Flexible stellarator optimization with Julia

Benjamin Faber

Julia for scientific computing

Julia: a new language for scientific computing
`https://julialang.org`

- Why use Julia:
  - Fast: C/Fortran like performance without the overhead
  - Dynamically typed: just-in-time compilation for optimized code without type specification
  - Multiple-dispatch: the same function can act on multiple types, reducing code bloat
  - Open-source, continually developed, achieving wide use/support within scientific computing community
- Effectively mixes good elements of object-oriented programming with functional programming and scripting languages
- **Natively** control every aspect of scientific work from Julia: fast linear algebra, visualization, file management, parallel programming...

Julia for stellarator optimization

- Need to support very general objective functions:
  - Different physics targets can have different requirements $\Rightarrow$ can lead to large overhead in statically typed languages
    - Adding a new physics module to STELLOPT is a non-trivial task
  - Dynamic typing allows for targets to be added/removed on the fly; multiple dispatch provides concise methods for calling targets:

```
ModuleTarget <: AbstractTarget
# General routine for computing objective functions
function computeTarget(target::AbstractTarget,kwargs...)
        ...
        computeTarget(target) # Calls module defined function
        ...
end
# Module defined routine
function computeTarget(::ModuleTarget,kwargs...)
```

Julia for stellarator optimization

- Ease in interfacing with legacy C/Fortran routines
- Directly access shared library routines (i.e. VMEC, SPEC)
- Significantly cuts down on number of lines of code vs. statically typed language:
  - C++ VMEC interface: $\sim$3000 lines of code
  - Julia VMEC interface: $\sim$ 800 lines of code
- No need for special libraries to access Fortran subroutines, variables (i.e. `f2py`, `f90wrap`)
- MPI interface exists to seamlessly pass communicators between routines
- Like Python, Julia provides REPL (read-eval-print loop) functionality
- Allows for direct manipulation, visualization of underlying data structures $\Rightarrow$ huge advantage to writing and debugging software

Optimization road map

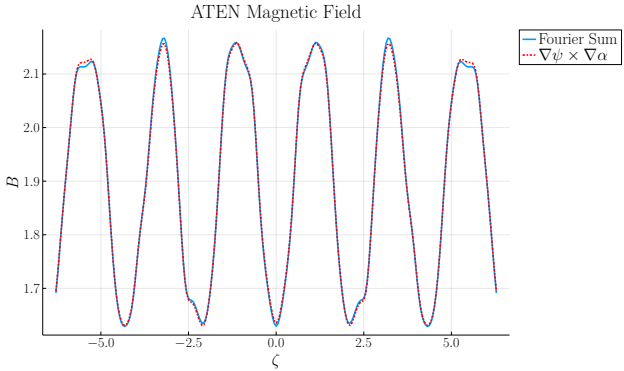
- Stellarator optimization road map
  - Interfaces with equilibrium solves: accomplished for VMEC (VMEC.jl)
  - General purpose routines for manipulating geometry data: PET.jl (Plasma Equilibrium Toolkit)
    - Provide routines for curvilinear coordinate transformations of vector data
    - Cartesian, cylindrical, PEST, VMEC, Boozer,...
    - Translate geometry data between equilibrium solvers, targets
  - Write physics targets in Julia
    - QS, ϵ_{eff}, Γ_c accomplished so far
  - Build distributed, scalable infrastructure around pre-built optimization libraries (`mango`, Optim.jl): lasso.jl (Lightweight Algorithms for Scalable Stellarator Optimization)

Design philosophies

- **Use libraries whenever possible**, the Julia package management system ensures compatibility
- For example, to compute radial dependence of VMEC data, use cubic B-splines from Interpolations.jl $\Rightarrow$ accurate, fast and memory efficient
  - Circumvents issues arising in STELLOPT with custom EZSpline library
- QuadGK.jl: optimized quadrature library used for accurate adaptive integration
- CoordinateTransformations.jl: provides template for extremely efficient coordinate transformations
- Combined, leads to more robust code base

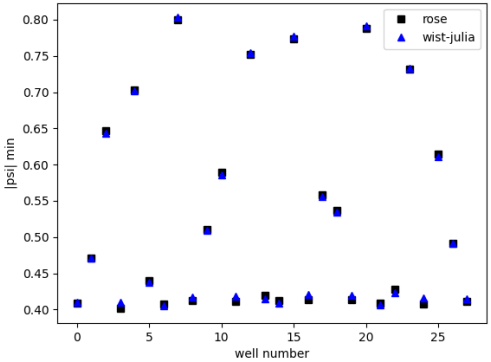
Example: VMEC.jl + PET.jl

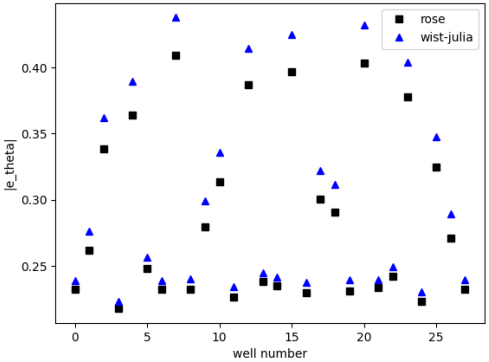
- Define a custom vector type to describe data in 3D space (e.g. $\nabla\psi$)
- Translate VMEC Fourier series data to 3D space in different coordinate systems
- Compute $B(\theta,\zeta) = \sum_{m,n} B^c_{mn} cos(m\theta - Nn\zeta)$ or construct $B = |\nabla\psi \times \nabla\alpha|$



Example: Γ_c target

- Proxy for evaluating fast particle transport (see Aaron's work)
- Compare geometry elements with ROSE





Next steps

- Finish `mango` interface
- Perform optimization with VMEC, `mango`, quasisymmetry and ϵ_{eff}
- Contributions are welcome!
- `https://gitlab.com/wistell`