closures-2d: Open Source Software Release

Tim Shaffer[1]

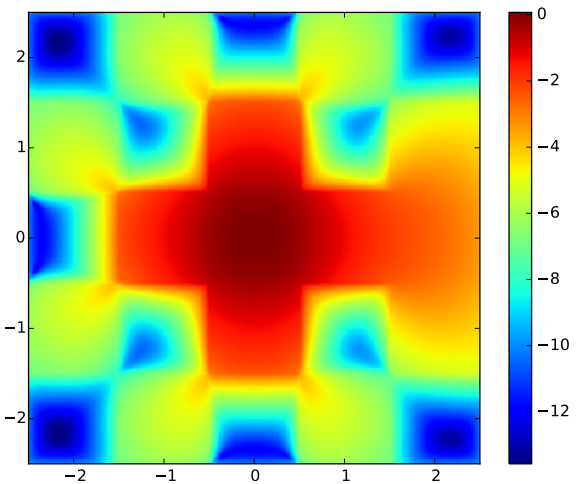
August 6, 2015

[1] Mentor: C. Kristopher Garrett

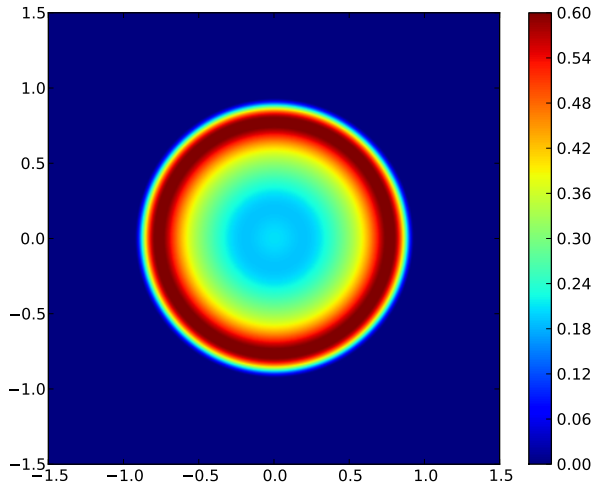
Goals

We are releasing software that tests angular approximations in kinetic transport simulations

- Based on code used for a previous publication (Garrett and Hauck 2013)
- Open source so that others can use/collaborate
- Modular, making it easy to implement new features
- Implements S_N, P_N, FP_N, M_N, PP_N, D_N (experimental)



P_N Order 3



FP_N (sspline) Order 3

Improvements

Algorithmic changes:

- Added experimental Lebedev quadrature
- Removed γ factor for ansatz correction in `momopt`
- Changed `momopt`'s regularization in case of bad condition number

Software-related improvements:

- Cross-platform build system
- Automated testing
- Better MPI communication
- Improved documentation
- Improved interface
- Bugfixes
- Profiling and optimization

Release

We are releasing this code as open source software.

Now uses SCons, a Python-based build system, rather than Makefiles

- Cross platform
- Sets up library search paths
- Intelligent compilation

Depends on:

- GSL
- BLAS
- LAPACK
- OpenMP (optional)
- MPI (optional)
- PAPI (optional)

What Are Kinetic Equations?

Macroscopic

- $\rho(x,t)$ – Density
- $u(x,t)$ – Velocity
- $E(x,t)$ – Kinetic Energy

Discretize x, t into 100 values:
4GB memory requirement

Mesoscopic

- $f(x,v,t)$ – Density with respect to space *and velocity*

Discretize x, v, t into 100 values:
800TB memory requirement

Macroscopic can be derived from mesoscopic

- $\rho(x,t) = \int_{\mathbb{R}^3} f \, \mathrm{d}v$
- $u(x,t) = \frac{1}{\rho} \int_{\mathbb{R}^3} vf \, \mathrm{d}v$
- $E(x,t) = \frac{1}{2} \int_{\mathbb{R}^3} \|v-u\|^2 f \, \mathrm{d}v$

What Are Kinetic Equations?

First used for rarefied gas dynamics (e.g. high altitude gases where collisions do not dominate the physics)

$$\partial_t f + v \cdot \nabla_x f = C(f)$$

where $\int C(f)\,\mathrm{d}v = 0$.

Integrate against v to get first Euler/Navier-Stokes equation

$$\partial_t \rho + \nabla_x \cdot (\rho u) = 0$$

Other areas:

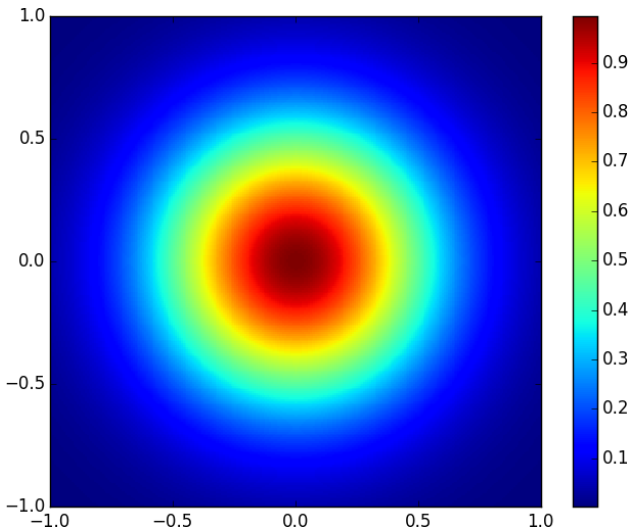
- Radiation transport
- Plasma simulations

Kinetic Problem

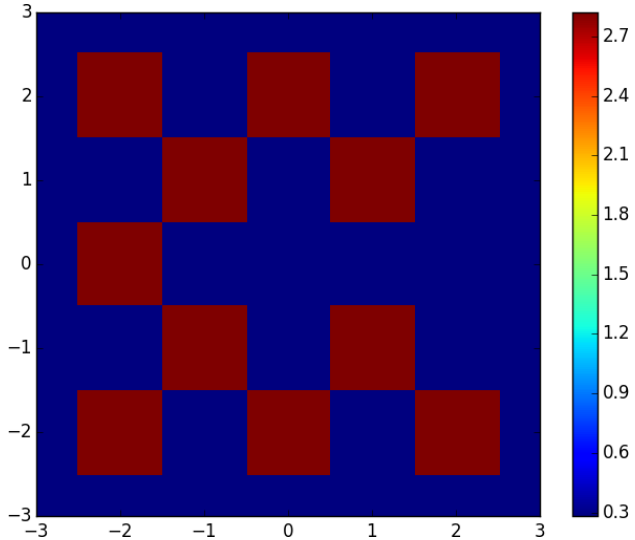
Unit Speed, Isotropic Scattering

$$\partial_t f + \Omega \cdot \nabla_x f = \frac{\sigma_s}{4\pi} \int_{\mathbb{S}^2} f \, \mathrm{d}\Omega - \sigma_t f$$

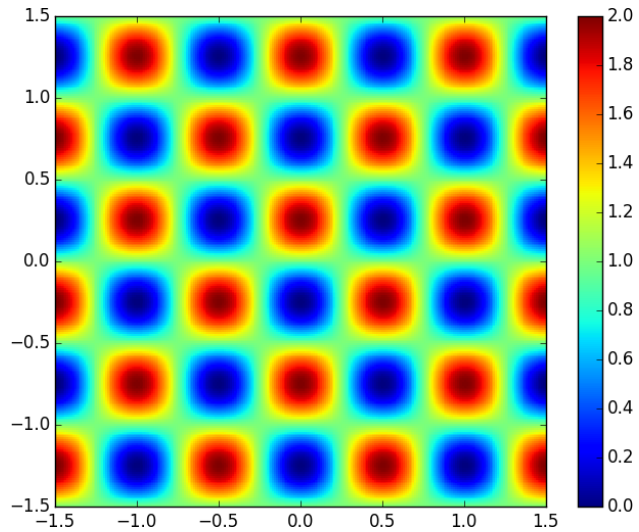
where $x \in \mathbb{R}^2$, $\Omega \in \mathbb{S}^2$, and σ_s,σ_t are the scattering and total cross sections.



Gaussian Initial Condition



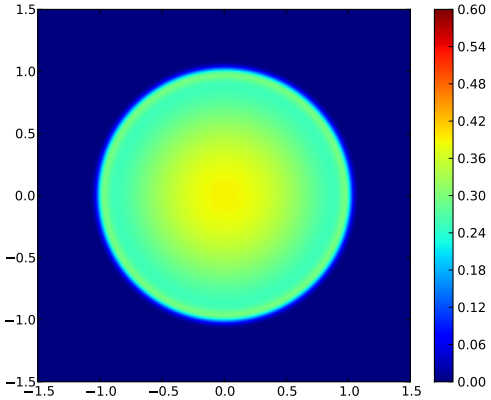
σ_T for Lattice Initial Condition



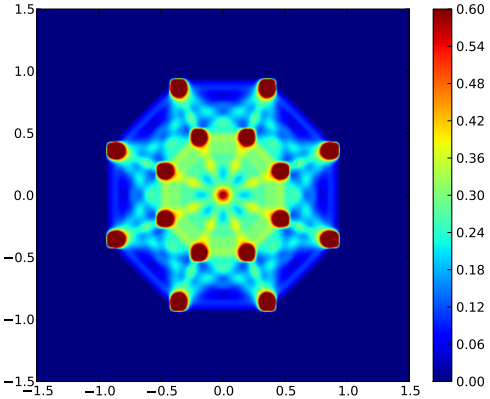
Smooth Initial Condition

kinetic

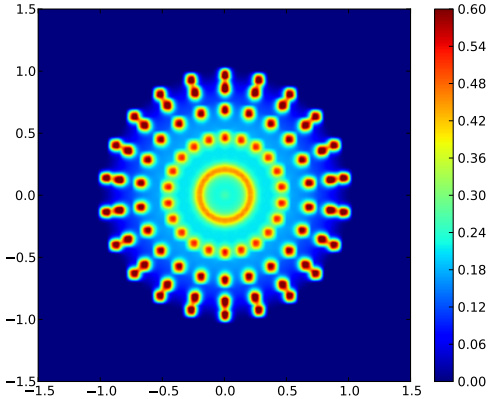
- Implements S_N
- Easy to compute
- Suffers from ray effects at low order



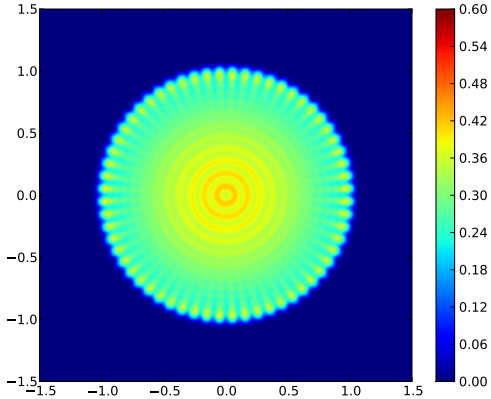
Analytic Solution



S_N Order 4



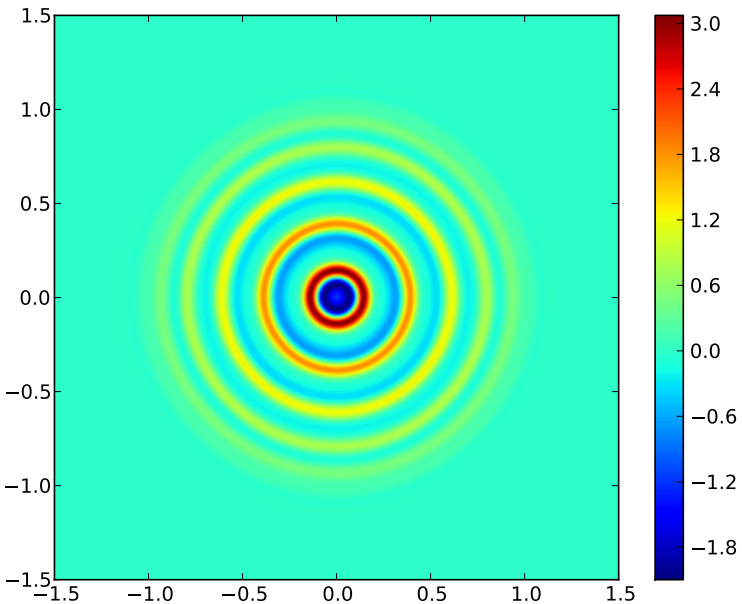
S_N Order 11



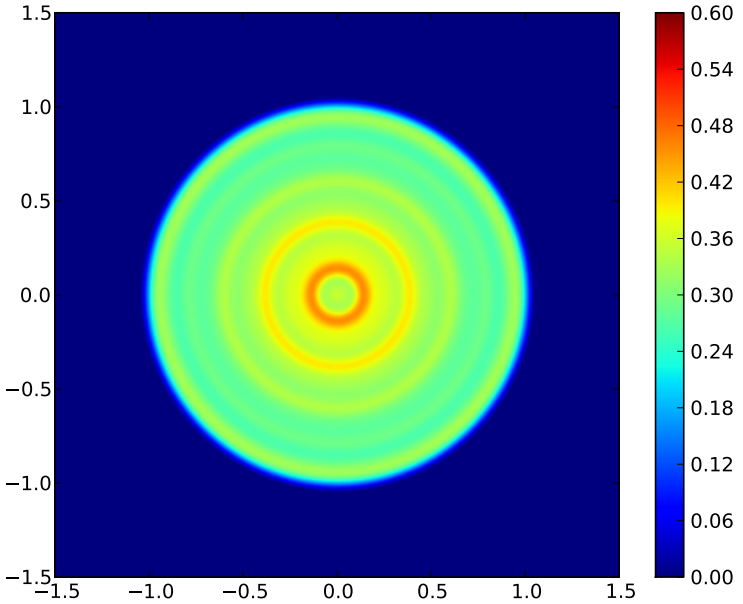
S_N Order 30

moment

- Implements P_N, FP_N
- Somewhat easy to compute
- Permits negative densities
- Suffers from oscillatory artifacts
- Filters can improve performance



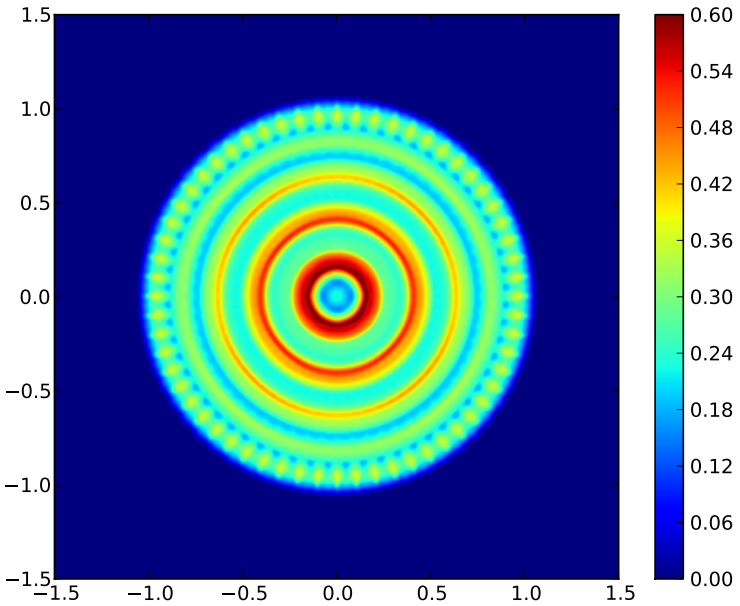
P_N Order 11



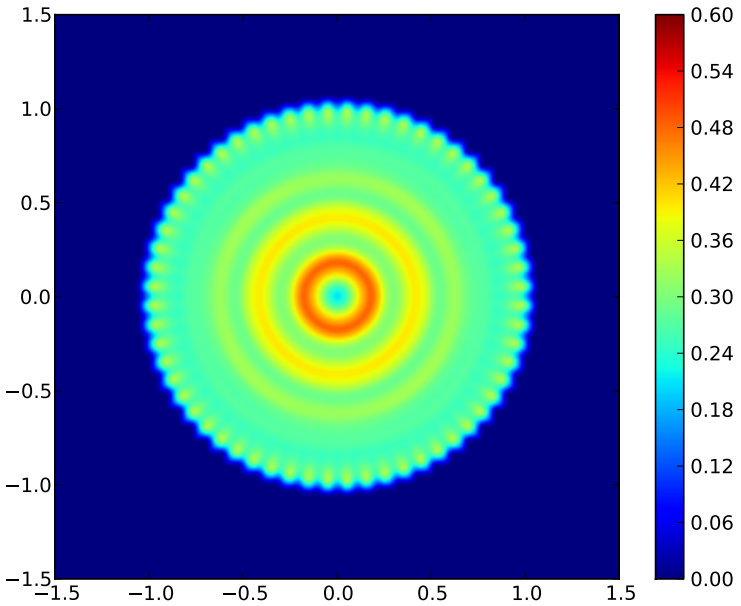
FP_N (Lanczos) Order 11

momopt

- Implements M_N and PP_N
- Difficult to compute
- Ensures positivity
- Requires additional optimization procedure



PP_N Order 11



M_N Order 11

Testing

We implemented several automated tests for the software. The testing code is written in Python and integrated with the build system.
The testing code carries out

- Regression tests – compare output to reference data included with the software
- Mass Conservation tests – check that total density remains the same
- Convergence tests – measure the effect of decreasing the cell size on the precision of the output

Convergence: `moment` (`sspline`)

Cell size	Relative Error	Order
$\Delta x/1$	0.015718	—
$\Delta x/2$	0.004056	1.954163
$\Delta x/4$	0.000958	2.082828
$\Delta x/8$	0.000189	2.340759

Performance

Added timing and profiling-related measurement capabilities

Tested additional optimizations

- Spatial blocking (reduce scheduling overhead, cache misses)
- CPU pinning (ccNUMA)
- Memory alignment

Caching Example: Matrix Multiplication

```
for(int i = 0; i < N; i++) {
for(int j = 0; j < N; j++) {
for(int k = 0; k < N; k++) {
    result[i][j] += left[i][k] * right[k][j]
}}}
```

The order of the loops does not matter algebraically.

Caching Example: Matrix Multiplication

Loop order	Time
ijk	8.139175
ikj	5.316213
jik	6.160773
jki	19.494467
kij	5.406215
kji	19.666042

Factor of 4 difference

Blocking

Cache utilization:

- Using a simple loop: move over each row and column, fetching values from main memory on *every iteration*
- Using a block-oriented loop: compute one section, fetching most nearby values from a CPU cache

OpenMP Scheduling:

- Using a simple loop: dispatch each cell to a (possibly different) core
- Using a block-oriented loop: dispatch an entire block to core, improving cache utilization and reducing scheduling overhead

With adjustments to main loop in flux calculations:

- ∼10% speedup
- Eliminate ∼30% L1 cache misses, ∼75% L2, ∼50% L3

Quadratures

Currently uses Chebyshev-Legendre quadrature.

- Product quadrature on the sphere

$$\int_{\mathbb{S}^2} \cdot \, \mathrm{d}\Omega = \int_{\mu=-1}^{1} \int_{\phi=0}^{2\pi} \cdot \, \mathrm{d}\phi \, \mathrm{d}\mu$$

- n Gauss-Legendre points on μ
- $2n$ equally spaced points on ϕ
- Exactly integrates to degree $2n - 1$ moments
- Easily optimized for symmetry

Quadratures

Added (experimental) Lebedev quadrature

- Constructed based on octahedral symmetry group
- Asymptotically optimal with respect to number of points (2/3 that of Chebyshev-Legendre)
- Structure does not lend itself to symmetry optimizations
- Negative weights break positivity

Optimization Problem

`momopt` uses nonlinear spectral methods, so updates entail solving an optimization problem for a given constant vector u and moments $m(\Omega)$.

$$\min_{\alpha} \int_{\mathbb{S}^2} \exp(\alpha^{\mathsf{T}} m)\, \mathrm{d}\Omega - \alpha^{\mathsf{T}} u$$

To use a Newton solver, we need

$$\text{Objective function } F(\alpha) = \int_{\mathbb{S}^2} \exp(\alpha^{\mathsf{T}} m)\, \mathrm{d}\Omega - \alpha^{\mathsf{T}} u$$

$$\text{Gradient } g(\alpha) = \int_{\mathbb{S}^2} m \exp(\alpha^{\mathsf{T}} m)\, \mathrm{d}\Omega - u$$

$$\text{Hessian } H(\alpha) = \int_{\mathbb{S}^2} m m^{\mathsf{T}} \exp(\alpha^{\mathsf{T}} m)\, \mathrm{d}\Omega$$

Now the estimated $\alpha_{i+1} = \alpha_i + td$ where $d = -H(\alpha)^{-1} g(\alpha)$.

γ factor

In `momopt`'s optimization steps, we compute an approximate α for a given u such that $u \approx \int_{\mathbb{S}^2} m \exp(\alpha^{\mathsf{T}} m)\, \mathrm{d}\Omega$.

- Needed to estimate error and try to correct via $\gamma = \frac{\exp(\bar{\alpha}^{\mathsf{T}} m)}{\exp(\hat{\alpha}^{\mathsf{T}} m)}$. Here, the numerator of γ is exact, and the denominator is our approximation.
- Required fine mesh and low tolerance on optimization to prevent non-realizability.

Instead, compute $\hat{u}$ such that $\hat{u} = \int_{\mathbb{S}^2} m \exp(\hat{\alpha}^{\mathsf{T}} m)\, \mathrm{d}\Omega$

- Adjust scaling to conserve mass
- Exact equality ensures realizability

Definition: Realizability

u is realizable if $\exists f > 0$ such that $u = \int_{\mathbb{S}^2} mf\, \mathrm{d}\Omega$

Fixed regularization

If cond($H(\alpha)$) is too large

- $H(\alpha)$ is difficult to invert
- Can't find the direction $d = -H(\alpha)^{-1}g(\alpha)$

Need to adjust the problem to make it more tractable

- Set isotropic initial guess
  $\alpha = (2\sqrt{\pi}\log\frac{1}{2\sqrt{\pi}}, 0, 0, \ldots, 0)$ for $\mathsf{M_N}$,
  $\alpha = (1 - 4\pi\delta, 0, 0, \ldots, 0)$ for $\mathsf{PP_N}$, where δ is a parameter of $\mathsf{PP_N}$
- For $i = 2, \ldots,$ multiply each u_i by $1 - r$, where r is a regularization constant with $0 < r < 1$

Source Code

The source code is available on Github

`https://github.com/ckrisgarrett/closures-2d`