higher if the polynomial system you're solving has a larger number of start_points. Defaults should perform decently.

```
{
"rolling_average_length": k
}
```

4.2.11 number_of_allowed_skips

One of the parameters controlling heuristics for keeping the number of points in the final sample low. Should be an integer p which is less than the rolling_average_length. Higher numbers correspond to fewer points, but more compute time. Defaults should perform decently.

```
{
"number_of_allowed_skips": p
}
```

4.2.12 variable_indices

For advanced users. If you have generated your own parameter homotopy files, and the relevant variables (in order) to extract are not the first *dimensionality* ones, you can list alternative variables (in whatever order) instead.

```
{
"variable_indices": [index1,index2,...,indexk]
}
```

5 Other considerations

5.1 Cleaning up interrupted runs

If a sampling run crashes for any reason, the current code does not clean up its computation files. Those files are placed in minimizer/computation_directory if doing an SSH based run. They are placed in folders of the form /tmp/tmp* otherwise. You can clean them up, for instance in the latter case, with

```
$ rm -rf /tmp/tmp*
```