A tour of **BifurcationKit.jl**

Journée Julia 2022

Romain VELTZ

Inria

Why Julia?

🤔

Outline

1. Introduction to Bifurcation Theory
2. Introduction to `BifurcationKit.jl`
3. Focus on equilibria
4. Focus on Periodic orbits
5. what about the docs?
6. The future of `BifurcationKit.jl`

1. Bifurcation Theory

Introduction to Bifurcation theory

Goal: predict new (time dependent) solutions to

$$\frac{dx}{dt} = F(x, p)$$

as function of a scalar parameter $p \in \mathbb{R}$.

Introduction to Bifurcation theory

Goal: predict new (time dependent) solutions to

$$\frac{dx}{dt} = F(x, p)$$

as function of a scalar parameter $p \in \mathbb{R}$.

- **(Equilibrium/Stationary point)** Encompasses finding roots

$$F(x^{eq}, p) = 0$$

Introduction to Bifurcation theory

Goal: predict new (time dependent) solutions to

$$\frac{dx}{dt} = F(x, p)$$

as function of a scalar parameter $p \in \mathbb{R}$.

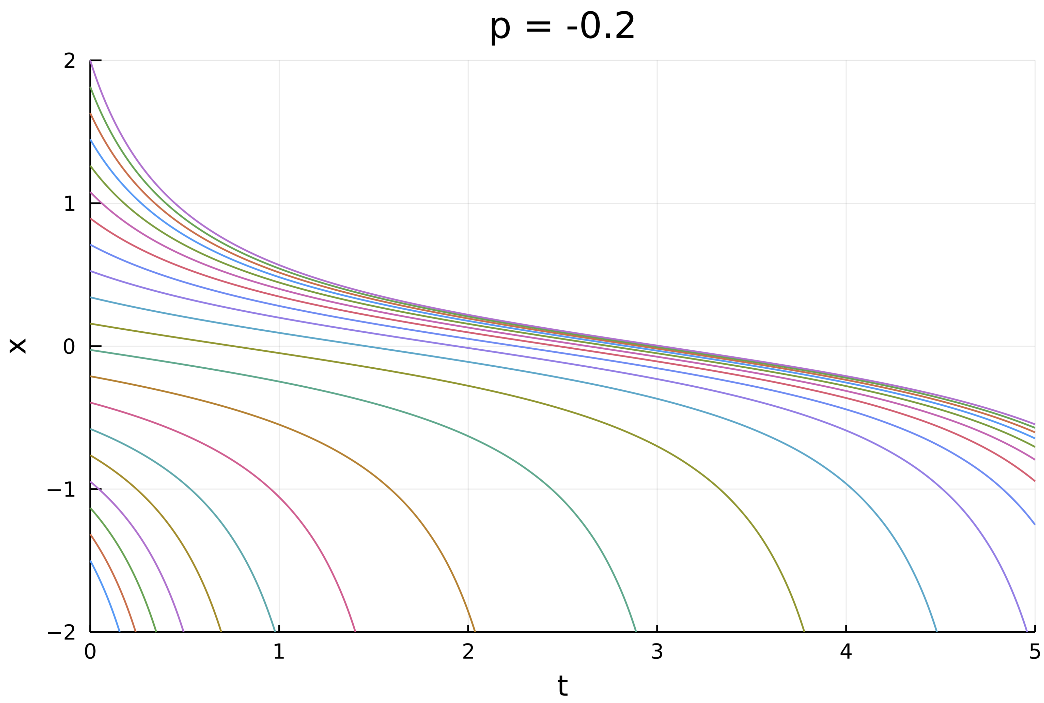
- **(Equilibrium/Stationary point)** Encompasses finding roots

$$F(x^{eq}, p) = 0$$

- indeed in this case $x(t) \equiv x^{eq}$ is a solution: $\frac{d}{dt}x(t) = 0 = F(x(t), p)$.

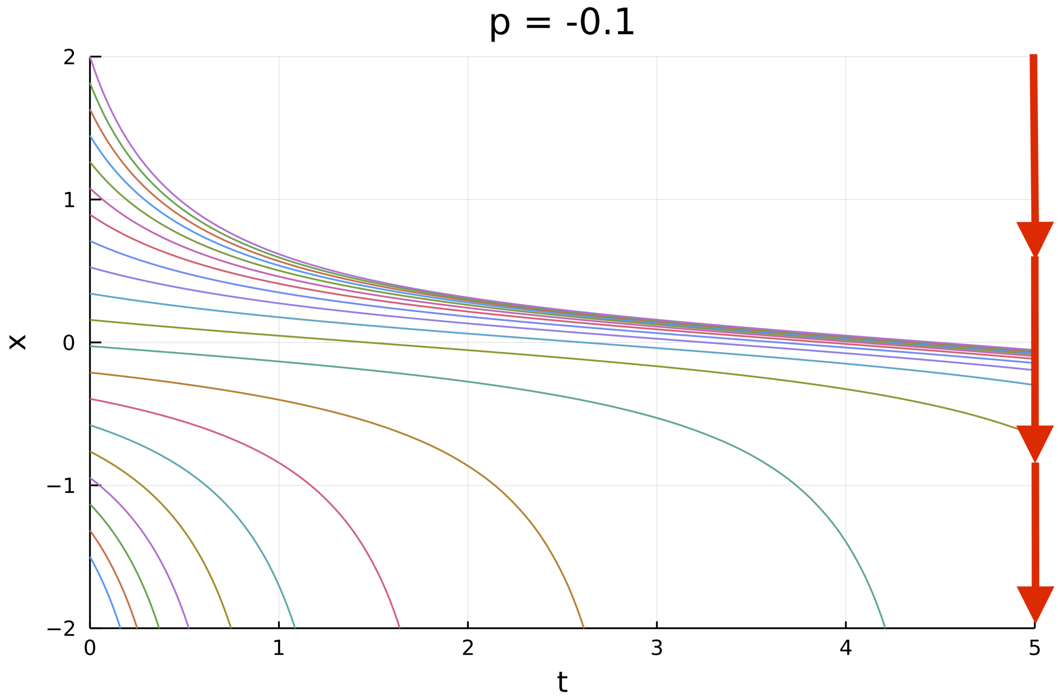
Example with Fold map

$$\frac{dx}{dt} = p - x^2 \in \mathbb{R}$$



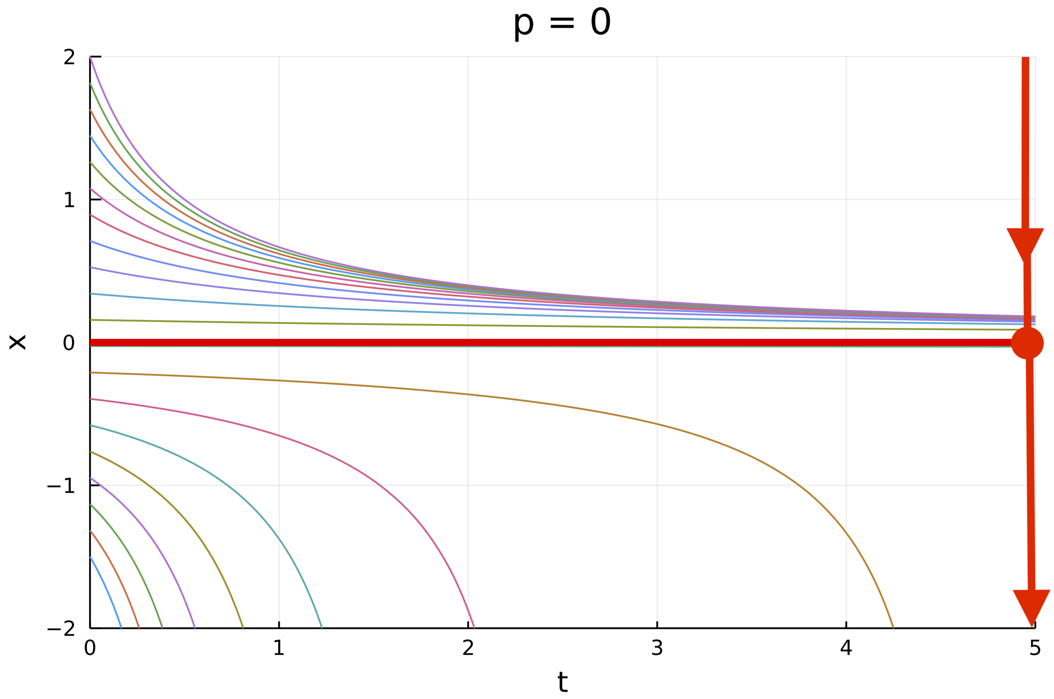
Example with Fold map

$$\frac{dx}{dt} = p - x^2$$



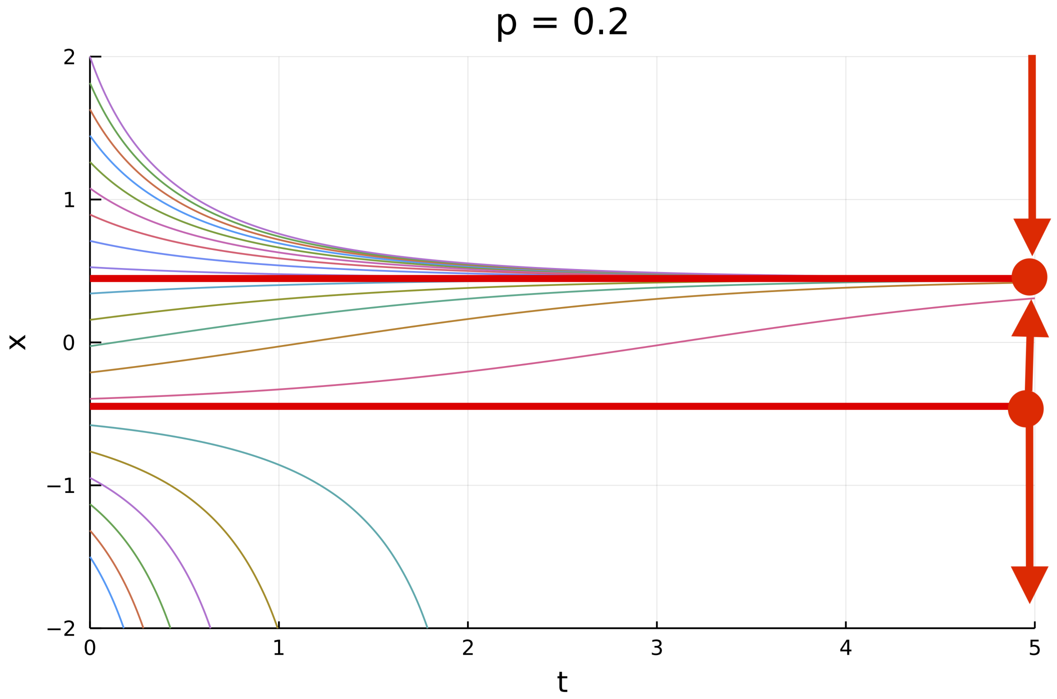
Example with Fold map

$$\frac{dx}{dt} = -x^2$$



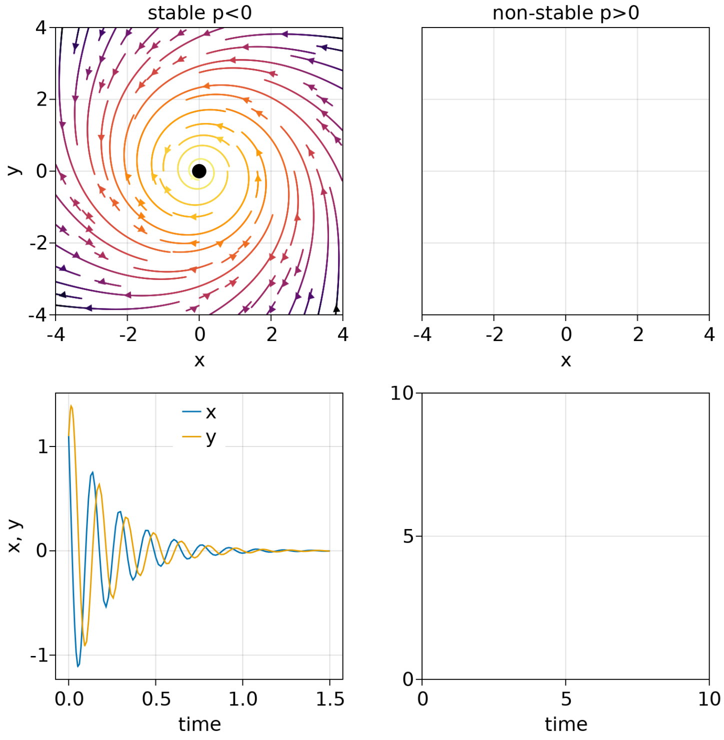
Example with Fold map (creation/annihilation of equilibria)

$$\frac{dx}{dt} = p - x^2$$



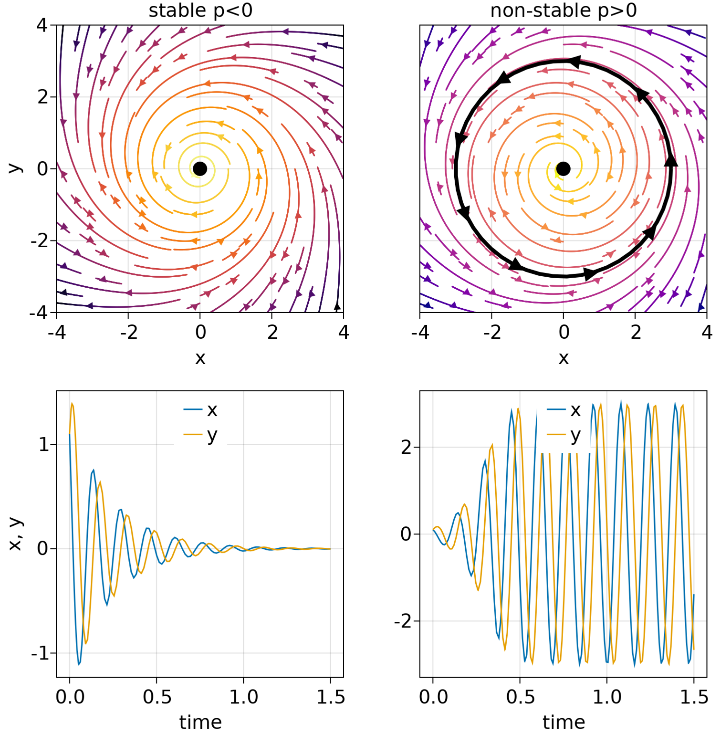
Example with Stuart-Landau oscillator

$$\frac{dz}{dt} = (p + i\omega - |z|^2)z \in \mathbb{C}, \quad z = x + iy, \quad p \in \mathbb{R}$$



Example with Stuart-Landau oscillator (Hopf bifurcation)

$$\frac{dz}{dt} = (p + i\omega - |z|^2)z \in \mathbb{C}, \quad z = x + iy, \quad p \in \mathbb{R}$$



Bifurcation theory amounts to

Bifurcation theory amounts to

1. continue (all) equilibria (zeroes) of $F(x, p)$ as function of scalar parameter p

Bifurcation theory amounts to

1. continue (all) equilibria (zeroes) of $F(x, p)$ as function of scalar parameter p
2. compute stability of the equilibria (given by number of eigenvalues $\lambda(p)$ with $\Re\lambda(p) > 0$)

Bifurcation theory amounts to

1. continue (all) equilibria (zeroes) of $F(x, p)$ as function of scalar parameter p

2. compute stability of the equilibria (given by number of eigenvalues $\lambda(p)$ with $\Re\lambda(p) > 0$)

3. detect a bifurcation point p_0 precisely, *i.e.* when $\Re\lambda(p_0) = 0$

   - Case $\lambda(p_0) = 0$: new equilibria (maybe)
   - Case $\lambda(p_0) = \pm i\omega$: nearby oscillations with frequency ω (maybe **Hopf** Bifurcation)

Bifurcation theory amounts to

1. continue (all) equilibria (zeroes) of $F(x, p)$ as function of scalar parameter p
2. compute stability of the equilibria (given by number of eigenvalues $\lambda(p)$ with $\Re\lambda(p) > 0$)
3. detect a bifurcation point p_0 precisely, *i.e.* when $\Re\lambda(p_0) = 0$
   - Case $\lambda(p_0) = 0$: new equilibria (maybe)
   - Case $\lambda(p_0) = \pm i\omega$: nearby oscillations with frequency ω (maybe **Hopf** Bifurcation)
4. switch to new branch of solutions and go to 1. (**Branch switching**)

Bifurcation theory amounts to

1. continue (all) equilibria (zeroes) of $F(x,p)$ as function of scalar parameter p
2. compute stability of the equilibria (given by number of eigenvalues $\lambda(p)$ with $\Re\lambda(p) > 0$)
3. detect a bifurcation point p_0 precisely, *i.e.* when $\Re\lambda(p_0) = 0$
    - Case $\lambda(p_0) = 0$: new equilibria (maybe)
    - Case $\lambda(p_0) = \pm i\omega$: nearby oscillations with frequency ω (maybe **Hopf** Bifurcation)
4. switch to new branch of solutions and go to 1. (**Branch switching**)

What is needed:

Bifurcation theory amounts to

1. continue (all) equilibria (zeroes) of $F(x,p)$ as function of scalar parameter p
2. compute stability of the equilibria (given by number of eigenvalues $\lambda(p)$ with $\Re\lambda(p)>0$)
3. detect a bifurcation point p_0 precisely, *i.e.* when $\Re\lambda(p_0)=0$
   - Case $\lambda(p_0)=0$: new equilibria (maybe)
   - Case $\lambda(p_0)=\pm i\omega$: nearby oscillations with frequency ω (maybe **Hopf** Bifurcation)
4. switch to new branch of solutions and go to 1. (**Branch switching**)

What is needed:

1. Compute jacobian (`ForwardDiff.jl`,...), solve (large) linear systems (`IterativeSolvers.jl`, `Krylov.jl`, `KrylovKit.jl`,...)

Bifurcation theory amounts to

1. continue (all) equilibria (zeroes) of $F(x, p)$ as function of scalar parameter p

2. compute stability of the equilibria (given by number of eigenvalues $\lambda(p)$ with $\Re\lambda(p) > 0$)

3. detect a bifurcation point p_0 precisely, *i.e.* when $\Re\lambda(p_0) = 0$

    - Case $\lambda(p_0) = 0$: new equilibria (maybe)
    - Case $\lambda(p_0) = \pm i\omega$: nearby oscillations with frequency ω (maybe **Hopf** Bifurcation)

4. switch to new branch of solutions and go to 1. (**Branch switching**)

What is needed:

1. Compute jacobian (`ForwardDiff.jl`,...), solve (large) linear systems (`IterativeSolvers.jl`, `Krylov.jl`, `KrylovKit.jl`,...)
2. Compute eigenvalues of large systems `Arpack.jl`, `ArnoldiMethod.jl`, `KrylovKit.jl`,...

Bifurcation theory amounts to

1. continue (all) equilibria (zeroes) of $F(x,p)$ as function of scalar parameter p

2. compute stability of the equilibria (given by number of eigenvalues $\lambda(p)$ with $\Re\lambda(p) > 0$)

3. detect a bifurcation point p_0 precisely, *i.e.* when $\Re\lambda(p_0) = 0$

   - Case $\lambda(p_0) = 0$: new equilibria (maybe)
   - Case $\lambda(p_0) = \pm i\omega$: nearby oscillations with frequency ω (maybe **Hopf** Bifurcation)

4. switch to new branch of solutions and go to 1. (**Branch switching**)

What is needed:

1. Compute jacobian (`ForwardDiff.jl`,...), solve (large) linear systems (`IterativeSolvers.jl`, `Krylov.jl`, `KrylovKit.jl`,...)
2. Compute eigenvalues of large systems `Arpack.jl`, `ArnoldiMethod.jl`, `KrylovKit.jl`,...
3. Bisection / Newton
4. See next

2. Introduction to `BifurcationKit.jl`

What is it?

Library for numerical bifurcation analysis of large dimensional equations $F(x, p) = 0$

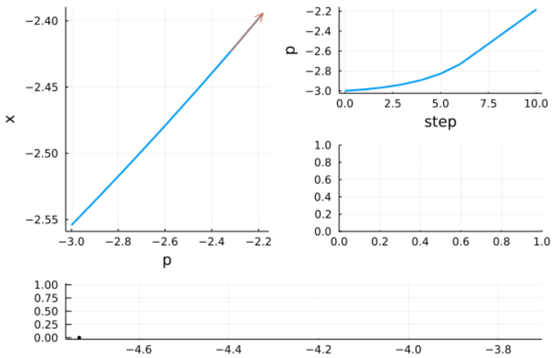
https://github.com/rveltz/BifurcationKit.jl (docs, tutorials, tests,...): see example later

Installation

```
] add BifurcationKit
```

Quick example

```julia
using BifurcationKit, Plots, Setfield

F = (x, par) -> @. par.p + x - x^3/3

opts = ContinuationPar(pMin = -3.,
    pMax = 1., detectBifurcation = 3)

params = (p = -3., q = 1.)

br, = continuation(F, [-2.], params,
    (@lens _.p), opts;
    recordFromSolution = (x,p)->x[1])

plot(br) #plot recipe
```



Why another library?

There are many good softwares already available.

- For continuation in **small dimension**, see DSWeb. One can mention the venerable AUTO-07p, or also, XPPAUT, MATCONT, PyDSTool, COCO and Bifurcations.jl.
- For **large scale** problems, there is the versatile pde2path but also COCO, Trilinos, CL_MATCONTL and the python libraries pyNCT and pacopy.
- For *deflated continuation*, there is defcont.

Why another library?

There are many good softwares already available.

- For continuation in **small dimension**, see DSWeb. One can mention the venerable AUTO-07p, or also, XPPAUT, MATCONT, PyDSTool, COCO and Bifurcations.jl.
- For **large scale** problems, there is the versatile pde2path but also COCO, Trilinos, CL_MATCONTL and the python libraries pyNCT and pacopy.
- For *deflated continuation*, there is defcont.

Reasons

Why another library?

There are many good softwares already available.

- For continuation in **small dimension**, see DSWeb. One can mention the venerable AUTO-07p, or also, XPPAUT, MATCONT, PyDSTool, COCO and Bifurcations.jl.

- For **large scale** problems, there is the versatile pde2path but also COCO, Trilinos, CL_MATCONTL and the python libraries pyNCT and pacopy.

- For *deflated continuation*, there is defcont.

Reasons

1. implement **new algorithms** for **large scale** problems

Why another library?

There are many good softwares already available.

- For continuation in **small dimension**, see DSWeb. One can mention the venerable AUTO-07p, or also, XPPAUT, MATCONT, PyDSTool, COCO and Bifurcations.jl.
- For **large scale** problems, there is the versatile pde2path but also COCO, Trilinos, CL_MATCONTL and the python libraries pyNCT and pacopy.
- For *deflated continuation*, there is defcont.

Reasons

1. implement **new algorithms** for **large scale** problems
2. take advantage of *unique* Julia ecosystem (DiffEq, AD, GPU, cluster...)

Why another library?

There are many good softwares already available.

- For continuation in **small dimension**, see DSWeb. One can mention the venerable AUTO-07p, or also, XPPAUT, MATCONT, PyDSTool, COCO and Bifurcations.jl.

- For **large scale** problems, there is the versatile pde2path but also COCO, Trilinos, CL_MATCONTL and the python libraries pyNCT and pacopy.

- For *deflated continuation*, there is defcont.

Reasons

1. implement **new algorithms** for **large scale** problems
2. take advantage of *unique* Julia ecosystem (DiffEq, AD, GPU, cluster...)
3. write generic code for CPU/GPU(/Cluster)

Why another library?

There are many good softwares already available.

- For continuation in **small dimension**, see DSWeb. One can mention the venerable AUTO-07p, or also, XPPAUT, MATCONT, PyDSTool, COCO and Bifurcations.jl.
- For **large scale** problems, there is the versatile pde2path but also COCO, Trilinos, CL_MATCONTL and the python libraries pyNCT and pacopy.
- For *deflated continuation*, there is defcont.

Reasons

1. implement **new algorithms** for **large scale** problems
2. take advantage of *unique* Julia ecosystem (DiffEq, AD, GPU, cluster...)
3. write generic code for CPU/GPU(/Cluster)
4. Specify the linear solver (dense, sparse, Matrix-Free), idem for eigensolver

Why another library?

There are many good softwares already available.

- For continuation in **small dimension**, see DSWeb. One can mention the venerable AUTO-07p, or also, XPPAUT, MATCONT, PyDSTool, COCO and Bifurcations.jl.
- For **large scale** problems, there is the versatile pde2path but also COCO, Trilinos, CL_MATCONTL and the python libraries pyNCT and pacopy.
- For *deflated continuation*, there is defcont.

Reasons

1. implement **new algorithms** for **large scale** problems
2. take advantage of *unique* Julia ecosystem (DiffEq, AD, GPU, cluster...)
3. write generic code for CPU/GPU(/Cluster)
4. Specify the linear solver (dense, sparse, Matrix-Free), idem for eigensolver
5. Should work on custom type (collocation, FEM, ...), not just `AbstractArray`

Why another library?

There are many good softwares already available.

- For continuation in **small dimension**, see DSWeb. One can mention the venerable AUTO-07p, or also, XPPAUT, MATCONT, PyDSTool, COCO and Bifurcations.jl.
- For **large scale** problems, there is the versatile pde2path but also COCO, Trilinos, CL_MATCONTL and the python libraries pyNCT and pacopy.
- For *deflated continuation*, there is defcont.

Reasons

1. implement **new algorithms** for **large scale** problems
2. take advantage of *unique* Julia ecosystem (DiffEq, AD, GPU, cluster...)
3. write generic code for CPU/GPU(/Cluster)
4. Specify the linear solver (dense, sparse, Matrix-Free), idem for eigensolver
5. Should work on custom type (collocation, FEM, ...), not just `AbstractArray`

$\Rightarrow$ develop fully automatic algorithms (for **memory limited** devices (GPU))

"Early" design decision

Use of `Setfield.jl` to specify the parameter axis.

Before

```
pars = (a=1., b=2., c=3.)
# continuation w.r.t. c
br = continuation( (x, p::Real) - > F(x, (pars..., c = p), x0, opts)
```

With `Setfield`:

```
# continuation w.r.t. c
br = continuation( F, x0, pars, (@lens _.c) , opts)
```

- Fundamental to deal with codim 2 continuation where you need to specify 2 parameter axis
- very useful for plot recipes

Other example

```
pars = [1., 2.]
(@lens _[1])
```

Can be used to modify immutable variables, structs...

Customizable: use of iterators

```julia
# functional we want to study
F = (x, par) -> @. par.p + x - x^3/3
params = (p = -3., q = 1.)

# parameters for the continuation
opts = ContinuationPar(pMin = -3., pMax = 1.)

# we define an iterator to hold the continuation routine
iter = ContIterable(F, [-2.], params, (@lens _.p), opts)

# variables to hold the results
resp = Float64[]; resx = Float64[]

# this is the PALC algorithm
for state in iter
    # we save the current solution on the branch
    push!(resx, getx(state)[1])
    push!(resp, getp(state))
end

# plot the result
plot(resp, resx; label = "", xlabel = "p")
```

Bonus: you can `copy(iter)` to perform additional steps (bisection) and come back to the parent continuation

Focus on stationary solutions

Two different numerical continuation algorithms

Goal find continuous curves of solutions $\gamma = (x(s), p(s))_{s\in I}$ to

$$F(x,p) = 0 \in \mathbb{R}^n, \quad x \in \mathbb{R}^n, p \in \mathbb{R} \tag{E}$$

using a Newton-Krylov solver from a **known solution** $X_0 := (x_0, p_0)$.

Two different numerical continuation algorithms

Goal find continuous curves of solutions $\gamma = (x(s), p(s))_{s\in I}$ to

$$F(x,p) = 0 \in \mathbb{R}^n, \quad x \in \mathbb{R}^n, p \in \mathbb{R} \tag{E}$$

using a Newton-Krylov solver from a **known solution** $X_0 := (x_0, p_0)$.

Deflated continuation

Find "all" solutions to (E)

Smart brute force by P. Farrell

Based on **deflated Newton**
$G_d(X) = \frac{G(X)}{\Pi_1^n(||X - X_i||^p + \alpha)}$

1. Find all solutions for $p = p_0$
2. Use them as guesses for $p = p_1 = p_0 + ds$ (Newton)
3. Find all solutions for $p = p_1$ (deflated Newton)

Two different numerical continuation algorithms

Goal find continuous curves of solutions $\gamma = (x(s), p(s))_{s\in I}$ to

$$F(x,p) = 0 \in \mathbb{R}^n, \quad x \in \mathbb{R}^n, p \in \mathbb{R} \tag{E}$$

using a Newton-Krylov solver from a **known solution** $X_0 := (x_0, p_0)$.

Deflated continuation

Find "all" solutions to (E)

Smart brute force by P. Farrell

Based on **deflated Newton** $G_d(X) = \frac{G(X)}{\Pi_1^n(||X - X_i||^p + \alpha)}$

1. Find all solutions for $p = p_0$
2. Use them as guesses for $p = p_1 = p_0 + ds$ (Newton)
3. Find all solutions for $p = p_1$ (deflated Newton)

Callable struct

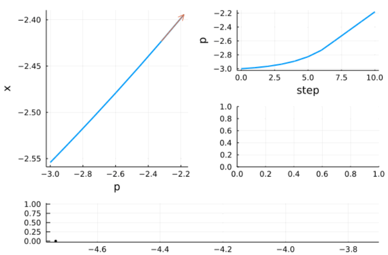
```
struct DeflationOperator{Tp <: Real, T
    "power"
    power::Tp
    "dot function"
    dot::Tdot
    "shift"
    α::T
    "roots"
    roots::Vector{vectype}
end
```

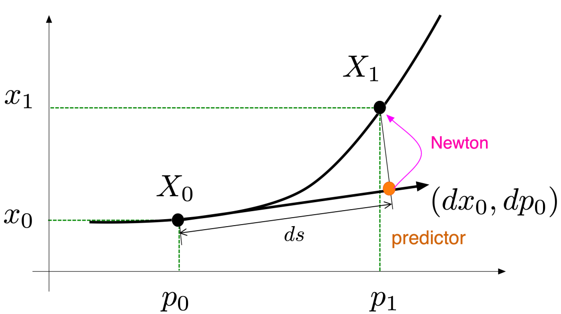
Two different numerical continuation algorithms

Goal find continuous curves of solutions $\gamma = (x(s), p(s))_{s\in I}$ to

$$F(x, p) = 0 \in \mathbb{R}^n, \quad x \in \mathbb{R}^n, p \in \mathbb{R} \tag{E}$$

using a Newton-Krylov solver from a **known solution** $X_0 := (x_0, p_0)$.





Pseudo arc-length continuation

Connected component of (x_0, p_0)

Solves (E) with constraint

$$N(x, p) = \frac{\theta}{length(x)}\langle x - x_0, dx_0\rangle$$
$$+(1-\theta)\cdot(p - p_0)\cdot dp_0 - ds = 0$$

1. tangent: (dx_0, dp_0) at X_0
2. predictor: $(x_1, p_1) =$
   $(x_0, p_0) + ds \cdot (dx_0, dp_0)$
3. correction: Newton to $[F, N]$

Two different numerical continuation algorithms

Goal find continuous curves of solutions $\gamma = (x(s), p(s))_{s\in I}$ to

$$F(x, p) = 0 \in \mathbb{R}^n, \quad x \in \mathbb{R}^n, p \in \mathbb{R} \tag{E}$$

using a Newton-Krylov solver from a **known solution** $X_0 := (x_0, p_0)$.

Bordered linear system (BLS)

$$\begin{bmatrix} d_xF & d_pF \\ d_xN & d_pN \end{bmatrix}$$

Inefficient linear solver example:

```
struct MatrixBLS <: AbstractBLS; end

function (lbs::MatrixBLS)(dxF, dpF,
                          dxN, dpN,
                          R, n)
    A = hcat(dxF, dpF)
    A = vcat(A, vcat(dxN, dpN))
    return A \ vcat(R, n), true, 1
end
```

Better ones available 😉

Pseudo arc-length continuation

Connected component of (x_0, p_0)

Solves (E) with constraint

$$N(x, p) = \frac{\theta}{length(x)}\langle x - x_0, dx_0\rangle$$
$$+(1-\theta)\cdot(p - p_0)\cdot dp_0 - ds = 0$$

1. tangent: (dx_0, dp_0) at X_0
2. predictor: $(x_1, p_1) =$ $(x_0, p_0) + ds \cdot (dx_0, dp_0)$
3. correction: Newton to $[F, N]$

Two different numerical continuation algorithms

Goal find continuous curves of solutions $\gamma = (x(s), p(s))_{s\in I}$ to

$$F(x,p) = 0 \in \mathbb{R}^n, \quad x \in \mathbb{R}^n, p \in \mathbb{R} \tag{E}$$

using a Newton-Krylov solver from a **known solution** $X_0 := (x_0, p_0)$.

Deflated continuation

Find "all" solutions to (E)

Smart brute force by P. Farrell

- memory intensive if many solutions
- difficult to parallelize
- loss of time on diverged Newton
- automatic branching

Pseudo arc-length continuation

Connected component of (x_0, p_0)

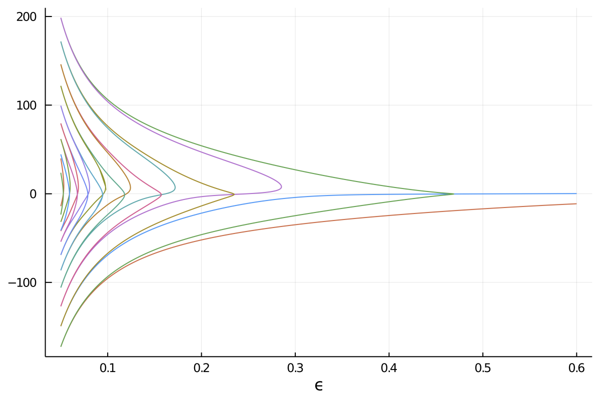
- "fast"
- small memory needed
- branching requires dedicated aglorithms

Two different numerical continuation algorithms: comparison

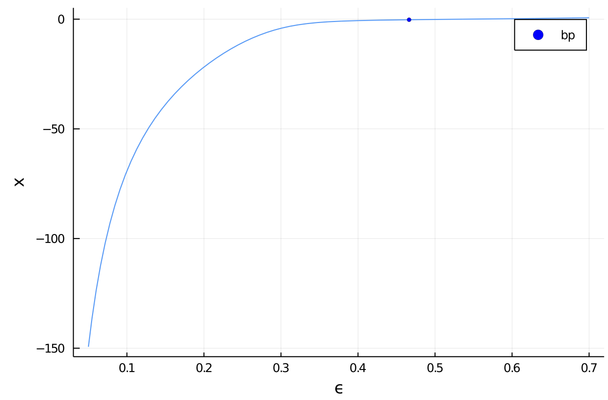
Let us consider the singular perturbation problem (see tutorials):

$$\epsilon^2 y'' + 2\left(1 - x^2\right) y + y^2 = 1, \quad y(-1) = y(1) = 0.$$

Deflated continuation



Pseudo arc-length continuation PALC



We now focus on PALC

Towards automatic bifurcation diagram

Automatic Branch Switching (aBS) at stationary bifurcation

New algorithm twist.

New solutions (may) emerge at **bifurcation points** *i.e.* when

$$d \equiv \dim \ker dF(x_0, p_0) > 0.$$

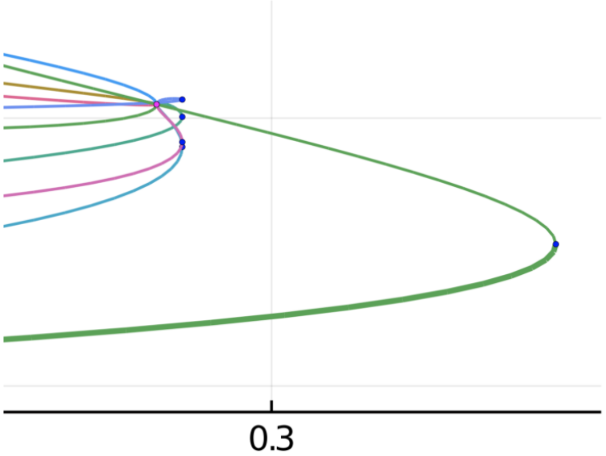
Lyapunov-Schmidt method gives equivalent **reduced equation** $\Phi(x_{Ker}, p) = 0 \in \mathbb{R}^d$

1. computes 3-jet of Φ
2. solves resulting polynomials equations
3. use solutions as guesses for Deflated Newton

See *[Wouters et al. :2019]*

$\Rightarrow$ Iterate to get automatic bifurcation diagram

```
jet = getJet(F)
br, = continuation(jet[1], jet[2], x0,
br1, = continuation(jet..., br, 1)
```



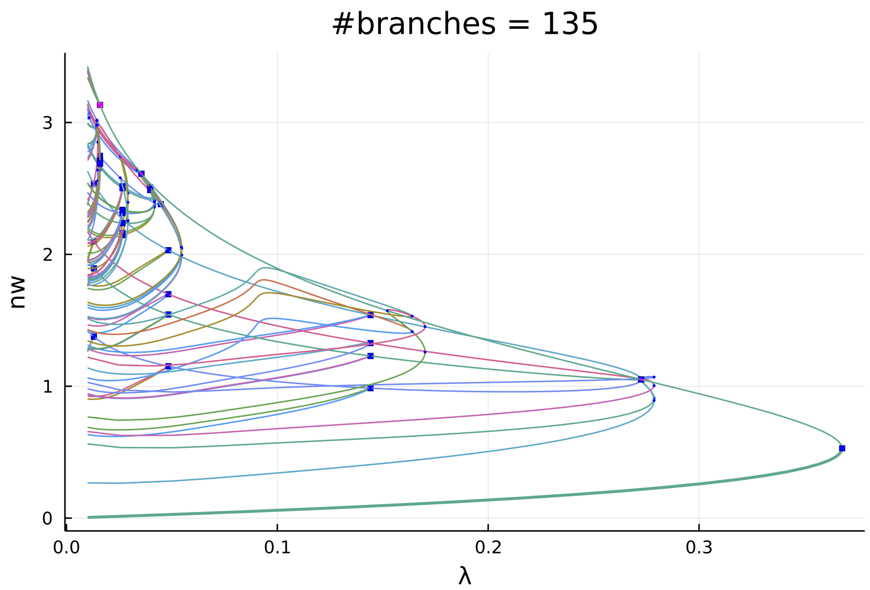
Automatic Bifurcation Diagram (aBD)

2d Bratu–Gelfand problem (see tutorials),

$$\Delta u = 10(u - \lambda e^u), \quad \Omega = (0,1)^2, \quad \partial_n u = 0 \ \partial\Omega$$

Sparse formulation (`DiffEqOperators.jl`), Eigen Solver from `KrylovKit.jl`

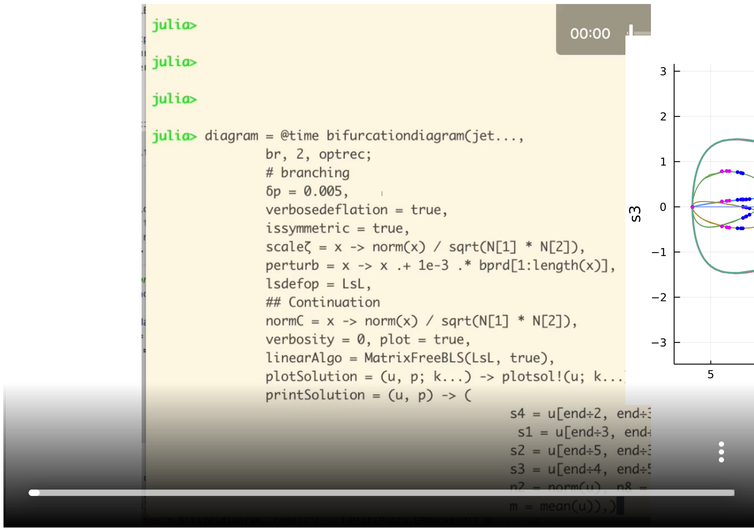
```
diagram = bifurcationdiagram(jet..., sol0, par_mit, (@lens _.λ), 5, opts)
```

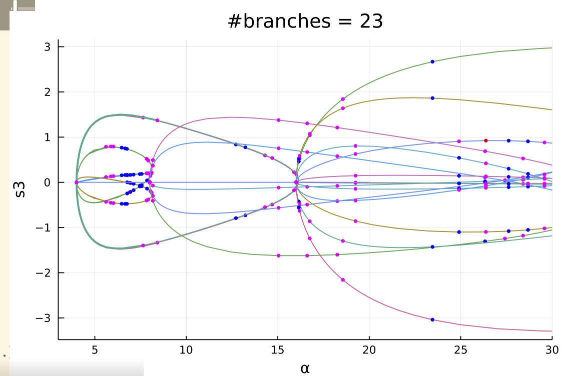


Automatic Bifurcation Diagram (aBD) on GPU

2d Kuramoto-Sivashinsky. Based on FFT (`CUDA.jl`), Eigen Solver from `KrylovKit.jl`

$$\frac{1}{\alpha}\left(2uu_x + 2uu_y + \Delta u\right) + 4\Delta^2 u = 0$$





Semi Automatic Bifurcation diagram entirely on GPU (1/2)

We solve the Neural Fields Equations (model of visual hallucinations)

$$\frac{d}{dt}V(x,t) = -V(x,t) + \int_\Omega W(x,y)S(\gamma V(y,t))dy,\ \ x,y \in \mathbb{R}^3, \quad W(x,y) \in \mathbb{R}$$

Constraints:

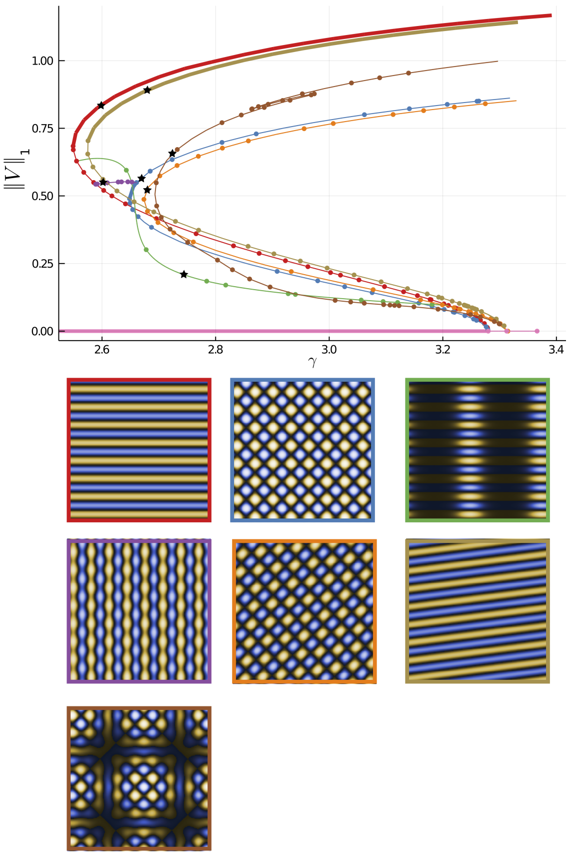
- need to be done on GPU
- lots of symmetries, difficult aBS
- you cannot keep in memory all Eigen elements

Details:

- runs **entirely** on GPU (V100 Tesla), 3d FFT! using `CUDA.jl`
- Linear solver (GMRES), Eigen solver `KrylovKit.jl`
- Bifurcation points located with **bisection**
- Reduced equation computed on the fly, **aBS** on GPU
- ~1e7 unknowns

⇒ One of the **few** bifurcation diagrams computed **entirely** on GPU

Semi Automatic Bifurcation diagram entirely on GPU (2/2)



Weak points...

- improve the tree structure which holds the bifurcation diagram (type stability)
- remove loops during computations
- write GUI (`Makie`) to improve navigation

Focus on periodic orbits

Computing periodic orbits (PO) 1/3

Trapeze method

We look for periodic orbits as solutions $(x(0), T)$ of

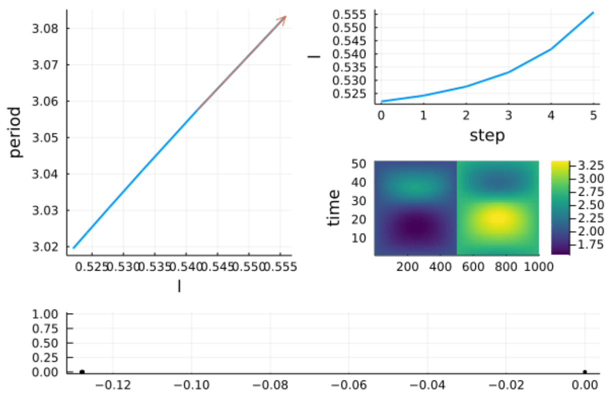
$$\frac{dx}{dt} = T \cdot F(x),\ \ x(0) = x(1).$$

By discretizing, we obtain ($h = T/m$)

$$\begin{aligned}
0 &= (x_j - x_{j-1}) - \tfrac{h}{2}\left(F(x_j) + F(x_{j-1})\right) \\
0 &= x_m - x_1 \\
0 &= \textstyle\sum_i \langle x_i - x_{\pi,i}, \phi_i \rangle = 0
\end{aligned}$$

- optimized code, reduced allocations
- 7 different linear solvers (dense, iterative, AD...)
- Floquet multipliers computation (not super precise for now)
- run on GPU
- Hopf, BPLC **aBS**

For the Brusselator 1d



Computing periodic orbits (PO) 2/3

(Multiple parallel) Standard Shooting method

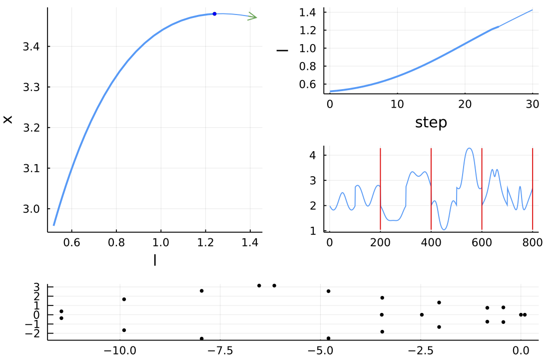
We aim at finding periodic orbits of $\frac{dx}{dt} = f(x)$ with flow $\phi^t(x_0)$ by solving

$$\phi^T(x) - x = 0, \quad s(x, T) = 0.$$

- 4 different linear solvers
- Floquet multipliers computation (not super precise for now)
- user defined flow
- wrapper to `DifferentialEquations.jl`
- Hopf **aBS**

Thanks to `DifferentialEquations.jl`, we are >= state of the art

For the 1d Brusselator



Computing periodic orbits (PO) 3/3

(Multiple) Poincaré Shooting method

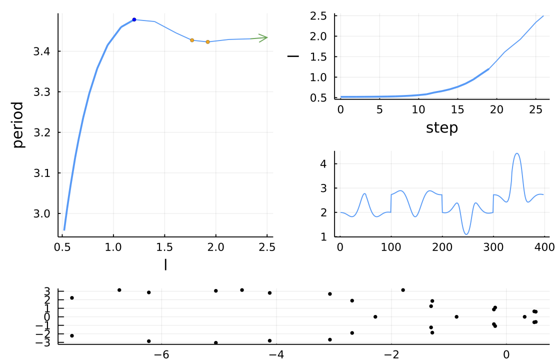
We aim at finding periodic orbits of $\frac{dx}{dt} = f(x)$ by solving a Poincaré return map equation

$$\Pi(x) - x = 0.$$

- needs to find a section but N-1 unknowns
- 4 different linear solvers
- Floquet multipliers computation (not super precise for now)
- wrapper to `DifferentialEquations.jl`
- Hopf **aBS**

Thanks to `DifferentialEquations.jl`, we are >= state of the art

For the 1d Brusselator



Let's talk about the docs

Automation

Documentation inside code

Documentation located https://github.com/bifurcationkit/BifurcationKitDocs.jl

- used to be part of `BifurcationKit.jl`
- based on `DocStringExtensions.jl` to generate DocStrings

```
"""
$(TYPEDEF)

Returns a variable containing parameters to affect
the `newton` algorithm when solving `F(x) = 0`.

# Arguments (with default values):
$(TYPEDFIELDS)

"""
@with_kw struct NewtonPar{T, L <: AbstractLinearSolver,
                             E <: AbstractEigenSolver}
    "absolute tolerance for `F(x)`"
    tol::T            = 1e-12
end
```

Documentation (online)

- documentation website based on `Documenter.jl`, hosted on github
- DocStrings are grouped in Library.html

```
using Documenter, BifurcationKit, Setfield

makedocs(doctest = false,
    sitename = "Bifurcation Analysis in Julia",
    format = Documenter.HTML(collapselevel = 1,assets = ["assets/indigo.css"]),
    authors = "Romain Veltz",
    pages = Any[
        "Home" => "index.md",
        "Tutorials" => "tutorials/tutorials.md",
        "Functionalities" => [
            "Plotting" => "plotting.md",
            ],
        "Library" => "library.md"])
deploydocs(
    repo = "github.com/bifurcationkit/BifurcationKitDocs.jl.git",
    devbranch = "main"
)
```

- tutorials and codes are (mostly) automatically generated
    - less figures to export
    - less error prone
    - proof that it works to the user
    - ...

The future of `BifurcationKit.jl`

More functionalities / Improvements

- computation of **periodic orbits** based on othogonal collocation with adpative mesh (*under test*)
- computation of **travelling waves** and their bifurcations (*under test*)
- computation of homoclinic trajectories and homoclinic bifucations (end of codim 2)
- improvements to **Deflated Continuation**
- deflated continuation applied to periodic orbits
- improve computation of Floquet coefficients `PeriodicSchurDecompositions.jl`
- add new continuation algorithms Moore-Penrose, ANM, ...

Code

- better use of `StaticArrays.jl`
- `Makie.jl` recipes

Software design 1/2

Hard nuts (mostly done)

- Change interface to `br = continuation(prob, PALC(), options)`
- problem and algo saved in `br`, simpler dispatch, modify `prob` and `alg` with `Setfield.jl`
- remove duplicated code
- less demanding on the user, more automation
- similar to `solve(odeprob, Tsit5())`.
- Allow to change the continuation algorithm very easily `br = continuation(prob, MoorePenrose(), options)`

Benefits

- easier interface to other problems
- easier interface to SciML
- `ODEProblem`? `DDEProblem`?

Software design 1/2

Hard nuts (mostly done)

Now

```
jet = getJet(Fb)
br, = continuation(jet[1]. jet[2], sol0, (1., 1.), (@lens _[1], opts;
    tangentAlgo = BorderedPred())
nf = computeNormalForm(jet..., br, 1)
br2 = continuation(jet..., br, 1)
```

Tomorrow

```
prob = BK.BifurcationProblem(Fb, sol0, (1., 1.), (@lens _[1])
br = continuation(prob, PALC(tangent = Bordered()), opts_br0)
nf = computeNormalForm(br, 1)
br2 = continuation(br, 1)
```

Software design 2/2

Tough nuts

1. custum types v.s. `AbstractArray` (AD?)
2. inplace / outplace methods
3. link between 1. and 2.
4. define interface

Goodies: tricks used in BK

Applying AD generated differentials to complex arguments

```
struct BilinearMap{Tm}
    bl::Tm
end

function (R2::BilinearMap)(dx1, dx2)
    dx1r = real.(dx1); dx2r = real.(dx2)
    dx1i = imag.(dx1); dx2i = imag.(dx2)
    return   R2(dx1r, dx2r) .- R2(dx1i, dx2i) .+
      im .* (R2(dx1r, dx2i) .+ R2(dx1i, dx2r))
end
(b::BilinearMap)(dx1::T, dx2::T) where {T <: AbstractArray{<: Real}} =
             b.bl(dx1, dx2)
```

Goodies: tricks used in BK

Dispatch on few fields

```
foo(br::ContResult{Tkind, Tbr, Teigvals, Teigvec, Biftype} ) where
                  {Tkind, Tbr, Teigvals, Teigvec, Biftype} = Tkind != Nothing
```

vs

```
foo(br::ContResult{Tkind} ) where {Tkind} = Tkind != Nothing
```

(one can use Abstract types too.)

Conclusion

- **highly tunable** library for ODE, PDE and other working on CPU/GPU
- many unique features (aBD, GPU, Shooting & Trapeze, codim 2) all in large dimensions
- "easy" interface with `ApproxFun.jl, Gridap.jl, FourierFlows.jl`...

Ideas

- interval arithmetics
- a GUI based on `Makie` to do all this interactively
- distributed computing

Thank you for your attention