NDA-QMC Results

April 3, 2021

```
[11]: include("nda_note.jl")
```

1 Using the notebook

The first cell in this notebook is an invisible markdown cell with LaTeX commands. The second is a code cell that sets up the packages you'll need to load. It's best to do a **run all** before experimenting with the solvers.

1.1 Contents

2 The equation

This note book compares various solvers for the problem in the Garcia/Siewert paper [1]. My formulation of the transport problem is taken from [7]. The equation for the angular flux ψ is

$$\mu \frac{\partial \psi}{\partial x}(x,\mu) + \Sigma_t(x)\psi(x,\mu) = \frac{1}{2}\left[\Sigma_s(x)\int_{-1}^{1}\psi(x,\mu')\,d\mu' + q(x)\right] \text{ for } 0 \le x \le \tau$$

The boundary conditions are

$$\psi(0,\mu) = \psi_l(\mu), \mu > 0; \psi(\tau,\mu) = \psi_r(\mu), \mu < 0.$$

The notation is

- ψ is intensity of radiation or angular flux at point x at angle $\cos^{-1}(\mu)$

- $\phi = \phi(x) = \int_{-1}^{1} \psi(x,\mu)\, d\mu$ is the scalar flux, the 0^{th} angular moment of the angular flux. - $\tau < \infty$, length of the spatial domain. - $\Sigma_s \in C([0,\tau])$ is the scattering cross section at x - $\Sigma_t \in C([0,\tau])$ is the total cross section at x - ψ_l and ψ_r are incoming intensities at the bounds - $q \in C([0,\tau])$ is the fixed source

2.1 Discretization

The discretization is plain vanilla S_N with diamond differencing. I'm storing fluxes (and later currents) at cell edges to make the boundary conditions for NDA a bit easier to deal with.

I'm using double Gaussian quadratures for the angular mesh. This means that the angles are N-point Gaussian quadratures on (-1,0) and on (0,1) for a total of 2N angles. The function **sn_angles.jl** in the /src directory sets up the angular mesh. It uses the package **FastGaussQuadrature.jl**. We will denote the weights and notes of the angluar mesh by w_j and μ_j for $j = 1, ..., 2N$.

I'm using a uniform spatial mesh $\{x_i\}_{i=1}^{N_x}$ where

$$x_i = (i-1)dx \text{ and } dx = \tau/(N_x - 1).$$

The function **sn_init.jl** in /src sets up all the mesh data and boundary conditions. It builds a Julia named tuple for me to pass around to the solvers.

The disretization approximates the angular flux $\psi_i^j \approx \psi(x_i, \mu_j)$ and the scalar flux

$$\phi_i \approx \phi(x_i).$$

The two approximations are related (in my cell-edge oriented code) by

$$\text{Flux Equation: } \phi_i = \sum_{j=1}^{2N} \psi_i^j w_j$$

The discrete equation is, with source $S_i = \Sigma_s(x_i) f_i + q(s_i)$

$$\begin{array}{lll} \text{Transport Sweep:} & & \\ \mu_j \frac{\psi_i^j - \psi_{i-1}^j}{dx} + \Sigma_t(x_i + dx/2) \frac{\psi_i^j + \psi_{i-1}^j}{2} & = \frac{S_i + S_{i-1}}{2} & \text{for } \mu_j > 0 \text{ and } \psi_1^j = \psi(0, \mu_j) \\ \mu_j \frac{\psi_i^j - \psi_{i+1}^j}{dx} + \frac{\psi_i^j + \psi_{i+1}^j}{2} & = \frac{S_i + S_{i+1}}{2} & \text{for } \mu_j < 0 \text{ and } \psi_{N_x}^j = \psi(\tau, \mu_j) \end{array}$$

So, source iteration begins with an initial iterate for the scalar flux ϕ, computes the discrete scalar flux with the equations **Transport_Sweep**, treating **f** as input data, and updates the scalar flux with the formula **Flux_Equation**. This can also be viewed as a linear equation for ϕ

$$\phi - \mathbf{K}\phi = f$$

and solved with a Krylov method like GMRES. The right side of the integral equation depends on the source q and the boundary data.

2.2 The Garcia-Siewert Example

In this example

$$\tau = 5, \Sigma_s(x) = \omega_0 e^{-x/s}, \Sigma_t(x) = 1, q(x) = 0, \psi_l(\mu) = 1, \psi_r(\mu) = 0.$$

2.3 Solvers

The linear and nonlinear solvers come from my Julia package SIAMFANLEQ.jl [3]. The documentation for these codes is in the Juila notebooks that accompany the package [2]. All of this is part of a book project [4]. The citations for these things may change as the book gets closer to publication. Please ask me before citing this stuff.

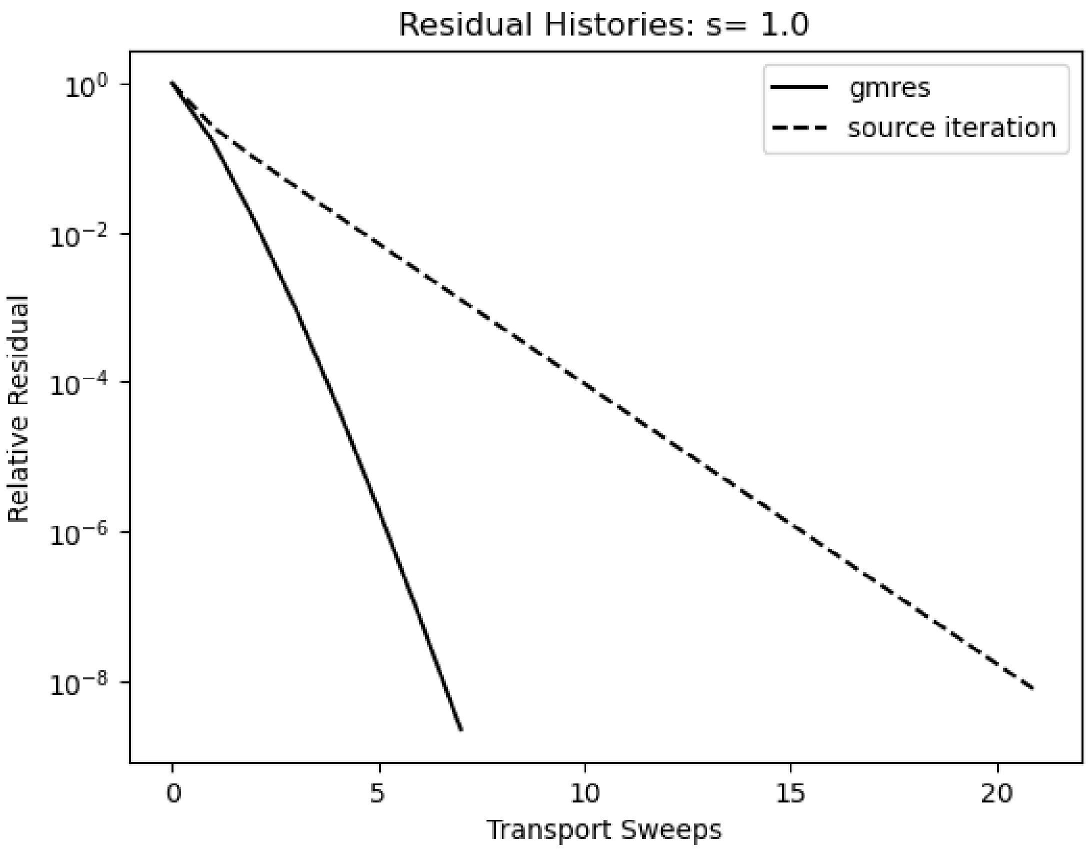
The important solvers for this notebook are the GMRES linear solver **kl_gmres.jl** and the Newton-Krylov nonlinear solver **nsoli.jl**.

2.4 GMRES and Source Iteration

The function `compare.jl` in the **src** directory runs the source iteration and GMRES, plots the residual histories, and tabulates the exit disbributions for comparision with Tables 1 and 2 in the paper. As a sanity check I print the L^∞ norm of the differences in the results from the two solvers.

Here is a run for the first columns in the tables.

[12]:
```
compare(1.0)
```



```
mu         I(0,-mu)        I(tau,mu)
0.05      5.89670e-01     6.07486e-06
```

```
 0.10     5.31120e-01     6.92514e-06
 0.20     4.43280e-01     9.64229e-06
 0.30     3.80307e-01     1.62338e-05
 0.40     3.32965e-01     4.38575e-05
 0.50     2.96091e-01     1.69371e-04
 0.60     2.66564e-01     5.73462e-04
 0.70     2.42390e-01     1.51281e-03
 0.80     2.22235e-01     3.24369e-03
 0.90     2.05175e-01     5.96035e-03
 1.00     1.90547e-01     9.77122e-03
Norm of result difference = 9.32897e-09
```

The functions all use precomputed data for the angles, spatial mesh, storage allocationf the angular flux, and various problem paramaters. The precomputed data lives in a named tuple `sn_data` which I create with `ns_init.jl`.

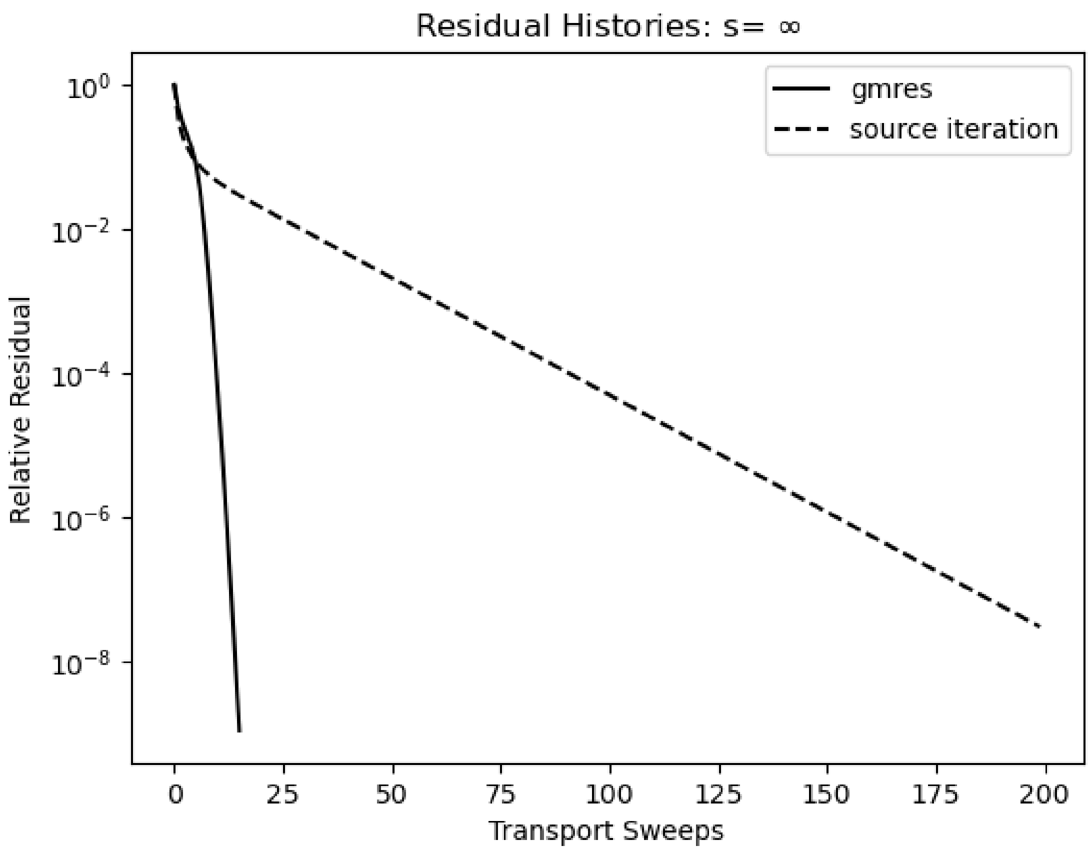
The iterations themselves use a transport sweep to do both source iteration and the liner residual computation for GMRES. In both cases a function `flux_map!` takes an input flux, does the transport sweep, and then takes the zeroth moment to return and output flux. The transport sweep is `transport_sweep.jl` it computes the angular flux from a given input flux and boundary conditions.

Here is function for the source iteration solver. It's pretty simple if you believe `flux_map.jl` does what I say it does. I think the QMC version of `flux_map.jl` would fit in with this pretty easiliy.

```
"""
 source_iteration(sn_data,s,tol)
 Source iteration example script for transport equation.

 This is one of the test cases from

 Radiative transfer in finite inhomogeneous plane-parallel atmospheres
 by Garcia and Siewert
 JQSRT (27), 1982 pp 141-148.

"""
function source_iteration(sn_data,s,tol=1.e-8)
    nx = 2001
    #
    # precomputed data
    #
    angles=sn_data.angles
    weihts=sn_data.weights
    itt = 0
    delflux = 1
    phi = zeros(nx)
    flux = zeros(nx)
    reshist = []
    while itt < 200 && delflux > tol
        flux = flux_map!(flux, sn_data)
        delflux = norm(flux - phi, Inf)
        itt = itt + 1
        push!(reshist, delflux)
        phi .= flux
    end
    #
    # Tabulate the exit distributions to check results.
    #
    return ( flux = flux, history= reshist)
end
```

Here are the results for the final column.

```
[13]: compare(Inf)
```



```
mu         I(0,-mu)        I(tau,mu)
0.05      8.97799e-01     1.02201e-01
0.10      8.87836e-01     1.12164e-01
0.20      8.69581e-01     1.30419e-01
0.30      8.52299e-01     1.47701e-01
0.40      8.35503e-01     1.64497e-01
```

```
 0.50      8.18996e-01      1.81004e-01
 0.60      8.02676e-01      1.97324e-01
 0.70      7.86493e-01      2.13507e-01
 0.80      7.70429e-01      2.29571e-01
 0.90      7.54496e-01      2.45504e-01
 1.00      7.38721e-01      2.61279e-01
Norm of result difference = 3.97115e-07
```

2.5 NDA

NDA (nonlinear diffusion acceleration) turns the linear problem into a nonlinear one. One gets an equation for the (**low-order**) flux f in terms of the moments of ψ. It is easier for me to explain this for the continuous problem. Be warned that my **boundary conditions are not standard**.

I'm taking this from [7], [6], and [5].

Given a flux $\phi(x)$ we begin by solving a **high-order** problem for an angular flux

$$\mu \frac{\partial \psi^{HO}}{\partial x} + \Sigma_t \psi^{HO}(x,\mu) = \frac{1}{2}\left[\Sigma_s \phi^{LO}(x) + q(x)\right],$$

with the same boundary conditions we used in the original problem.

The next step is to compute high-order fluxes

$$f^{HO}(x) = \int_{-1}^{1} \psi(x,\mu')\, d\mu'$$

and currents

$$J^{HO}(x) = \int_{-1}^{1} \psi(x,\mu')\mu'\, d\mu'.$$

We use these to compute

$$\hat{D} = \frac{J^{HO} + \frac{1}{3}\frac{d\phi^{HO}}{dx}}{\phi^{HO}},$$

If ϕ is the solution to the **low-order** problem,

$$\frac{d}{dx}\left[-\frac{1}{3\Sigma_t}\frac{d\phi}{dx} + \hat{D}\phi\right] + (\Sigma_t - \Sigma_s)\phi = q,$$

then ϕ is the scalar flux and we have solved the transport equation. The boundary conditions are tricky if we store fluxes and currents at cell centers. If we have cell edge fluxes, then boundary conditions

$$\phi(0) = \phi^{HO}(0), \phi(\tau) = \phi^{HO}(\tau)$$

work well.

The low-order prolbem is nonlinear because ϕ^{HO}, J^{HO} and $\hat{D}$ depend on ϕ. We can make this explicit (writing $\hat{D}(\phi)$ and that helps when it's time to write code. Write the nonlinear equation as $F(\phi) = 0$ where

$$F(\phi) = \frac{d}{dx}\left[-\frac{1}{3\Sigma_t}\frac{d\phi}{dx} + \hat{D}(\phi)\phi\right] + (\Sigma_t - \Sigma_s)\phi - q.$$

We can solve this equation efficiently with Newton-GMRES if we have a good preconditioner. We use fast solver for the high-order term in the low-order equation. The preconditioner in [7], [6], [5], is a bit more complicated.

We can also formulate the low-order problem as a fixed point problem by viewing the solution of the low-order problem ϕ^{LO} as a transformation of ϕ. So given ϕ^{LO} compute the high-order flux, current, and $\hat{D} = \hat{D}(\phi^{LO})$. Then solve

$$\frac{d}{dx}\left[-\frac{1}{3\Sigma_t}\frac{d\phi}{dx} + \hat{D}(\phi^{LO})\phi\right] + (\Sigma_t - \Sigma_s)\phi = q,$$

For ϕ. This is a **linear** equation for ϕ because ϕ^{LO} is input. Express this as

$$\phi = G(\phi^{LO})$$

We have solved the problem if $\phi = \phi^{LO}$. So a Picare or fixed point iteration is

$$\phi_{n+1} = G(\phi_n).$$

One can also solve

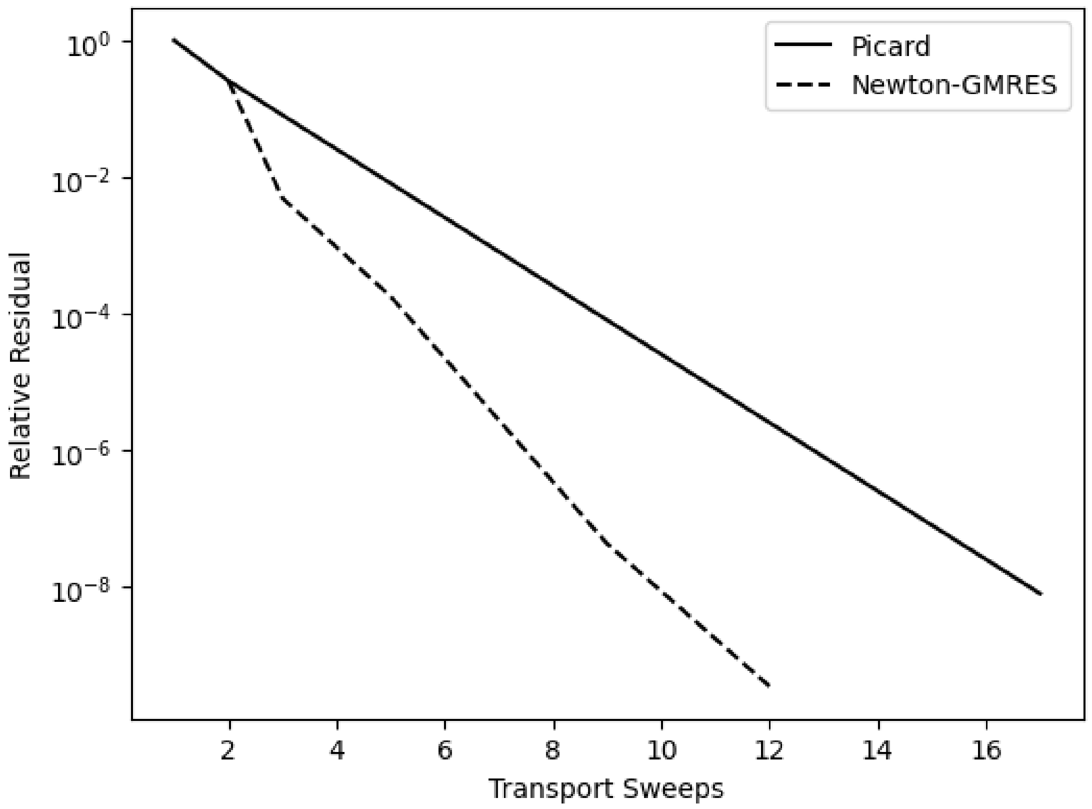
$$F(\phi) = \phi - G(\phi)$$

with a Newton-Krylov method. This is **not what we did** in [7] because MC is not deterministic and we had to use a formula for the derivative of $\hat{D}$ with respect to ϕ. Using that formula to differentiate G would be a real pain. We can do this for QMC and use a finite-difference Jacobian-vector product. The advantage is that the conditioning of the linear system is better because we precondition when we solve low-order problem.

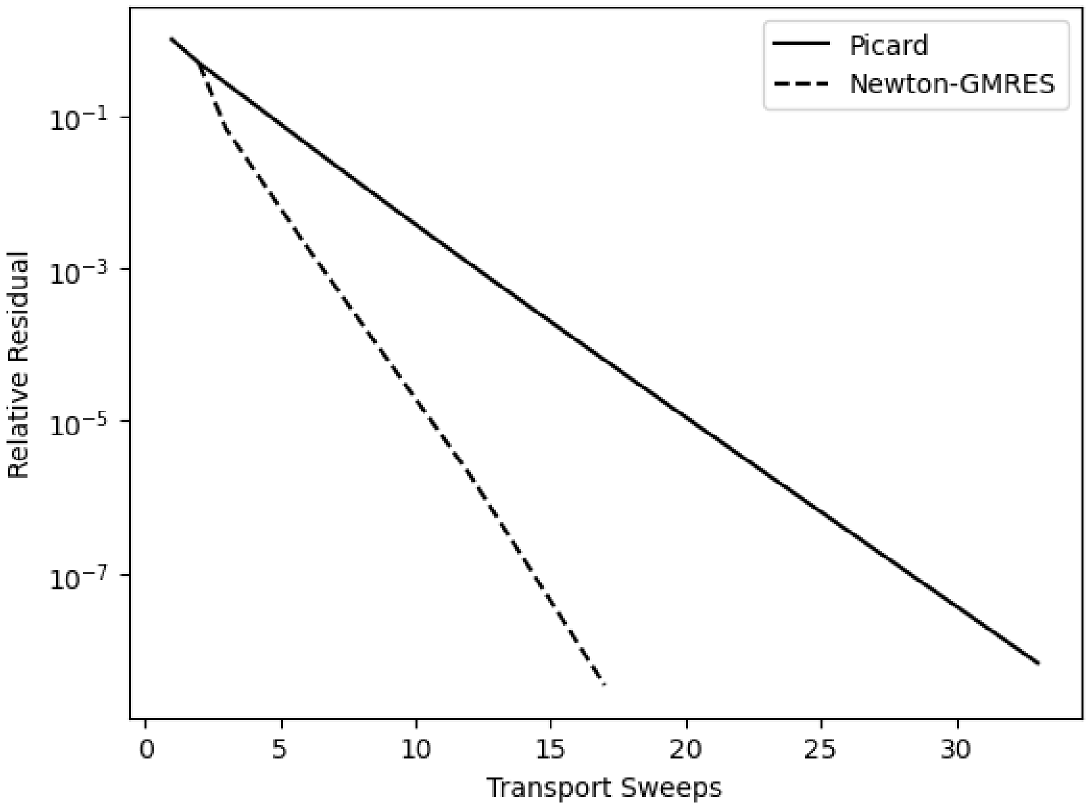
One subtle point for Newton is that we take on Picard iteration before starting Newton. The reason we do this is that the initial iterate $\phi = 0$ is really bad and Newton, while converging, takes many iterations to fix that. A single Picard sorts that out.

The plots below compare the two approaches. The limited data in this note book indicate that GMRES is best for the $s = 1$ case and NDA-Newton is best for $s = \infty$. The iteration histories overlap at the start because the first step for both is that single Picard. Note that I'm not counting Newton iterations but transport sweeps.

[14]:
```
# Run for s=1
nda_compare(1.0);
```



```
[15]: # and for s=Inf
nda_compare(Inf);
```



2.6 QMC and NDA

What we need from QMC are

- cell average fluxes and currents, which I think we have and
- cell average spatial derivatives for the high-order flux.

As you can see from [7], it was not entirely trivial to get the spatial derivatives.

Any ideas? I am ready to try a hack job once I have your QMC code.

What I need from QMC is something that takes the cell average low-order flux and returns cell average high-order fluxes and currents. I'm pretty sure I can conform to the way you want me to tell you how many cells and the number of samples.

[ ]:

```
```

References

[1] R.D.M. Garcia and C.E. Siewert, *Radiative transfer in finite inhomogeneous plane-parallel atmospheres*, J. Quant. Spectrosc. Radiat. Transfer, 27 (1982), pp. 141–148.

[2] C. T. KELLEY, *Notebook for Solving Nonlinear Equations with Iterative Methods: Solvers and Examples in Julia*. https://github.com/ctkelley/NotebookSIAMFANL, 2020. IJulia Notebook.

[3] ——, *SIAMFANLEquations.jl*. https://github.com/ctkelley/SIAMFANLEquations.jl, 2020. Julia Package.

[4] ——, *Solving Nonlinear Equations with Iterative Methods: Solvers and Examples in Julia*, 2020. Unpublished book ms, under contract with SIAM.

[5] D. A. KNOLL, H. PARK, AND K. SMITH, *A new look at nonlinear acceleration*, Nuclear Science and Engineering, 99 (2008), pp. 332–334.

[6] ——, *Application of the Jacobian-free Newton-Krylov method to nonlinear acceleration of transport source iteration in slab geometry*, Nuclear Science and Engineering, 167 (2011), pp. 122–132.

[7] JEFF WILLERT, C. T. KELLEY, D. A. KNOLL, AND H. K. PARK, *Hybrid deterministic/Monte Carlo neutronics*, SIAM J. Sci. Comp., 35 (2013), pp. S62–S83.