Hellow,

I found two flaws.

1. Four Lagrangian interpolation functions in namespace Polylib may have significant errors when evaluation near interpolation points. This problem affects interpolation operations' accuracy.

   These four functions are the follows:

   double hgj (const int i, const double z, const double *zgj, const int np, const double alpha, const double beta),

   double hgrjm (const int i, const double z, const double *zgrj, const int np, const double alpha, const double beta),

   double hgrjp (const int i, const double z, const double *zgrj, const int np, const double alpha, const double beta),

   double hglj (const int i, const double z, const double *zglj, const int np, const double alpha, const double beta).

   The reason of this problems is this. These four functions using formulas like this one

$$h_j(z) = \begin{cases} \dfrac{(1-z^2)P_{np-2}^{\alpha+1,\beta+1}(z)}{((1-z_j^2)[P_{np-2}^{\alpha+1,\beta+1}(z_j)]' - 2z_j P_{np-2}^{\alpha+1,\beta+1}(z_j))(z-z_j)} & \text{if } z \neq z_j \\ 1 & \text{if } z = z_j \end{cases},$$

   used in hglj. To judge whether z equals z_j, a parameter EPS= 100*DBL_EPSILON is adopted as a threshold, and EPS = 2.E-14. Problem happens when z is very close to z_j but their distance is still greater than EPS, since at this situation the formula in h_j contains subtraction of two very close (double) float numbers, which causes heavy loss of accuracy.

   One example is in Gauss Lobatto interpolation function hglj with n=12, I evaluate its value at point -0.81927932164452633, which differs from the third zero -0.81927932164400674 by 5E-13>EPS. Function hglj gives the value 1.00015479671952878, of which the error is 1E-4.

   I use a new parameter EPSERR=3E-3 as threshold specially for these four functions to judge whether z is close to z_j enough, and if abs(z-z_j)<EPSERR, I adopt a traditional Lagrange interpolation function $\mathrm{h_j}(z) = \prod_{\substack{0\le m<n \\ m\neq j}} \frac{z-z_m}{z_j-z_m}$ to evaluate its value. This gives a very good overall accuracy.

2. In function Nektar::FieldUtils::InputXml::Process ,the “if” statement at line 356 version 4.4.0. I think it should be this if(m_requireEquiSpaced) but not if(m_requireEquiSpaced || vm.count("output-points")). Since in the current codes, when --output-points and --noequispaced options are used simultaneously, the –equispaced option will not work.

And still, I have a question. In function Nektar::Extrapolate::AccelerationBDF when solving incompressible Navier-Stokes equation using higher order Neumann pressure boundary condition, why should the backward differentiation formula start after m_pressureCalls > 2? since when m_pressureCalls=2, we can already obtain a first order accuracy value of $\frac{\partial u}{\partial t}$. I changed that number to 1, like the following codes

```
if (m_pressureCalls > 1)
{
    int acc_order = min(m_pressureCalls-1,m_intSteps); …
```

and it seems nothing goes wrong in my simulation.

Best wishes

Ankang