PyTRT: a Python/C++ framework for transport methods development

Seth Johnson

University of Michigan, Ann Arbor

May 6, 2011

PyTRT

MICHIGAN

NEUP Nuclear Energy University Programs

U.S. Department of Energy

Outline

1. Introduction
2. Object-oriented boundary conditions
3. Python
4. Conclusions

Design goals

Primary mission

To graduate in a timely manner.

- rapid, error-free implementation of new methods,
- easy definition of multiple-method test problems,
- high-performance solver kernels to run the problems quickly,
- powerful data analysis tools driven by user needs (me), and
- automated generation of high-quality figures.

Capabilities

Cartesian product of methods:

- Steady-state, linear time-dependent, and nonlinear (semi-implicit) TRT
- 1-D, Flatland, 2-D
- Monte Carlo, S_N transport, diffusion, P_1, anisotropic diffusion

Analysis:

- Lineout
- Angleout
- Matrixout
- Silo for VisIT

Techniques

Reliability:

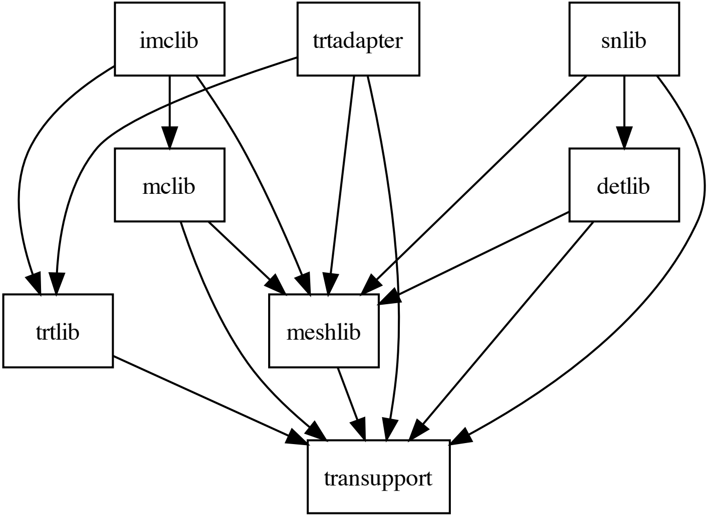
- CMake build process
- Git for version control
- Unit tests (regression)
- Design By Contract
- Modular design
- Trilinos for linear algebra*

Code reuse:

- Template on geometry, etc.
- Python wrapper handles linearization
- Python handles all the stuff that only needs to happen once

*Two-edged sword in terms of reliability

Modular design



Outline

1. Introduction
2. Object-oriented boundary conditions
3. Python
4. Conclusions

Object oriented boundary conditions

- Abstract `BoundaryCondition` class:
    - At least one `apply` method
    - Other methods like `getIncidentSourceRate`
- Generic `BcManager` class:
    - The problem definition contains a `BcManager` object
    - Vector that maps `BoundaryFace` to `BoundaryCondition*`
    - Functors to help with modifying multiple boundary conditions

Diffusion/constructed matrix

```
void MyBc::apply(const BoundaryFaceT& bface,
                 ProxyVector& vec) const
{
    const CellT& cell = *insideBoundaryCell(&bface);
    source.getFlux()[ cell ] += /* some value */;
}

void apply(const BoundaryFaceT& bface,
           Operator& matrix) const
{
    const CellT& cell = *insideBoundaryCell(&bface);
    matrix.startRow( matrix.getFlux()[cell] );
    matrix.pushRowElement( matrix.getFlux()[cell], /*val*/);
    matrix.finishRow();
}
```

Diffusion/constructed matrix

Called after initializing source vector:

```
bcs.apply( <Traits::BcManagerT, ProxyVector>(sourceVec) );
```

Called after the internal part of the diffusion matrix has been built:

```
startMatrix( ACCUMULATE );
bcs.apply(BcOperatorApplier<Traits>(*this));
finishMatrix();
```

S_N boundary conditions

```
void ReflectingBoundaryCondition::apply(
        const BoundaryFaceT& bface,
        Vector& source ) const
{
    bool onNegBoundary = bface.getFace()->onNegBoundary();
    unsigned int axis = bface.getFace()->getAxis();
    FluxDiscreteT& sourceFlux = source.getBoundaryFlux()[ bfac
    for (QuadratureSetT::const_iterator angle = qs_.begin();
            angle != qs_.end(); ++angle)
    {
        if (isPositive(angle->getOmega()[axis]) == onNegBounda
            sourceFlux[ *angle ]
                = sourceFlux[ qs_.getReflectedAngle(*angle, ax
        }
    }
}
```

MC boundary conditions

```
template<class ProblemTraits_T>
void VacuumBoundaryCondition<ProblemTraits_T>::apply(
        const BoundaryFaceT& bface,
        ParticleT& particle,
        Tally& tally) const
{
    tally.particleExited( bface, particle );
    particle.markDestroyed();
}
```

Outline

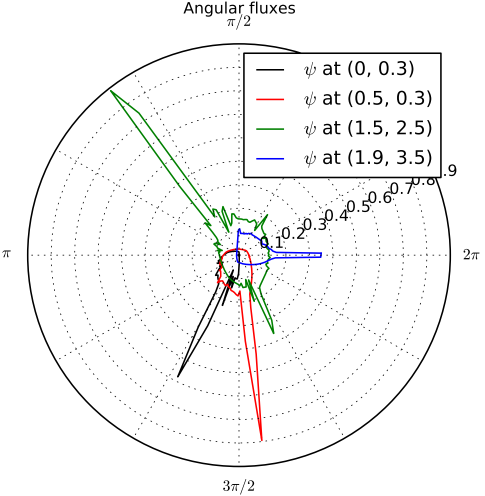
Python structure

- `Manager` class emits C++ problem depending on module passed to it (duck typing)
- `Solver` handles time stepping, callbacks, user feedback, etc.
- Callbacks include Silo output, liveplot, lineout, angleout, MC particle tally info, Δt vs. t, etc.
- Lineout etc. use PyTables to store HDF5 data and metadata

High-level "glue" written in Python:

- Flux-limited diffusion
- Linearization scheme
- Multigrid management
- Time-dependent "events"

Angleout


Angular fluxes
$\pi/2$
π
2π
$3\pi/2$
0.1
0.2
0.3
0.4
0.5
0.6
0.7
ψ at (0, 0.3)
ψ at (0.5, 0.3)
ψ at (1.5, 2.5)
ψ at (1.9, 3.5)

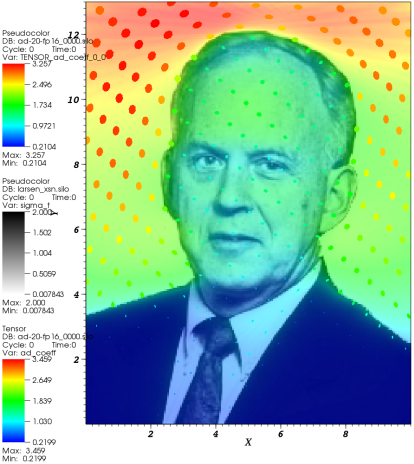
Outline

Availability

Sponsored by the public, available to the public. (Simplified BSD license.)

pytrt.org

Questions?



This material is based upon work supported under a National Science Foundation Graduate Research Fellowship and a Department of Energy Nuclear Energy University Programs Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation or the Department of Energy Office of Nuclear Energy.