This project is built on `f2py` and the paradigm that code compiled in fortran is faster than python, yet python is much more convenient that fortran, particularly given Jupyter notebooks. `f2py` means `Fortran` to `python` [Peterson, 2009].

If the package is correctly installed on the system, then it just need be imported at the head of any python script:

```
import tstrippy
```

Then, the user must select and/or load the initial conditions. In the code, I provide the masses, sizes, and kinematics of the globular cluster catalog [Baumgardt and Hilker, 2018]; the galactic potential parameters for the second model of [Pouliasis et al., 2017]; and a galactic reference frame:

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

The user must pick the initial conditions. For instance, if the user wants to integrate the globular clusters, they must use the MW reference frame to convert the IRCS coordinates to a Galactocentric reference frame using *astropy*. Or, the user can generate a plummer sphere if wishing to simulate a cluster:

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

where `NP` is the number of particles, `halfmassradius` is the half mass radius of the system, `massHost` is the total mass of the plummer sphere, and `G` is of course, the gravitational constant. The values must be in the same unit basis.

Then, the integrator must be prepared. All of the arguments can be packaged into a list and then unpacked in the function's arguments. Here is an example of initializing the integrator for a stream in a potential with a 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` tells the code which potential model to use and also passes it's parameters. `setinitialkinematics` takes the initial positions and velocities of all the particles whose orbits are to be determined. `integrationparameters` gives the initial time, timestep, and number of steps for the integration. `inithostperturber` takes the timestamps, positions, velocities, mass, and characteristic radius of the host globular cluster. `initgalacticbar` is similar to `setstaticgalaxy`. It expects the name of the bar model, the parameters for the potential, and the parameters for the spin, which are coefficients to a polynomial: $\theta(t) = \theta_0 + \omega t + \dot{\omega} t^2 + \ddot{\omega} t^3 + \ldots$, which allows the user to pass a galactic bar with a spin rate that changes in time. `setbackwardorbit` changes the signs of the velocities to send the bodies backward, and ensures that the internal clock of the integrator decreaes: $t_i = t_0 - i \cdot \Delta t$.

The user has two options of writing out data during the integration:

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

conceptually, there are two paradigms. We can either same the orbit of each particle into its own file, *or* save snapshots of the simulations at different times. Both functions take the same arguments, `nskip` controls how many integration timesteps to skip before writing out data. `myoutdir` is the output directory of where to save the files and `myoutname` is the base file name, which then gets appended an integer. The resultant filenames would be something like: `../dir/temp0.bin`, `../dir/temp1.bin`, `.....`/`dir/tempN.bin`, where $N = N_{\text{step}}/N_{\text{skip}}$. `initwriteparticleorbits` writes the same number of files as the number of particles, and during the integration step it returns to said file and appends it with a particles full phase-space information and timestep. This is intended for saving the trajecotries of the globular clusters. `initwritestream` on the other hand, is intended for saving the positions and velocities of all the particles at a single timestamp. Note that the data are saved in Fortran binary files. They must be parsed knowing their structure. There is a scipy routine that reads Fortran files, `scipy.io.FortranFile`. However, I have written a custom script that parses the binary files using `numpy.frombuffer` to avoid a scipy dependency.

After all the parameters the user has two options for integrating. The two options match paradigm for being interested either in orbits of individual bodies, or snapshots of the entire simulation. To save the whole orbit, the user must:

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

`leapfrogintime` must be used with caution. The resultant data size can be extremely large if the number of steps. For example, if we wanted to integrate the whole globular cluster system at the highest temporal resolution to resolve the internal dynamics of the denesest globular cluster: we would have a data volume of:

$$7 \times N_p \times N_{\text{step}} \times 8 \text{ Byte} \approx 450\ Gb, \quad (1)$$

if $N_{\text{step}} \approx 10^7$. This is much larger than the RAM. This breaks even easier if considering a stream even if at modest timestep size. However, `leapfrogintime` is useful when wanting to look at orbits of a few particles, or for a short period of time. It is very useful when quickly exploring parameters in a Jupyter notebook. When looking to get high resolution data, writing the data out must happen.

To not store any of the intermediate positions in the integrator during the integration we can do this:

```
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()` just goes to the final positions. The resultant positions must be copied otherwise they will be deleted after deallocating the integrator, which must be done to ensure that the kernel doesn't crash:

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

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.

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.