1 Tstrippy

This project is built on `f2py`, based on the paradigm that code compiled in Fortran is significantly faster than Python, while Python, especially within Jupyter notebooks, is far more convenient for development and visualization. `f2py` stands for *Fortran to Python* [Peterson, 2009]. The name, `t-strip-py`, is short for `T`idal `Strip`ping `Py`thon.

`F2py` only supports `F77/F90/F95`. So we choose `F90` since it introduces *modules*. A module is similar to Object Oriented Programming. A Module has its own data, and subroutines that can modify said data. However, modules do not have ineritences, and there can not be more than one instantiation of a module, as there can be many instances of the same class within the same program.

`tstrippy` has five modules:

- `integrator`: the main module. This module stores the positions and velocities and computes the forces and time evolution of the orbits. It also writes out intermitten data during the simulations, if desired. It interfaces with the other modules within the code.
- `potentials`: The analytical potentials that are used for the force computations. At the moment, it includes: `Plummer`, `Hernquist`, `AllenSantillian`, `MiyamotoNagai`, `longmuralibar` [Long and Murali, 1992] and the composite potential `pouliasis2017pii`. The user can also access this module directly in python if they are interested in, for example, computing the energy of a body in post production.
- `hostperturber`: this module handles the host globular cluster. It stores the orbit's timestamps, positions, and velocities. It coordinates the timestamps of the cluster with the integrator's internal clock. It computes the forces on the star-particles by the perturber.
- `perturbers`: this module practically does the same as the host perturber, but can handle multiple different clusters. When computing the force on each particle, it loops over all clusters. If Object Oriented Programming was an option, both `pertubers` and `hostperturbers` would be children of the same parent class.
- `galacticbar`: This module stores the bar's parameters corresponding to the potential model. It also stores `barpoly` which are the polynomial coefficients for angular displacement as a function of time:
$$\theta(t) = \theta_0 + \omega t + \dot{\omega}t^2 + \ddot{\omega}t^3 + \dots$$

1.1 Minimum example

If the package is properly installed on the system, it can be imported at the top of any Python script:

```python
import tstrippy
```

Next, the user must load or define the initial conditions. The code provides:

- The masses, sizes, and kinematics of the globular cluster catalog from Baumgardt and Hilker [2018];
- The galactic potential parameters for model II of Pouliasis et al. [2017]; and
- A galactic reference frame.

```python
GCdata      = \
    tstrippy.Parsers.baumgardtMWGCs().data
MWparams    = \
    tstrippy.Parsers.potential_parameters.pouliasis2017pii()
MWrefframe  = \
    tstrippy.Parsers.potential_parameters.MWreferenceframe()
```

The user must then select the system to integrate. For example, to integrate the orbits of observed globular clusters, one must convert the ICRS coordinates to a Galactocentric frame using `astropy` and the provided MW reference frame. Alternatively, to simulate a star cluster, one can generate a Plummer sphere:

```
xp,yp,zp,vxp,vyp,vzp=\
tstrippy.ergodic.isotropicplummer(G,massHost,halfmassradius,NP)
```

Here, `NP` is the number of particles, `halfmassradius` is the system's half-mass radius, `massHost` is the total mass of the Plummer sphere, and `G` is the gravitational constant. All values must be in the same unit system.

The integrator must then be initialized. All parameters are passed via lists that are unpacked at the function call. Here is an example of initializing the integrator for a stellar stream in a potential that includes a rotating galactic bar:

```
tstrippy.integrator.setstaticgalaxy(*staticgalaxy)
tstrippy.integrator.setinitialkinematics(*initialkinematics)
tstrippy.integrator.setintegrationparameters(*integrationparameters)
tstrippy.integrator.inithostperturber(*hostperturber)
tstrippy.integrator.initgalacticbar(*galacticbar)
tstrippy.integrator.setbackwardorbit()
```

- `setstaticgalaxy` specifies the static potential model and passes its parameters.
- `setinitialkinematics` provides the initial positions and velocities of the particles.
- `setintegrationparameters` defines the initial time, timestep, and number of steps.
- `inithostperturber` specifies the globular cluster's trajectory and mass as a function of time.
- `initgalacticbar` defines a rotating bar. It takes the name of the bar model, potential parameters, and spin parameters.
- `setbackwardorbit` reverses the velocity vectors and sets the internal clock to count down: $t_i = t_0 - i \cdot \Delta t$. For the usecase presented in this work, `setbackwardorbit` is used for computing the globular cluster orbits and not for the star-particles.

The user can choose between two output modes during integration:

```
tstrippy.integrator.initwriteparticleorbits(nskip,myoutname,myoutdir)
tstrippy.integrator.initwritestream(nskip,myoutname,myoutdir)
```

Conceptually, these represent two output paradigms:

- `initwriteparticleorbits` saves the full orbit of each particle to an individual file.
- `initwritestream` saves full snapshots of all particles at selected timesteps.

Both functions take:

- `nskip`: number of timesteps to skip between outputs;
- `myoutname`: the base file name;
- `myoutdir`: the output directory.

The output files will be named like: `../dir/temp0.bin`, `../dir/temp1.bin`, ..., up to `../dir/tempN.bin`, where $N = N_{\text{step}}/N_{\text{skip}}$. Note that the files are written in Fortran binary format. Although `scipy.io.FortranFile` can read them, I use a custom parser based on `numpy.frombuffer` to avoid the SciPy dependency.

Once all parameters are set, the user can proceed with integration using one of two methods:

Full orbit integration (in memory)

```
xt,yt,zt,vxt,vyt,vzt=\
    tstrippy.integrator.leapfrogintime(Ntimestep,nObj)
timestamps=\
    tstrippy.integrator.timestamps.copy()
```

`leapfrogintime` stores the full orbit of each particle in memory. This is useful for a small number of particles or short integrations—e.g., rapid parameter studies in a notebook. However, for large simulations it can be prohibitively memory-intensive. For instance, integrating all globular clusters at high time resolution might require:

$$7 \times N_p \times N_{\text{step}} \times 8 \text{ Byte} \approx 450 \text{ GB} \tag{1}$$

if $N_{\text{step}} \approx 10^7$. This will likely exceed system RAM.

Final state only

```
tstrippy.integrator.leapfrogtofinalpositions()
xf   = tstrippy.integrator.xf.copy()
yf   = tstrippy.integrator.yf.copy()
zf   = tstrippy.integrator.zf.copy()
vxf  = tstrippy.integrator.vxf.copy()
vyf  = tstrippy.integrator.vyf.copy()
vzf  = tstrippy.integrator.vzf.copy()
finaltime=tstrippy.integrator.currenttime.copy()
```

`leapfrogtofinalpositions()` performs the integration but only returns the final phase-space coordinates. These arrays must be copied before deallocating memory:

```
tstrippy.integrator.deallocate()
```

Deallocating is necessary to avoid memory leaks or crashes in Jupyter when rerunning code cells.

References

H. Baumgardt and M. Hilker. A catalogue of masses, structural parameters, and velocity dispersion profiles of 112 Milky Way globular clusters. MNRAS, 478(2):1520–1557, August 2018. doi: 10.1093/mnras/sty1057.

Kevin Long and Chigurupati Murali. Analytical Potentials for Barred Galaxies. ApJ, 397:44, September 1992. doi: 10.1086/171764.

Pearu Peterson. F2py: a tool for connecting fortran and python programs. *International Journal of Computational Science and Engineering*, 4(4):296–305, 2009. doi: 10.1504/IJCSE.2009.029165.

E. Pouliasis, P. Di Matteo, and M. Haywood. A Milky Way with a massive, centrally concentrated thick disc: new Galactic mass models for orbit computations. A&A, 598:A66, February 2017. doi: 10.1051/0004-6361/201527346.