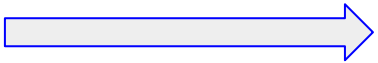
Abstract Fortran in NIMROD Discussion

NIMROD team meeting
Sherwood, April 21st 2018
Auburn, AL

What is abstraction? Why should we do it?

With abstraction we can rewrite hard-coded blocks loop from this:

```
DO ibl=1,nrbl
  rb(ibl)%be%fs …
ENDDO
DO ibl=nrbl+1,nbl
  tb(ibl)%be%fs …
ENDDO
```



To this:

```
DO ibl=1,nbl
  bl(ibl)%be%fs …
ENDDO
```

Concrete type of bl is determined at runtime

- NIMROD is *already* designed/structured in an abstract way -- changes aren't that radical

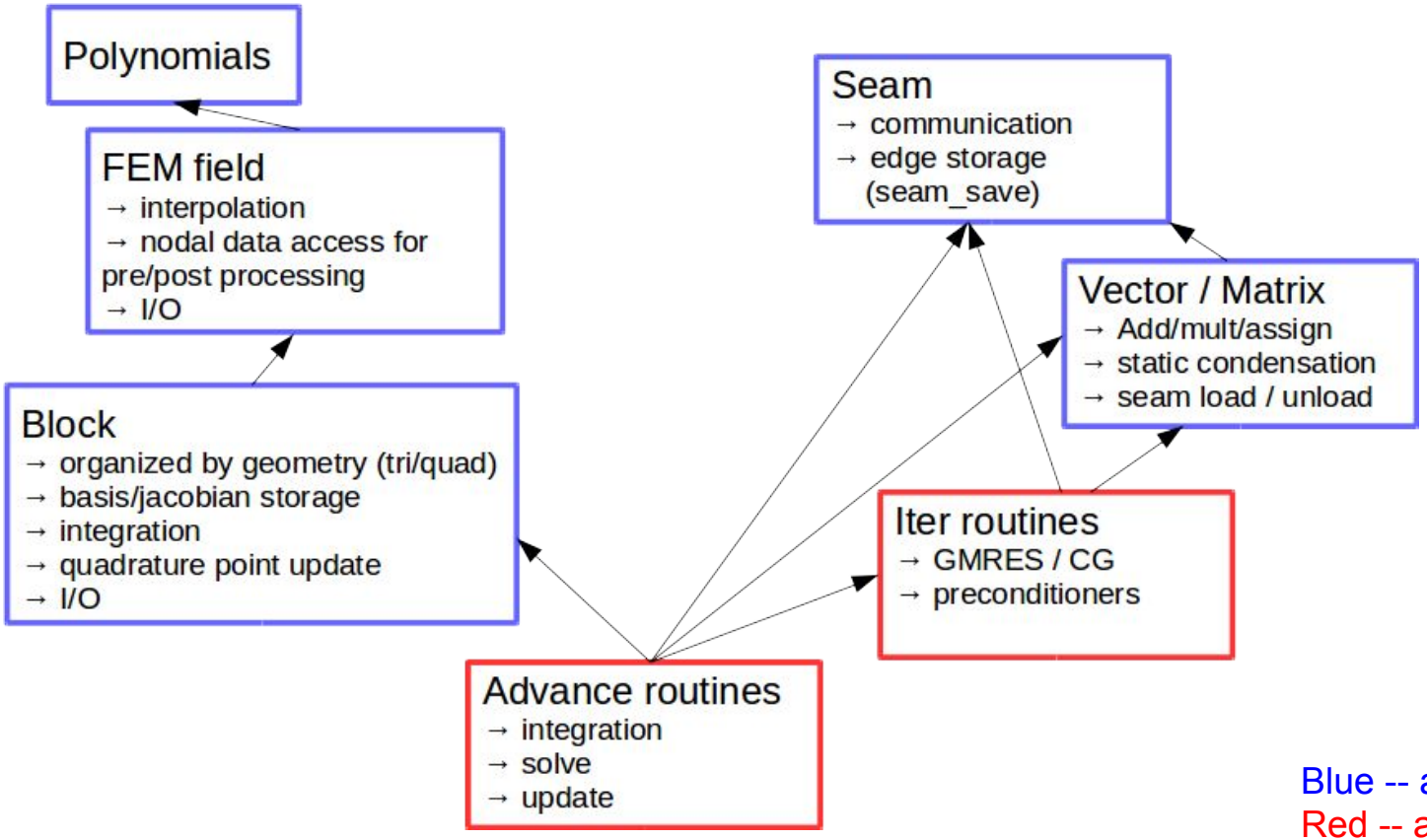
What are the goals of this effort?

- Goals are oriented towards physics objectives
- Optimization for continuum kinetic (CK) computations
  - Many CTTS CK-related planned -- NTMs, RWMs and energetic particles
  - Other CK-related efforts: RMPs, neoclassical flows/current
- Flexibility with meshing
  - E.g. tokamak calculation with structured closed-flux region grid coupled to unstructured grid that conforms to the limiter/divertor geometry
  - Potentially significant for disruption computations (VDEs, ideal and locked modes)
- Flexibility with basis functions
  - E.g. H(curl) and H(div) elements
  - Enable exploration of additional numerical methods

Fortran 90 or 2008/2013?

- Abstraction can be done in Fortran 90
- Fortran 2008/2013 adds useful object oriented functionality:
  - Blocks/Fields are cleanly described as self-contained objects
  - Use of abstract base class defines an interface that instantiations must conform to
  - “Program to interface, not an implementation”
  - Easier to maintain/extend/test code base
  - Consider an interface to “fields” (presently H1 lagrange_quad):
    - Basic functionality: interpolation of basis (evals)
      - Requires polynomial evaluation
    - Extended functionality: initialization (alloc/dealloc/set values at nodes)
- But don’t use object oriented program if not needed!
  - Abstract interfaces (callback) may be more appropriate
  - Present example: integrand routines
  - Planned example: generalized iterative solver routines (GMRES/CG)

Flow-chart overview of NIMROD design



Blue -- abstract types
Red -- abstract interfaces

Discussion outline

1. Present Status
   - Polynomials, fields, blocks, I/O
   - Unit testing, documentation (ford)
   - Abstractions for continuum kinetics and H(curl)/H(div) elements
2. Imminent development
   - Vectors, matrices, seams, iter routines
   - Timers
   - Nimeq and Nimrod
3. Long-term development
   - Boundary condition infrastructure, map_mod
   - Regression testing
   - Input (place grid information in dump file?)
   - Nimset, fluxgrid/fgnimeq, nimplot, etc.

Fields (status)

- Functionality: Evaluate basis functions and store values at nodes
- Example: lagrange_quad.f90
- New abstract base class: nodal.F90
- Draft implementation complete:
  - open to discussion
  - Planning changes

Fields (open issues)

- Should fields save alpha functions?
  - Separating alpha storage from qp storage

H(curl) motivated adjustments

- Need concept of nqty separate from concept of number of spatial dimensions
- Tests need to be more general
- qpfr/z could be r/z * appropriate differential operator

Polynomials

- Gauss-Legendre and Lobatto-Legendre cardinal functions fully implemented with tests
- Extendible with common interface
  - `eval` and `eval_d` for cardinal function evaluation
  - `getnodes` for nodal locations
  - Initialized for given polynomial degree
- Evaluation implementation uses dense linear algebra by precomputing Legendre coefficients of cardinal functions (same as before with ‘gll’)
- Calls level 2 BLAS when available
- Can be included in fields for 1-D function evaluation
- Should be thread-safe except for initialization

Blocks

- Stores nodal and quadrature fields and Jacobian, controls I/O
- Presently handles qp_updates
- H(curl) and H(div) qp_updates use the Jacobian differently than H1 elements
  - Plan: bind qp_update to field and pass Jacobian from the block
  - Related issue: nqty for derivatives with H(curl) and H(div) is not the same as the nqty for the field
  - Plan: Also move qp_alloc to be bound to nodal fields
- Stores basis functions with label
  - Initialization is now deferred to nodal fields
- make_matrix/get_rhs needs implementation
  - Will move assembly code to corresponding vector/matrix types

Documentation and Unit tests

- Documentation with ford (FORtran Documentation)
  - Doc-string generation for code interface
  - Pages with markdown syntax
- Unit tests
  - Good development pattern: program interface as test, code it interface
  - Help prevent code regression
  - Exercise abstract interfaces

Discussion outline

1. Present Status
   - Polynomials, fields, blocks, I/O
   - Unit testing, documentation (ford)
   - Abstractions for continuum kinetics and H(curl)/H(div) elements
2. Imminent development
   - Vectors, matrices, seams, iter routines
   - Timers
   - Nimeq and Nimrod
3. Long-term development
   - Boundary condition infrastructure, map_mod
   - Regression testing
   - Input (place grid information in dump file?)
   - Nimset, fluxgrid/fgnimeq, nimplot, etc.

Vectors and Matrices

- Next to implement
- Vector type should be straightforward
  - add, mult, equal
  - Move assembly implementation as function
  - Move seam load/unload implementation as function
- Matrix type
  - Static condenstation, matvec
  - Move assembly implementation as function
  - Move seam load/unload implementation as function
  - Static con: a CK implementation should speed up computations

Seams

- Seam load-unload will be incorporated into vector/matrix types
- Switch to F2013 from F90 based?

Should seams be incorporated into a more complete mesh type?

- Mesh responsibilities
  - Parallel Partitioning
  - Geometry - mapping and jacobian
  - Connectivity - local connectivity for unstructured meshes, partition connectivity is sufficient for structured meshes
  - DoF mapping - need for each type of element; H1, H(curl), H(div)
- Somewhat conflicts with current responsibilities of blocks
- Concepts of mesh are well developed and various implementations exist
  - Using a mesh library could accelerate use of new mesh geometries, but may be more disruptive to the code base overall
  - Homegrown implementation could fit in the code base better, but could be a large development investment

Iter routines

- Present CG/GMRES routines have many versions:
  - REAL - COMPLEX - linear - nonlinear - CK
  - Recent development: callbacks to threed_dot_mgt routines for mat-vec dot product eval
  - Allows for BC and regularity during dot evals
  - Planned: callback to preconditioner routines (unify linear/nonlinear/CK routines)
- Sparsity pattern creation is complication to abstraction
  - Code presently in iter_utils will be incorporated in to matrix classes for abstraction
  - Perhaps there is the possibility to streamline with seam structures for the matrix?
    - E.g. one-way matrix element communication

NIMEQ and NIMROD

- Need to reintegrate code with NIMROD ASAP
- Present (tentative) timeline: Fluid advance by summer, CK advance by DPP
- Plan to update NIMEQ first as test-bed

Discussion outline

1. Present Status
   - Polynomials, fields, blocks, I/O
   - Unit testing, documentation (ford)
   - Abstractions for continuum kinetics and H(curl)/H(div) elements
2. Imminent development
   - Vectors, matrices, seams, iter routines
   - Timers
   - Nimeq and Nimrod
3. Long-term development
   - Boundary condition infrastructure, map_mod
   - Regression testing
   - Nimset, fluxgrid/fgnimeq, nimplot, etc.

Boundary infrastructure

- Plan for NIMDEVEL-like lagrange_edge infrastructure but using abstract fields with ndim-1
- Initially plan to target surface integrals and dirichlet/”no_*_bc” operations
- Would like to add regions specified at initialization to seam0
  - Boundary conditions are then specified as an array corresponding of size nregions
  - Standard from other codes

Mapmod

- Map_mod is a nimdevel-only infrastructure:
  - solves the “inverse mapping problem” R,Z -> ix,iy
  - Also facilitates global storage of fields (e.g. for field line traces)
- Map_mod is predicated on a global structured grid
- Map_mod enables easy-parallelism for field-line tracing, particles pushing and integral closures
- Potential path forward, two versions:
  - One for a globally structured (fast)
  - One for a general mesh (using linked list?)

Continuous Integration (git?)

- With the inclusion of unit tests, “Continuous Integration” (CI) ideas could be used
- CI is well-integrated into most git hosting systems and free or paid services are available
- CI provides additional “checks” during development, i.e. it can be required that unit tests pass in order to merge functionality with ‘trunk’

Propose to move abstractBlock development to GitHub or GitLab to leverage this and provide more opportunity for code review. (Open to suggestions on **comparable** SVN workflows.)

Nimset/fluxgrid/fgnimeq/nimeq

- Present (in nimdevel) these are all coupled
- Plan:
  - Separate nimset from fluxgrid/nimeq
  - Potentially add common infrastructure library
    - Field profiles
    - Seam creation
    - Allocation (?)
  - Move fgnimeq fully into nimeq
    - Historically: separated for code maintenance
    - Keep fluxgrid separate but only nimeq uses it