Time domain self-force with discontinuous Galerkin code: `SelfForce1D`

Peter Diener[1]
in collaboration with
Barry Wardell[2] and Niels Warburton[2]

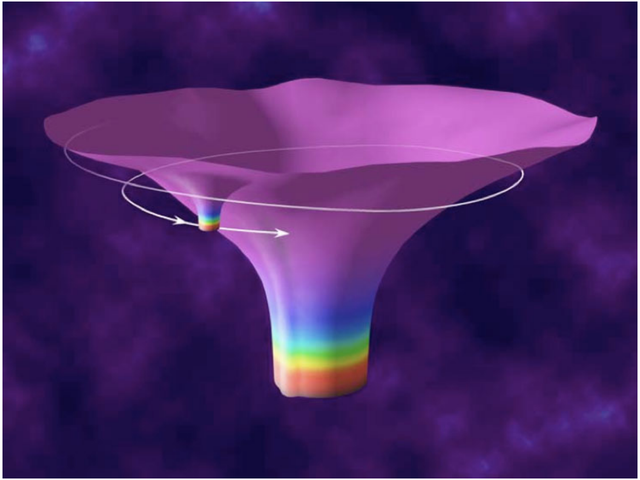
[1]Louisiana State University
[2]University College Dublin

July 29, 2021
North American Einstein Toolkit School, ~~UIUC, Urbana Champaign~~ online

Extreme Mass Ratio Inspirals.

- Supermassive black holes are surrounded by a cluster of stars.
- Some of these will be black holes or neutron stars.
- Can be brought onto highly eccentric orbits by two-body interactions.
- Energy and angular momentum losses through gravitational wave emission shrinks the orbit until the small object plunges into the supermassive black hole.
- Eccentricity will decrease over time but will most likely still be significant just before the plunge.
- Such systems are expected to be very important events for the space based gravitational wave detector LISA.
- Analytically and numerically it is possible to use perturbation theory and the point particle approximation by decomposing the field into a singular and a regular part.
- The particle perturbs the spacetime and interacts with it's own perturbations to accelerate the orbit causing the inspiral.
- The simpler scalar charge case can be used as a sandbox for development of new numerical techniques.

Extreme Mass Ratio Inspirals.



Artist's impression of an EMRI. Credit NASA

A small compact object in orbit around a supermassive black hole.

SelfForce-1D

SelfForce-1D is an open source code for performing time domain self-force computations.

- Evolves the scalar wave equation (metric perturbation equations are being added) in a Schwarzschild space-time (Kerr is being added).
- Fields are decomposed into Spherical Harmonics resulting in 1+1 dimensional PDE's to be solved using the Method of Lines.
- Nodal Discontinuous Galerkin method being used to discretize the PDE's in the radial direction. Can handle non-smooth features easily.
- Point particle treatment through the Effective Source for generic orbits in a world-tube approach (retarded field outside, regular field inside).
- Different coordinate systems are used in different parts of domain: Hyperboloidal near horizon and $\mathscr{I}^+$, time dependent or Tortoise in between. That is, the computational domain covers everything between the horizon and $\mathscr{I}^+$.

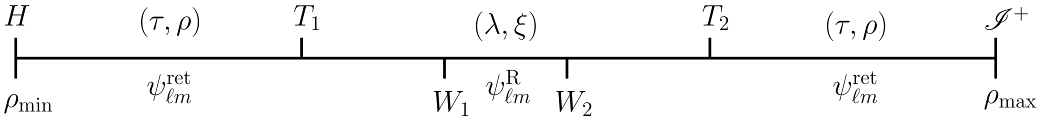
SelfForce-1D (cont)

- Generic orbits evolved using direct geodesic integration (with forces) or through the osculating orbits framework (also with forces).
- Runge-Kutta and Adams-Bashford-Moulton multi-value methods can be used for time integration.
- Self-force can be extracted from the regular field at the particle location.
- Other observers can extract the fields at the horizon and at $\mathscr{I}^+$.
- Can use initial data calculated in the frequency domain for eccentric geodesics for low ℓ-modes in order to avoid having to evolve for a long time before initial transient leaves the computational domain.
- Back reaction can be turned on but there are still some issues with instabilities.
- Written mostly in object oriented Fortran with the effective source in C++ (Barry) and initial data in Python (Niels).

Code requirements

- Needs a fairly modern Fortran compiler. Has been successfully compiled with `gfortran 7.3.0` and `ifort 17.0.7` and newer (however Ubuntu's gfortran fails with internal compiler error).
- Needs a C++ compiler. Either `g++` or `icpc` should work.
- Needs `Python` as it uses `SCons` for building and initial data files are produced by a `Python` code.
- Needs `BLAS`/`LAPACK` (small subset only for some small matrix eigenvalues and inversions).
- Needs `GSL` (for Spherical Harmonics and fitting).
- Documentation of classes available at: `https://www.cct.lsu.edu/~diener/SelfForce1D/Doc/index.html` (produced by `FORD`).

The computational setup



Between H and T_1 ingoing hyperboloidal coordinates (τ, ρ) are used to evolve the retarded field $\psi_{\ell m}^{\mathrm{ret}}$.
Between T_1 and W_1 time dependent coordinates (λ, ξ) are used to evolve the retarded field $\psi_{\ell m}^{\mathrm{ret}}$.
Between W_1 and W_2 time dependent coordinates (λ, ξ) are used to evolve the regular field $\psi_{\ell m}^{\mathrm{R}}$.
Between W_2 and T_2 time dependent coordinates (λ, ξ) are used to evolve the retarded field $\psi_{\ell m}^{\mathrm{ret}}$.
Between T_2 and $\mathscr{I}^+$ outgoing hyperboloidal coordinates (τ, ρ) are used to evolve the retarded field $\psi_{\ell m}^{\mathrm{ret}}$.

Future improvements and developments.

- Other systems of equations are in the pipeline:
  1. Teukolsky in both Schwarzschild (REU student Sarah Skinner, 2019) and Kerr (REU student Sho Gibbs, 2020 and Samantha Hardin, 2021).
  2. Metric perturbations in Lorenz gauge (Samuel Cupp)
  3. Regge-Wheeler-Zerilli metric perturbations (Samuel Cupp, me)
- Checkpointing/restart (REU student Mary Ogborn, 2020).
- High level code documentation explaining how all the classes fit together.
- The code is part of the Einstein Toolkit and the Black Hole Perturbation Toolkit.
- Hopefully the code will be a useful community resource that will inspire new developments that will be contributed back to the toolkits.
- Available at: `https://bitbucket.org/peterdiener/selfforce-1d`

Extra 1: The design.

- Relies on object oriented programming ideas to expose and exploit modularity whenever possible.
- Implemented in modern Fortran 2003/2008.
- One of the basic concepts is an abstract `Equation` class that knows nothing about the actual equations but defines the interface to certain type bound procedures (like C++ member functions) that any derived `Equation` class has to provide.
- On top of this, different types of `Equation` classes that know about the data structures they need (i.e. ODE or PDE equations) can be defined.
- On top of these, actual equations systems (geodesic evolution, osculating orbits evolution and scalar wave equation) can finally be defined.
- The time integrator need only know about the type bound procedures as defined in the abstract `Equation` class (and implemented in the actual equation classes) and hence is completely agnostic about the underlying data structures.
- Communication between equations are done through external data types where different equation classes can write and read data without knowing about each other.

Extra 2: The abstract equation class

```
type, abstract :: equation
  integer :: ntmp
  character(:), allocatable :: ename
contains
  procedure (eq_init_interface), deferred, pass :: init
  procedure (eq_rhs_interface), deferred, pass :: rhs
  procedure (eq_set_to_zero_interface), deferred, pass :: set_to_zero
  procedure (eq_update_vars_interface), deferred, pass :: update_vars
  procedure (eq_save_globals_1), deferred, pass :: save_globals_1
  procedure (eq_save_globals_2), deferred, pass :: save_globals_2
  procedure (eq_load_globals), deferred, pass :: load_globals
  procedure (eq_output), deferred, pass :: output
end type equation
```

Other classes can then extend this class, provide some of the routines and defer other routines to the next level.

Extra 3: Interface for update_vars

```
subroutine eq_update_vars_interface ( this, source, dest, source2, &
                                              scalar, scalar2 )
  class(equation), target, intent(inout) :: this
  integer(ip), intent(in) :: source
  integer(ip), intent(in) :: dest
  integer(ip), optional, intent(in) :: source2
  real(wp), optional, intent(in) :: scalar
  real(wp), optional, intent(in) :: scalar2
end subroutine eq_update_vars_interface
```

Here source, source2 and dest can be -1 (RHS), 0 (VAR) or $1 \ldots \mathrm{ntmp}$ (TMP).

This can be used to perform an update like $\mathrm{VAR} = \mathrm{scalar} * \mathrm{RHS} + \mathrm{scalar2} * \mathrm{TMP}$.

Every class that provides an actual implementation of an equation has to contain a routine that implements the update_vars interface.