Recent Advancements in Differential Equation Solver Software

CHRIS RACKAUCKAS

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

DifferentialEquations.jl: Research Platform for Production

- ~300 methods available, including wrappers to C/Fortran methods
  - **Platform for reproducible research and benchmarking**
- MPI+GPU Compatibility
- Implicit, IMEX, multirate, symplectic, exponential integrators, etc.
- **Adaptive high order methods for stochastic differential equations**
- Stiff state-dependent delay differential equation discontinuity tracking
- Mix in Gillespie simulation (Continuous-Time Markov Chains)
- Automatic sparsity detection and optimization
- Arbitrary code injection through callbacks

Native Julia methods routinely benchmark as one of the fastest libraries in most categories (caveat category: large (>1000) ODE/DAE systems)

3 Major Areas

- Neural Differential Equations
- A Hackable Model Compiler (ModelingToolkit.jl)
- Improvements to basic numerical methods

There is a new field that is merging AI and domain-specific modeling: Scientific ML/AI

SCIENTIFIC AI: DOMAIN MODELS WITH INTEGRATED MACHINE LEARNING

HTTPS://WWW.YOUTUBE.COM/WATCH?V=FGFX8CQHDQA

THE ESSENTIAL TOOLS OF SCIENTIFIC MACHINE LEARNING

HTTP://WWW.STOCHASTICLIFESTYLE.COM/THE-ESSENTIAL-TOOLS-OF-SCIENTIFIC-MACHINE-LEARNING-SCIENTIFIC-ML/

What is the mathematical structure of machine learning?

Neural Networks = Nonlinear Regression

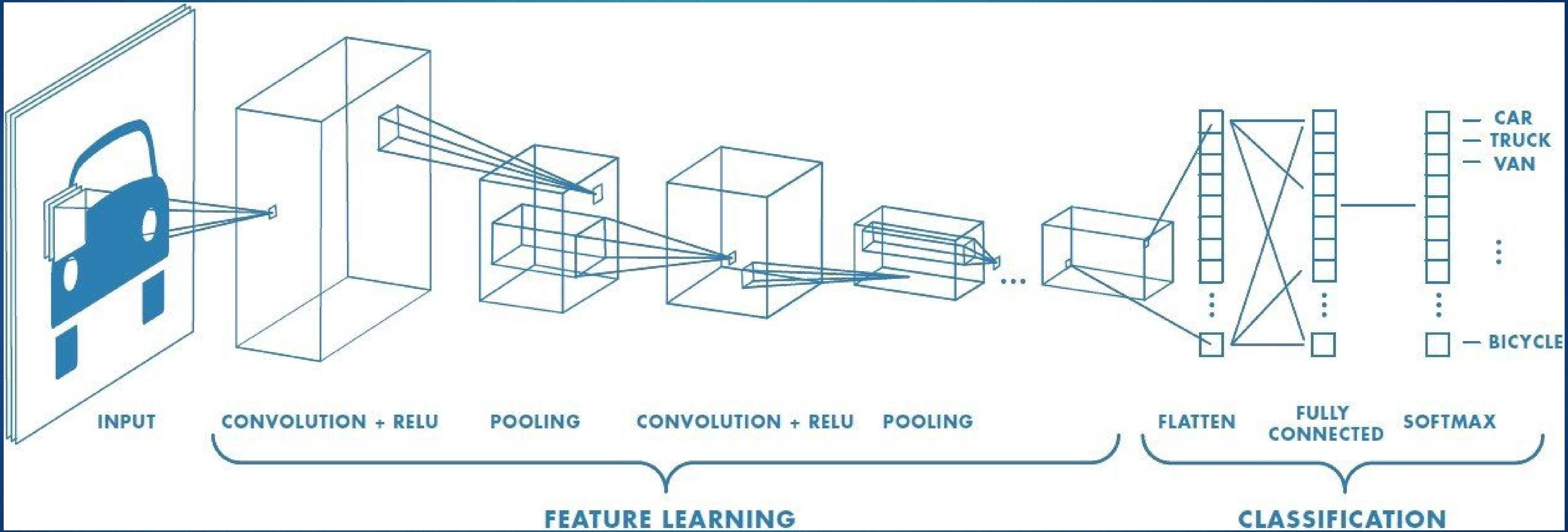
- Polynomial: $e^x = a_1 + a_2 x + a_3 x^2 + \cdots$
- Nonlinear: $e^x = 1 + \frac{a_1 \tanh(a_2 x)}{a_3 x - \tanh(a_4 x)}$
- Neural Network: $e^x \approx W_3 \sigma(W_2 \sigma(W_1 x + b_1) + b_2) + b_3$. Train the weights (W, b)

Universal Approximation Theorem

NEURAL NETWORKS CAN GET ϵ CLOSE TO ANY $R^n \to R^m$ FUNCTION

Neural Networks Overcome “the curse of dimensionality”

Not Quite a Black Box: Convolutional Neural Networks Encode (Spatial) Structure



A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, Sumit Saha

Now let’s generalize this idea to scientific structures

Latent (Neural) Differential Equations

NEURAL ORDINARY DIFFERENTIAL EQUATION:

$$u' = f(u, p, t)$$

LET f BE A NEURAL NETWORK

Training a neural differential equation: DiffEqFlux.jl

- Solve the differential equation
- Compute the gradient of the solution with respect to the parameters defining the neural network
  - Adjoint sensitivity analysis
  - Differentiable programming
- Update the neural network and repeat

Automatically Learning the Model

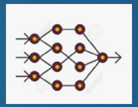
```julia
using DiffEqFlux, OrdinaryDiffEq, Flux, Plots

# Generate data from a real ODE
u0 = Float32[2.; 0.]; datasize = 30
tspan = (0.0f0,1.5f0)
function trueODEfunc(du,u,p,t)
    true_A = [-0.1 2.0; -2.0 -0.1]
    du .= ((u.^3)'true_A)'
end
t = range(tspan[1],tspan[2],length=datasize)
prob = ODEProblem(trueODEfunc,u0,tspan)
ode_data = Array(solve(prob,Tsit5(),saveat=t))

# Define a Neural ODE
dudt = Chain(x -> x.^3,
             Dense(2,75,tanh),
             Dense(75,2))
n_ode(x) = neural_ode(dudt,x,tspan,AutoTsit5(Rosenbrock23(autodiff=false)),saveat=t,reltol=1e-7,abstol=1e-9)
function predict_n_ode()
  n_ode(u0)
end
loss_n_ode() = sum(abs2,ode_data .- predict_n_ode())

# Train the Neural ODE to match the data
data = Iterators.repeated((), 200)
cb = function () #callback function to observe training
  display(loss_n_ode()); cur_pred = Flux.data(predict_n_ode())
  p1 = scatter(t,ode_data[1,:],label="data",legend=:bottomright); scatter!(p1,t,cur_pred[1,:],label="prediction")
  p2 = scatter(t,ode_data[2,:],label="data",legend=:top); scatter!(p2,t,cur_pred[2,:],label="prediction")
  display(plot(p1,p2,layout=(2,1)))
end
Flux.train!(loss_n_ode, Flux.params(dudt), data, Nesterov(0.0005), cb = cb)
```

```
julia>
```

The real power comes from incorporating known structure into the ML framework (Mixed Neural Differential Equation)

Mix Neural Networks Into DiffEqs!

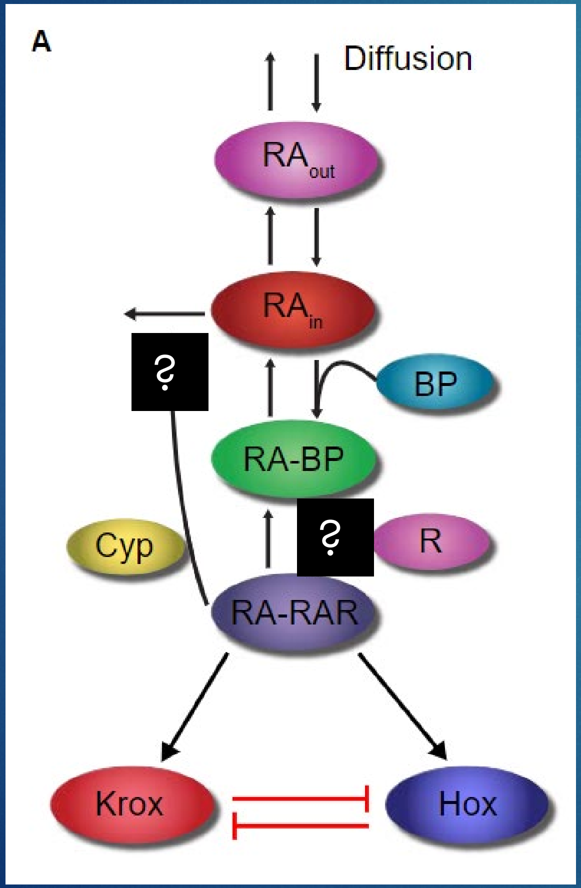
```julia
using DiffEqFlux, Flux, OrdinaryDiffEq

u0 = param(Float32[0.8; 0.8])
tspan = (0.0f0,25.0f0)
ann = Chain(Dense(2,10,tanh), Dense(10,1))

p1 = Flux.data(DiffEqFlux.destructure(ann))
p2 = Float32[-2.0,1.1]
p3 = param([p1;p2])
ps = Flux.params(p3,u0)

function dudt_(du,u,p,t)
    x, y = u
    du[1] = DiffEqFlux.restructure(ann,p[1:41])(u)[1]
    du[2] = p[end-1]*y + p[end]*x
end
prob = ODEProblem(dudt_,u0,tspan,p3)
function predict_adjoint()
  diffeq_adjoint(p3,prob,Tsit5(),u0=u0,saveat=0.0:0.1:25.0)
end
loss_adjoint() = sum(abs2,x-1 for x in predict_adjoint())
Flux.train!(loss_adjoint, ps, Iterators.repeated((), 10), ADAM(0.1))
```

$$\frac{dx}{dt} =$$ 

$$\frac{dy}{dt} = p_1 y + p_2 x$$

Fit the "mixed neural differential equation" using the same method!

ML-Assisted Model Discovery



The chemical reactions imply an evolution of:

$$d[RA_{out}] = (\beta - b[RA_{out}] + c[RA_{in}])dt,$$

$$d[RA_{in}] = \left(b[RA_{out}] + \delta[RA - BP] - \left(\gamma[BP] + \eta + \; ? \qquad - c\right)[RA_{in}]\right)dt + \sigma dW_t,$$

$$d[\mathrm{RA} - \mathrm{BP}] = (\gamma[BP][RA_{in}] + \lambda[BP][RA - RAR] - \; ? \qquad dt,$$

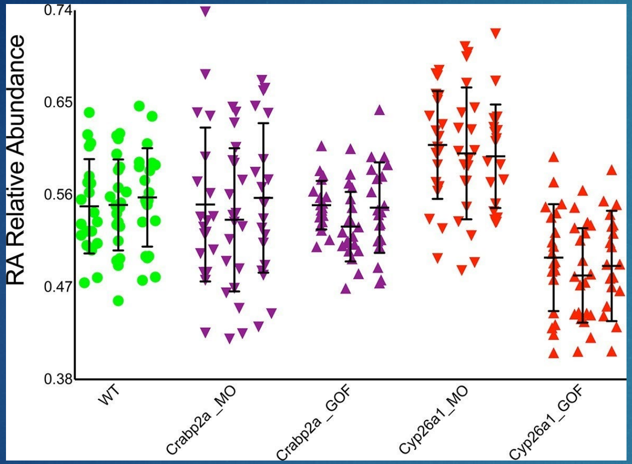
$$d[RA - RAR] = \; ? \qquad - \lambda[BP][RA - RAR])dt,$$

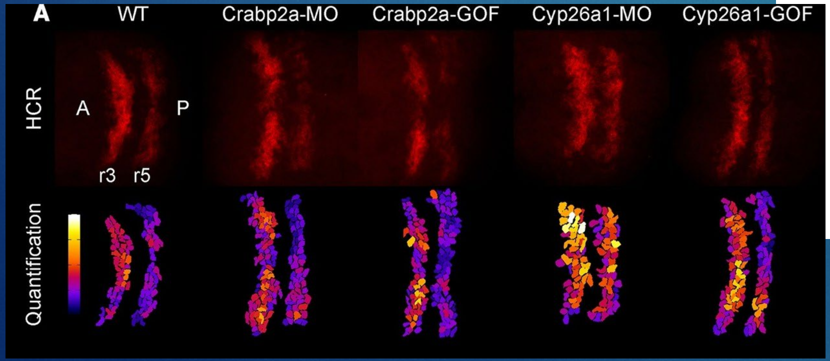
$$d[RAR] = \; ? \qquad dt,$$

$$d[BP] = (a - \lambda[BP][RA - RAR] - \gamma[BP][RA_{in}] + (\delta + \nu[RAR])[RA - BP] - u[BP])dt,$$

Biologically-Informed Neural Network

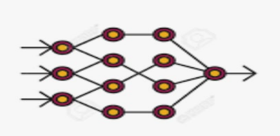
Data

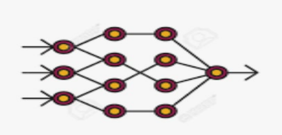


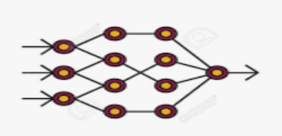


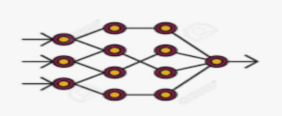
Find neural networks so the model matches the data

$$d[RA_{out}] = (\beta - b[RA_{out}] + c[RA_{in}])dt,$$

$d[RA_{in}] = \Big(b[RA_{out}] + \delta[RA-BP] - \big(\gamma[BP] + \eta +$  $- c\big)[RA_{in}]\Big)dt + \sigma dW_t,$

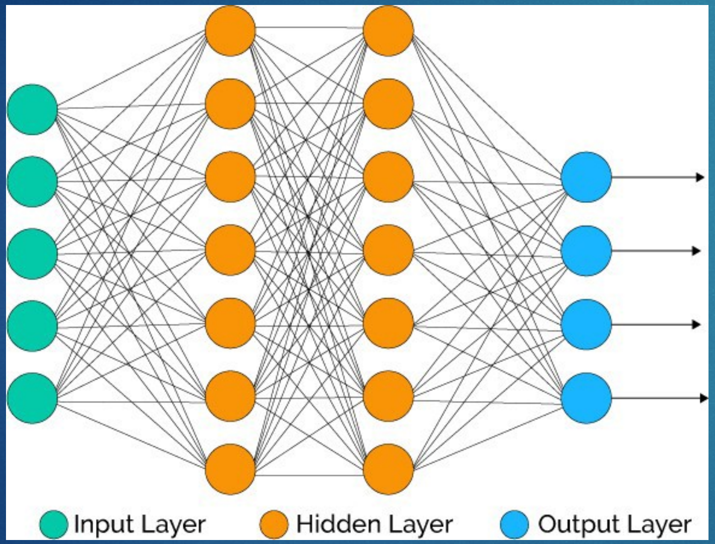
$d[\mathrm{RA-BP}] = (\gamma[BP][RA_{in}] + \lambda[BP][RA-RAR] -$  $dt,$

$d[RA-RAR] =$  $-\lambda[BP][RA-RAR])dt,$

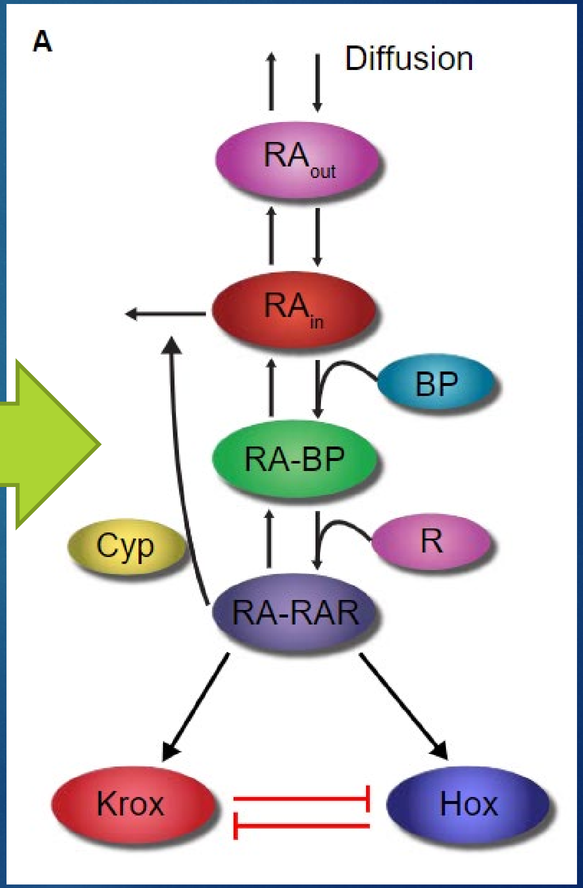
$d[RAR] =$  $dt,$

$$d[BP] = (a - \lambda[BP][RA-RAR] - \gamma[BP][RA_{in}] + (\delta + \nu[RAR])[RA-BP] - u[BP])dt,$$

Interpretability of Neural Differential Equations



Analyze the Jacobian/Hessian

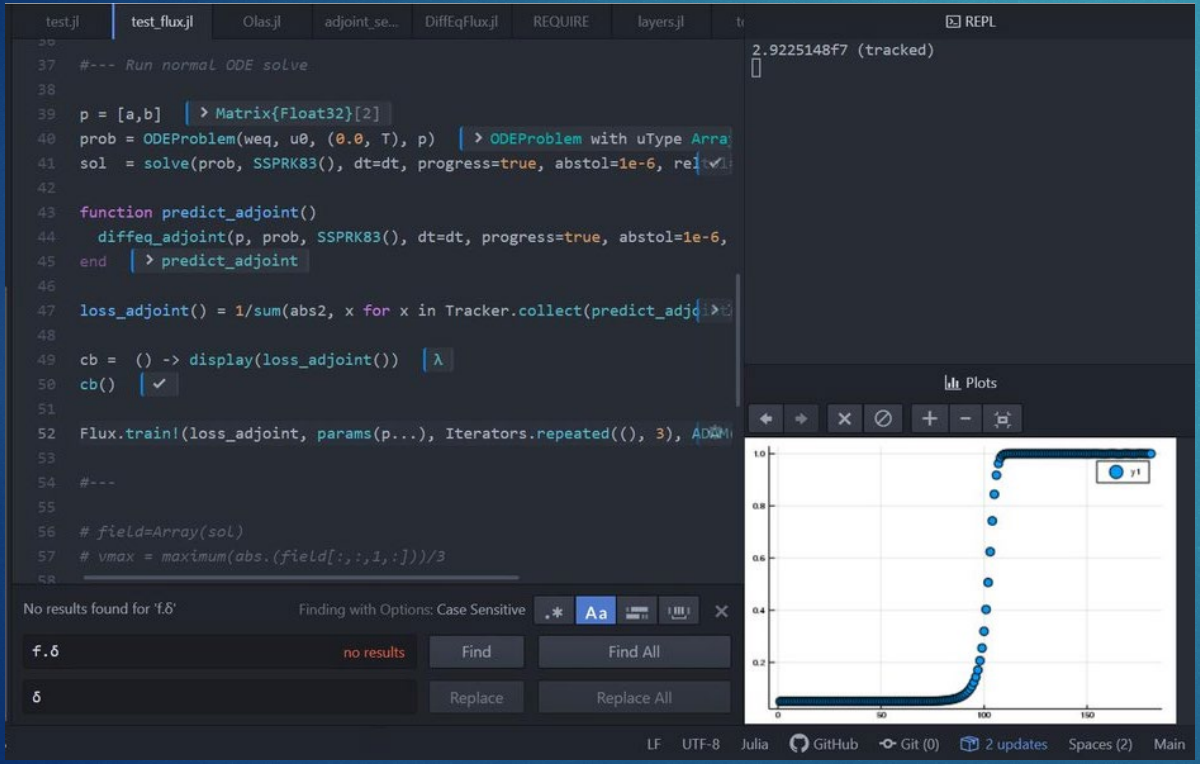


Data-Efficient Physics-Embedded Machine Learning

Nonlinear Optimal Control as a Mixed Neural ODE

- $x' = f(x(t), u(t), t)$
- Minimize $J = \Phi\left(x(t_0), t_0, x\left(t_f\right), t_f\right) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt$
- Example: $x(t)$ is the location of an automated drone, $u(t)$ is the controller, find what the controller should be such that the vehicle goes to the right place for the least energy.
- Neural ODE Approach: Make $u(t)$ be a neural network. Find the neural network s.t. $x(t)$ correctly evolves

Neural PDEs for Acceleration of Navier-Stokes



- Boussinesq Equations (Navier-Stokes) are used in climate models

$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \mathrm{Pr}\nabla^2 \mathbf{u} + b\hat{z}$$

$$\frac{\partial b}{\partial t} + \mathbf{u} \cdot \nabla b = \nabla^2 b + Fe^z$$

- People attempt to solve this by "parameterizing", i.e. getting a 1-dimensional approximation through averaging:

$$\left(\frac{\partial}{\partial t} + \mathbf{u}.\nabla - \nabla^2\right) c' - \frac{\partial}{\partial z}\overline{w'c'} = -w'\frac{\partial \overline{c}}{\partial z}$$

where $\overline{w'c'}$ is unknown.

- Instead of picking a form for $\overline{w'c'}$(the current method), replace it with a neural network and learn it from small scale simulations! Discretize. Result: Neural ODE.

Solving 1000 dimensional PDEs: Hamilton-Jacobi-Bellman, Nonlinear Black-Scholes

- Semilinear Parabolic Form (Diffusion-Advection Equations, Hamilton-Jacobi-Bellman, Black-Scholes)

$$\frac{\partial u}{\partial t}(t,x)+\frac{1}{2}\mathrm{Tr}\left(\sigma\sigma^{\mathrm{T}}(t,x)(\mathrm{Hess}_x u)(t,x)\right)+\nabla u(t,x)\cdot\mu(t,x)+f\left(t,x,u(t,x),\sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\right)=0 \qquad [1]$$

Then the solution of Eq. **1** satisfies the following BSDE (cf., e.g., refs. 8 and 9):

$$\begin{aligned}&u(t,X_t)-u(0,X_0)\\&=-\int_0^t f\left(s,X_s,u(s,X_s),\sigma^{\mathrm{T}}(s,X_s)\,\nabla u(s,X_s)\right)ds\\&+\int_0^t[\nabla u(s,X_s)]^{\mathrm{T}}\,\sigma(s,X_s)\,dW_s.\end{aligned} \qquad [3]$$

- Make $(\sigma^T \nabla \mathrm{u})(t,X)$ a neural network.
- Solve the resulting SDEs and learn $\sigma^T \nabla \mathrm{u}$ via:

$$l(\theta)=\mathbb{E}\left[\left|g(X_{t_N})-\hat{u}\left(\{X_{t_n}\}_{0\le n\le N},\{W_{t_n}\}_{0\le n\le N}\right)\right|^2\right].$$

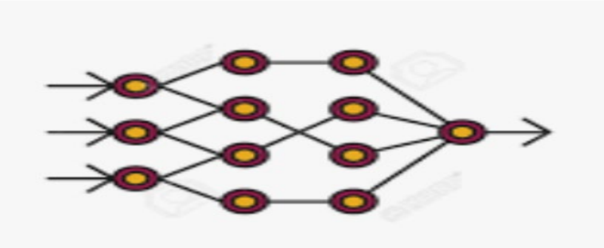
Simplified:

- Transform it into a Backwards SDE.
- The unknown is a function!
- Learn the unknown function via neural network.
- Once learned, the PDE solution is known.

Solving high-dimensional partial differential equations using deep learning, 2018, PNAS, Han, Jentzen, E

Represent 1000 dimensional PDEs as a 1000 dimensional neural SDE

$$
\begin{aligned}
dX_t &= \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \\
dU_t &= f(t, X_t, U_t, \sigma^T(t, X_t)\nabla u(t, X_t))dt \\
&\quad +
\end{aligned}
$$



$$dW_t,$$

- Solving the PDE = training the neural network
- As a neural SDE, we can solve with higher order (less neural network evaluations), adaptivity, etc.

Submitting to AISTATS 2020

DiffEqFlux.jl

Unified Framework for Scientific ML

The first (mixed) Neural Differential Equation solver. Supports:

- Neural ODEs
- Neural SDEs (SDDEs)
- Neural DAEs
- Neural DDEs
- Stiff Equations
- Hybrid Equations
- Adjoints via reverse-mode AD and adjoint sensitivity analysis
- **Data-Efficient and Physical Machine Learning**

ModelingToolkit: An Open Source Compiler + IR for Models and Transformations

Defining Principles

- A structured and documented IR for describing models
- "Model transforms" are compiler passes IR -> IR
- Extendable high level "context" system
- **Let others write domain-specific languages on top**
  - DSLs should not have to define simplification, differentiation, etc!

```julia
using ModelingToolkit

# Define some variables
@parameters t σ ρ β
@variables x(t) y(t) z(t)
@derivatives D'~t

eqs = [D(x) ~ σ*(y-x),
       D(y) ~ x*(ρ-z)-y,
       D(z) ~ x*y - β*z]

de = ODESystem(eqs)
generate_function(de, [x,y,z], [σ,ρ,β])

## Which returns:
:((##363, u, p, t)->begin
        let (x, y, z, σ, ρ, β) = (u[1], u[2], u[3], p[1], p[2], p[3])
            ##363[1] = σ * (y - x)
            ##363[2] = x * (ρ - z) - y
            ##363[3] = x * y - β * z
        end
    end)
```

Automatically convert numerical functions to symbolic

```
function lorenz(du,u,p,t)
 du[1] = p[1]*(u[2]-u[1])
 du[2] = u[1]*(p[2]-u[3]) - u[2]
 du[3] = u[1]*u[2] - p[3]*u[3]
end    > lorenz

using ModelingToolkit    ✓
@parameters t σ ρ β    (t(), σ(), ρ(), β())
@variables x(t) y(t) z(t)    (x(t()), y(t()), z(t()))
u = [x,y,z]    > Operation[3]
du = similar(u)    > Operation[3]
p = [σ,ρ,β]    > Operation[3]
lorenz(du,u,p,t)    x(t()) * y(t()) - β() * z(t())
du    v Operation[3]
        σ() * (y(t()) - x(t()))
        x(t()) * (ρ() - z(t())) - y(t())
        x(t()) * y(t()) - β() * z(t())
@derivatives Dx'~x Dy'~y Dz'~z    (> Differential, > Differential, > Differential)
J = [Dx(du[1]) Dy(du[1]) Dz(du[1])
     Dx(du[2]) Dy(du[2]) Dz(du[2])
     Dx(du[3]) Dy(du[3]) Dz(du[3])]    > 3×3 Array{Operation,2}:
J = expand_derivatives.(J)    v 3×3 Array{Expression,2}:
                                       σ() * -1            σ()   Constant(0)
                                  ρ() - z(t())   Constant(-1)   x(t()) * -1
                                       y(t())         x(t())      -1 * β()
```

Current Research


```julia
using ModelingToolkit, DiffEqOperators, DiffEqBase, LinearAlgebra

# Define some variables
@parameters t x
@variables u(..)
@derivatives Dt'~t
@derivatives Dxx''~x
eq  = Dt(u(t,x)) ~ Dxx(u(t,x))
bcs = [u(0,x) ~ - x * (x-1) * sin(x),
           u(t,0) ~ 0, u(t,1) ~ 0]

domains = [t ∈ IntervalDomain(0.0,1.0),
           x ∈ IntervalDomain(0.0,1.0)]

pdesys = PDESystem(eq,bcs,domains,[t,x],[u])
discretization = MOLFiniteDifference(0.1)
prob = discretize(pdesys,discretization)

using OrdinaryDiffEq
sol = solve(prob,Tsit5(),saveat=0.1)
```

- Components and Pantelides for large DAE systems
- An extendable framework for automated PDE discretizations
- Methods for (nonlinear) model order reduction
- Transformations for SDEs
- Tearing and other structural optimizations

Now assume the form of the differential equation is “good”

CAN WE IMPROVE THE SOLVERS?

Non-Stiff Methods are still being improved!

The Structure of a Runge-Kutta Method

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \,,$$

where

$$\begin{aligned}
k_1 &= f(t_n, y_n), \\
k_2 &= f(t_n + c_2 h, y_n + h(a_{21} k_1)), \\
k_3 &= f(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2)), \\
&\ \vdots \\
k_s &= f(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \cdots + a_{s,s-1} k_{s-1})).
\end{aligned}$$

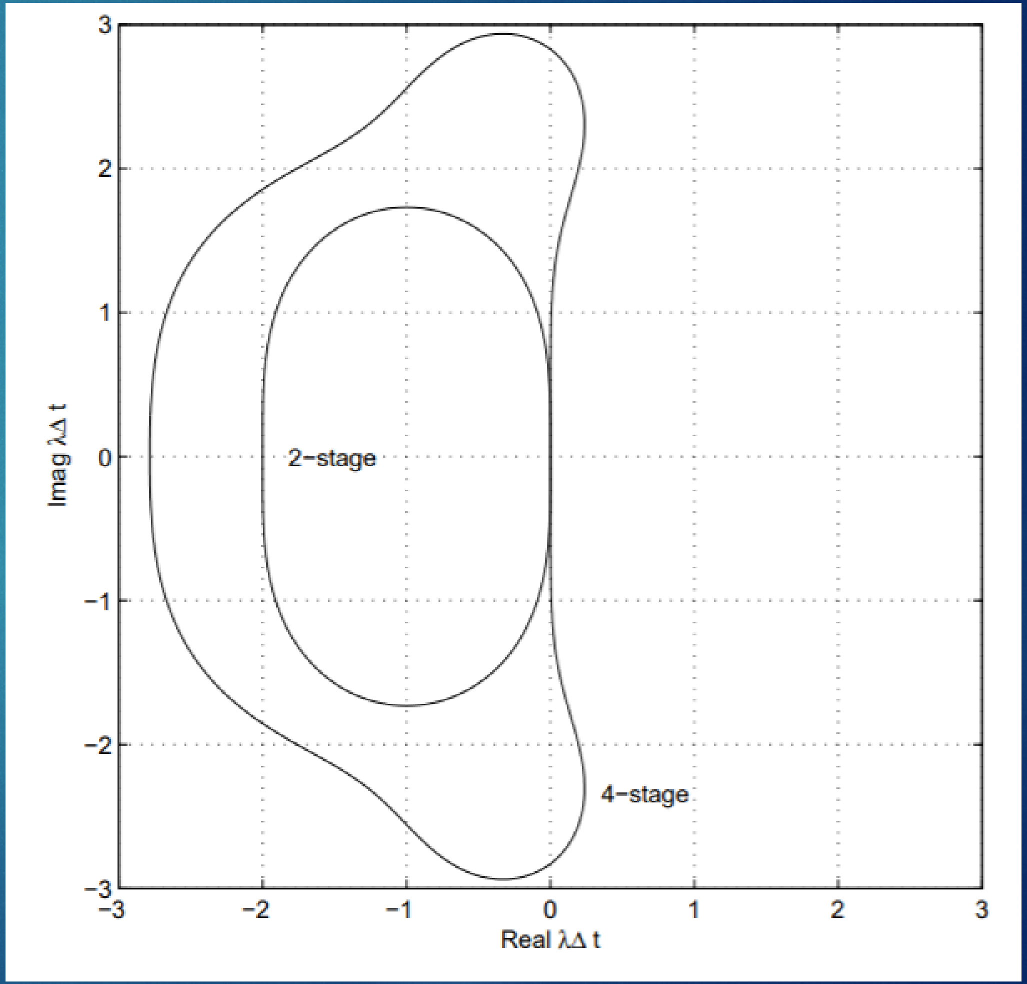
$$\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & & \ddots & & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Ways to Judge an RK Method

Optimization of next order coefficients

$$
\begin{gathered}
b_2a_{21} + b_3[a_{31} + a_{32}] + b_4[a_{41} + a_{42} + a_{43}] = 1/2 \\
b_2a_{21}^2 + b_3[a_{31} + a_{32}]^2 + b_4[a_{41} + a_{42} + a_{43}]^2 = 1/3 \\
b_2a_{22} + b_3[a_{21}a_{32} + a_{33}] + b_4[a_{21}a_{42} + a_{43}(a_{31} + a_{32}) + a_{44}] = 1/6 \\
b_2a_{21}^3 + b_3[a_{31} + a_{32}]^3 + b_4[a_{41} + a_{42} + a_{43}]^3 = 1/4 \\
b_2a_{21}a_{22} + b_3[\frac{1}{2}a_{21}^2a_{32} + (a_{31} + a_{32})(a_{21}a_{32} + a_{33})] + \frac{1}{2}b_4[a_{21}^2a_{42} \\
+a_{43}(a_{31} + a_{32})^2 + 2(a_{41} + a_{42} + a_{43})(a_{21}a_{42} + (a_{31} + a_{32})a_{43} + a_{44})] = 1/6 \\
b_3a_{22}a_{32} + b_4[a_{21}a_{32}a_{43} + a_{22}a_{42} + a_{33}a_{43}] = 1/24 \\
b_2a_{21}^4 + b_3[a_{31} + a_{32}]^4 + b_4[a_{41} + a_{42} + a_{43}]^4 = 1/5 \\
3b_2a_{21}^2a_{22} + b_3[a_{21}^3a_{32} + 3(a_{31} + a_{32})^2(a_{21}a_{32} + a_{33})] + b_4[a_{21}^3a_{42} \\
+(a_{31} + a_{32})^3a_{43} + 3(a_{41} + a_{42} + a_{43})^2(a_{21}a_{42} + (a_{31} + a_{32})a_{43} + a_{44})] = 7/20 \\
b_3a_{21}^2a_{32}(a_{31} + a_{32}) + b_4[(a_{41} + a_{42} + a_{43})(a_{21}^2a_{42} + (a_{31} + a_{32})^2a_{43})] = 1/15 \\
\frac{1}{2}b_2a_{22}^2 + b_3[a_{21}a_{32}(\frac{1}{2}a_{21}a_{32} + a_{22} + a_{33}) + a_{22}a_{32}(a_{31} + a_{32}) + \frac{1}{2}a_{33}^2] \\
+b_4[\frac{1}{2}a_{21}^2(a_{32}a_{43} + a_{42}^2) + (a_{31} + a_{32})(a_{21}(a_{32}a_{43} + a_{42}a_{43}) + a_{43}(a_{33} + a_{44}) \\
+\frac{1}{2}(a_{31} + a_{32})a_{43}^2) + a_{21}a_{42}(a_{22} + a_{44}) + (a_{21}a_{32}a_{43} + a_{22}a_{42} \\
+a_{33}a_{43})(a_{41} + a_{42} + a_{43}) + \frac{1}{2}a_{44}^2] = 11/120 \\
b_4a_{22}a_{32}a_{43} = 1/120
\end{gathered}
$$

Stability



Dormand-Prince 5th Order (1980)

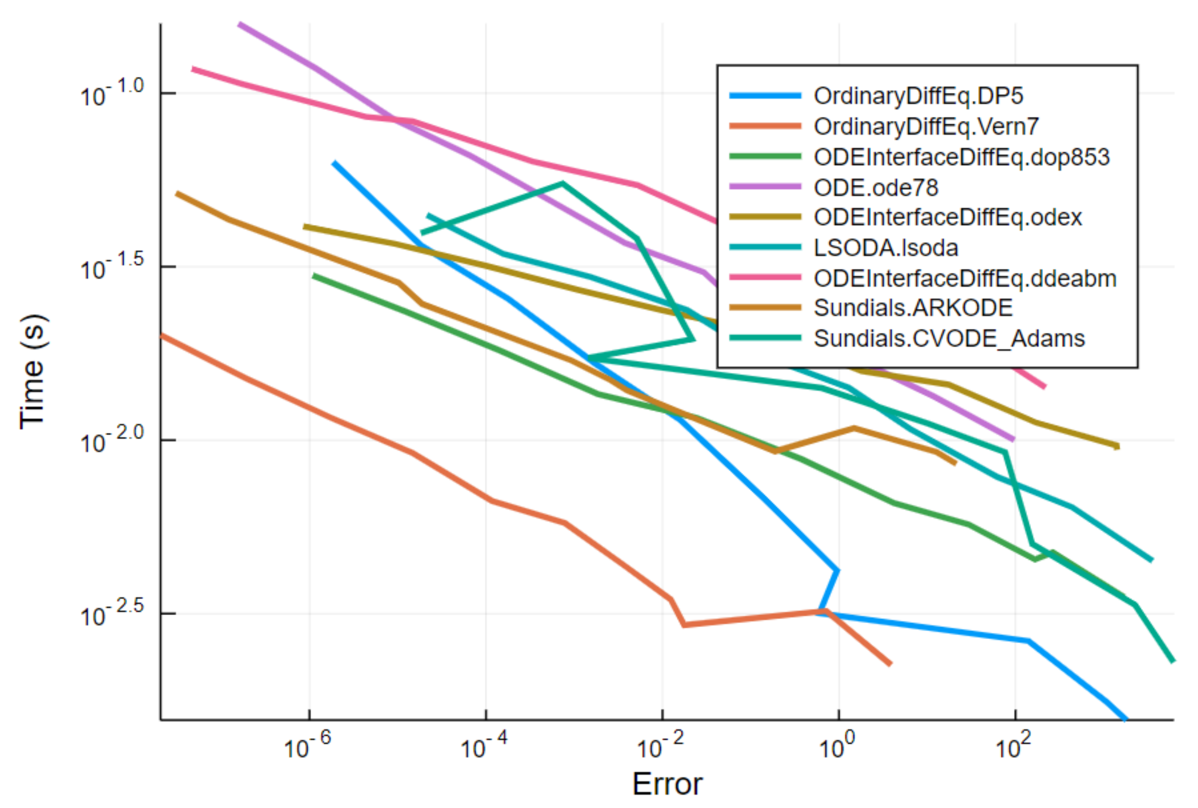
0							
1/5	1/5						
3/10	3/40	9/40					
4/5	44/45	−56/15	32/9				
8/9	19372/6561	−25360/2187	64448/6561	−212/729			
1	9017/3168	−355/33	46732/5247	49/176	−5103/18656		
1	35/384	0	500/1113	125/192	−2187/6784	11/84	
	35/384	0	500/1113	125/192	−2187/6784	11/84	0
	5179/57600	0	7571/16695	393/640	−92097/339200	187/2100	1/40

Advancements since 2010

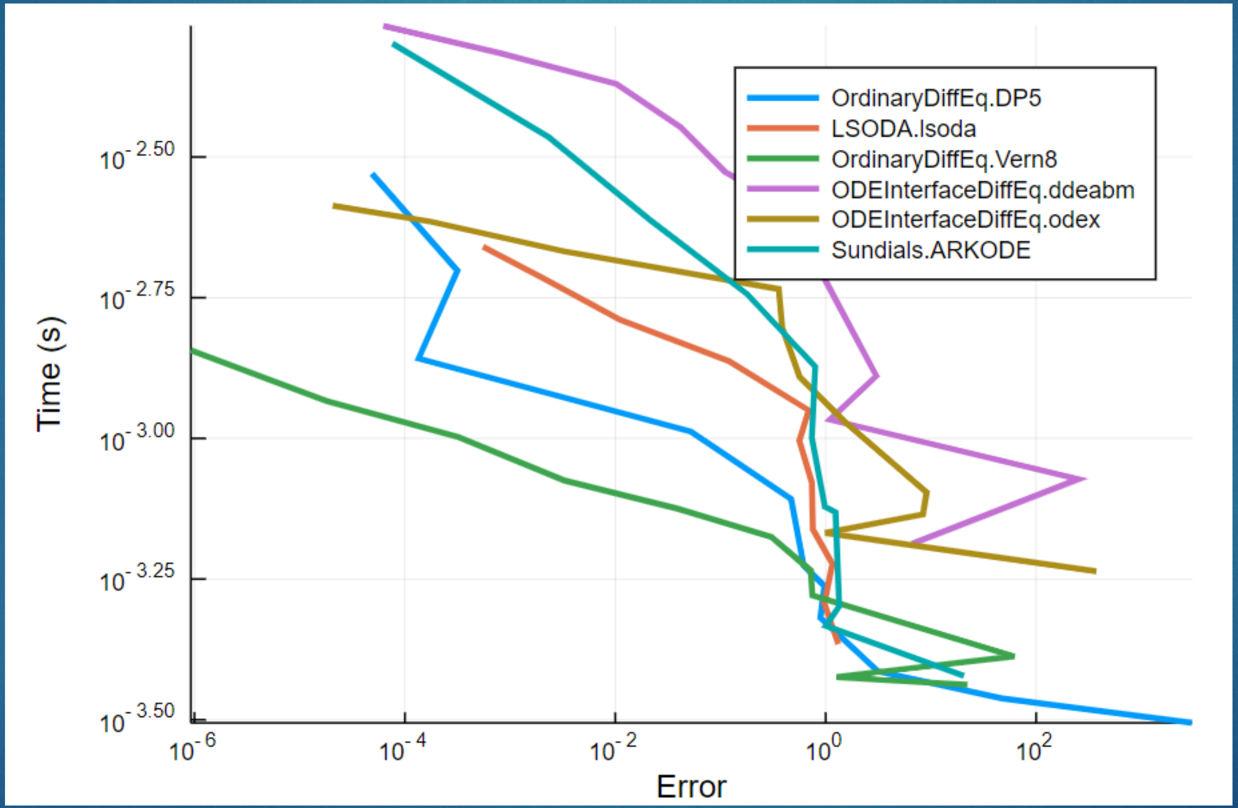
Recent methods, Tsit5 and Vern#, reduce the number of assumptions made in coefficient optimization, leading to more optimal solutions (>2010)

Methods specialized for wave equations, low-dispersion results, extended monotonicity equation for PDEs (SSPRK), etc. are hot topics in new high order Runge-Kutta methods (2017)

100x100 Linear ODEs



3-Body Problem (CVODE_Adams fails)



And parallelism is not well exploited!

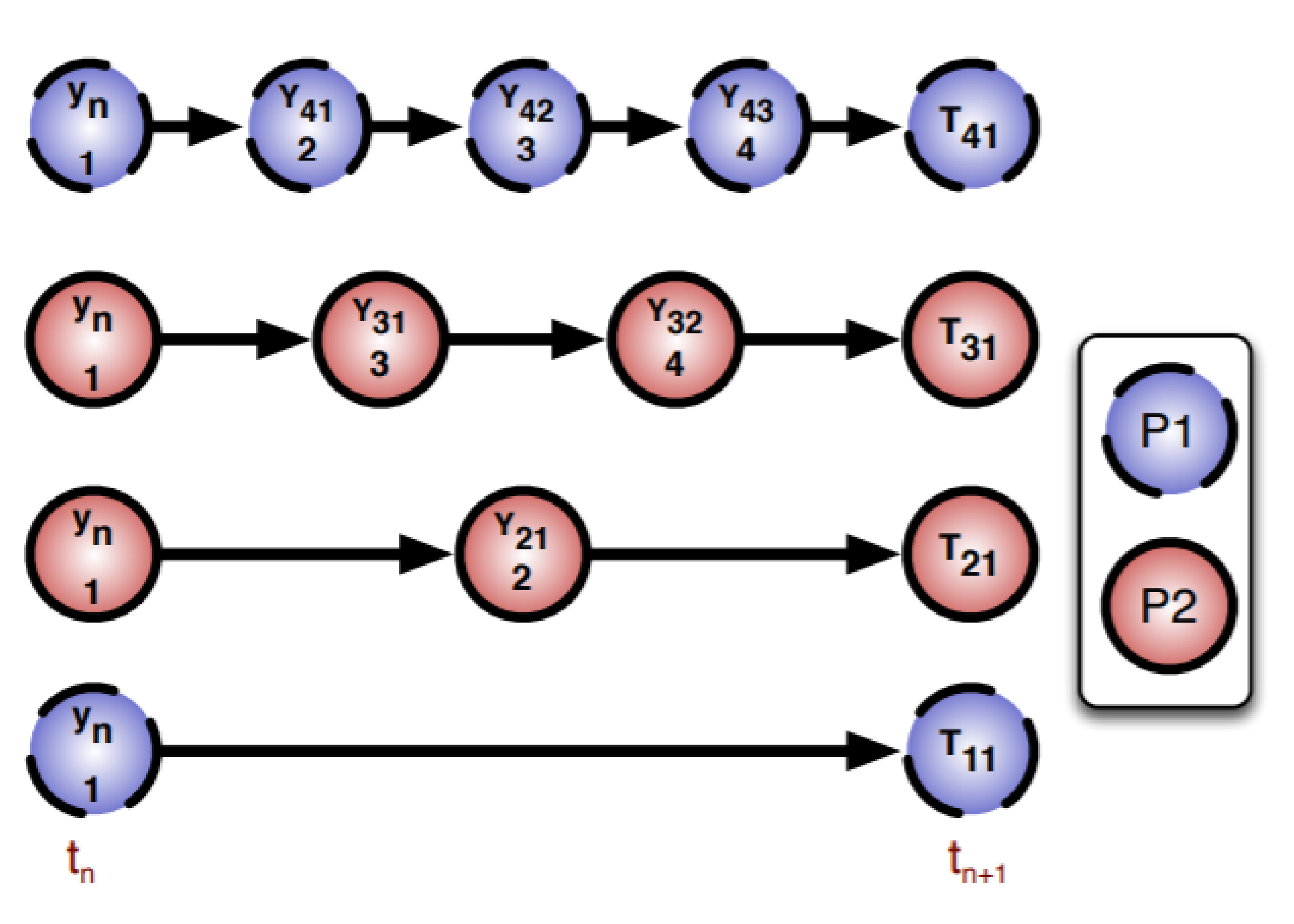
Pervasive Allowance of Within-Method parallelism through Julia

Zero GPU/Distributed message passing done by the solver!

```
@muladd function perform_step!(integrator, cache::Tsit5Cache, repeat_step=false)
  @unpack t,dt,uprev,u,f,p = integrator
  @unpack c1,c2,c3,c4,c5,c6,a21,a31,a32,a41,a42,a43,a51,a52,a53,a54,a61,a62,a63,a64,a65,a
  @unpack k1,k2,k3,k4,k5,k6,k7,utilde,tmp,atmp = cache
  a = dt*a21
  @. tmp = uprev+a*k1
  f(k2, tmp, p, t+c1*dt)
  @. tmp = uprev+dt*(a31*k1+a32*k2)
  f(k3, tmp, p, t+c2*dt)
  @. tmp = uprev+dt*(a41*k1+a42*k2+a43*k3)
  f(k4, tmp, p, t+c3*dt)
  @. tmp = uprev+dt*(a51*k1+a52*k2+a53*k3+a54*k4)
  f(k5, tmp, p, t+c4*dt)
  @. tmp = uprev+dt*(a61*k1+a62*k2+a63*k3+a64*k4+a65*k5)
```

Multithreading Extrapolation

Simultaneous Euler stepping of different step sizes



Ketcheson, 2016

Parallel Runge-Kutta methods

$$\mathbf{A} = \begin{bmatrix} 0 & & & & \\ \times & 0 & & & \\ \times & 0 & 0 & & \\ \times & \times & \times & 0 & \\ \times & \times & \times & 0 & 0 \end{bmatrix}$$

5 stages

But only 3 steps in parallel

DiffEqGPU.jl: Automatic GPU-based parameter parallelism of high level code

```julia
using DiffEqGPU, CuArrays, OrdinaryDiffEq, Test

function lorenz(du,u,p,t)
 @inbounds begin
     du[1] = p[1]*(u[2]-u[1])
     du[2] = u[1]*(p[2]-u[3]) - u[2]
     du[3] = u[1]*u[2] - p[3]*u[3]
 end
 nothing
end

CuArrays.allowscalar(false)
u0 = Float32[1.0;0.0;0.0]
tspan = (0.0f0,100.0f0)
p = (10.0f0,28.0f0,8/3f0)
prob = ODEProblem(lorenz,u0,tspan,p)
prob_func = (prob,i) -> remake(prob,p=rand(Float32,3).*p)
monteprob = EnsembleProblem(prob, prob_func = prob_func)

#Performance check with nvvp
# CUDAnative.CUDAdrv.@profile
@time solve(monteprob,Tsit5(),EnsembleGPUArray(),trajectories=100_000,saveat=1.0f0)
```

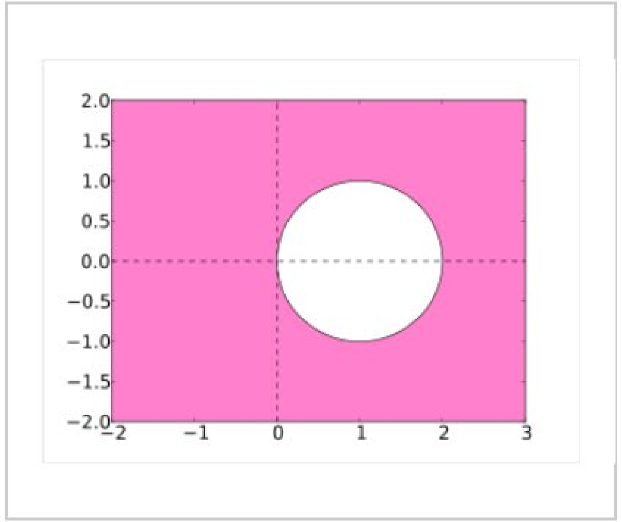
- Demonstrates a 12x-90x speedup over multithreaded ODE solves by using just 1 GPU.
- Handles stiff and non-stiff hybrid ODEs/SDEs/DDEs/DAEs with adaptive timestepping.
- Support coming soon:
  - Multiple GPUs
  - Mixed with multithreading/distributed
- Result: take existing hybrid ODE simulations written in Julia and automatically GPU+distributed+multithreaded parallelize it for the user!
- Requires relatively small systems (<50 DEs?)

Stiff ODEs: Fall of the BDF

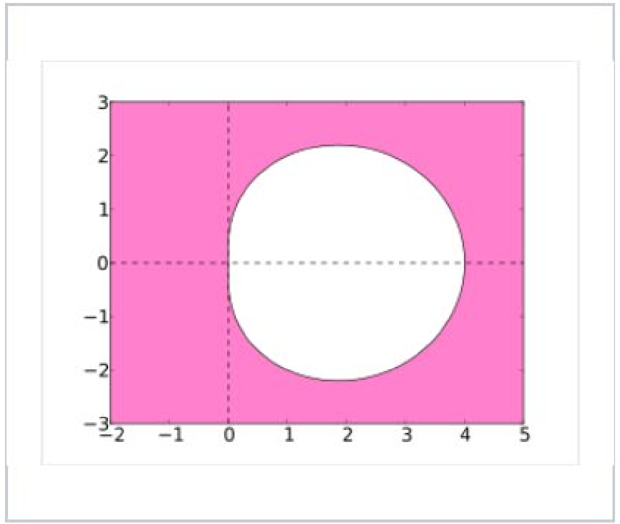
WHAT'S COMING TO GET GEAR'S METHOD.

Evolution of Gear's Method

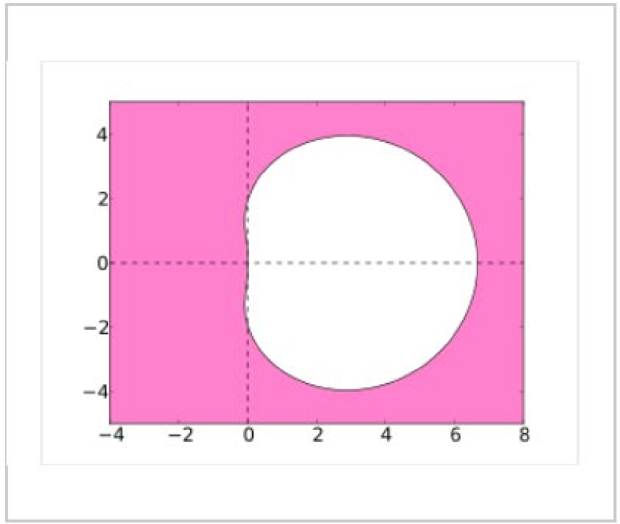
- GEAR: Original code. Adaptive order adaptive time via interpolation
  - Lowers the stability!
- LSODE series: update of GEAR
  - Adds rootfinding, Krylov, etc
- VODE: Variable-coefficient form
  - No interpolation necessary.
- CVODE: VODE rewritten in C++
  - Adds sensitivity analysis



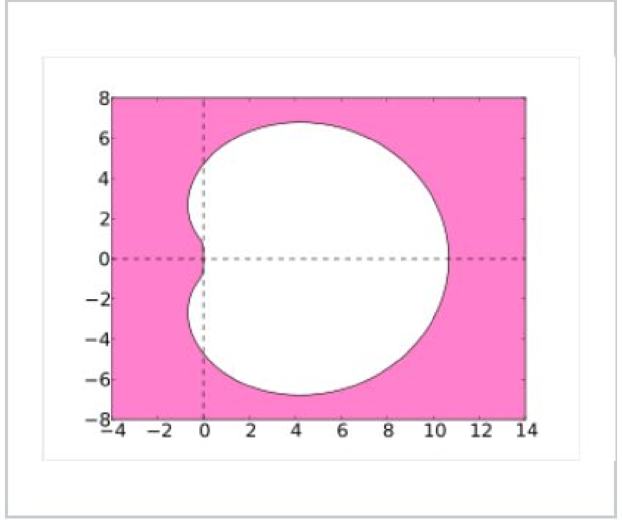
BDF1



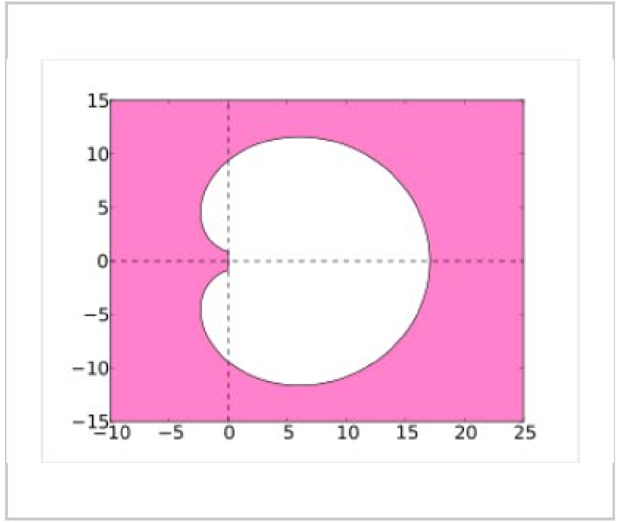
BDF2



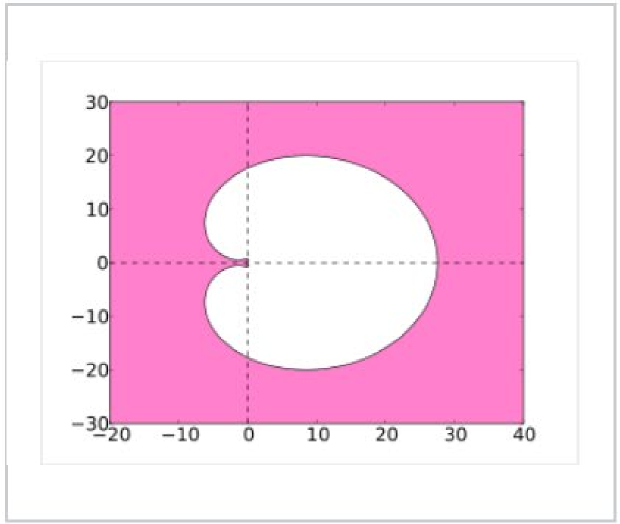
BDF3



BDF4



BDF5



BDF6

Problems with BDF

BDF is a multistep method

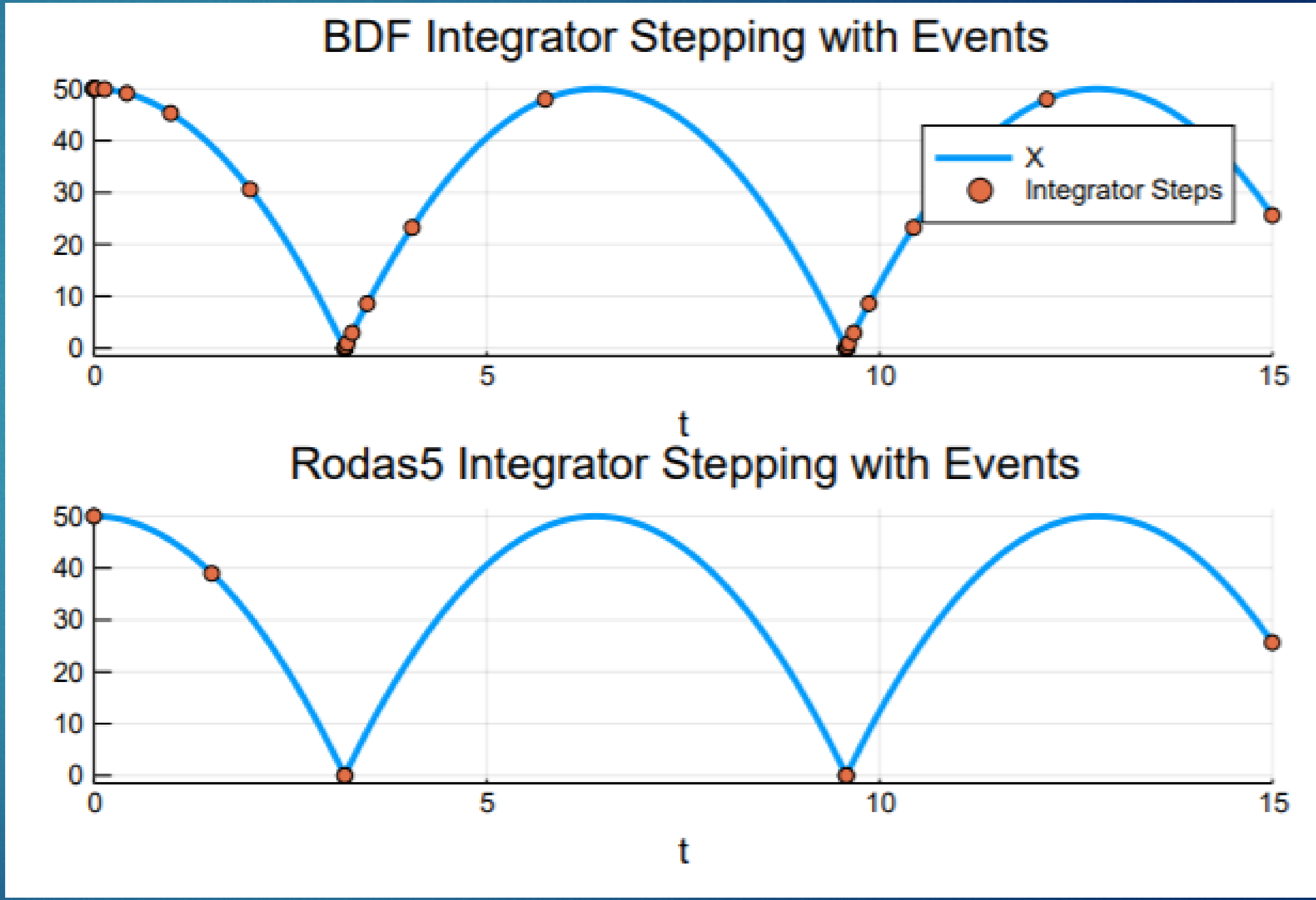
- Needs “Startup Steps”
- Inefficient with events

It is only L-stable up to 2nd order

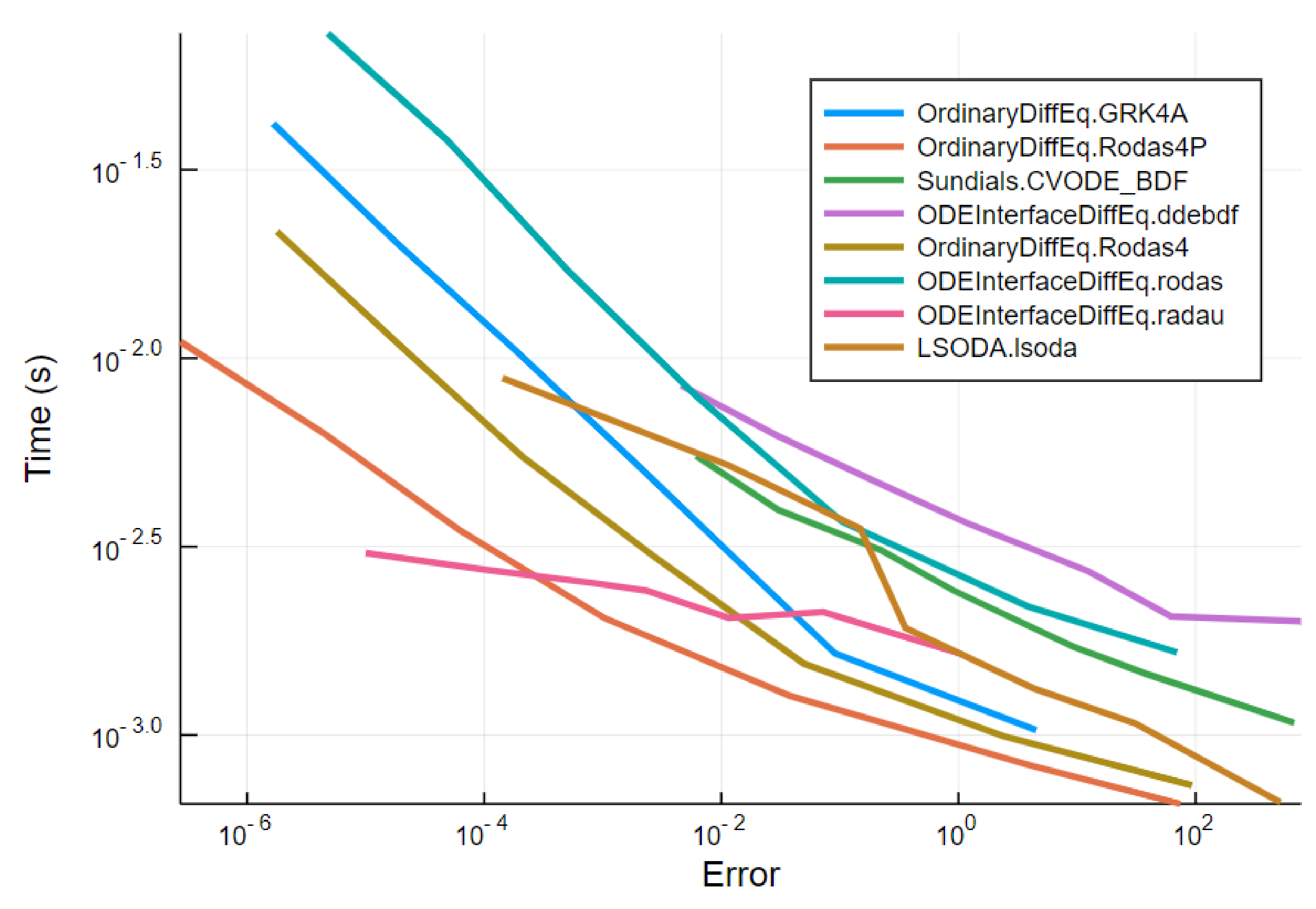
Has high truncation error coefficients

Implicit

- Requires good step predictors



Orego Benchmarks



Rosenbrock Methods

Aren't new! (ode23s)

Can fix a lot of problems:

- Exploit sparse factorization
- No step predictions required
- Can optimize coefficients to high order

Con: Needs accurate Jacobians

- Answer: AD (or symbolic)

$$Wk_1 = F(y_n)$$

$$Wk_2 = F\left(y_n + \frac{2}{3}hk_1\right) - \frac{4}{3}hdJk_1$$

$$y_{n+1} = y_n + \frac{h}{4}(k_1 + 3k_2)$$

ODE Problems can fall into different classes

Physical Modeling

SecondOrderODEProblem(f,u0,tspan,p)

- $u'' = f(u, p, t)$

PartitionedODEProblem(f1,f2,v0,u0,tspan,p)

- $v' = f_1(t, u)$
- $u' = f_2(v)$

HamiltonianODEProblem(H,p0,q0,tspan,p)

- $H(p, q)$

PDE Discretizations

SplitODEProblem(f1,f2,u0,tspan,p) (IMEX)

- $u' = f_1(u, p, t) + f_2(u, p, t)$

SemilinearODEProblem(A,f2,u0,tspan,p)

- $u' = Au + f(u, p, t)$

LocalSemilinearODEProblem(A,f2,u0,tspan,p)
$u' = Au + f.(u, p, t)$

SDIRK Methods can treat half of the problem as explicit, decreasing the nonlinear solver cost

Exponential Runge-Kutta

Explicit methods for stiff equations

- Small enough: Build matrix exponential
- Large enough: Krylov exp(t*A)*v

$$U_{ni} = e^{c_i h_n L_n} u_n + h_n \sum_{j=1}^{i-1} a_{ij}(h_n L_n) N_n(U_{nj}),$$

$$u_{n+1} = e^{h_n L_n} u_n + h_n \sum_{i=1}^{s} b_i(h_n L_n) N_n(U_{ni})$$

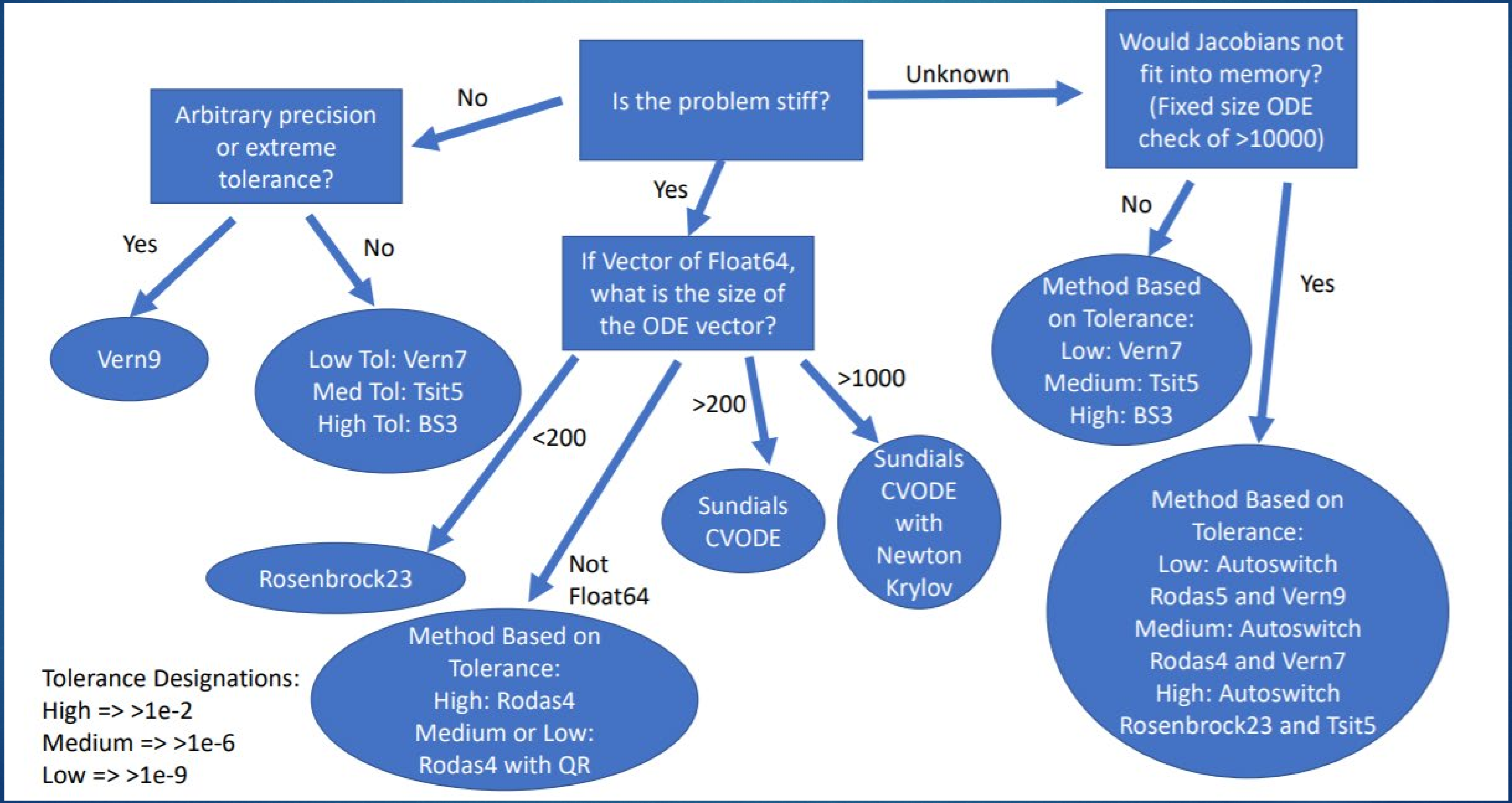
Crossover Question: Can we automatically divide equations for IMEX/Multirate methods?

Current Results (DiffEqBenchmarks.jl)

- <20-30 stiff ODEs = High order Rosenbrock
- <2000 ODEs = Optimized SDIRK Methods
- >2000 (general) ODEs = SUNDIALS BDF ☹
  - Stabilized Explicit methods in PDE contexts
  - IMEX methods when a split is known
  - …

Ongoing research to crack the code for more types of systems

Putting it together for users: polyalgorithms



Conclusion

- Today you can solve differential equations
- Tomorrow you will likely be able to solve them much faster
  - Neural-embedded methods for simplifying models
  - A compiler for researching transformations methods in context
  - Ongoing improvements to numerical methods

Students, want a paid summer position?

- Contact me for Google Summer of Code development.
- No Julia experience is required.
- https://julialang.org/soc/ideas-page

Industry interested in this research?

- Contact me to help fund development in JuliaDiffEq!
- We need industry sponsors/interest for CSSI grants, please let us know