SEREN - Version 1.0.0

David Hubber, Christopher Batty & Andrew McLeod

January 10, 2011

Contents

1 Overview

SEREN is a Smoothed Particle Hydrodynamics (SPH) code designed for solving slef-gravitating hydrodynamical problems in astrophysics, particularly in the fields of star and planet formation. SEREN largely grew from DRAGON, the star formation SPH code written by Simon Goodwin at Cardiff. Although many routines have significantly diverged from the original DRAGON versions, there are still several features that have survived. SEREN has also been designed to be compatible with DRAGON in its features and file formats.

The basic elements of SEREN can be used to simulate any problem involving hydrodynamics and gravity, but SEREN also contains many specialized features for star formation problems. The main features present in SEREN 1.0.0 include :

- Smoothed Particle Hydrodynamics (Standard SPH or conservative 'grad-h' SPH)
- Self-gravitating SPH / N-body dynamics
- Isothermal, barotropic or polytropic equations of state
- Octal-spatial (Barnes-Hut) neighbour-searching and gravity tree
- Simple sink particles (i.e. gravity only) or minimum-h value
- Different particle types (gas, inter-cloud, boundary and CDM particles)
- 1, 2 or 3 dimensions
- Periodic boundary conditions (independent for each dimension) or spherical wall
- Hierarchical block timesteps, with neighbour-checking for safety
- Euler, 2nd order Runge-Kutta, Leapfrog-KDK, Leapfrog-DKD and Predictor-Corrector integration schemes
- Artificial viscosity with Balsara switch, time-dependence and Keplerian pattern-matching
- Artificial conductivity with switches
- N-body evolution of sinks using 4th order Hermite integrator at termination of SPH
- Radiative cooling approximation of Stamatellos et al. (2007)
- Parallelized using OpenMP
- Bash script to run automated batch tests
- Output compatable with Splash

Features currently in development, or implemented but not full tested

- MPI parallelization (McLeod)
- Combined radiative cooling and Flux-limited Diffusion (Stamatellos)
- Sinks with smoother (non-integer) accretion of particles (Hubber, Walch & Whitworth)
- Re-distributing angular momentum in sink particles (Walch, Hubber & Whitworth)
- Hybrid 4th-order hermite-scheme for N-body integration combined with SPH for background gas evolution (Hubber)
- Grouped particle tree walks (Hubber)
- Ewald method for periodic gravity (McLeod & Hubber)
- Implement Riemann solver for all EOSs (Hubber)

- Binary-mass tree (Whitworth & Hubber)
- Ionizing radiation with HEALPix (Bisbas & Hubber)
- Remove unconsequential outlying particles from simulation (Hubber)

Features planned for future versions include :

- Physical viscosity
- Jet feedback from protostars
- Wind feedback from high-mass stars
- 2nd fluid component (e.g for ion/dust particles)
- Particle Splitting
- Ideal/Non-ideal MHD
- Divergence cleaning/Euler potentials

Suspended/abandoned features (i.e. features implemented in the past where development has either been suspended, or abandoned but not removed from the code as of yet but may do so in the future)

- Ideal MHD

2 Using SEREN

2.1 Obtaining SEREN via git

SEREN can be obtained using *git* (http://git-scm.com/), which is a new version-control software written by Linux-kernel author, Linus Torvalds. Further information will be added here about using git, both to download SEREN and to automatically update it, in the near future.

2.2 Compiling and running SEREN

SEREN has been designed so to be compiled with GNU make. The user must specify a number of compiler options, which are set at the head of the Makefile (see Section **??** for more information). In order to compile, a compatible compiler must be specified in the first line of the Makefile. SEREN has been successfully tested on the following operating systems and compilers.

- GNU/Linux
  - f95 - NAG f95 compiler (Linux workstations)
  - g95 - g95 compiler (Linux workstations)
  - gfortran - GNU Fortran compiler (Linux workstations)
  - ifort - Intel Fortran compiler (Merlin cluster)
  - pgf90 - Portland group Fortran compiler (Coma cluster)
  - pgf95 - Portland group Fortran compiler (Iceburg cluster)
- Mac OS X (1.4, 1.5 & 1.6)
  - g95 - g95 compiler
  - gfortran - GNU Fortran compiler

Once all the other Makefile options have been set to their desired values, SEREN is compiled by GNU make with the command

```
make seren
```

GNU make will compile the source code of SEREN and produce the executable program seren. Prior to performing a simulation, the user must set all simulation parameters in the file params.dat (See section **??** for more information) and provide an initial conditions file in the appropriate format. To run SEREN , the user must type

```
./seren
```

SEREN will read in the default parameters file params.dat before performing the simulation.

2.2.1 Command-line arguments

SEREN has a number of optional command-line arguments that can be invoked to change the behaviour of the SEREN executable. The behaviour can depend on several factors, particularly what Makefile options have been invoked while compiling SEREN .

Table 1: List of all command-line arguments available in SEREN

Argument	**Behaviour**
-d, -D	SEREN prints out the debug output column data format to screen and then exits without running any simulation. (N.B. The same information is printed to the runid.params file when a simulation is performed using SEREN)

Argument	Behaviour
-diag	SEREN prints out the column data format to screen of the diagnostics file (enabled with the -DDEBUG_DIAGNOSTICS flag in the Makefile).
-h, -H, -help	SEREN prints out all available command-line options
-m, -M	SEREN prints out the Makefile options used to compile the code to screen and then exits without running any simulation. (N.B. The same Makefile options are printed to the runid.params file when a simulation is performed using SEREN)
-s, -S, -sinks, -stars	SEREN prints out the column data format to screen for the sink files.
-v, -V, -version	SEREN prints out current version number
paramsfile	SEREN reads the parameters file paramsfile instead of the default params.dat

2.2.2 Restarting simulations

If a simulation is terminated for some reason, then it can be restarted by simply running SEREN without any modification to the params.dat file. Each simulation generates a file runid.restart which contains the name of the last snapshot to be outputted. SEREN will search for this file, and if it exists, it will read the snapshot name contained and restart the simulation from that point. If you do not wish to restart the simulation from this point, but want a fresh run, then you should delete the runid.restart file. If you wish to restart a simulation from a different snapshot, you can delete the runid.restart file and alter some of the parameters in the params.dat file, such as the restart logical flag (See Section **??**).

2.3 Makefile

The head of the Makefile contains the complete list of compilation variables that are available. Most variables have two or more possible values which must be entered in the Makefile. If an illegal value is entered, then make will halt during compilation, or the program will stop during runtime (see the routine /main/sanitycheck.F90). The Makefile is technically split into two separate files; Makefile, which contains the user options, and makefiletail.mk, which processes all the selected options to compile the code. The full list of all Makefile variables and possible options is given in the table below.

Table 2: List of all available Makefile options in SEREN

Variable	**Options**	
F90	f95 :	NAG f95 compiler (Linux)
	g95 :	free (not gnu) f95 compiler (Linux, Mac OS X)
	gfortran :	gnu f95 compiler (Linux, Mac OS X)
	pgf90 :	Portland Group compiler (Coma cluster)
	mpif90 :	Portland Group compiler (Coma cluster)
	ifort :	Intel Fortran compiler (Merlin cluster)
VERSION_NO		Version no. string
SRCDIR		Absolute path of main SEREN directory (default $(PWD)/src)
EXEDIR		Absolute path of location for SEREN executable (default $(PWD))
OPTIMISE	0 :	No compiler optimization
	1 :	-O1 optimization
	2 :	-O2 optimization
	3 :	-O3 optimization
OPENMP	0 :	Compile as serial code
	1 :	Compile using OpenMP directives
PROFILE	0 :	No profiling
	1 :	Subroutine profiling (Invokes -pg option to be used by gprof)
	2 :	Line profiling (Invokes -pg -g options to be used by gprof and gdb)
DEBUG	0 :	Debug flags switched off
	1 :	Debug flags switched on to level 1
	2 :	Debug flags switched on to level 2
	3 :	Debug flags switched on to level 3
NDIM	1 :	One-dimensional
	2 :	Two-dimensional
	3 :	Three-dimensional
PRECISION	SINGLE :	Single precision for main real variables
	DOUBLE :	Double precision for main real variables
INFILE_FORMAT	ALL :	Include routines to read all possible file formats
	DRAGON :	Only include DRAGON-format reading routines
	SEREN :	Only include SEREN-format reading routines
	ASCII :	Only include column-ASCII format reading routine

Variable	Options	
OUTFILE_FORMAT	ALL :	Include routines to write all possible file formats
	DRAGON :	Only include DRAGON-format writing routines
	SEREN :	Only include SEREN-format writing routines
PERIODIC	1 :	Periodic boundary conditions (Note : must be set to 1 if any of X_BOUNDARY, Y_BOUNDARY, Z_BOUNDARY or SPHERICAL_MIRROR are set to any value other than 0)
	0 :	No periodic boundary conditions
X_BOUNDARY	PERIODIC :	Periodic box in x-dimension
	WALL :	Walls in LHS and RHS directions of x-dimension
	0 :	No periodicity in x-dimension
Y_BOUNDARY	PERIODIC :	Periodic box in y-dimension
	WALL :	Walls in LHS and RHS directions of y-dimension
	0 :	No periodicity in y-dimension
Z_BOUNDARY	PERIODIC :	Periodic box in z-dimension
	WALL :	Walls in LHS and RHS directions of z-dimension
	0 :	No periodicity in z-dimension
SPHERICAL_MIRROR	1 :	Spherical mirror to reflect particles that exceed some give radial distance
	0 :	No spherical mirror
CYLINDRICAL_MIRROR	1 :	Cylindrical mirror to reflect particles that exceed some given distance about z-axis
	0 :	No spherical mirror
SPH_SIMULATION	1 :	Perform SPH simulation
	0 :	No SPH simulation
NBODY_SPH_SIMULATION	1 :	Perform hybrid N-body/SPH simulation
	0 :	No hybrid simulation
NBODY_SIMULATION	1 :	Perform N-body simulation
	0 :	No N-body simulation
SPH	STANDARD :	Use traditional SPH formulation (Monaghan 1992)
	GRAD_H_SPH :	Use 'grad-h' SPH formulation (Springel & Hernquist 2002; Price & Monaghan 2004)
SPH_INTEGRATOR	RK :	2nd order Runge-Kutta integration scheme
	LFKDK :	2nd order Leapfrog kick-drift-kick scheme
	LFDKD :	2nd order Leapfrog drift-kick-drift scheme
	PC :	2nd order Predictor-Corrector scheme
KERNEL	M4 :	M4 kernel (Monaghan & Lattanzio 1985)
	M4TC :	M4 kernel with modifed 1st derivative (Thomas & Couchman 1992)
	QUINTIC :	Quitic kernel (Morris 1996)
	QUINTICTC :	Quintic kernel with modified 1st derivative (c.f. Thomas & Couchman 1992)

Variable	Options	
	GAUSSIAN_3H :	Gaussian kernel truncated at 3 h
HFIND	NUMBER :	Determine h by number of neighbours
	MASS :	Determine h by total mass of neighbours
	CONSTANT :	Use contant smoothing length
MINIMUM_H	1 :	Set a minimum smoothing length
	0 :	Allow any smoothing length
HYDRO	1 :	Hydro forces switched on
	0 :	No hydro forces
THERMAL	ISOTHERMAL :	Isothermal equation of state
	BAROTROPIC :	Barotropic equations of state
	POLYTROPIC :	Polytropic equation of state
	ENERGY_EQN :	Solve energy equation
	CENTRAL_STAR :	...
	RAD_WS :	Whitworth & Stamatellos radiative transfer method (Stamatellos et al. 2007)
SINK_POTENTIAL_WS	1 :	Use gravitational potential from sink in radiative cooling calculations
	0 :	Ignore gravitational potential from sinks for cooling
AMBIENT_HEATING_WS	1 :	External ambient heating source (e.g. CMB)
	0 :	No ambient heating
SINK_HEATING_WS	STAR_HEATING :	
	STAR_SIMPLE_HEATING :	
	HDISC_HEATING :	
	0 :	
FLUX_LIMITED_DIFFUSION	1 :	Switch on flux-limited diffusion for hybrid radiation scheme (Forgan et al. 2009). Only activated when THERMAL = RAD_WS option is used (experimental).
	0 :	No flux-limited diffusion
IONIZING_RADIATION	0 :	No ionizing sources
	SINGLE_STATIC_SOURCE :	Single static source of ionizing radiation (Bisbas et al. 2009)
	MULTIPLE_SINK_SOURCES :	Sink particles act as sources of ionizing radiaiton (experimental)
RIEMANN_SOLVER	1 :	Use iterative Riemann solver (Van Leer 1977?, Cha & Whitworth 2003). Currently unstable and deactivated.
	0 :	No Riemann solver
ARTIFICIAL_VISCOSITY	MON97 :	Monaghan (1997) artificial viscosity
	AB :	Standard α-β viscosity (Monaghan & Gingold 1983)
	0 :	No artificial viscosity
VISC_TD	1 :	Use time-dependent value of α (Morris & Monaghan 1997)
	0 :	Constant value for α
BALSARA	1 :	Use Balsara switch (Balsara 1995)
	0 :	No Balsara switch
PATTERN_REC	1 :	Use Keplarian pattern-matching (Cartwright & Stamatellos 2010)

Variable	Options	
	0 :	Do not use pattern matching
ARTIFICIAL_CONDUCTIVITY	0 :	No artificial conductivity
	PRICE2008 :	Artificial conductivity with constant $\alpha_{\rm COND}$ (Price 2008)
	WADSLEY2008 :	Wadsley et al. (2008) conductivity
EXTERNAL_PRESSURE	0 :	No external pressure
	1 :	Simple external pressure formulation
MHD	0 :	No magnetic fields
	IDEAL :	Ideal MHD (not functioning yet)
INDUCTION_EQN	0 :	No induction equation
	STANDARD :	Use standard induction equation
RESISTIVITY	0 :	No artificial resistivity
EXTERNAL_FORCE	0 :	No external forces
	PLUMMER :	Plummer sphere potential
	UDS :	Uniform density sphere potential
	NFW1996 :	Navarro, Frenk & White (1996) potential
GRAVITY	0 :	No gravitational forces computed
	KS :	Kernel-softened gravity for 2-body forces
	NBODY :	Newton's gravitational law for all 2-body forces
EWALD	1 :	Ewald periodic gravity switched on
	0 :	No Ewald corrections
REMOVE_OUTLIERS	1 :	Remove outlying particles from the simulation (Experimental)
	0 :	No removal of outliers
SINKS	0 :	No sinks
	SIMPLE :	Simple (i.e. only gravitating) sinks (Bate, Bonnell & Price 1995)
	NO_ACC :	Simple sinks with no accretion
	SMOOTH_ACC :	Sinks with smooth accretion (experimental)
SINK_RADIUS	FIXED_ABSOLUTE :	Absolute value (in AU in params.dat file; same for all sinks)
	FIXED_HMULT :	Multiple of mean h at sink density (same for all sinks)
	HMULT :	Multiple of h at sink density (individual values for sinks)
SINK_REMOVE_ANGMOM	1 :	Deposit sink ang. mom. on nearby particles (Experimental)
	0 :	Sink particle retain ang. mom. of accreted particles
SINK_GRAVITY_ONLY	1 :	Only consider sinks are sources of gravity
	0 :	Consider all physical sources of gravity
NBODY_INTEGRATION	HERMITE4 :	4th-order Hermite integration scheme (Makino & Aarseth 1992)
BINARY_STATS	1 :	Calculate binary statistics and output to file
	0 :	No binary calculations
TREE	0 :	No tree (all quantities calculated by direct summation)
	BH :	Use Barnes-Hut tree (octal-spatial; Barnes & Hut 1985)

Variable	Options	
	BINARY :	Use Binary-number tree (Not fully functioning yet)
MULTIPOLE	0 :	No higher-order multipole terms
	QUADRUPOLE :	Include quadrupole moment terms in gravity calculations
	OCTUPOLE :	Include both octupole and quadrupole moment terms
MAC	GEOMETRIC :	Use standard Barnes-Hut geometric opening angle criterion (Barnes & Hut 1985)
	GADGET :	Use Gadget-style higher-order moment criterion (Springel et al. 2002)
	GADGET2 :	Use Gadget 2.0 moment criterion (Springel 2005)
	EIGEN :	Use Eigenvalues of Q tensor to compute appropriate MAC (Hubber et al. 2010)
REORDER	PARTICLES :	Re-order particles in arrays according to tree-walk order
	0 :	No re-ordering
CELL_WALK	1 :	Walk the tree with a group of particles (Experimental)
	0 :	Walk the tree with individual particles
SORT	INSERTION :	Use insertion sort for sorting lists
	HEAP :	Use heapsort for sorting lists
TIMESTEP	ADAPTIVE :	Block timestep levels adjusted at resync
	FIXED :	Fixed block timestep levels, with maximum level set by the $\mathrm{dt\_fixed}$ parameter
	RESTRICTED :	Timestep levels can only take certain values (dt_fixed parameter times integer power of 2), but are readjusted at resync
CHECK_NEIB_TIMESTEP	2 :	Ensure neighbours have similar timesteps
	1 :	As option 2, but doesn't change timestep in middle of current step
	0 :	No neighbour timestep comparison
NEIGHBOURLISTS	1 :	Store neighbour lists in memory
	0 :	Do not store neighbour lists in memory
TIMING_CODE	1 :	Use custom subroutines to produce timing statistics
	0 :	No timing output
DIMENSIONLESS	1 :	Dimensionless simulation (all scaling variables set to zero)
	0 :	Scaled variables (units specified in $\mathrm{params.dat}$)
TEST	FREEFALL :	Freefall collapse test
	SPIEGEL :	Spiegel (ref??) test
	BINARY :	Orbitting binary stars test
	PLUMMER :	Plummer sphere stability test
	0 :	No test flags

2.4 Debug flags

The SEREN Makefile contains a number of debug flags below the main options which can be switched on or off by uncommenting them or commenting them out. Most of the debug flags produce verbose output of each routine, and in some cases produce extra files with more important information. The full list of debugging options with additional output is shown in the table below.

Table 3: List of special debugging options available in SEREN

Variable	**Options**
DEBUG_DIV_A	
DEBUG_ACCRETE	
DEBUG_ALLOCATE_MEMORY	
DEBUG_BHTREEBUILD	
DEBUG_BHTREESTOCK	
DEBUG_BHTREEWALK	
DEBUG_BHTREEGRAVITY	
DEBUG_BINARYPROPERTIES	Ouput binary properties to screen when calculated
DEBUG_BINARY_SEARCH	
DEBUG_BLOCK_TIMESTEPS	Outputs occcupation of timestep levels
DEBUG_COPY_PARTICLE_DATA	
DEBUG_CREATE_SINK	
DEBUG_CREATE_HP_SOURCE	
DEBUG_DENSITY	
DEBUG_DIAGNOSTICS	Output diagnostic in file 'run_id.diag'
DEBUG_DIV_V	
DEBUG_DUDTRAD	
DEBUG_ENERGY_EQN	
DEBUG_FOLIATE	
DEBUG_FORCES	Records individual grav, hydro, magnetic forces
DEBUG_FREEFALL	
DEBUG_GATHER_NEIB	
DEBUG_GET_NEIB	
DEBUG_GRAD_H_SPH	
DEBUG_HEAPSORT	
DEBUG_HERMITE4	
DEBUG_H_GATHER	
DEBUG_H_GATHER_DENSITY	
DEBUG_H_GUESS	
DEBUG_HP_IF	
DEBUG_HP_OUTPUT	Outputs various ionization properties to files
DEBUG_HP_SPLIT_ACTIVE_RAYS	
DEBUG_HP_WALK_ALL_RAYS	
DEBUG_HP_WALK_RAY	
DEBUG_INTEGRATE	
DEBUG_KERNEL	Outputs file 'kernel.dat'
DEBUG_MHD	
DEBUG_NBODYSETUP	
DEBUG_PARAMETERS	
DEBUG_PLOT_DATA	Outputs regular debug files with snapshot files. Files are simple column-data files where the information of each column is written to the run_id.params file.

Variable	Options
DEBUG_RAD	Outputs 'run_id.rad' file for RAD_WS tests
DEBUG_REDUCE_TIMESTEP	
DEBUG_REMOVE_OUTLIERS	
DEBUG_RSPH_OUTPUT	Outputs files in RSPH format
DEBUG_SINKCORRECTION	
DEBUG_SINK_REMOVE_ANGMOM	
DEBUG_SINK_SEARCH	
DEBUG_SINK_TIMESTEP	
DEBUG_SKELETON	
DEBUG_SPH_UPDATE	
DEBUG_SWAP_PARTICLE_DATA	
DEBUG_TIMESTEP_SIZE	
DEBUG_TRACK_PARTICLE	Outputs single file ('track1.dat') which contains large amount of information (same as that ouputted by debug files including the time) of one single chosen particle (set by parameter ptrack in params.dat file) which is printed every timestep.
DEBUG_TREE_BUILD	
DEBUG_TREE_GRAVITY	
DEBUG_TREESTOCK	
DEBUG_TREEWALK	
DEBUG_TYPES	
DEBUG_VISC_BALSARA	
DEBUG_VISC_PATTERN_REC	

2.5 Parameter file

Unlike DRAGON, SEREN contains all simulation information in a single parameter file, called params.dat. The information contained in the parameters file in version 1.0 is shown in the following table.

Table 4: List of user parameters in SEREN

Variable	Type	Description
run_id	char(256)	Run identifier string
run_dir	char(256)	Output directory name
in_file	char(256)	Name of initial conditions file
in_file_form	char(50)	Format of initial conditions file
out_file_form	char(50)	Format of output snapshot files
restart	logical	Is this a restart or a new run?
com_frame	logical	Change to centre of mass frame?
rseed	int	Random number seed
ptrack	int	i.d. of tracked particle
sph_endtime	DP	End time of SPH simulation
nbody_sph_endtime	DP	End time of hybrid N-body/SPH simulation
nbody_endtime	DP	End time of N-body simulation
firstsnap	DP	Time of first snapshot
snaptime	DP	Snapshot time interval (in real time)
ntempstep	int	Temporary snapshot interval (in integer steps)
ndiagstep	int	Integer time interval between diagnostic output
nsinkstep	int	Sink output time interval (in integer time)
nsnapstep	int	Snapshot time interval (in integer time)
courant_mult	DP	Courant timestep multiplication factor
accel_mult	DP	Acceleration timestep multiplication factor
sink_mult	DP	Sink accel. timestep multiplication factor
nbody_timemult	DP	Timestep factor for N-body simulations
nlevels	int	Number of multiple timestep levels
dt_fixed	DP	Fixed ref. time for creating timestep levels
runit	char(256)	Length scaling unit
munit	char(256)	Mass scaling unit
tunit	char(256)	Time scaling unit
vunit	char(256)	Velocity scaling unit
aunit	char(256)	Acceleration scaling unit
rhounit	char(256)	Density scaling unit
sigmaunit	char(256)	Column density scaling unit
Punit	char(256)	Pressure scaling unit
funit	char(256)	force scaling unit
Eunit	char(256)	Energy scaling unit
momunit	char(256)	Momentum scaling unit
angmomunit	char(256)	Angular momentum scaling unit
angvelunit	char(256)	Angular velocity scaling unit
dmdtunit	char(256)	Accretion rate scaling unit
Lunit	char(256)	Luminosity scaling unit
kappaunit	char(256)	Opacity scaling unit
Bunit	char(256)	Magnetic field (B-field) scaling unit
Qunit	char(256)	Electric charge unit
Junit	char(256)	Current density unit
uunit	char(256)	Specific internal energy unit

Variable	Type	Description
rscale	DP	Length scaling factor
mscale	DP	Mass scaling factor
periodic_min(1)	PR	Size of periodic box in x-dimension
periodic_max(1)	PR	Size of periodic box in x-dimension
periodic_min(2)	PR	Size of periodic box in y-dimension
periodic_max(2)	PR	Size of periodic box in y-dimension
periodic_min(3)	PR	Size of periodic box in z-dimension
periodic_max(3)	PR	Size of periodic box in z-dimension
rspheremax	PR	Radius of spherical wall
psphere	int	Mirror origin id (0 : co-ordinates origin; p : SPH particle; -s : sink particle)
pp_gather	int	Neighbours required to determine *h*
hmin	PR	Minimum allowed smoothing length
h_fac	PR	grad-h density-h iteration factor
isotemp	PR	Temperature for isothermal, barotropic EOSs (K)
rhobary	PR	Adiabatic density for barotropic density (cgs units)
gamma	PR	Ratio of specific heats
mu_bar	PR	Mean gas particle mass (in a.m.u.)
Kpoly	PR	Polytropic constant
Pext	PR	External pressure
alpha	PR	α-viscosity value
beta	PR	β-viscosity value
alpha_min	PR	Minimum value of α
abserror	PR	Absolute error fraction in GADGET MAC
thetamaxsqd	PR	Maximum opening angle squared (Geometric MAC)
nbuildstep	int	Frequency of DRAGON tree builds (in integer time units)
rhosink	PR	Sink formation density (cgs units)
sinkrad	PR	Sink radius (in units of *h* or in AU depending on options)
nsearchstep	int	No. of integer timesteps between sink search
rho_search	logical	Calculate density for selecting sink candidates
potmin_search	logical	Only consider particles at potential minimum
hill_sphere_search	logical	Hill spheres of sinks must not overlap
energy_search	logical	Only create sinks from bound objects
div_v_search	logical	Only create sinks from converging objects
div_a_search	logical	Do not create sinks if particles are accelerating apart
timescale_search	logical	Compare timescales for sink formation
energy_accrete	logical	Only accrete bound particles
alpha_ss	PR	Sunyaev-Shakura viscosity parameter
smooth_accrete_frac	PR	Fraction of mass for instant accretion
smooth_accrete_dt	PR	Timestep fraction for instant accretion
f_accretion	PR	Fraction of accretion energy radiated as luminosity
feedback_tdelay	PR	Time delay between sink formation and feedback
feedback_minmass	PR	
nbody_frac	PR	Fraction of mass accreted before switching to N-body
ptemp0	PR	Disc temperature at r = 1 AU from star (K)
temp_inf	PR	Disc temperature at infinity (K)
ptemp_r0	PR	Temperature softening radius ($\ll$ 1 AU)
ptemp_q	PR	Temperature power law index
fcolumn	PR	Column polytrope factor

Variable	Type	Description
nionallstep	int	Integer steps inbetween HEALPix walk
f1	PR	Integration step accuracy variable
f2	PR	HEALPix resolution factor
f3	PR	Temperature smoothing parameter
f4	PR	Density interpolation parameter
Tneut	PR	Temperarure of neutral gas
Tion	PR	Temperature of ionized gas
Xfrac	PR	Fraction of hydrogen
a_star	PR	Recombination coefficient
N_LyC	PR	No. of ionizing photons per second
rstatic	PR(1:3)	Location of single static ionizing source
lmax_hp	PR	Maximum allowed number of HEALPix levels

3 Generating initial conditions

SEREN contains a large number of small programs which can be used to generate initial conditions to run simulations. These programs are contained in the sub-directory /seren/ic and can be compiled. To compile any initial conditions program of some name ic_name, simply type

make ic_name

Some of the initial conditions programs require their own separate parameters file, a template of which can be found in the seren/datafiles sub-directory. These parameters files must be copied into the main seren run directory in order to be accessed by the initial conditions program. To run the initial conditions program, type

./ic_name

3.1 ic_BB

ic_BB sets-up the Boss-Bodenheimer test (Boss & Bodenheimer 1979), i.e. a uniform density sphere with an azimuthal density perturbation in solid-body rotation. Program reads in a unifrom density sphere of unit radius (centred at the origin), scales to the required density and radius, and then adds the azimuthal perturbation and a solid-body velocity field. The original Boss-Bodenheimer test considered simply an isothermal EOS, but many subsequent studies have used barotropic and other EOSs. Parameters read in from file BBparams.dat.

Required Makefile options :

- NDIM = 3
- SPH = STANDARD/GRAD_H_SPH
- THERMAL = ISOTHERMAL/BAROTROPIC/RAD_WS
- HYDRO = 1
- GRAVITY = KS
- DIMENSIONLESS = 0

Variable	Type	Description
in_file	char(256)	Input filename (file containing uniform density sphere of unit radius)
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
mass	PR	Mass of cloud
munit	char(256)	Mass unit
rcloud	PR	Radius of cloud
runit	char(256)	Length unit
temp_cloud	PR	Temperature of cloud
angvel	PR	Angular velocity of cloud
angvelunit	char(256)	Angular velocity unit
mpert	integer	Order of azimuthal perturbation (usually mpert=2)
amp	PR	Amplitude of density perturbation (usually 0.1 or 0.5)

3.2 ic_core

ic_core creates a spherically symmetric density distribution for any given density function (as a function of radial distance). Currently only contains the distribution for a plummer-like density profile and a radial power-law density function. Requires the params file core.dat.

Required Makefile options :

- NDIM = 3
- HYDRO = 1
- GRAVITY = KS
- DIMENSIONLESS = 0

3.3 ic_jeans

ic_jeans sets-up the Jeans instability test (Hubber et al. 2006) which tests the ability of SEREN to resolve the Jeans gravitational instability. Program reads in a relaxed unit cube (with the cube placed in positive octant) and stretches the particle distribution to produce a 1-D sinusoidal density perurbation. Currently reads in parameters from the command line rather than via a separate parameters file.

Required Makefile options :

- NDIM = 3
- PERIODIC = 1
- PERIODIC_X = 1
- PERIODIC_Y = 1
- PERIODIC_Z = 1
- SPH = STANDARD/GRAD_H_SPH
- THERMAL = ISOTHERMAL
- HYDRO = 1
- GRAVITY = KS
- EWALD = 1
- DIMENSIONLESS = 1

Variable	Type	Description
in_file	char(256)	Input filename (File containing unit-uniform density sphere)
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
npert	int	No. of wavelengths
amp	PR	Amplitude of sinuosoidal perturbation

3.4 ic_KH

ic_KH creates the initial conditions for the Kelvin-Helmholtz instability test. Requires KHparams.dat file.

3.5 ic_lattice_cube

ic_lattice_cube creates a cubic-lattice distribution of particles with side-length length and ppd particles in each dimension. Therefore the total number of particles in the lattice is $\mathrm{ppd}^{\mathrm{NDIM}}$. In 1-D, the program produces a uniformly-spaced line of particles, in 2-D a square-grid of particles, and in 3-D a cubic lattice. The lattice extends from 0 to length in each dimension. Parameters are currently read in from the command-line.

Required Makefile options :

- NDIM = 1/2/3
- DIMENSIONLESS = 1

Variable	Type	Description
ppd	integer	Particles per dimension in lattice (Must be a positive integer)
length	PR	Total length of lattice edge (For a unit cube, length = 1)
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format

3.6 ic_NTSI

ic_NTSI generates the initial conditions for the non-linear thin-shell instability (NTSI) test. Requires the parameters file NTSIparams.dat.

3.7 ic_polytrope

Creates a finite polytrope/infinite polytrope with surrounding medium from a uniform-density sphere of unit radius centred at the origin. For an isothermal polytrope (e.g. a Bonner-Ebert sphere), the inputted sphere is divided into 4 regions; the polytrope (self-gravitating gas), the gas envelope (self-gravitating gas), the surrounding inter-cloud medium (non-self gravitating gas) and a static outer-wall (boundary particles). The outer-three regions are optional depending on the parameters selected in $\mathrm{polytrope.dat}$.

Required Makefile options :

- NDIM = 3
- THERMAL = ISOTHERMAL/POLYTROPIC/BAROTROPIC
- HYDRO = 1
- GRAVITY = KS
- DIMENSIONLESS = 0

Variable	Type	Description
in_file	char(256)	Input filename (File containing unit-uniform density sphere)
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
isocloud	logical	Flag true if isothermal polytrope (if true, THERMAL must equal ISOTHERMAL)
etapoly	PR	Polytropic index
xi_bound	PR	Dimensionless cloud boundary (6.35 for a mariginally stable Bonner-Ebert sphere)
mpoly	PR	Mass of cloud
munit	char(256)	Mass unit (e.g. $\mathrm{m\_sun}$)
rho0	PR	Central density of cloud (Only if $\mathrm{mflag} = \mathrm{rho0}$)
rhounit	char(256)	Density unit
mflag	char(20)	Set the total mass (mass) or central density (rho0) of the polytrope
Kpoly	PR	Polytropic constant, or a_0^2 for isothermal polytrope
vunit	char(256)	Velocity unit (unit of isothermal speed of gas if isothermal polytrope is selected)
menvelope	PR	Mass of gas envelope around polytrope (distributed uniformly around the polytrope with the same density and pressure as the polytrope at its surface)
micm	PR	Mass of IcM envelope which surrounds gas (distributed uniformly around the polytrope/gas envelope with the same density and pressure as the polytrope at its surface)
hboundary	PR	Size of static boundary zone (in units of the mean smoothing length; should be 3 or 4 to ensure no edge effects occur for interior gas particles)

3.8 ic_radtest

$\mathrm{ic\_radtest}$ creates the initial conditions to perform the Masunaga-Inutsuka test (Masunaga & Inutsuka ????) using the radiative cooling method of Stamatellos et al. (2007; RAD_WS option).

3.9 ic_random_cube

ic_random_cube creates a line, sheet or cube (depending on the dimensionality) of randomlly-placed particles. Distributes particles between 0 and length in each dimension. Parameters are currently read in from the command-line rather than a separate parameters file.

Required Makefile options :

- NDIM = 1/2/3
- DIMENSIONLESS = 1

Variable	Type	Description
ptot	int	Total number of particles
length	PR	Total length of lattice edge
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format

3.10 ic_replicate_cubes

Loads in a unit cube (from 0 to 1 in each dimension) and creates nrepeat periodic replicas in each dimension. The larger cube is then scaled to a unit cube itself. Used to create large-relaxed uniform density fields from smaller files. Parameters are read in from the command-line rather than a separate parameters file.

Required Makefile options :

- NDIM = 1/2/3
- DIMENSIONLESS = 1

Variable	Type	Description
in_file	char(256)	Input filename (File containing unit cube)
in_file_form	char(256)	Input file format
nrepeat	int	No. of replicas in each dimension (must be a positive integer)
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format

3.11 ic_RT

Generates initial conditions for Rayleigh-Taylor instability test. Prepares two layers of gas with different densities in hydrostatuc balance on top of each other with a sinusoidal density perturbation to seed the instability. A cubic grid of particles is generated rather than reading in a file. Parameters are read in from the file $\mathrm{RTparams.dat}$.

Required Makefile options :

- NDIM = 2
- PERIODIC = 1
- PERIODIC_X = 1
- PERIODIC_Y = 1
- THERMAL = ENERGY_EQN
- HYDRO = 1
- GRAVITY = 0
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
pertmode	char(20)	Perturbation mode (1=velocity, 2=boundary)
ppd1,ppd2	int	Particles per dimension
nlayers1,nlayers2	int	No. of layers of particles (in y-direction)
rho1,rho2	PR	Densities
Press1	PR	Pressure
acc_grav	PR	External y-gravitational acceleration
gamma	PR	Ratio of specific heats
xsize	PR	x..
amp	PR	Amplitude of y-velocity perturbation
lambda	PR	Wavelength of perturbation
pp_gather	PR	Required no. of SPH neighbours
hmin	PR	Minimum smoothing length
h_fac	PR	'grad-h' SPH factor

3.12 ic_sedov

Creates initial conditions for Sedov blast-wave test from inputted unit-uniform density sphere. Requires inputting a unit-sphere. A 'point-explosion' is added by giving the central particle and its neighbours a total energy of unity (weighted by the kernel from the centre, while the rest of the particles equally share an energy of total 10^{-6}. Parameters are read in from the file $\mathrm{sedovparams.dat}$.

Required Makefile options :

- NDIM = 3
- PERIODIC = 0
- PERIODIC_X = 0
- PERIODIC_Y = 0
- PERIODIC_Z = 0
- THERMAL = ENERGY_EQN
- HYDRO = 1
- GRAVITY = 0
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
in_file	char(256)	Input filename (File contains unit sphere)
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
rho0	PR	Density of sphere
radius	char(20)	Radius of sphere after rescaling

3.13 ic_shocktube

Generates initial conditions for general 2-part shocktube tests (e.g. Sod 1978). Reads in two relaxed cubic density distribution, creates periodic replicas in the x-direction to elongate the shocktube and sets particle properties to create the desired test problem. Parameters are read in from the file $\mathrm{sodparams.dat}$.

Required Makefile options :

- NDIM = 1/2/3
- PERIODIC = 1
- PERIODIC_X = 1
- PERIODIC_Y = 1
- PERIODIC_Z = 1
- THERMAL = ISOTHERMAL/ENERGY_EQN
- HYDRO = 1
- ARTIFICIAL_VISCOSITY = AB/MON97
- GRAVITY = 0
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
file1	char(256)	Input filename
file1_form	char(256)	Input file format
file2	char(256)	Input filename
file2_form	char(256)	Input file format
p1, p2	int, int	No. of particles in file 1, 2
n1, n2	int, int	No. of replicas for LHS/RHS
rho1, rho2	PR, PR	Density of LHS/RHS layers
Press1, Press2	PR, PR	Pressure for LHS/RHS
x1, x2	PR, PR	x
y1, y2	PR, PR	y
z1, z2	PR, PR	z
v1(1), v2(1)	PR, PR	vx
v1(2), v2(2)	PR, PR	vy
v1(3), v2(3)	PR, PR	vz
B1(1), B2(1)	PR, PR	Bx
B1(2), B2(2)	PR, PR	By
B1(3), B2(3)	PR, PR	Bz

3.14 ic_sphere

Creates a spherical distribution of particles of unit radius and centred on the origin containing an exact number of particles. First, loads in a unit cube and then iterates to find the radius that contains the correct number of particles. Finally the spherical cut is rescaled and placed at the origin. Will fail to find the required number of particles if the inputted unit cube has too few particles. Sphere parameters are read in from the file sphereparams.dat.

Required Makefile options :

- NDIM = 3
- PERIODIC = 0
- PERIODIC_X = 0
- PERIODIC_Y = 0
- PERIODIC_Z = 0
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
in_file	char(256)	Input filename
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
rcloud	PR	Required radius of sphere
nwant	int	Required number of particles in sphere

3.15 ic_vel_pert.F90

Adds a variety of perturbations to any inputted (spherical) density distribution. Requires parameters file velpert.dat.

Required Makefile options :

- NDIM = 3
- DIMENSIONLESS = 0

Variable	Type	Description
in_file	char(256)	Input filename
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
densmode	char(20)	Mode of density perturbation (not used yet)
amp	PR	Amplitude of azimuthal perturbation (not used yet)
mpert	integer	Azimuthal perturbation mode (not used yet)
fenhance	PR	Density enhancement factor (increase all particle masses by fenhance; used to make stable polytropes unstable)
vpower	PR	Turbulent velocity power spectrum index
eturb	PR	Ratio of turbulent to gravitational energy
ngrid	integer	No. of grid points for vel field (determines resolution of velocity field; must be a multiple of 2)
iseed1	integer	Random No. seed 1
iseed2	integer	Random No. seed 2
velradmode	char(20)	Radial velocity mode (energy, dvdr or none)
dvdr	PR	Radial velocity gradient
erad	PR	Ratio of radial kinetic to gravitational energy
velrotmode	char(20)	Rotational mode (energy, angmom, angvel or none)
angmom	PR	Total angular momentum (if velrotmode = angmom)
angmomunit	char(256)	angular momentum unit
angpower	PR	Angular velocity power law (angular velocity is a function of axial distance, $\omega \propto r^{angpower}$)
angvel	PR	Angular velocity
angvelunit	char(256)	Angular velocity unit
erot	PR	Ratio of rotational kinetic energy to gravitational energy

4 Running the SEREN bash test script

SEREN contains a bash script designed to run batches of tests of SEREN for development and debugging purposes. The script, and all related files for running the tests, is located in the /seren/testsuite sub-directory. In the testsuite directory, there is the test-seren.sh bash script and further sub-directories which contain files used by test-seren.sh when performing batch tests.

A test is launched from the command line as in the following example :

./test-seren.sh -gfortran -openmp -debug1 -test POLYRAD1-AB

The current list of command line options for the script are (**TBD**) :
The list of tests currently set-up for use with the test script are (**TBD**) :

Table 5: List of automated tests in SEREN

Test name	**Description**
ADSOD-3D	Classic SOD test of two initially static columns of gas in contact which then interact forming a shock. Gas is non-radiative so the energy equation is solved and no energy escapes the system (i.e. it is adiabatic).
BURRAU1	Burrau 3-body problem (Burrau 19??); also known as the Pythagorean problem. Three stars with masses 3, 4 and 5 placed on the corners of a right-angled triangle all with zero-velocity and allowed to evolve until the system dissolves into a single star and a binary star.
COL-3D	Two columns of uniform density gas collide supersonically to produce a dense shocked layer.
EIGEN1	Gravitational force accuracy using eigenvalue MAC
FIGURE8	Figure-8 3-body test for N-body integrator (????).
FREEFALL1	Free-fall collapse test.
GEO1	Gravitational force accuracy using geometric MAC
ISOFREEFALL1	Isothermal free-fall collapse test
KH-2D	2D Kelvin-Helmholtz instability test
NTSI-2D	2D Non-linear thin shell instability test
POLYRAD1	Masunaga & Inutsuka (????) collapse test
SEDOV-3D	Sedov blast wave test (Sedov 19??).
SHEAR-2D	2-D shearing layer test.
SIT1-AB	A variation of the Boss-Bodenheimer (1979) test. A uniform-density spherical cloud is given a sinusoidal azimuthal density perturbation and a solid-body rotationaal velocity field such that it collapses to form a dense filament with a star on each end and eventually bound binary system.
STATPOLY1	Relax a polytropic gas to hydrostatic balance.

5 Coding style of SEREN

5.1 Design philosophy of SEREN

SEREN is a highly modular code written in Fortran 90 which comprises of several layers of subroutine calls in performing basic simulations. Each subroutine is designed to perform one single task. If a long procedure consists of a number of independent steps (i.e. not using the same local variables), then it is broken down into a sequence of smaller subroutines. Also, each .F90 file contains one single subroutine (with the exception of sanitycheck.F90 which has two extra smaller subroutines for clarity).

For the benefit of anyone reading through the source code, or for those wishing to develop new routines, we discuss here in detail some of the more important coding conventions that are used in SEREN . We do not discuss the particular features of any one subroutine (since each routine is extensively commented), but focus on the style used in most subroutines of SEREN .

5.2 Macros

SEREN uses C-like macros throughout the source code, both for the clarity (by reducing the number of lines) and the efficiency and runtime speed of the code. Macros are strings (conventionally in upper case as in C) which are substituted for some user-defined value or expression by the *pre-processor*, i.e. before the compiler generates machine code from the source code. This can improve the runtime performance somewhat by removing common variable references.

Macros are defined in two separate locations in SEREN . Some are defined in the Makefile (e.g. NDIM). Most macros however are defined in the header file /headers/macros.h. In order to make use of the macros, we must import the file /headers/macros.h into the subroutine by way of the pre-processor command #include "macros.h". The majority of macros in SEREN are straight-forward numerical substitutions of important information, such as array sizes or physical constants.

5.2.1 Function-like macros

Function-like macros are macros that look like functions/subroutines by their syntax, but work by the substitution of a string of commands, rather than calling a subroutine elsewhere in memory (thereby eliminating the extra cost associated with a subroutine call). In SEREN , we use function-like macros as a compact and concise way of writing debugging information to the screen when in debug mode. For example, we define the debug1 macro in the following way.

```
#ifdef DEBUG1
   #define debug1(x) write (6,*) x
   #else
   #define debug1(x)
   #endif
```

If we wished to write debug information to screen (e.g. in order to indicate the current location in the code), we could write in long-hand:

```
#ifdef DEBUG1
   write(6,*) "Calculating smoothing lengths"
   #endif
```

In SEREN , we can instead write the short-hand form

```
debug1("Calculating smoothing lengths")
```

If the DEBUG1 compiler flag is defined in the Makefile, then the debug1() macro is replaced with write(6,*) "Calculating smoothing lengths". If DEBUG1 is not defined in the Makefile, then debug1() macro is replaced with nothing. For subroutines (particularly those in development) that contain many debugging statements, these macros allow us to write code with more clarity and fewer lines. We use four levels of debug macros, which are all defined in /headers/macros.h.

5.3 Real variable types

Rather than hard-wiring in the precision of real variables in the source code, SEREN allows the user to specify the precision through one of the options in the Makefile (PRECISION). The precision is controlled by several lines in the module definitions (in modules.F90) :

```
    integer, parameter :: DP = selected_real_kind(p=15)
    integer, parameter :: SP = selected_real_kind(p=6)
#if defined(DOUBLE_PRECISION)
    integer, parameter :: PR = DP
#else
    integer, parameter :: PR = SP
#endif
```

The first two lines use the intrinsic selected_real_kind function to define the precision independent of the processor type (i.e. whether it is 32-bit or 64-bit). The conditional compilation section then defines the precision used in the code (i.e. PR) depending on the option selected in the Makefile. Any real variable in the code must be defined in the following way, e.g.

```
    real(kind=PR) :: drmag
```

If we wish to declare a double or single precision variable irrespective of the general precision (e.g. any summation variables in /main/diagnostics.F90 always use double precision), then we use DP or SP in place of PR.

If we wish to convert a variable to a real variable of required precision, we must specify the kind (i.e. PR, DP or SP) as a second parameter in the real function, e.g. to convert the integer variable i to a real variable of precision PR, we write

```
    ireal = real(i,PR)
```

5.4 Particle data arrays

SEREN mainly uses simple arrays to store particle data. However, data which are important in gravity calculations are stored differently. The position, mass and smoothing length information are grouped together in a single array, parray(1:NDIM+2,1:ptot). The position of particle p is stored in the elements parray(1:NDIM,p), the mass is stored in the element parray(MASS,p), and the smoothing length is stored in the element parray(SMOO,p) (See /seren/headers/macros.h for macro definitions).

5.5 Particle types

SEREN accomodates the following particle types:

- Static boundary particles (pboundary)
- Non-gravitating inter-cloud medium (IcM) particles (picm)
- Self-gravitating gas particles (pgas)
- Dark-matter particles (pcdm)
- Dust particles (pdust)
- Ion particles (pion)
- Sink particles (stot)

where the variable names indicate the number of each particle type present in the simulation. All data for the first three (boundary, IcM and self-gravitating gas particles) are stored in the main arrays, which contain ptot elements where ptot = pboundary + picm + pgas + pcdm. The data is stored such that the first pboundary elements contain the information for boundary particles, the next picm elements contain the information for

the IcM particles, and the next pgas elements contain the information for the gas particles, and the final pcdm elements contain the information for the cdm particles. Although provision has been made for their use in future versions of SEREN , dust and ion particles are not currently active.

When using different particle types, care must be taken when looping over particles in the code. If all particles are self-gravitating gas particles, then all loops would simply range from 1 to ptot, but with different types particles this is not always the case. Consider some common examples:

- Hydro forces - We only need to loop over all IcM and gas particles (since boundary particles do not move). We therefore only loop from the first IcM particle (i.e. $\mathrm{p} = \mathrm{pboundary} + 1$) to the very last gas particle (i.e. $\mathrm{p} = \mathrm{ptot}$). We therefore define the variable

  $\mathrm{phydrostart} = \mathrm{pboundary} + 1$
  (in the subroutine /main/types.F90). Any loops over hydro particles will thus range from phydrostart to ptot.

- Gravitational forces - We only need to loop over all self-gravitating gas particles. As with hydro forces, we define the variable

  $\mathrm{pgravitystart} = \mathrm{pboundary} + \mathrm{picm} + 1$
  in /main/types.F90 which points to the first gas particle. Any loops over gravitating particles will thus range from pgravitystart to ptot.

The sink particles are stored in separate data structures, since they can have many additional properties that are not possessed by normal SPH particles and thus require their own data structures. We use Fortran types (equivalent to C structures) to hold sink data. The main array that contains each sink structure is called sinkdata and elements can be accessed using the Fortran % notation (e.g. the mass element of sink s is $\mathrm{sinkdata(s)\%m}$).

6 Units

Dimensionless units are used in numerical simulations so that all values are as close to unity as possible, to avoid having very large or very small values that may result in significant rounding errors. SEREN contains a flexible system of units which allows the user to select from a wide range of commonly used astrophysical units, or easily construct their own set of units. All variables related to units and scaling are determined in units.F90.

SEREN uses a slightly different approach to scaling than DRAGON (which may cause some initial confusion since some variables with common names between the two have different functions). Each quantity, X, has three scaling variables associated with it: Xunit, Xscale and X_SI.

- Xunit is a string which contains the name of the unit that the quantity X is measured in; e.g. for length units, runit may take the values pc, au, r_sun, etc. All Xunit strings are defined in the parameters file. The available options in version 1.0 of the code are given in the following table:
- Xscale is a real variable that allows us to convert between physical and code units. In order to convert any variable from physical to code units (where the physical variable is measured in units specified by Xunit), then we divide the physical unit by Xscale. Conversely, to convert any code variable to physical units, we multiply the code value by Xscale
- X_SI is a real variable that allows us to convert between the unit specified by Xunit and S.I. units. In order to convert from Xunit to S.I. units, we multiply the variable (in units of Xunit) by X_SI.

In a self-gravitating code like SEREN , we choose a set of units so as to make the gravitational constant G equal to unity. As with DRAGON, we are free to choose the values of rscale and mscale. The value of tscale is then set to ensure $\mathrm{G} = 1$ in the new system of units. Therefore, the value of tscale can be obtained using

$$tscale \times t\_SI = \frac{(rscale \times r\_SI)^{3/2}}{\sqrt{G \times mscale \times m\_SI}} \tag{1}$$

where G is the gravitational constant in c.g.s. units, i.e. $\mathrm{G} = 6.67 \times 10^{-8}\,\mathrm{cm^3\,g^{-1}\,s^{-2}}$. All other scaling factors can be determined in a similar way using:

$$vscale \times v\_SI = \frac{rscale \times r\_SI}{tscale \times t\_SI} \tag{2}$$

$$ascale \times a\_SI = \frac{rscale \times r\_SI}{(tscale \times t\_SI)^2} \tag{3}$$

$$rhoscale \times rho\_SI = \frac{mscale \times m\_SI}{(rscale \times r\_SI)^3} \tag{4}$$

$$sigmascale \times sigma\_SI = \frac{mscale \times m\_SI}{(rscale \times r\_SI)^2} \tag{5}$$

$$Pscale \times P\_SI = \frac{mscale \times m\_SI}{rscale \times r\_SI \times (tscale \times t\_SI)^2} \tag{6}$$

$$fscale \times f\_SI = \frac{mscale \times m\_SI \times rscale \times r\_SI}{(tscale \times t\_SI)^2} \tag{7}$$

$$Escale \times E\_SI = \frac{mscale \times m\_SI \times rscale \times r\_SI}{(tscale \times t\_SI)^2} \tag{8}$$

$$momscale \times mom\_SI = \frac{mscale \times m\_SI \times rscale \times r\_SI}{tscale \times t\_SI} \tag{9}$$

$$angmomscale \times angmom\_SI = \frac{mscale \times m\_SI \times rscale^2 \times r\_SI^2}{tscale \times t\_SI} \tag{10}$$

$$dmdtscale \times dmdt\_SI = \frac{mscale \times m\_SI}{tscale \times t\_SI} \tag{11}$$

$$Lscale \times L\_SI = \frac{Escale \times E\_SI}{tscale \times t\_SI} \tag{12}$$

$$kappascale \times kappa\_SI = \frac{(rscale \times r\_SI)^2}{mscale \times m\_SI} \tag{13}$$

In MHD, we must also introduce the unit of charge and associated units such as magnetic field and current density. As with gravitational problems, we can scale the units of the magnetic field such that the permiability of free space, μ_0, is equal to unity. **to be completed**.

$$Jscale \times J\_SI = \frac{Qscale \times Q\_SI}{tscale \times t\_SI \times rscale^2 \times r\_SI^2} \tag{14}$$

Table 6: List of unit options in SEREN

Xunit	**Options**	**Description**
runit	mpc	megaparsecs
	kpc	kiloparsecs
	pc	parsecs
	au	astronomical units
	r_sun	solar radii
	r_earth	Earth radii
	km	kilometres
	m	metres
	cm	centimetres
munit	m_sun	solar masses
	m_jup	Jupiter masses
	m_earth	Earth masses
	kg	kilograms
	g	grams
tunit	gyr	gigayears
	myr	megayears
	yr	years
	day	days
	sec	seconds
vunit	km_s	kilometres per second
	au_yr	astronomical units per year
	m_s	metres per second
	cm_s	centimetres per second
aunit	km_s2	kilometres per second squared
	au_yr2	astronomical units per year squared
	m_s2	metres per second squared
	cm_s2	centimetres per second squared
rhounit	m_sun_pc3	solar masses per cubic parsec
	g_cm3	grams per cubic centimetre
sigmaunit	m_sun_pc2	solar masses per parsec squared
	g_cm2	grams per centimetre squared
Punit	Pa	pascals
	bar	bars

Xunit	Options	Description
	g_cms2	grams per centimetre per second squared
funit	N	newtons
	dyn	dynes
Eunit	J	joules
	erg	ergs
	GJ	gigajoules
momunit	m_sunkm_s	solar masses kilometres per second
	m_sunau_yr	solar masses astronomical units per year
	kgm_s	kilomgram metres per second
angmomunit	kgm2_s	kilogram metres squared per second
	gcm2_s	gram centimetres squared per second
angvelunit	rad_s	radians per second
dmdtunit	m_sun_myr	solar masses per megayear
	m_sun_yr	solar masses per year
	kg_s	kilograms per second
	g_s	grams per second
Lunit	L_sun	solar luminosities
kappaunit	m2_kg	metre squared per kilogram
	cm_g	centimetre per gram
Bunit	tesla	tesla
	gauss	gauss
Qunit	C	coulombs
Junit	C_m2_s	coulombs per second per metre squared
uunit	J_kg	Joules per kilogram
	erg_g	ergs per gramme

7 File formats

SEREN 1.0.0 uses both the DRAGON file format and the native SEREN file format for reading in initial conditions and writing out snapshots. Unlike in DRAGON, the format of the initial conditions file need not be the same as the format of the output snapshots. This is controlled by the two input parameters in the parameters file, in_file_form and out_file_form. The possible values for these parameters are :

- dragon_form - Formatted (ASCII) DRAGON snapshot files
- dragon_unform - Unformatted (binary) DRAGON snapshot files
- seren_form - Formatted (ASCII) SEREN snapshot files
- seren_unform - Unformatted (binary) SEREN snapshot files (Not yet working)

As well as standard snapshot files, SEREN can also produce simple ASCII output which is useful for debugging purposes.

8 Structure of code

8.1 Basic directory structure

Subroutines in SEREN are not all contained in a single source directory, but are grouped in several sub-directories depending on their purpose. In version 1.0, the following sub-directories exist :

- /seren/src/advance - integration routines
- /seren/src/analyse - analysis routines
- /seren/src/BHtree - Barnes-Hut octal tree subroutines
- /seren/src/binarytree - Binary-number tree subroutines
- /seren/datafiles - Contains initial conditions parameters files
- /seren/docs - Contains latest version of the userguide
- /seren/src/gravity - subroutines that calculate gravitational forces
- /seren/src/headers - macro and modules files
- /seren/src/healpix - HEALPix ioniaztion routines
- /seren/src/ic - programs to generate initial conditions for regularly used configurations (e.g. relaxed rectangular cubes, spheres)
- /seren/src/io - subroutines that read and write files
- /seren/src/main - contains important and commonly used subroutines
- /seren/src/mhd - contains magneto-hydrodynamics routines
- /seren/src/nbody - N-body routines
- /seren/src/nbody_sim - N-body simulation subroutines
- /seren/src/nbody_sph_sim - Hybrid N-body/SPH simulation routines
- /seren/src/radiation - contains radiation transfer subroutines
- /seren/src/setup - contains subroutines called during initialization of SEREN .
- /seren/src/sinks - subroutines that search for, create and advance sink particles.

- /seren/src/sorts - subroutines for sorting lists
- /seren/src/sph - subroutines that perform important SPH functions
- /seren/src/sph_sim - SPH simulation routines (e.g. h-finding, neighbour searching)
- /seren/src/tests - test programs
- /seren/src/timestep - timestepping routines
- /seren/testsuite - bash script for running batch tests of seren

9 Variable conventions

In SEREN , the names of all commonly used local variables are kept as consistent as possible between different subroutines. Here we list the names of all common local integer and real variables.

9.1 Integer variables

c	Cell identifier
cc	Child cell identifier
i	Auxiliary counter (often used when looping over neighbour lists)
k	Dimension counter
l	Level counter (for BH tree and HEALPix)
p	Particle identifier
pp	Neighbour identifier
pp_pot	No. of potential neighbours (e.g. after a tree walk)
pp_templist(1:pp_limit)	Temporary list of neighbour identifiers
pp_tot	Total number of neighbours for particle p
s	Sink particle identifier
ss	Secondary sink counter

9.2 Real variables

dr(1:NDIM)	Relative position vector
drmag	Distance
drsqd	Distance squared
dr_unit(1:NDIM)	Unit position vector
hp	Smoothing length of particle p
hpp	Smoothing length of neighbouring particle pp
mp	Mass of particle p
mpp	Mass of neighbouring particle pp
ms	Mass of sink particle s
invdrmag	Reciprocal of distance, i.e. 1 / drmag
invdrsqd	Reciprocal of distance squared, i.e. 1 / drsqd
invhp	Reciprical of smoothing length of p, i.e. 1 / hp
invhpp	Reciprical of smoothing length of pp, i.e. 1 / hpp
qc(1:NQUAD)	Quadrupole moment tensor for cell c
rp(1:NDIM)	Position of particle p
rpp(1:NDIM)	Position of neighbouring particle pp
rs(1:NDIM)	Position of sink particle s
sound_p	Sound speed of particle p
sound_pp	Sound speed of neighbouring particle pp
up	Specific internal energy of particle p
upp	Specific internal energy of neighbouring particle pp
vp(1:NDIM)	Velocity of particle p
vpp(1:NDIM)	Velocity of neighbouring particle pp
vs(1:NDIM)	Velocity of sink particle s