DGM Developer's Guide

S. Scott Collis, Curtis C. Ober

February 5, 2014

Abstract

The DGM developer's guide is a collection of notes for developer's to help document and educate developer's with conventions and coding practices for the development of DGM.

Chapter 1

Basics

1.1 Formatting

The basic formatting conversions are:

1. Text will fit in 80 columns. Many editing packages, webviewing, and other GUIs default to 80 columns. This way text wrapping can be avoided.

1.2 Use of Ordinal, int, Size, LocalSize, GlobalSize, Scalar

From Scott: Here is a quick version:

- DGM::Scalar (the type held by Elements to represent the PDE solution)
- DGM::Real (a real number, used for coordinates and such), currently
- DGM::Real is DGM::Scalar but one day DGM::Scalar could be
- std::complex<DGM::Real> for example.
- DGM::Ordinal ≡ DGM::LocalSize and is used to index on-node quantities
- DGM::Size ≡ DGM::GlobalSize and is used to index full-scale quantities

Notes:

1. In general, DGM::Ordinal and DGM::Size must be okay to be either signed or unsigned.
2. The basic types are Ordinal and Size but we sometimes may typedefs (or template on) LocalSize and GlobalSize as this can make the code easier to decipher.
3. All CMC::Vector are indexed on DGM::Ordinal since this must fit on one core.
4. If you want a maximally 64bit version then you build with DGM::Ordinal = size_t and DGM::Size = unsigned long long. This works great, but due to limits in Trilinos, we cannot enable it when this combination.

int is only used when it *must* be (like for MPI and Epetra) and for return values that get communicated back to the OS. When we need too, we use boost::numeric_cast<int> to convert Ordinal and Size to ints and this will catch overflows.

It is bad when int or unsigned are used directly (even locally deep in a kernel) and I've had to tease a number of these out as they may lead to overflows one day.

From Drew: Running ROL in DGM.

1. To choose ROL optimizers, change the `opttype` parameter in `root.inp` to 4.

2. The specific optimization algorithm used is determined through parameters in `root.rol`. For an example of `root.rol` see

   `/dgm/examples/dakota/heat-dgm/heat.rol.`

3. The ROL-DGM interface has the feature to mimic either DGM or PEopt `root.ocs` output. This output specification is controlled through the parameter `output_type` in `root.inp`. The parameter `output_type` corresponds to an unsigned integer. Independent of the value of `output_type`, the ROL optimizer creates `root.ocs` in ROL format. If `output_type` is 1, then we create the file `root_dgm.ocs` using DGM-style output. If `output_type` is 2, then we create the file `root_peopt.ocs` using PEopt-style. If `output_type` is 0 or greater than 2, then we only create `root.ocs`. Note that in DGM style output, the "`Alpha`" corresponds to the norm of the optimization step, not the linesearch parameter size. Similarly, for peopt-style output, "`||dx||`" corresponds to the norm of the optimization step, not the preconditioned step. Furthermore, for PEopt-style output, we do not include "`LSIter`" or the Krylov columns because ROL manages these components internally.