TEXAS A&M UNIVERSITY
Dwight Look College of Engineering
Department of Nuclear Engineering

Research Memo

From: Zachary M. Prince, Jean C. Ragusa
To: Distribution
Date: March 8, 2016

Subject: Predictor Corrector IQS

1. Standard IQS Review

Multigroup diffusion equation with delayed neutron precrusors:

$$\begin{aligned}\frac{1}{v^g}\frac{\partial \phi^g}{\partial t} =& \frac{\chi_p^g}{k_{\mathit{eff}}}\sum_{g'=1}^{G}(1-\beta)\nu^{g'}\Sigma_f^{g'}\phi^{g'} - \left(-\vec{\nabla}\cdot D^g\vec{\nabla} + \Sigma_r^g\right)\phi^g \\ &+ \sum_{g'\neq g}^{G}\Sigma_s^{g'\to g}\phi^{g'} + \sum_{i=1}^{I}\chi_{d,i}^g\lambda_i C_i \ , \quad 1 \le g \le G\end{aligned} \tag{1}$$

$$\frac{dC_i}{dt} = \frac{\beta_i}{k_{eff}}\sum_{g=1}^{G}\nu^g\Sigma_f^g\phi^g - \lambda_i C_i \ , \quad 1 \le i \le I \tag{2}$$

IQS method involves factorizing flux ($\phi^g(\vec{r},t)$) into space-time dependent shape ($\varphi^g(\vec{r},t)$) and time-dependent amplitude ($p(t)$). This results coupled sytem of shape diffusion and point reactor kinetics for amplitude:

$$\begin{aligned}\frac{1}{v^g}\frac{\partial \varphi^g}{\partial t} =& \frac{\chi_p^g}{k_{\mathit{eff}}}\sum_{g'=1}^{G}(1-\beta)\nu^{g'}\Sigma_f^{g'}\varphi^{g'} - \left(-\vec{\nabla}\cdot D^g\vec{\nabla} + \Sigma_r^g + \frac{1}{v^g}\frac{1}{p}\frac{dp}{dt}\right)\varphi^g \\ &+ \sum_{g'\neq g}^{G}\Sigma_s^{g'\to g}\varphi^{g'} + \frac{1}{p}\sum_{i=1}^{I}\chi_{d,i}^g\lambda_i C_i \ , \quad 1 \le g \le G\end{aligned} \tag{3}$$

$$\frac{dp}{dt} = \left[\frac{\rho-\bar{\beta}}{\Lambda}\right]p + \sum_{i=1}^{I}\bar{\lambda}_i\xi_i \tag{4}$$

$$\frac{d\xi_i}{dt} = \frac{\bar{\beta}_i}{\Lambda} - \bar{\lambda}_i\xi_i \quad 1 \le i \le I \tag{5}$$

Where the functional coefficients are calculated using the space-/time-dependent shape function as follows:

$$\frac{\rho-\bar{\beta}}{\Lambda} = \frac{\sum_{g=1}^{G}\left(\phi^{*g}, \frac{\chi_p^g}{k_{eff}}(1-\beta)\sum_{g'=1}^{G}\nu^{g'}\Sigma_f^{g'}\varphi^{g'} + \sum_{g'\neq g}^{G}\Sigma_s^{g'\to g}\varphi^{g'} - \left(-\vec{\nabla}\cdot D^g\vec{\nabla}+\Sigma_r^g\right)\varphi^g\right)}{\sum_{g=1}^{G}\left(\phi^{*g}, \frac{1}{v^g}\varphi^g\right)} \tag{6}$$

$$\frac{\bar{\beta}}{\Lambda} = \sum_{i=1}^{I}\frac{\bar{\beta}_i}{\Lambda} = \sum_{i=1}^{I}\frac{1}{k_{eff}}\frac{\sum_{g=1}^{G}(\phi^{*g}, \chi_{d,i}^g\beta_i\sum_{g'=1}^{G}\nu^{g'}\Sigma_f^{g'}\varphi^{g'})}{\sum_{g=1}^{G}\left(\phi^{*g}, \frac{1}{v^g}\varphi^g\right)} \tag{7}$$

$$\bar{\lambda}_i = \frac{\sum_{g=1}^{G}(\phi^{*g}, \chi_{d,i}^{g}\lambda_i C_i)}{\sum_{g=1}^{G}(\phi^{*g}, \chi_{d,i}^{g} C_i)} \tag{8}$$

The shape diffusion is solved at the beginning time step with a already known amplitude, then the PRKE is evaulted at small (micro) time steps until the next large (macro) time step. The final amplitude from the PRKE is used to solve the shape diffusion equation. Since the PRKE parameters would be more accurate with this new shape, they are recomputed and the PRKE and shape evaluations are iterated. The system is iterated until the IQS error is converged to 0:

$$\text{Error}_{\text{IQS}} = \left| \frac{\left(\phi_g^*, \frac{1}{v_g}\varphi_g^n\right)}{\left(\phi_g^*, \frac{1}{v_g}\varphi_g^0\right)} - 1 \right| \to 0 \tag{9}$$

2. **Predictor-Corrector IQS**

The Predictor-Corrector version of IQS factorizes the flux and derives the PRKE the same way as the standard version, but the evaluation of the coupled system is different. This version first solves the flux diffusion (represented by Equations 1 and 2) to get a predicted flux. The predicted flux at this step is then converted to shape by rescaling as follows:

$$\varphi_{n+1}^{g} = \underbrace{\phi_{n+1}^{g}}_{\text{predicted}} \frac{K_0}{K_{n+1}} \tag{10}$$

Where:

$$K_{n+1} = \sum_{g=1}^{G}\left(\phi^{*g}, \frac{1}{v^g}\phi_{n+1}^{g}\right) \tag{11}$$

$$K_0 = \sum_{g=1}^{G}\left(\phi^{*g}, \frac{1}{v^g}\varphi_{n+1}^{g}\right) = \sum_{g=1}^{G}\left(\phi^{*g}, \frac{1}{v^g}\phi_{0}^{g}\right) \tag{12}$$

The PRKE parameters are then computed with this shape using Equations 6 - 8 and interpolated over the macro step, then the PRKE is evaluated. With the newly computed amplitude, the shape is rescaled and the corrected flux is evaluated:

$$\underbrace{\phi_{n+1}^{g}}_{\text{corrected}} = p_{n+1} \times \varphi_{n+1}^{g} \tag{13}$$

The precursors need to then be updated, using any of a variety of methods previously discussed in the "Precursor Integration" memo. Theta method evaluation looks like:

$$C^{n+1} = \frac{1-(1-\theta)\Delta t\lambda}{1+\theta\Delta t\lambda}C^n + \frac{(1-\theta)\Delta t\beta}{1+\theta\Delta t\lambda}S_f^n + \frac{\theta\Delta t\beta}{1+\theta\Delta t\lambda}S_f^{n+1} \tag{14}$$

Where:

$$S_f^n = \sum_{g'=1}^{G} \left(\nu^{g'}\Sigma_f^{g'}\right)_n \underbrace{\phi_n^g}_{\text{corrected}} \tag{15}$$

There is no iteration necessary for this method and in turn is much simpler and faster than the standard IQS.

3. **Rattlesnake Implementation**

In order to preserve the already implemented standard version of IQS, another executioner called IQStmp was created that can be specified in the input deck. This executioner references the original IQS executioner to execute the PRKE. Because the diffusion solve is flux instead of shape, when IQStmp is specied, the IQS removal kernel ($\frac{1}{v^g}\frac{1}{p}\frac{dp}{dt}$) is bypassed, while all the postprocessors are still executed. However, it is difficult to rescale the flux to shape before the PRKE parameter postprocessor are executed. So the parameters are computed using the full flux, but amplitude is space independent and comes out of the integrals. As seen in parameter definitions, when shape is replaced with flux, the amplitude comes out of the integral and cancels out. So instead of rescaling shape, the predicted flux is rescaled as follows:

$$\underbrace{\phi_{n+1}^g}_{\text{corrected}} = \underbrace{\phi_{n+1}^g}_{\text{predicted}} \frac{K_0}{K_{n+1}} p_{n+1} \tag{16}$$

How it is implemented in Rattlesnake is as follows:

```
IQStmp::endStep(Real /*input_time*/)
{
  IQS::endStep();
  Real z = (*_Lambda)/_Lambda_orig;
  _nl.update();
  _nl.solution().close();
  _nl.solution().scale(_X.get_values()[0]/z);
  _nl.solution().close();
  _nl.update();
}
```

However, how to update the precursors auxilary variable is still not understood. The predicted easiest way is to simply re-evaluate the auxkernel, instead of doing something complicated in the executioner.

ZMP,JCR:zmp,jcr

Distribution:
Yaqi Wang, INL, *yaqi.wang@inl.gov*

Cy:
File