Documentation

Components of Julia RINO implementation

Required functionality:

- Deriving taylor approximations from functions, derivatives, gradients, and jacobians.
- Automatic differentiation to compute derivatives, gradients and jacobians of functions at affine points.
- Converting affine forms to intervals.
- Modal interval arithmetic: in particular we want to evaluate the mean-value extension (multiplication, addition, and matrix vector multiplication).

A critique of existing Julia components:

automatic forward differentiation https://github.com/JuliaDiff/ForwardDiff.jl
documentation: http://www.juliadiff.org/ForwardDiff.jl/stable/

Points:

- `ForwardDiff.derivative()` (as well as `.gradient()`, `jacobian()`) does not return a function, but rather produces a method that can be used to evaluate the derivative. This method only accepts subtypes of Real.
- There is an outstanding concern that library ForwardDiff is unusable for functions that have inputs and outputs that are not type Real:
  - It is possible to compute the $df(x)/dx$ where x is affine, provided that the type Affine is made a subset of type Real. However, this leads to some concerns (see section on Affine Arithmetic).
- RINO obtains taylor series from derivatives (and gradients, jacobians). As of now there is no clear way to do this in Julia.
  - It seems possible to change the source code of ForwardDiff to allow for arbitrary inputs.

intervals https://github.com/JuliaIntervals/IntervalArithmetic.jl
documentation: https://juliaintervals.github.io/IntervalArithmetic.jl/stable/

Points:

- Originally I considered implementing modal intervals (using the quantified modal interval interpretation $([a, b], Q) \in \mathbb{IR} \times \{\forall, \exists\}$) on top of IntervalArithmetic library. However IntervalArithmetic only admits outer approximations for intervals. Outer approximations are are done automatically within each interval operation.
- There are two possible approaches:
  - Modify IntervalArithmetic directly so that it admits and inner approximations of intervals. It may be possible to utilize exists macros that set approximations in package.
  - Create a completely new IntervalArithmetic library. In this case we may only need to implement modal interval multiplication and addition (matrix vector multiplication can invoke interval multiplication).

affine https://github.com/JuliaIntervals/AffineArithmetic.jl
documentation: none

Points:

- This library is currently a thin implementation (only implemented +, -, *, /, ==, zero, one, range) so I am implementing an affine arithmetic library from scratch. This library is being actively developed. Around mid-July, the went from one to four files.
- Any Julia library for affines must correspond with C++'s `aaflib` library. In particular it should use the same approximation Chebyshev approximation method for non-affine operations, as well as all elementary functions and binary arithmetic operations.

- Any implementation of affine forms must be able to save deviation coefficients in a compact manner. Otherwise, vectors for coefficients will increase linearly by the number of (non-affine) operations (something approaching $O(n^2)$ space considering affine forms proliferate).

taylor series https://github.com/JuliaIntervals/TaylorSeries.jl
documentation: http://www.juliadiff.org/TaylorSeries.jl/stable/

Points:

- Interestingly, TaylorSeries does not provide a method that takes a function as input to evaluate a taylor approximation. Instead it provides types Taylor1 and TaylorN that once passed as variables to a function generates a taylor approximation as output. This infers that we can obtain a taylor approximation of a derivative of f when passing f and an instance of Taylor1 to `ForwardDiff.derivative()`.
- TaylorSeries supports arbitrary types as coefficients (limitations?).

taylor models https://github.com/JuliaIntervals/TaylorModels.jl
documentation: https://juliaintervals.github.io/TaylorModels.jl/stable/

Points:

- Is built on top of the TaylorSeries module to provide rigorous bounds for the enclosure representing the error term of a taylor approximation.
- It may be possible to use TaylorModels to obtain an outer approximation of the Jacobian at each time step, I am hesitant on making this library a component of the Julia RINO port until I understand more on how inner approximations are computed in RINO.
- This is the leading package in Julia for validated numerics, and hence I'm interested in benchmarking RINO with TaylorModels.