## PETSc Perspective on PDE Solving

Portable Extensible Toolkit for Scientific computing

Solving generic PDEs can be broken down into a series of steps:

| | | |
|---|---|---|
| 0. | Define continuous problem (domain, BCs, etc.) | continuous PDE |
| 1. | Discretize the domain | system of ODEs |
| 2. | Solve ODEs using time integrators | system of non-linear eqns |
| 3. | Solve nonlinear equations | system of linear eqns |
| 4. | Precondition the linear system | preconditioned " " |
| 5. | Solve preconditioned linear system | |

PETSc has capabilities to handle all of this for it's users. Each step has a different module/object associated with it:

| | | | |
|---|---|---|---|
| 1. | Discretization | DM | |
| 2. | Time stepping | TS | Solvers |
| 3. | Non-linear Solver | SNES | Solvers |
| 4. | Preconditioning | PC | Solvers |
| 5. | Linear Equation | KSP | Solvers |

## PETSc DM

Handles the connection between a discretization and PETSc's Solvers.

Specifically, it handles:

- Global ↔ Local data communication
- Boundary condition removal from system (ie Strong BCs)
- Vector index ordering
- Geometric multigrid
- Other discretization specific tasks (interpolation, graph coloring, etc.)

## PETSc Solvers

The libCEE-Fluids code uses the full Solver stack. As such, the primary interface between libCEED and PETSc is through the TS module.

The TS module solves problem of the form:

$$F(t, u, u_{,t}) = H(t, u) \quad , \quad u(x, t_0) = u_0$$

Users provide functions to evaluate F and/or H and PETSc calls them when necessary.

$F(t, u, u_{,t})$ is the IFunction

$H(t, u)$ is the RHSFunction

For explicit methods, we assume $F = u_{,t}$ and only H is provided.
For implicit methods, F must be given and H is optional.

To solve non-linear equation, PETSc requires the Jacobian of F (aka the Tangent matrix).
It requires the form:

$$\frac{dF}{du} = F_{,u_{,t}}(t, u, u_{,t})\sigma + F_{,u}(t, u, u_{,t})$$

The Jacobian of F is stored as PETSc Mat object, which is created by a function IJacobian.

> IFunction, RHSFunction, IJacobian constitute the primary interface between libCEED and PETSc.

We use CeedOperators to perform the functions.

## Matrix-Free Jacobian

libCEED-Fluids can use a matrix-free approach to evaluate Jacobian matvecs.
But different from PHASTA method.

FDMF: $\frac{\partial G}{\partial Y}\underset{\sim}{v} = \frac{G(\underset{\sim}{Y} + \varepsilon \underset{\sim}{x}) - G(\underset{\sim}{Y})}{\varepsilon}\underset{\sim}{v} = \frac{G(\underset{\sim}{Y} + \varepsilon \underset{\sim}{x}) - G(\underset{\sim}{Y})}{\varepsilon}$

libCEED uses the total derivative of the residual combined storing intermediate data from residual calculation to compute exact Jacobian matvec

Remember: Total Derivative is the change of inputs contracted with the Jacobian,

$\therefore$ Jacobian matvec = total derivative

when the change of inputs is the vector to be multiplied by the Jacobian.

$$G(\underset{\sim}{Y}, \underset{\sim}{Y}_t) \xrightarrow{d} dG(\underset{\sim}{Y}, \underset{\sim}{Y}_t, d\underset{\sim}{Y}, d\underset{\sim}{Y}_t) = G_{,\underset{\sim}{Y}}(\underset{\sim}{Y}, \underset{\sim}{Y}_t)\, d\underset{\sim}{Y} + G_{,\underset{\sim}{Y}_t}(\underset{\sim}{Y}, \underset{\sim}{Y}_t)\, d\underset{\sim}{Y}_t$$

Ignoring stabilization, $G_{,\underset{\sim}{Y}_t} = I$ and without loss of accuracy, express

$$d\underset{\sim}{Y}_t = \sigma\, d\underset{\sim}{Y}$$

We end up with:

$$dG(\underset{\sim}{Y}, d\underset{\sim}{Y}) = G_{,\underset{\sim}{Y}}(\underset{\sim}{Y})\, d\underset{\sim}{Y} + \sigma\, d\underset{\sim}{Y}$$

We store $\underset{\sim}{Y}$ and other intermediate values to accelerate Jacobian calculation.

$dG$ is calculated without forming $G_{,\underset{\sim}{Y}}$.

Comparison

| FDMF | Total derivative MF |
|---|---|
| Cheap ($G(\underset{\sim}{Y} + \varepsilon \underset{\sim}{X})$) | pretty cheap ($\approx G(\underset{\sim}{Y})$) |
| No Jacobian code* | Require calculating and coding Jacobian |
| Approximate Jacobian | Exact Jacobian |