Optimizing for ITG growth rates with StellaratorOptimization.jl

Benjamin Faber, Joey Duff, Aaron Bader

March 11, 2022

- `lasso.jl` has been renamed to the more descriptive `StellaratorOptimization.jl`
- Registered with the Julia package manager; available to install March 14
  - *Caution:* Packages are currently under **active** development, and may still undergo rapid changes
- Accompanying `StellaratorOptimizationMetrics.jl` package registered containing a collection of useful optimization metrics
  - Γ_c and ϵ_{eff} metrics migrated here
  - Adding MHD infinite-n ballooning stability metric (rewrite of COBRAVMEC, finished), $\nabla B : \nabla B$
- Tested with MPI, multithreaded parallelization on single node, multinode testing/debugging ongoing at NERSC
- Integration with the `PlasmaViz.jl` package for visualization ongoing

Improvements to StellaratorOptimzation.jl

- `StellaratorOptimization.jl` substantially more lightweight: *no* explicit dependencies on equilibrium solver, physics target packages
- New packages interact with `StellaratorOptimization.jl` by *extending* existing methods through multiple dispatch
- Key to success: providing new types that wrap existing modules/functions
  - Solvers and physics targets retain separate scopes → no pollution of variables/functions!
- `EquilibriumWrapper`:

```
struct EquilibriumWrapper{T, E <: AbstractMagneticEquilibrium}
  eq::E
  input::Dict
  translator::Dict
  derived_geometries::Dict
end
```

- `VMEC.jl` implements methods for computing new equilibria, updating input and translating state vector info

Improvements to StellaratorOptimization.jl

- No longer necessary to wrap physics targets in submodules: *any* function can be called as a target
- Target modules (distinct variable scopes) need to only extend a minimum of two `StellaratorOptimization.jl` methods:
  - `AbstractModuleWrapper`:

```
struct _ModuleID <: StellOpt.AbstractModuleWrapper end
export _ModuleID
```

  - `evaluate_target_expression`:

```
function StellOpt.evaluate_target_expression(::Union{T, Type{T}},
                                             method::Expr
                                            ) where {T <: _ModuleID}
    return eval(method)
end
```

Improvements to StellaratorOptimization.jl

- Simplified input: input given provided in executable Julia syntax
  - Reduces need for special parsing, substantially simplifies target definitions
  - Allows for assignment of variables are function arguments
- Setup a `StellaratorOptimization.Problem`; add equlibria, derived geometry types, physics targets and optimization variables
- Derived geometries, targets are stored as `Expressions` and evaluated in appropriate module scope when necessary
  - Derived geometries are cached in dictionary, passed to appropriate targets to prevent recalculation

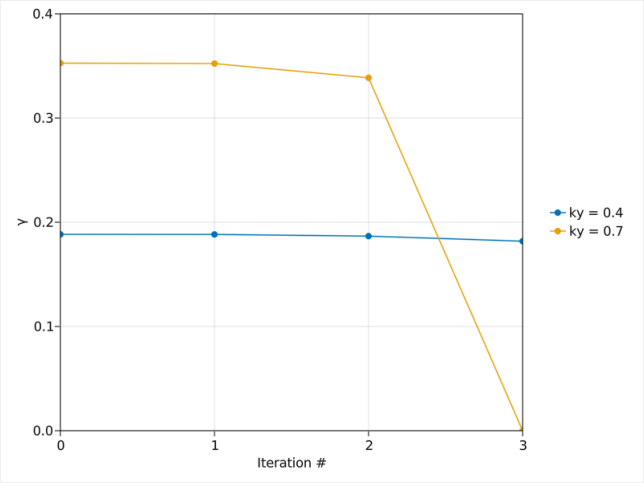
INPUT EXAMPLE

```
comms = setup_comms(procs_per_group, MPI.COMM_WORLD)
prob = Problem{Float64}()
opt = Optimizer(ConjugateGradient(), Optim.Options())
vmec_nml = VMEC.read_input_namelist("/path/to/vmec/input")
wrapper = EquilibriumWrapper{Float64, Vmec}(vmec_nml)
add_equilibrium_wrapper!(prob, wrapper)
vmec_s05 = @add_derived_geometry!(prob, VmecSurface(0.5, equilibrium = wrapper))
vmec_s08 = @add_derived_geometry!(prob, VmecSurface(0.8, equilibrium = wrapper))
@add_target!(prob, VMEC.quasisymmetry_deviation(1, 4, 16, 16,
             geometry = vmec_s05), 0.0, 1.0, "qs_vmec(s=0.5)")
@add_target!(prob, VMEC.quasisymmetry_deviation(1, 4, 16, 16,
             geometry = vmec_s08), 0.0, 1.0, "qs_vmec(s=0.8)")
zeta_min = 0.0; zeta_max = 100*pi; delta_zeta = 2*pi/100;
@add_target!(prob, StellOptMetrics.GammaC.gamma_c_target(geometry = vmec_s08,
             zeta_min, delta_zeta, zeta_max, 100, false, true),
             0.0, 1.0, "GammaC(s=0.8)")
add_variable!(prob, wrapper, "rbc(1,2)", finite_diff_step=0.001)
add_optimizer!(prob, opt); residual = optimize_mpi(prob, comms)
```

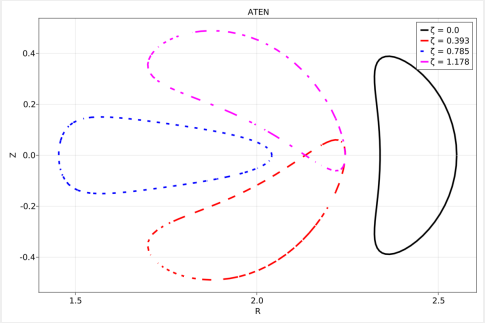
Flexible optimization

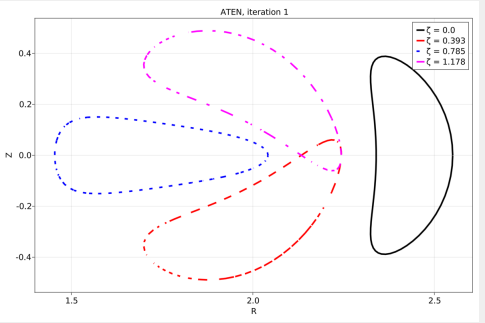
- Wrappers + multiple dispatch: no hardwired dependencies
- For example: optimization with local 3D equilibrium (`NE3DLE.jl`)
  - `NE3DLE` extends `StellaratorOptimization.jl` methods for `EquilibriumWrapper{T, NE3DLESurface}`
  - Only requires boilerplate routines in one small file, does not require any modifications to internal `NE3DLE.jl` routines
- Plug and play into `StellaratorOptimization.jl`, no compilation necessary
- Through `PlasmaEqulibriumToolkit.jl`, `NE3DLESurface` types can be used easily with physics targets, like Γ_c, quasisymmetry deviation, and `PlasmaTurbulenceSaturationModel.jl`
- First local 3D optimizations underway (J. Duff)

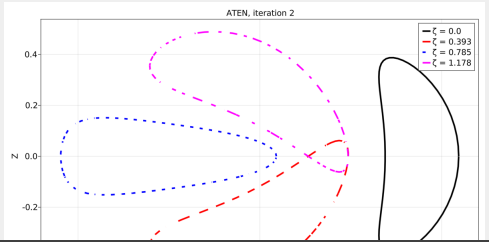
GROWTH RATES

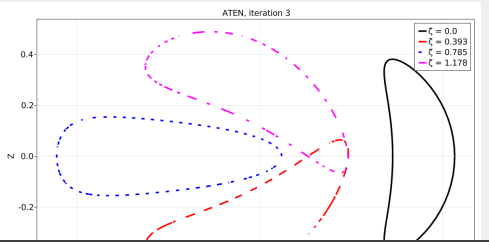


EQUILIBRIA


ATEN
ζ = 0.0
ζ = 0.393
ζ = 0.785
ζ = 1.178
Z
R


ATEN, iteration 1
ζ = 0.0
ζ = 0.393
ζ = 0.785
ζ = 1.178
Z
R


ATEN, iteration 2
ζ = 0.0
ζ = 0.393
ζ = 0.785
ζ = 1.178
Z


ATEN, iteration 3
ζ = 0.0
ζ = 0.393
ζ = 0.785
ζ = 1.178
Z