Numerical Relativity: Starting from Scratch
– Sample Codes –

Thomas W. Baumgarte and Stuart L. Shapiro

January 20, 2021

This web page contains links to the two python sample codes presented in *Numerical Relativity: Starting from Scratch*, as well as plotting routines that help visualize the results. The first code, `puncture.py`, solves the *constraint equations* of Chapter 3 and constructs initial data describing black holes, while the second code, `maxwell.py`, provides an example for how to solve the *evolution equations* of Chapter 4 – in this case Maxwell's equations.

Both scripts, as well as the accompanying visualization tools, were written for python3, but should run with some earlier versions of python also. Depending on the computer platform, and on which libraries have been installed (in particular `matplotlib`), readers may want to try different versions of python. We provide these routines as simple python scripts, but readers could also upload these routines into a `jupyter` notebook, of course, or their favorite IDE.

A brief disclaimer before we get started: Coding is always an exercise in compromise, in which a number of different goals, including efficiency, readability, and generalizability, are weighed against each other. In the scripts provided here we have not paid much attention to efficiency – presumably, somebody interested in a large-scale, highly efficient code would not use python anyway. Instead we have tried to make the scripts as readable as possible. We hope that the reader can easily identify equations in the text (we have even coded up some operators, rather than using python libraries, so that the reader can see these operators "at work"), can run and modify the scripts (see, e.g., the exercises in the book), and can get a sense of how the numerical algorithms described in Appendix B can be used to solve prototypes of typical problems arising in numerical relativity.

1 Constructing Initial Data: `puncture.py`

The script `puncture.py` implements the puncture method of Section 3.4 to construct initial data describing black holes. We refer to Section B.2.3 for a complete description of the variables and parameters used by the code.

Once downloaded, the script can be run at command line with

```
python3 puncture.py
```

As we discussed above, the user could alternatively upload the file into a `jupyter` notebook or an IDE and run the script that way. Executing the script should produce puncture initial

data for a single black hole with certain default parameters, including a numerical grid with `n_grid` $= 16$ uniform grid points in each dimension, and the outer boundary at `x_out` $= 4.0$, in units of the black hole's puncture mass $\mathcal{M}$, in each direction. Running the script with the flag `-h`, i.e.

```
python3 puncture.py -h
```

will return a list of all parameters, their default values, and flags that can be used to overwrite these default values. For example, to produce data with `n_grid` $= 24$ and `x_out` $= 6.0$, call

```
python3 puncture.py -n_grid 24 -x_out 6.0
```

Other flags allow the user to change the position and momentum of the black hole, as well as parameters for the numerical iteration.

The script will produce a data file called `Puncture_<n_grid>_<x_out>.data`, where, in the above example, `<n_grid>` would be replaced by 24, and `<x_out>` by 6.0. The data file contains values of the function u (see Section 3.4) as a function of x and y on a plane of constant z as close to the equatorial plane as possible.

The data can be visualized with the routine `puncture_plot.py`. Running this script with the flag `-h`, i.e.

```
python3 puncture_plot.py -h
```

again displays all options and their defaults. An example for a black hole with linear momentum $\bar{P}^i = (1, 0, 0)$ located at the origin of the coordinate system is shown in Fig. 3.1.

2 Solving the Evolution Equations: `maxwell.py`

The script `maxwell.py` solves Maxwell's equations in vacuum and in a flat Minkowski spacetime. It adopts the initial data of equation (B.55) to produce a snapshot of a dipolar electromagnetic wave, and then evolves these data in time – either with the "original" or a "reformulated" version of Maxwell's equations (see Section 4.1.1). We refer to Section B.3.2 for a complete description of the variables and parameters used by the code.

At command line, the script can be run with

```
python3 maxwell.py
```

in which case it adopts a number of default options, for example for the number of grid points `<n_grid>` (set to 26, which includes two ghost points), and the location of the outer boundary `x_out` (set to 6.0, in units of λ). By default, the script also evolves the original version of Maxwell's equations. To see all options, their default values, and how to overwrite them, run `maxwell.py` with the flag `-h`. For example, to evolve the reformulated set of Maxwell's equations to a time `t_max = 150`, storing data at intervals of `t_check = 1.0`, both in units of λ, run `maxwell.py` with

```
python3 maxwell.py -reform -t_max 150.0 -t_check 1.0
```

The script will produce two different types of output files.

One output file, named `Max_orig_<n_grid>_<x_out>_constraints.data`, lists the L2-norm of the constraint violations $\mathcal{C}$ (see equations B.59 and B.60) as a function of time, evaluated at time intervals `t_check` (see Figs. 4.1 and B.2 for examples).

A second set of output files, named `Max_reform_<n_grid>_<x_out>_fields_<t>.data`, lists "snapshots" of the electromagnetic fields as well as the constraint violations as a function of x and y for the smallest value of z at times `<t>`; these files are also created at time intervals `t_check`. These data files can be used to produce animations of the electromagnetic field with the script `maxwell_animation.py`. Executing

```
python3 maxwell_animation.py
```

for example, the script will search for the above "snapshot" data files in the current directory and will produce an animation for the x-component of the electric field $\mathbf{E}$ from these data. We again caution the reader that results may depend on platform and installations of python and its libraries; if python3 does not work, earlier versions may. The script `maxwell_animation.py` can also be run with a number of different options (as before, they are listed when the flag `-h` is used); in particular, it can produce either animations or "stills" for different components of the electric field $\mathbf{E}$ or the vector potential $\mathbf{A}$.

Snapshots of the electric field $\bar{E}^y$ at some representative times are shown in Fig. B.1; they show the electromagnetic wave propagating through the numerical grid and disappearing though the outer boundaries.