`closures-2d`
Design Documentation

C. Kristopher Garrett, Tim Shaffer

July 30, 2015

Contents

1 Purpose

This software implements various numerical methods for approximating the angular variable in kinetic transport in order to study performance, accuracy, and robustness on common test problems. Whereas fluid equations describe a vector of quantities dependent on space, kinetic transport equations describe a scalar density of particles depending on both space and velocity. Such numerical methods occur frequently in production codes modeling certain physical systems, such as neutron transport inside a reactor, neutrino transport in supernova simulations, photon transport, and rarefied gas dynamics. Rather than simulating a specific physical system, this software solves simpler problems that capture issues commonly encountered in full simulations. Comparing the performance of these numerical methods on our simplified problems can give researchers insight into the behavior of similar methods in more complicated implementations.

In addition to evaluating the methods themselves, this software is written to allow study of performance with modern high-performance computing resources. This software optionally uses OpenMP and Message Passing Interface (MPI) to carry out parallel and concurrent computations. Studying the behavior of numerical methods with such acceleration techniques provides reference for implementations on large-scale problems.

This software also supports profiling measurements such as timings, flop rates, memory rates, and cache misses. Using statistics about runtime performance, we examined code optimizations, bottlenecks, etc. that occur in our implementations.

2 Problem

This software simulates particles with unit speed that scatter isotropically according to a simple kinetic equation. The equation governing particle behavior takes the form

$$\partial_t f + \Omega \cdot \nabla_x f + \sigma_t f = \frac{\sigma_s}{4\pi} \langle f \rangle \tag{1}$$

where $f(x, \Omega, t)$ is the density of particles, $x \in \mathcal{D} \subset \mathbb{R}^2$ is position, $\Omega \in \mathbb{S}^2 \subset \mathbb{R}^3$ (the unit sphere) is the direction of travel, $\sigma_t(x) \geq \sigma_s(x)$ are the total and scattering cross sections, and $\langle \cdot \rangle$ is shorthand for integration over $\mathbb{S}^2$. The boundary conditions implemented are either zero inflow boundary conditions or periodic boundary conditions depending on the initial condition discussed in Section 2.1.

2.1 Initial Conditions

The following initial conditions are implemented in this software.

2.1.1 Gaussian Initial Condition

In the Gaussian initial condition, a two dimensional Gaussian function is placed with its peak at the center of the grid. Each point on the initial grid is set to

$$f(x, y, \Omega, t = 0) = \max\left(\frac{1}{2\pi\sigma_g^2} e^{-(x^2+y^2)/(2\sigma_g^2)}, \texttt{floor}\right), \tag{2}$$

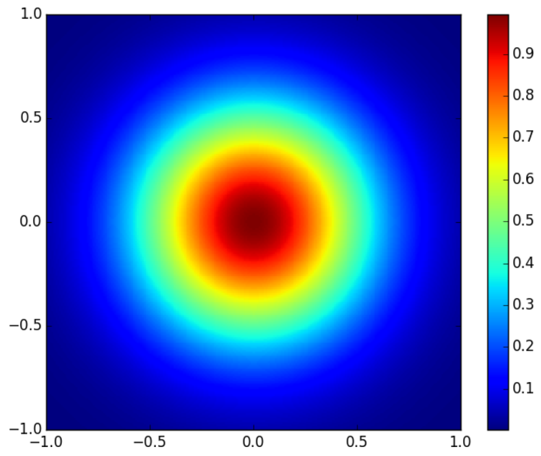

Figure 1: Gaussian Initial Condition

where σ_g is set by the user. The cross sections σ_s and σ_t are constants defined by the user. The zero inflow boundary condition is used. Figure 1 shows the initial condition of $\langle f \rangle$ for $\sigma_g = 0.4$.

2.1.2 Delta Initial Condition

The limiting case, as $\sigma \to 0$, is the Delta initial condition. This condition is meant to simulate an initial pulse of particles distributed isotropically along an infinite line in space. The points on the initial grid are initialized to

$$f(x, y, \Omega, t = 0) = \frac{1}{4\pi}\delta(x, y), \tag{3}$$

approximating an infinitely narrow source at the origin. The source doesn't have the $\frac{1}{4\pi}$ factor, is it introduced by scaling later? σ_S and σ_T are uniformly set to the configured value for sigma.

2.1.3 Lattice Initial Condition

The Lattice initial condition corresponds to a checker board pattern of highly scattering and highly absorbing regions [2, 3]. This configuration is reminiscent of a small section of a nuclear reactor core. The Lattice initial condition leaves the grid initially empty, but has the most complicated scattering pattern of the initial conditions. σ_S and σ_T are set to 1 at all positions, except for several blocks arranged throughout the grid at which $\sigma_S = 0$ and $\sigma_T = 10$.

2.1.4 Smooth Initial Condition

The Smooth initial condition is primarily intended for testing convergence. This configuration initializes the grid points with a periodic boundary. Each point in

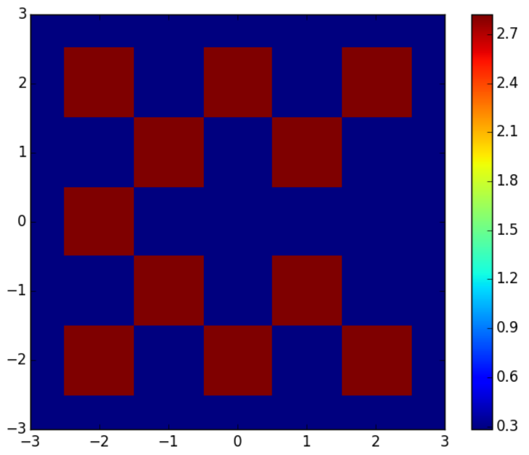

Figure 2: σ_T for Lattice Initial Condition

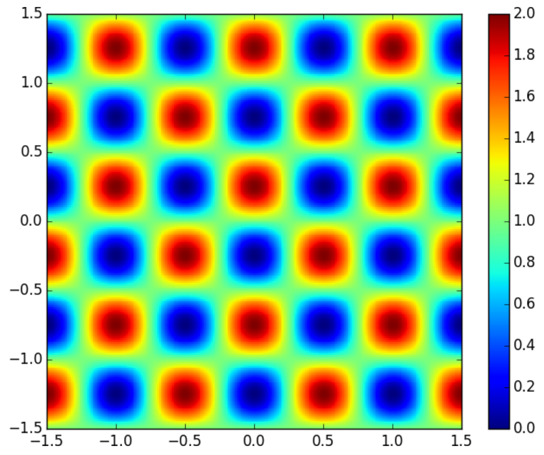

Figure 3: Smooth Initial Condition

the initial grid is set to

$$f(x, y, \Omega, t = 0) = 1 + \sin(2\pi x)\cos(2\pi y). \tag{4}$$

σ_S and σ_T are uniformly set to the configured value for sigma.

3 Angular Approximations

3.1 Discrete Ordinates

The `kinetic` solver is an implementation of the discrete ordinates method, also known as S_N. Our implementation uses a Chebyshev-Legendre quadrature on the unit sphere. A Lebedev quadrature is also implemented for this solver, but is considered experimental. Details of the quadratures are given in Section 4.1. Let $\{\Omega_1, \dots, \Omega_Q\} \in \mathbb{S}^2$ be be a set of nodes with corresponding weights $\{w_1, \dots, w_Q\}$. Then from Equation 1

$$\partial_t f_q + \Omega_q \cdot \nabla_x f_q + \sigma_t f_q = \frac{\sigma_s}{4\pi} \sum_{q'=1}^{Q} w_{q'} f_{q'}, \tag{5}$$

where $f_q(x, t) \approx f(x, \Omega_q, t)$ for $q = 1, \dots, Q$.

This implementation uses Heun's method to achieve second order convergence. Edge values are computed via upwinding. For approximate slopes, the double minmod limiter is used.

3.2 Moment Solvers

Show basic setup. Then define $\mathcal{E}(\mathbf{u})$ for each type. The `moment` solver uses standard spectral methods with Equation 1. The real spherical harmonics serve as an orthonormal basis of L^2 with respect to $\mathbb{S}^2$ for the expansion of the moments. Let $\mathbf{m}(\Omega) = (Y_{0,0}, Y_{1,-1}, Y_{1,0}, Y_{1,1}, \dots, Y_{N,-N}, \dots, Y_{N,N})^T$ be a vector of spherical harmonics up to and including degree N. The moments with respect to $\mathbf{m}$ are given by

$$\mathbf{u}^{\text{exact}}(\mathbf{x}, t) = \langle \mathbf{m}(\Omega) f(\mathbf{x}, \Omega, t) \rangle \tag{6}$$

since $u_{\ell,m} = f_{\ell,m}$ and the collision operator is diagonalized. Separate exact u from approximate u. The exact moment system is given by

$$\partial_t \mathbf{u} + \nabla_x \cdot \langle \Omega \mathbf{m} f \rangle = \mathrm{D}\mathbf{u}, \tag{7}$$

but this formulation is not closed. Thus f is replaced with a P_N moment closure $\mathcal{E}(\mathbf{u})$ such that $\langle \mathbf{m}\mathcal{E}(\mathbf{u}) \rangle = \mathbf{u}$. In this case,

$$\mathcal{E}(\mathbf{u}) = \sum_k u_k m_k = \mathbf{u}^T \mathbf{m} \tag{8}$$

yielding the closed moment system

$$\partial_t \mathbf{u} + \nabla_x \cdot \langle \Omega \mathbf{m} \mathcal{E}(\mathbf{u}) \rangle = \mathrm{D}\mathbf{u}. \tag{9}$$

It is only necessary to use the spherical harmonics $Y_{\ell,m}$ such that $\ell + m$ is even. State why...

3.2.1 Entropy Minimization

Suppose $\mathcal{E}(\mathbf{u})$ is obtained as an entropy minimization

$$\mathcal{E}(\mathbf{u}) = \arg\min_{g \in L^1} \langle \eta(g) \rangle \quad \text{subject to} \quad \langle \mathbf{m} g \rangle = \mathbf{u} \tag{10}$$

Fill in the rest here.

3.2.2 $\mathbf{P}_N$

$\mathcal{E}(\mathbf{u}) = \mathbf{m}^T \mathbf{u}$. Can get this by (1) truncating, (2) least squares, (3) minimizing entropy. Fill in a little bit.

Various filters can also be applied to suppress oscillations in spherical harmonics.

- The Hauck filter is described in [4]. Let N be the moment order and ω be the filter tune, and define
$$\alpha = \frac{\omega}{N^2(\sigma_T L + N)^2}. \tag{11}$$
Scale each moment by
$$\frac{1}{1 + \alpha n^2 (n+1)^2} \tag{12}$$
for the n^{th} moment.

- The Spline filter is given in [5]. With moment order N and filter tune σ_e, let
$$s = \frac{-\sigma_e \Delta t}{\log F(N/(N+1))} \tag{13}$$
where $F(x) = 1/(1 + x^4)$. The scale factor is given by
$$F(n/(N+1))^s. \tag{14}$$

- The Lanczos filter is taken from [5]. Again with moment order N and filter tune σ_e, let
$$s = \frac{-\sigma_e \Delta t}{\log L(N/(N+1))} \tag{15}$$
where
$$L(x) = \begin{cases} 1 & \text{when } x = 0 \\ \dfrac{\sin x}{x} & \text{otherwise.} \end{cases} \tag{16}$$
Now the moment scale factor is
$$L(n/(N+1))^s. \tag{17}$$

3.3 M_N and PP_N

One serious drawback of the P_N method is the possibility of non-realizable solutions with negative densities, oscillatory approximations of nonsmooth solutions. Can just put the definitions for η and other stuff here for M_N and PP_N.

Nonlinear approaches can ensure positivity, but come with increased complexity and computational cost. In general, entropy-based moment closures, like the `momopt` solver, can be cast in the framework of the following minimization problem

$$\mathcal{E}(\mathbf{u}) = \underset{g \in L^1}{\arg\min} \langle \eta(g) \rangle \quad \text{subject to} \quad \langle \mathbf{m} g \rangle = \mathbf{u} \tag{18}$$

where η is a smooth, strictly convex, coercive[1] function. For the `momopt` solver, $\eta(\mathbf{r}) = \mathbf{r} \log \mathbf{r} - \mathbf{r}$ and its Legendre dual[2] $\eta * (\mathbf{s}) = \mathbf{e}^{\mathbf{s}}$.

The solution to Equation 18, if it exists, is given by

$$\mathcal{E}(\mathbf{u}) = \eta_*'(\hat{\alpha}(\mathbf{u})^T \mathbf{m}) \tag{19}$$

where η_* is the Legendre dual of η and $\hat{\alpha}(\mathbf{u})$ solves the dual problem

$$\hat{\alpha}(\mathbf{u}) = \underset{\alpha \in \mathbb{R}^n}{\arg\min} \left\{ \langle \eta_*(\alpha^T \mathbf{m}) \rangle - \alpha^T \mathbf{u} \right\}. \tag{20}$$

4 Implementation

4.1 Quadratures

Let's go over this. Some of the wording is not quite right. The Chebyshev-Legendre quadrature [1] is used for numerical integration over $\mathbb{S}^2$ in all of the solvers. This quadrature is constructed from an n point Gauss-Legendre rule, which exactly integrates smooth functions of order less than $2n-1$. Since this is a 2D code, it is only necessary to integrate over the upper half of $\mathbb{S}^2$, which is divided into n layers along the z axis. Abscissae and weights from GSL are arranged in a circle around $\mathbb{S}^2$ at each layer, giving n^2 points on the upper half of $\mathbb{S}^2$. The Chebyshev-Legendre quadrature is not optimal with respect to number of points, but is simple to implement. The Lebedev quadrature, for example, is optimal in number of points, and an experimental implementation is available for the `kinetic` solver. This quadrature, however, uses a more complicated arrangement of points, making exploiting symmetries in the integral more difficult. In addition, the points of the Chebyshev-Legendre quadrature are simply the Cartesian product of n angles spaced around a circle and n values of z. This structure could allow the evaluation of integrals via the Chebyshev-Legendre quadrature to be optimized more easily than the Lebedev quadrature, which is arranged nontrivially and becomes denser with increasing order.

[1] We define a function η to be coercive if $\lim_{r\to\infty} \frac{\eta(r)}{|r|} = \infty$

[2] The Legendre dual of η is given by $\eta_*(s) = rs - \eta(r)$ where $s = \eta'(r)$. By differentiating this relation, one can show that $r = \eta_*'(s)$. Thus η' and η_*' are inverses of each other.

The fixed-order Gauss-Legendre integration points and weights from GSL will be referred to as μ_i, and w_i, respectively. Next, the azimuthal angles of quadrature, ϕ_k, are calculated as

$$\phi_k = \frac{(k+0.5)\pi}{Q} \tag{21}$$

for $k \in \mathbb{Z}$ and quadrature order Q such that $0 \leq k < 2Q$, placing $2Q$ points evenly around the unit circle, and the points are mapped into cylindrical coordinates ξ_i, η_i.

For q_1, q_2 such that $0 \leq q_1 < Q/2$ and $0 \leq q_2 < 2Q$, let $i = 2q_1Q + q_2$. Now

$$W_i = \frac{2\pi w_{q_1}}{Q} \tag{22}$$

$$\xi_i = \sqrt{1-\mu_{q_1}^2}\cos\phi_{q_2} \tag{23}$$

$$\eta_i = \sqrt{1-\mu_{q_1}^2}\sin\phi_{q_2}. \tag{24}$$

Once the quadrature points and weights are computed, each integral over $\mathbb{S}^2$ can be calculated as

$$\int_{\mathbb{S}^2} f(x,y,\Omega,t)\,\mathrm{d}\Omega = \sum_{i=1}^{Q} W_i f(x,y,\omega_i,t) \tag{25}$$

where ω_i is the direction (ξ_i, η_i).

4.2 Flux Calculations

Is this worth including/expanding? All the solvers compute flux by examining values of neighboring cells in the x and y directions. The exact operations vary with each solver.

4.3 Time

Time steps are carried out via Heun's method which is an explicit two-stage Runge-Kutta method that is second order accurate. If $f(x,y,\Omega,t)$ is the density a a given position, direction, and time, then we apply two time steps in succession to obtain a first approximation $\tilde{f}(x,y,\Omega,t+2\Delta t)$. Now the calculated time step is the average of the approximation and the starting value, that is

$$f(x,y,\Omega,t+\Delta t) = \frac{1}{2}\Big(f(x,y,\Omega,t) + \tilde{f}(x,y,\Omega,t+2\Delta t)\Big). \tag{26}$$

Notice Heun's method is the average of two Euler steps which puts it into the category of a strong stability preserving (SSP) method. SSP methods preserve properties satisfied by the Euler method. In particular, the software uses this method to ensure positivity for some methods.

4.4 Space

All the implementation stuff should be here. Not in the appendix. Add proof of positivity for S$_N$ and M$_N$ and PP$_N$. Space is discretized using the finite volume method. The problem domain is decomposed into a regular grid of cells with an additional halo of ghost cells to enforce the boundary condition and allow synchronization with other nodes. At each update, the program first computes the flux at each cell, then updates the cell values according to the kinetic transport equation. Several of the solvers guarantee positivity on the grid.

The minmod preserves positivity. Need to show this. Some of the solvers employ a double minmod slope limiter to reduce oscillations introduced by the spatial scheme. This function is defined as

$$\operatorname{minmod}(x,y) = \operatorname{sgn}'(x)\max\Big(0, \min\big(|x|, y\operatorname{sgn}'(x)\big)\Big) \tag{27}$$

where

$$\operatorname{sgn}'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0. \end{cases} \tag{28}$$

In particular, note that $\operatorname{minmod}(x,y)$ is

- 0 if $xy < 1$
- $\min\{x,y\}$ if $x, y > 0$
- $-\min\{|x|,|y|\}$ if $x, y < 0$.

For the `kinetic` solver (implementing S_N), the timestep Δt is chosen with respect to the cell size so as not to violate the CFL condition:

$$\Delta t = \frac{c\Delta x\Delta y}{2(\Delta x + \Delta y)}. \tag{29}$$

where c is the configured CFL factor. To eliminate the possibility of negative density due to scattering, an additional term is included in the CFL check. Is this implemented, or did we just talk about it?

The `momopt` solver also ensures positivity as part of the flux calculations. The flux is based on an ansatz grid with strictly positive values. For M_N, the ansatz grid value a at each cell and direction is given by

$$a = \exp(\alpha^T m). \tag{30}$$

For PP_N,

$$a = \begin{cases} \dfrac{1}{2}k + \dfrac{1}{2}\sqrt{k^2 + 4\delta} & \text{if } k > 0 \\ \dfrac{-\delta}{\frac{1}{2}k - \frac{1}{2}\sqrt{k^2+4\delta}} & \text{if } k \leq 0 \end{cases} \tag{31}$$

where $k = \alpha^T m$.

4.5 Optimization Procedure

...

5 Program Layout

Make this nicer. Introduce each solver here (i.e. kinetic_solver) as associated with for instance S_N. Describe binary file formats. All options controlling the runtime operation of the program reside in `input.deck`. Comments prefixed with `#` are allowed, and options not used by the selected solver are ignored. Available options are listed in Section **??**, and example options for the various solvers are included in the `examples/` directory. The outputs are binary files with extensions `.sn`, `.pn`, and `.opt`, depending on the solver. Python code for reading and working with these files is provided in `util/formats.py`.

The common functionality is split up among the files in `src/`.

- `main.cpp` – Entry point of program
- `comm.cpp` – Controls MPI communication
- `utils.cpp` – Helper functions used throughout the code

The solver code has an approximately common layout, e.g. in `src/moment/`

- `moment_init.cpp` – Read config, set up quadrature and filters, etc.
- `moment_boundaries.cpp` – Communicate boundary data with other nodes
- `moment_update.cpp` – Solve flux, update the grid with time, etc.
- `moment_output.cpp` – Write out results

When using MPI, `comm.cpp` communicates boundary data between nodes. `Solver::getInnerBoundaries` is first called to obtain grid data on the boundaries of the current node's region of the domain. Next, each node trades boundary data with its neighbors to the north, south, east, then west via `MPI_ISend` and `MPI_IRecv`. Finally, the node calls `Solver::setOuterBoundaries` to update its grid with the data from neighboring nodes. Note scaling has not been tested for MPI.

6 Results

...

References

[1] Kendall Atkinson. Numerical integration on the sphere. *J. Austral. Math. Soc. Ser. B*, 23:332–347, 1982.

[2] T. A. Brunner and J. P. Holloway. Two-dimensional time-dependent Riemann solvers for neutron transport. *J. Comp Phys.*, 210:386–399, 2005.

[3] Thomas A. Brunner. Forms of approximate radiation transport. Technical Report SAND2002-1778, Sandia National Laboratories, 2002.

[4] Ryan G. McClarren and Cory D. Hauck. Robust and accurate filtered spherical harmonics expansions for radiative transfer. *Journal of Computational Physics*, 229(16):5597 – 5614, 2010.

[5] David Radice, Ernazar Abdikamalov, Luciano Rezzolla, and Christian D. Ott. A new spherical harmonics scheme for multi-dimensional radiation transport I: static matter configurations. *Journal of Computational Physics*, 242(0):648 – 669, 2013.