Two approaches to speeding up dynamics simulation for a low dimension mechanical system

S.G. Orlov, A.K. Kuzin, N.N. Shabrov

Computer technologies in engineering dept.
Peter the Great St. Petersburg Polytechnic University

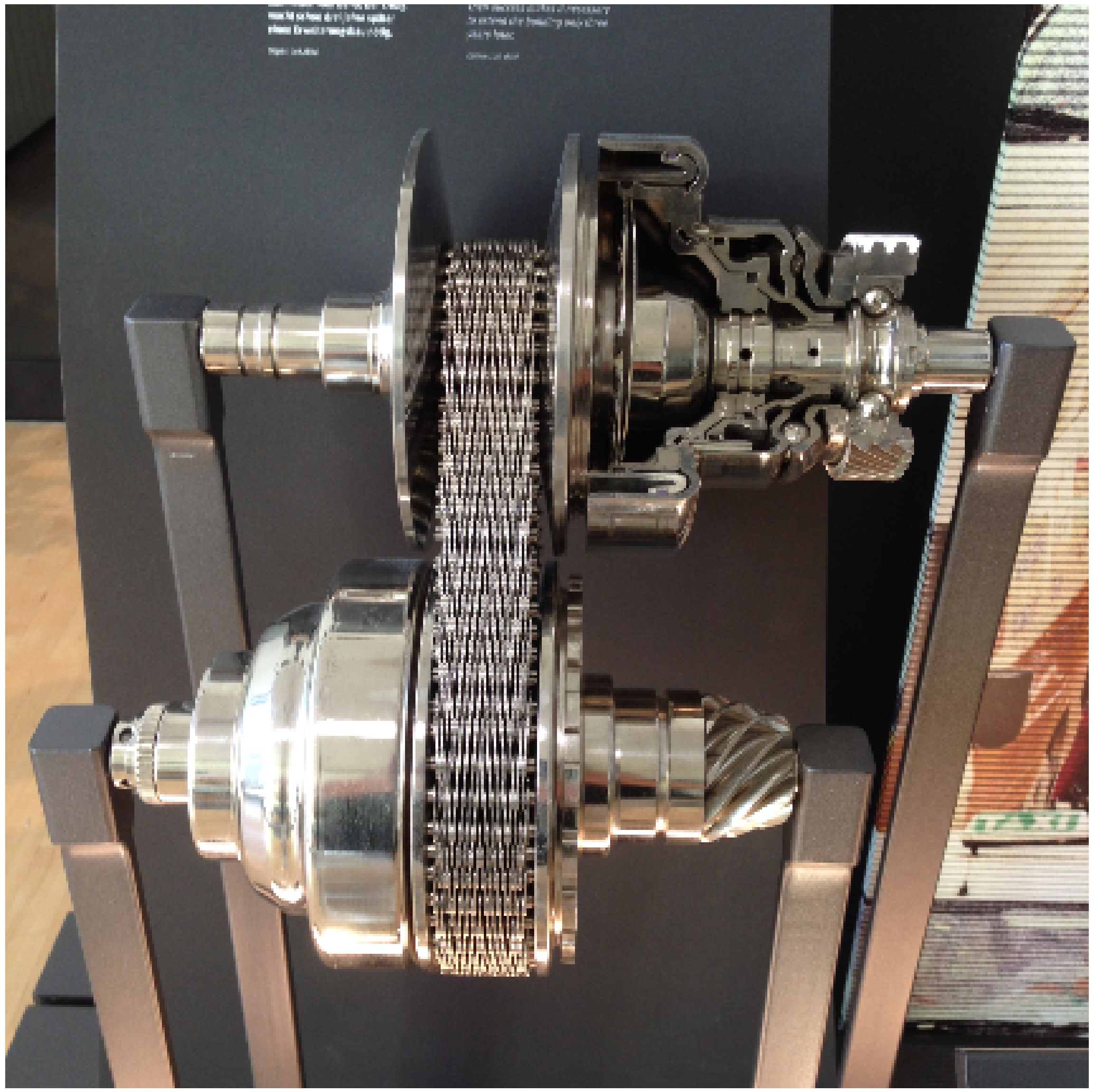
Russian Supercomputing Days
Sept. 25-26 2017, Moscow, Russia

Outline

- Model overview
- Parallelization
  - OpenMP for ODE right hand side
  - Results
- Exploring numerical methods
  - Jacobian eigenvalue analysis
  - Explicit (RK 4-8, GBS, extrapolated Euler)
  - Semi-implicit (W1, SW2-4, extrapolated W1)
  - Completely implicit (trapezoidal rule)
  - Stabilized explicit (DUMKA)
- Conclusions

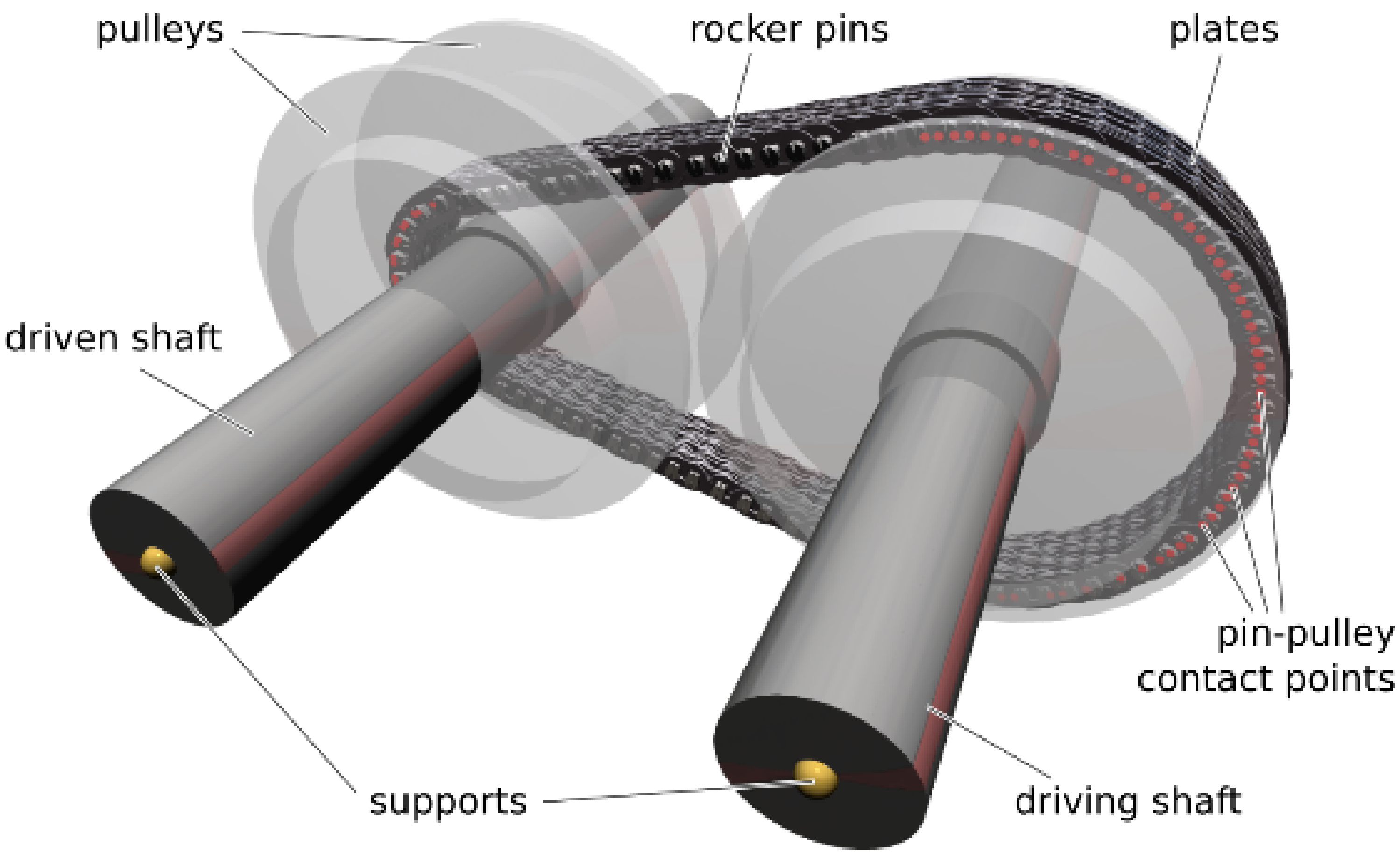
Model overview

Real device

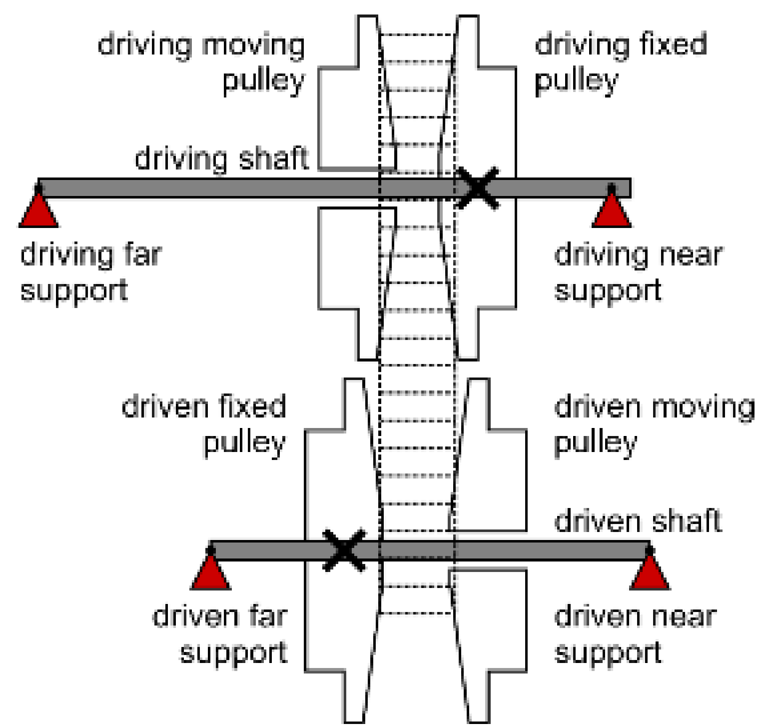


Model overview

3D view



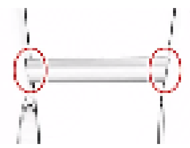
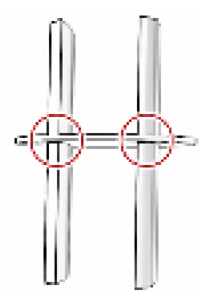
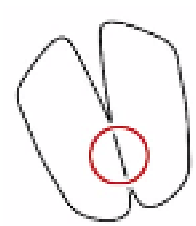
top view

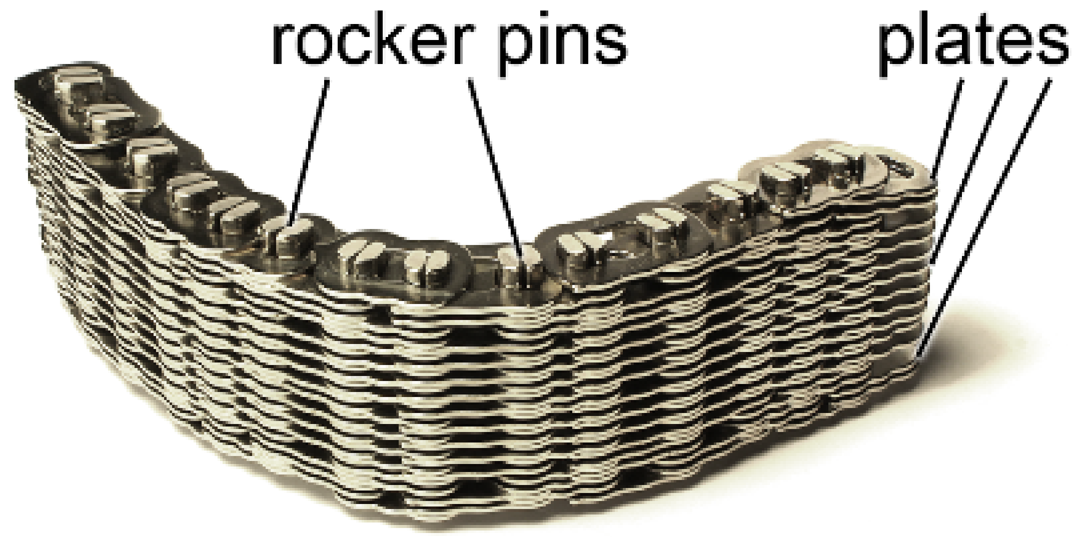


Model overview

The system works like this:

Model overview

- The chain consists of plates and rocker pins
- Each pin has two halves rolling over each other
- There are many contact interactions
    - pin — pulley 
    - pin — plate 
    - pin — pin 

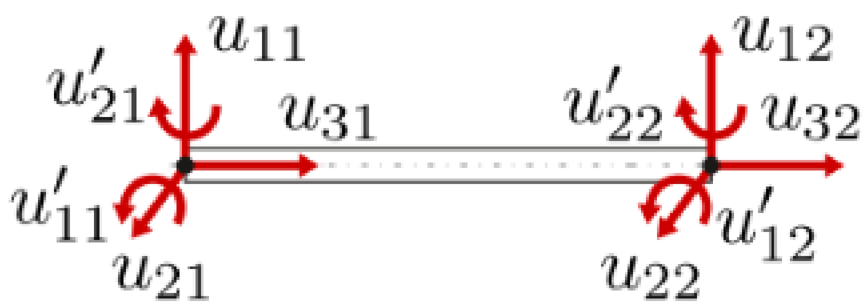


Model overview

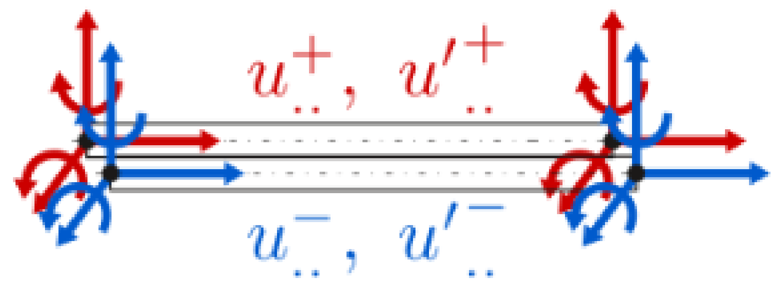
Pins, plates, and shafts are elastic

Model overview

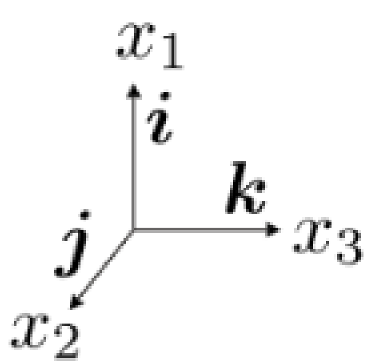
21 generalized coordinates per chain link



pin half



pin



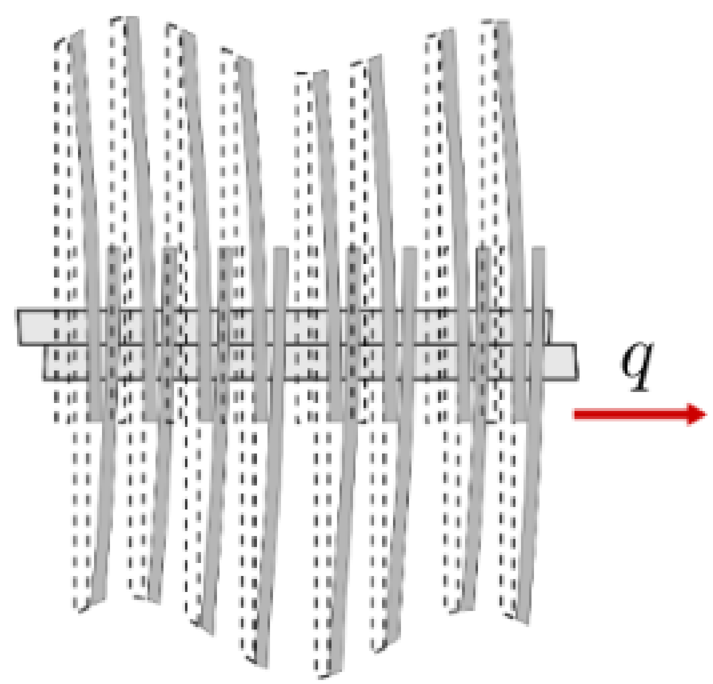
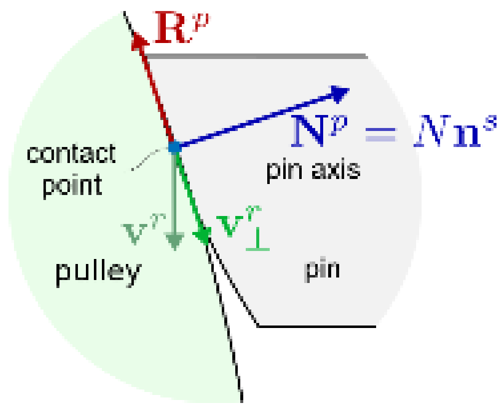


plate pack

Model overview

There is contact friction

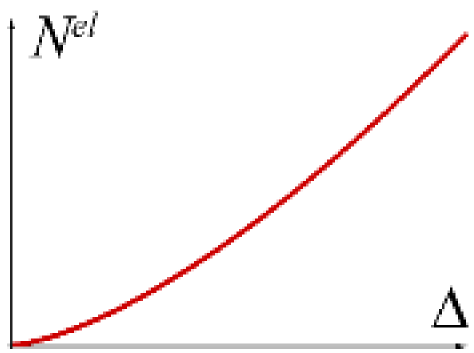
Contact forces



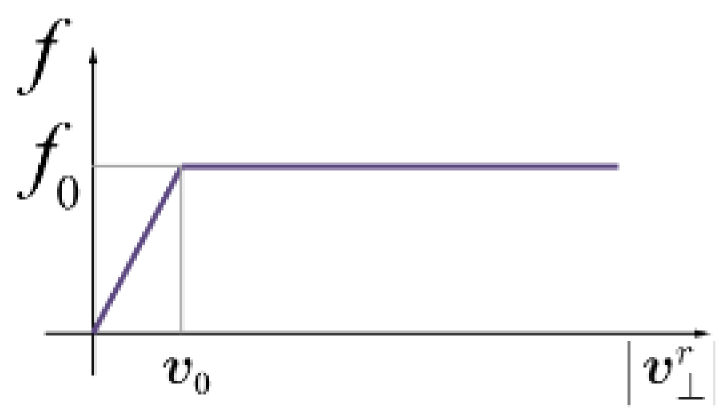
Formulas

$$
\begin{aligned}
&\mathbf{F}^p = \mathbf{N}^p + \mathbf{R}^p, \quad \mathbf{F}^s = \mathbf{N}^s + \mathbf{R}^s \\
&\mathbf{N}^p = -\mathbf{N}^s = N\mathbf{n}^s, \quad N = N^{el} + N^d, \\
&N^{el} = c\Delta^{3/2}, \quad N^d = b\dot{\Delta} \\
&\mathbf{R}^p = -\mathbf{R}^s = -f\left(|\mathbf{v}^r_\perp|, N^{el}\right) N^{el}\tau_\perp, \\
&\tau_\perp = \mathbf{v}^r_\perp / |\mathbf{v}^r_\perp|, \quad \mathbf{v}^r_\perp = (\mathbf{I} - \mathbf{n}^s\mathbf{n}^s)\cdot(\mathbf{v}^p - \mathbf{v}^s)
\end{aligned}
$$

Normal force law (Hertz)

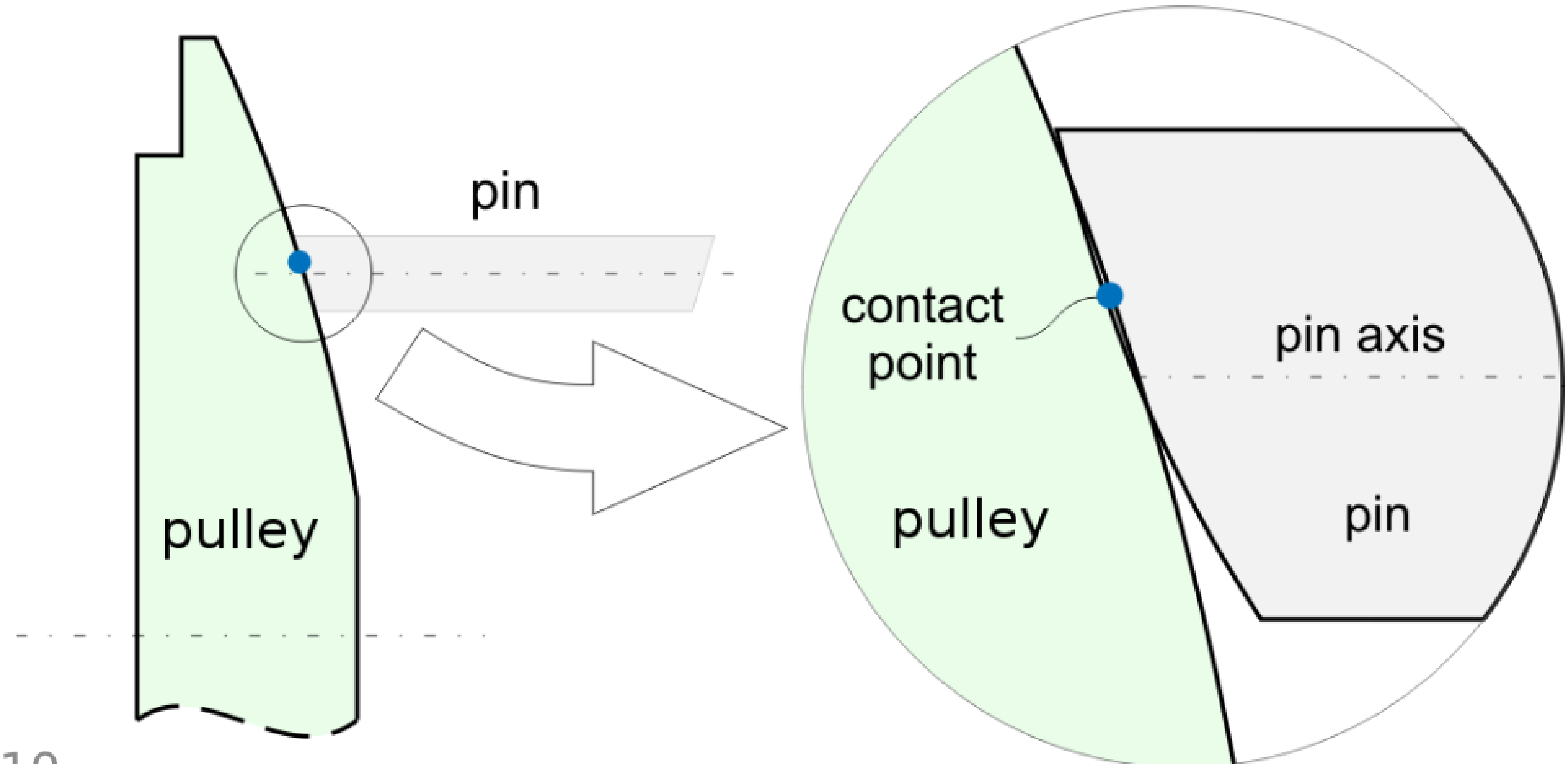


Friction law (nonsmooth!)



Model overview

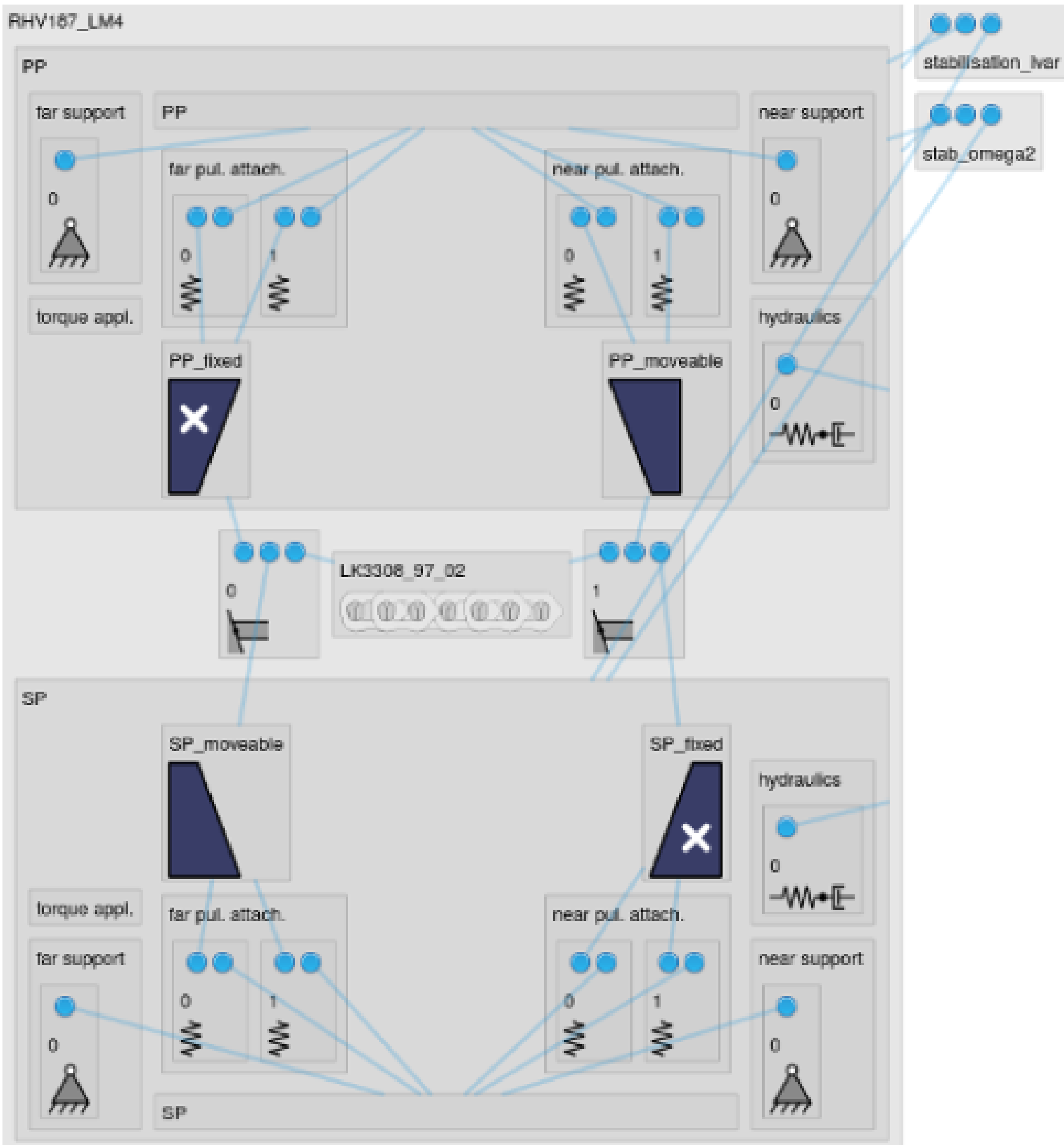
Pin-pulley contact surfaces are locally quadratic



Equations of motion

- Lagrange equations: $\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = \tilde{Q}$
- lead to $\mathbf{A}(q)\,\ddot{q} = \tilde{F}(t, q, \dot{q}) \quad \Rightarrow \quad \ddot{q} = F(t, q, \dot{q})$
  - The inertia matrix $\mathbf{A}$ is sparse block-diagonal
  - Sometimes it really depends on q
- In the normal form, ODE system is $\dot{x} = f(t, x)$
  - $\begin{matrix} q \equiv u \\ \dot{q} \equiv v \end{matrix}, \quad x = \begin{bmatrix} u \\ v \end{bmatrix}, \quad f = \begin{bmatrix} v \\ F(t, u, v) \end{bmatrix}$

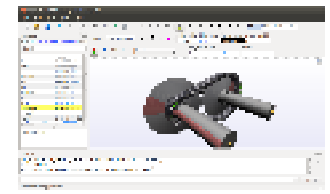
Model overview

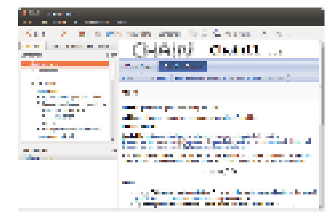


- Heterogeneous system
    - different parts

The problem

- Software product with docs, fancy GUI, scripting, postprocessing, visualization, etc., and support.





- But it runs slow
  - 1 real time second costs ~10 hours CPU time

- The goal
  - Make it run at least 100x faster

Parallelization

- Problem features
  - Tiny memory requirements (just 3600 vars)
    - Data most likely fits into cache
  - Several different parts in model
    - Including chain consisting of 80+ similar blocks
    - And 300+ similar contact pairs
  - $f(t, x)$ costs ~1 ms for single thread
  - Events (open/close contacts)
  - Object oriented C++ code
    - Not HPC-friendly memory organization
    - Complicated memory access patterns

Parallelization

- Solving IVP for $\dot{x} = f(t,x) = [v, F(t,u,v)]^T$
- Currently using explicit RK4 scheme

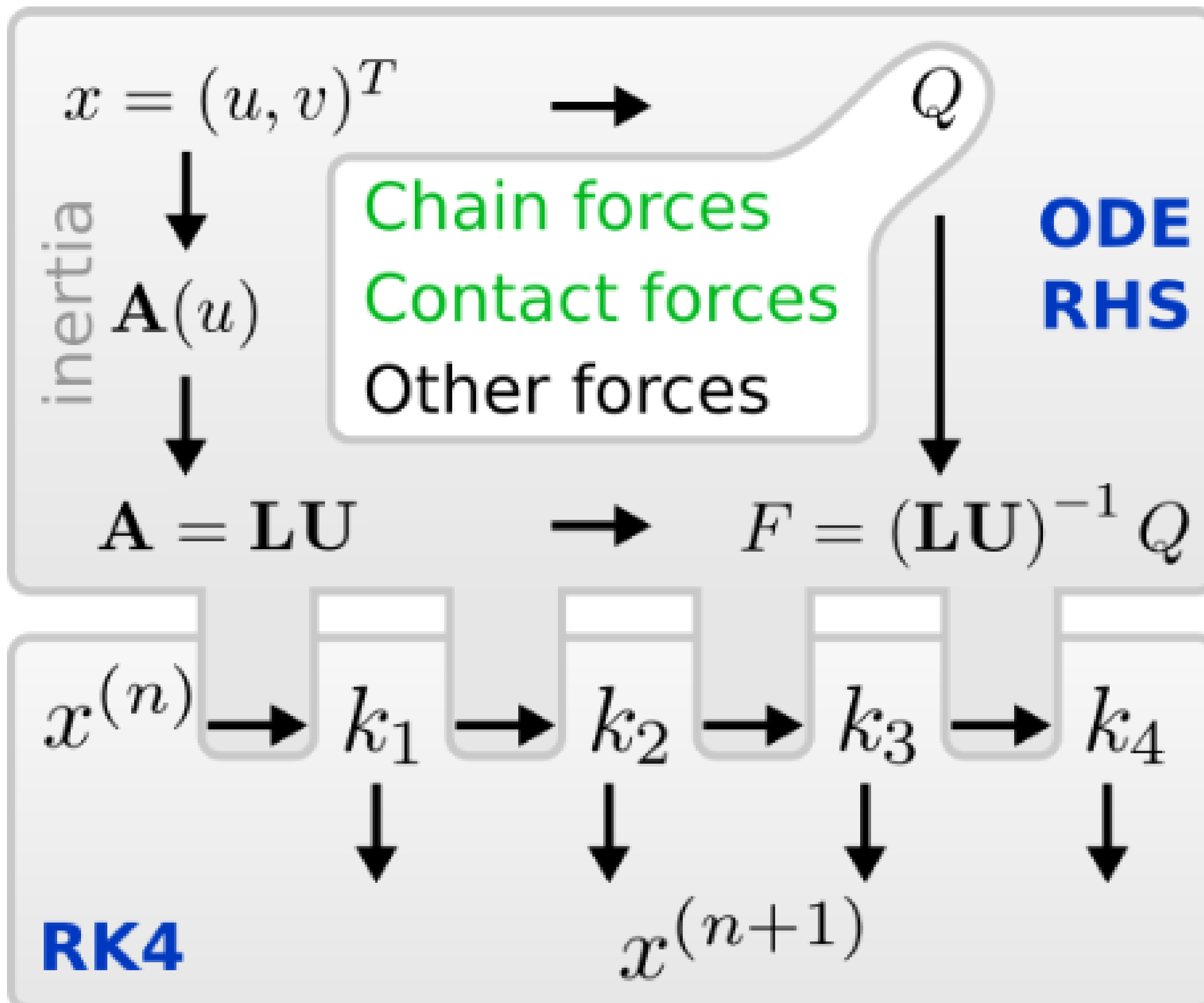
$$
\begin{aligned}
k_1 &= f\left(t^{(n)}, x^{(n)}\right), \\
k_2 &= f\left(t^{(n)} + \tfrac{h}{2}, x^{(n)} + \tfrac{h}{2}k_1\right), \\
k_3 &= f\left(t^{(n)} + \tfrac{h}{2}, x^{(n)} + \tfrac{h}{2}k_2\right), \\
k_4 &= f\left(t^{(n)} + h, x^{(n)} + hk_3\right), \\
x^{(n+1)} &= x^{(n)} + \tfrac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).
\end{aligned}
$$

- Model has about 1800 generalized coordinates
  - x dimension is about 3600
- Parallelizing $F(t,u,v)$ evaluation

Parallelization

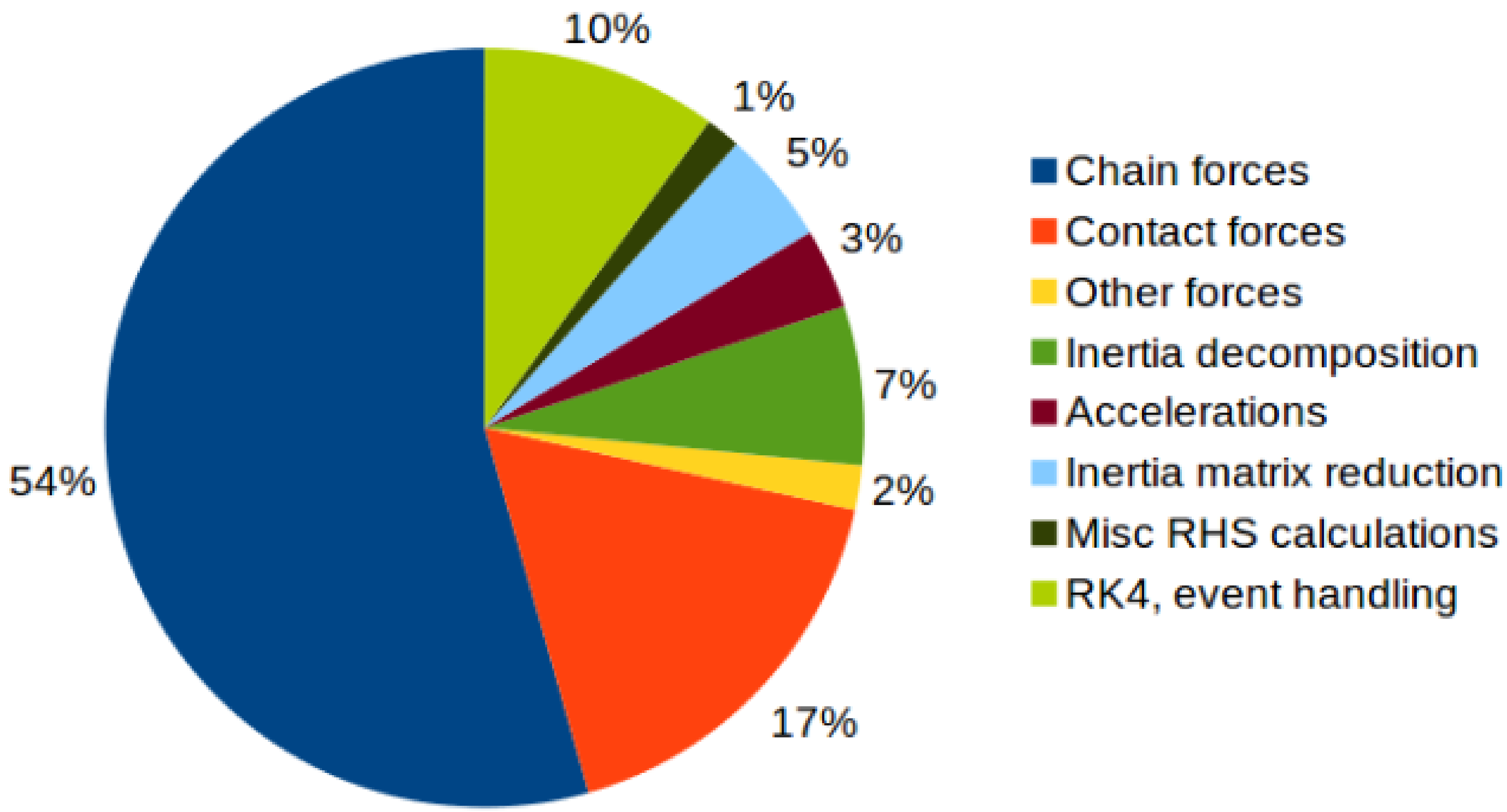
- First parallelize
  - Chain forces
  - Pin-pulley contact forces

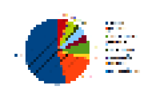
Big tasks within one RK4 step



Parallelization

Sequential code



more detailed: 

Parallelization

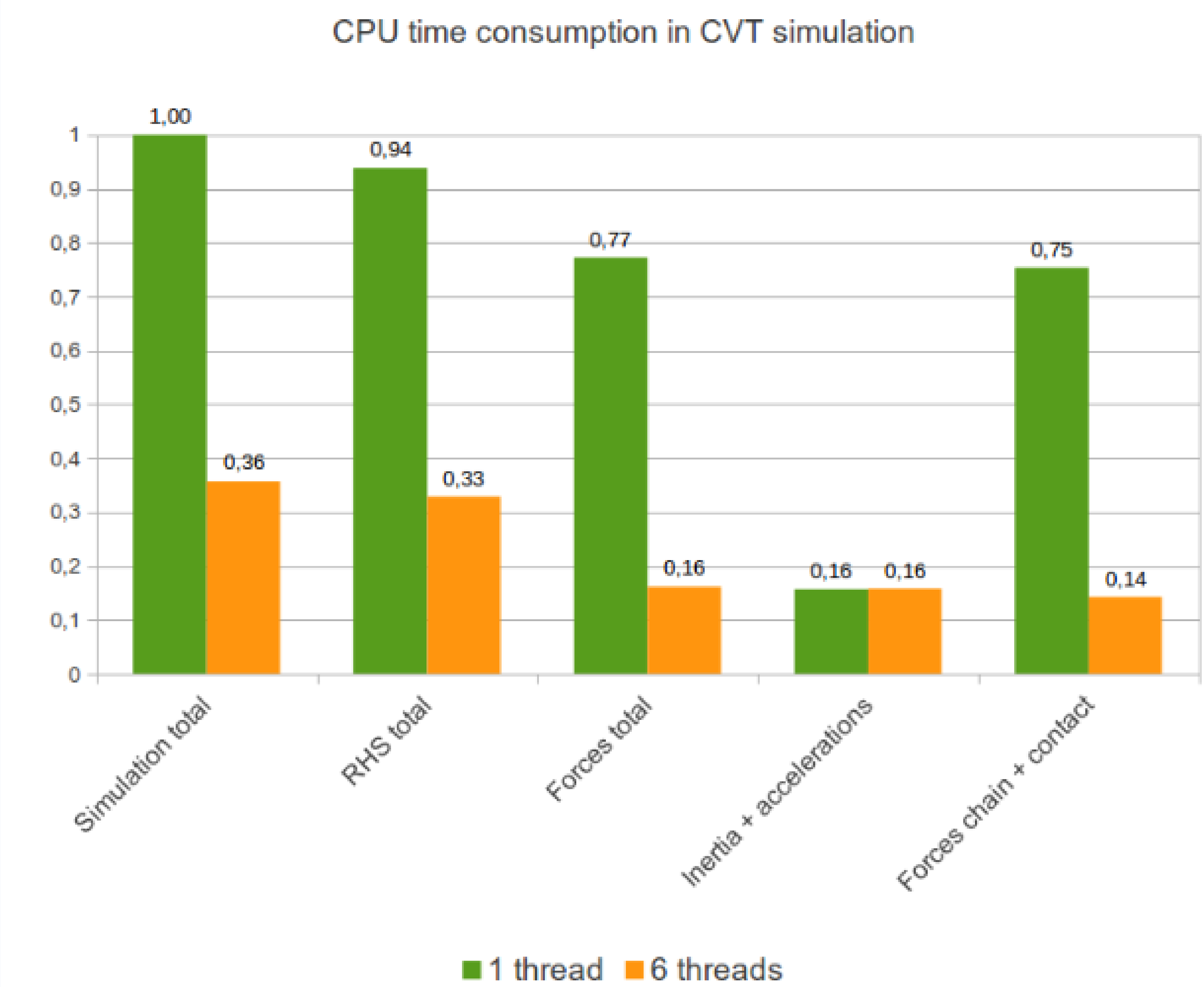
- Targeting SMP & NUMA architectures
  - Single nodes (now)
  - Clusters, with new runtime from HLRS (future)
    - This project is part of planned joint Russian-German project by St. Petersburg Polytechnical university and HLRS
- Using OpenMP
  - Thread-based parallelism (now)
  - Task-based parallelism (future)

Parallelization

Hardware parameters and OS/GCC versions

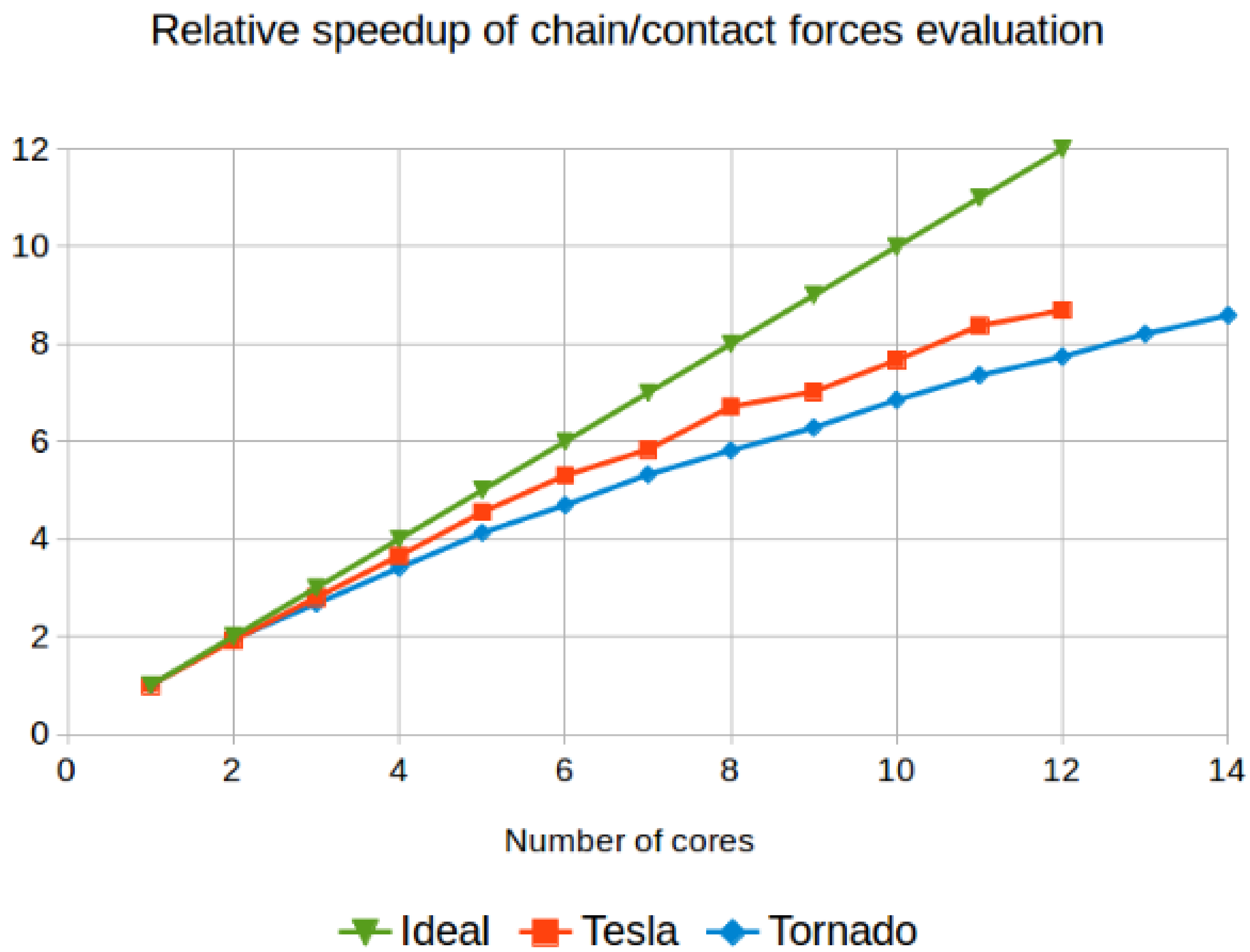
	Tesla	Tornado
Cores per socket	6	14
Sockets	2	2
NUMA Nodes	2	2
CPUs	Intel Xeon CPU X5660 2.80GHz	Intel Xeon CPU E5-2697 v3 2.60GHz
Linux	Ubuntu 16.04.4 LTS	CentOS Linux release 7.0.1406 (Core)
GCC version	5.4.0	5.4.0

Parallelization



All cores were explicitly assigned with GOMP_CPU_AFFINITY variable so only one NUMA node was used

Parallelization

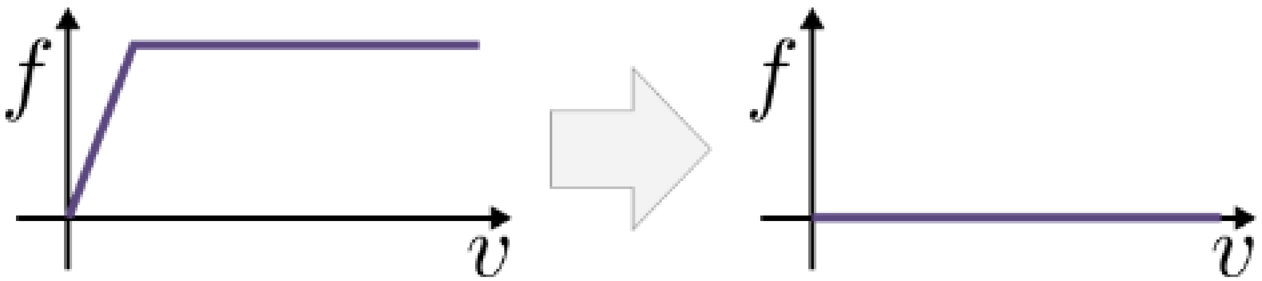


All cores were explicitly assigned with GOMP_CPU_AFFINITY variable so only one NUMA node was used if possible

Jacobian eigenvalue analysis

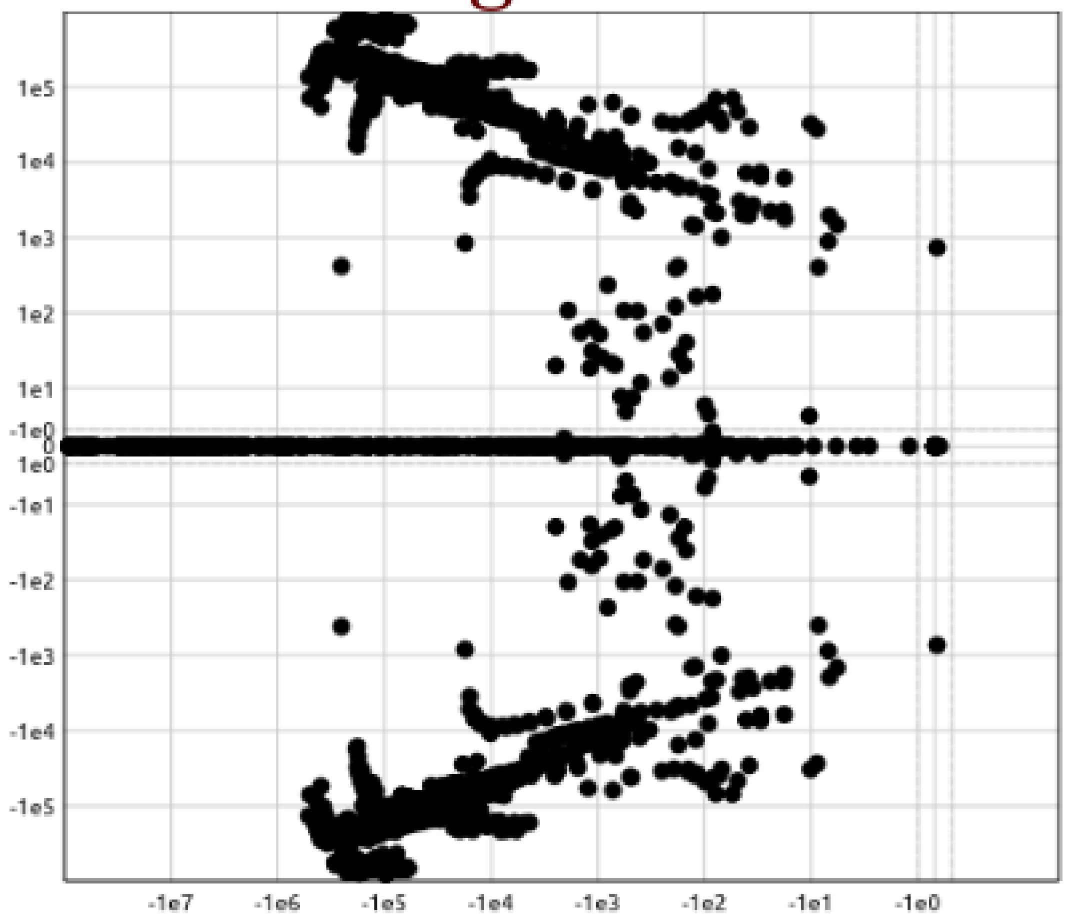
- System appears to be mildly stiff
- Natural frequencies up to 10^6 1/s
- Real negative λ up to -10^8 1/s
  - These are due to friction
    - Pin-pin friction at driving chain branch is the worst case
- Jacobian changes fast

Jacobian eigenvalue analysis

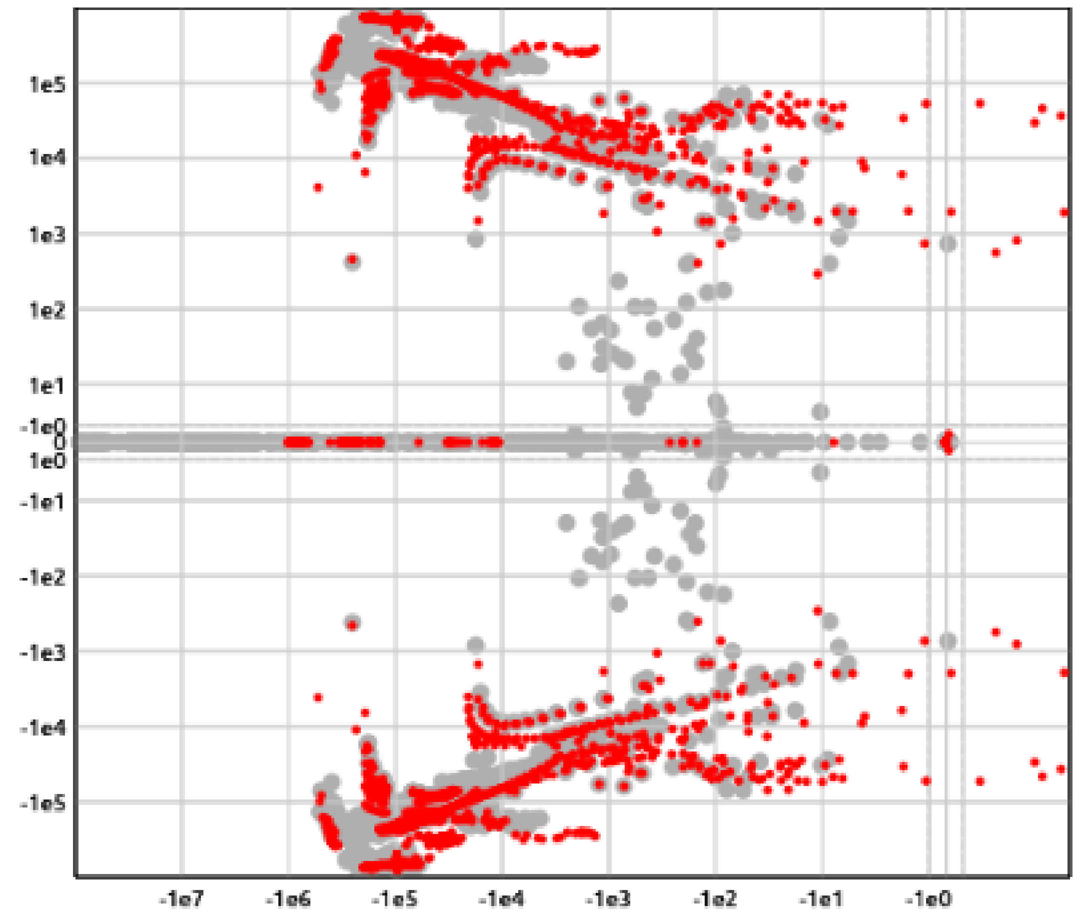


Disable friction at all

original



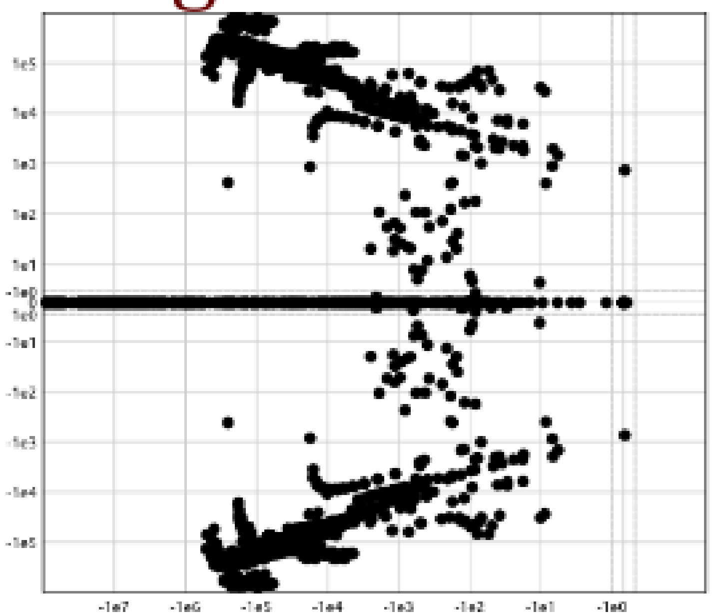
no friction



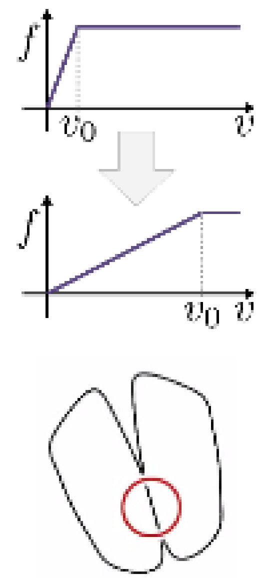
Jacobian eigenvalue analysis

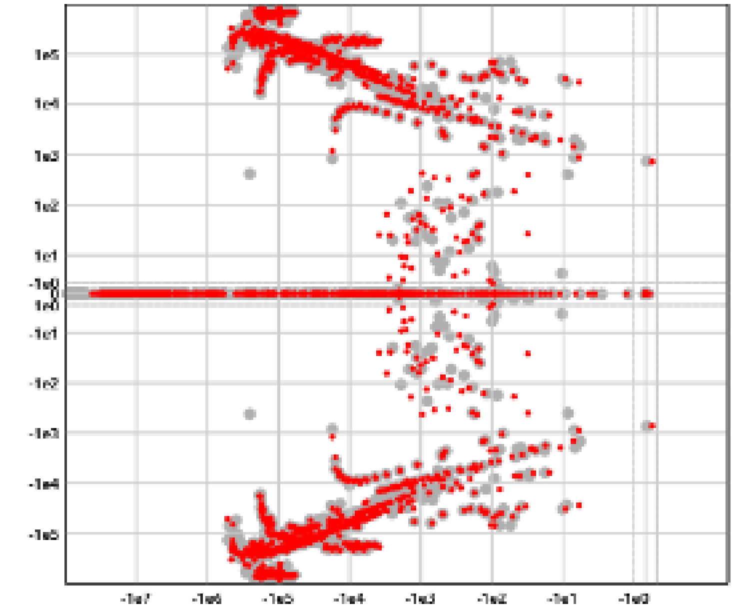
Decrease slope of linear friction part

original

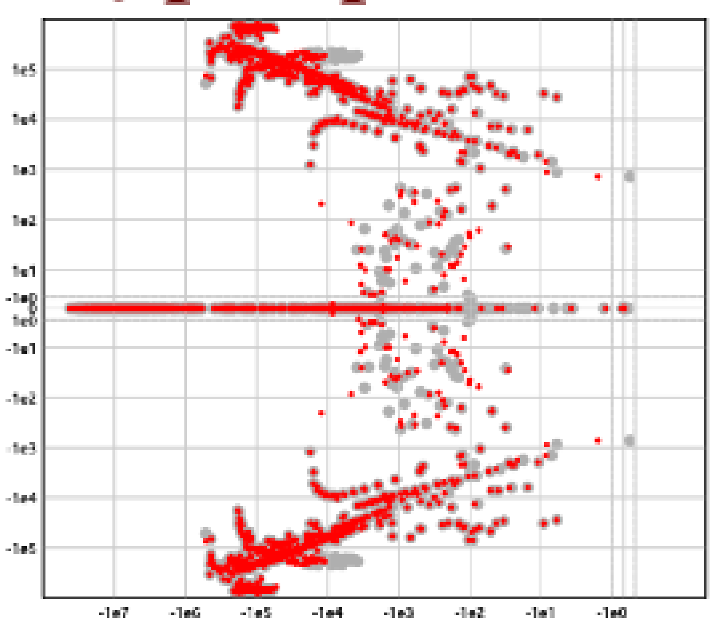


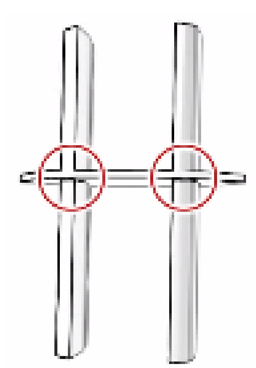
x10 v_0 pin-pin



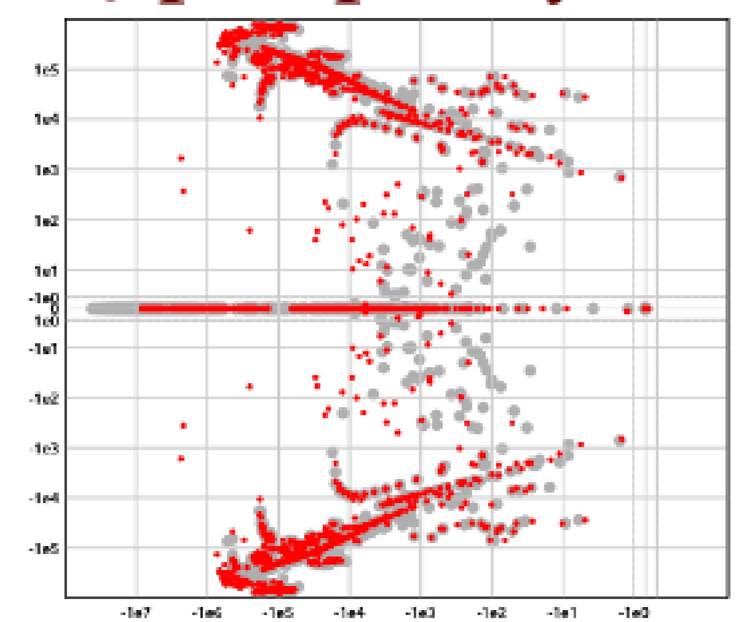


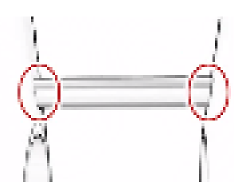
x10 v_0 pin-plate





x10 v_0 pin-pulley





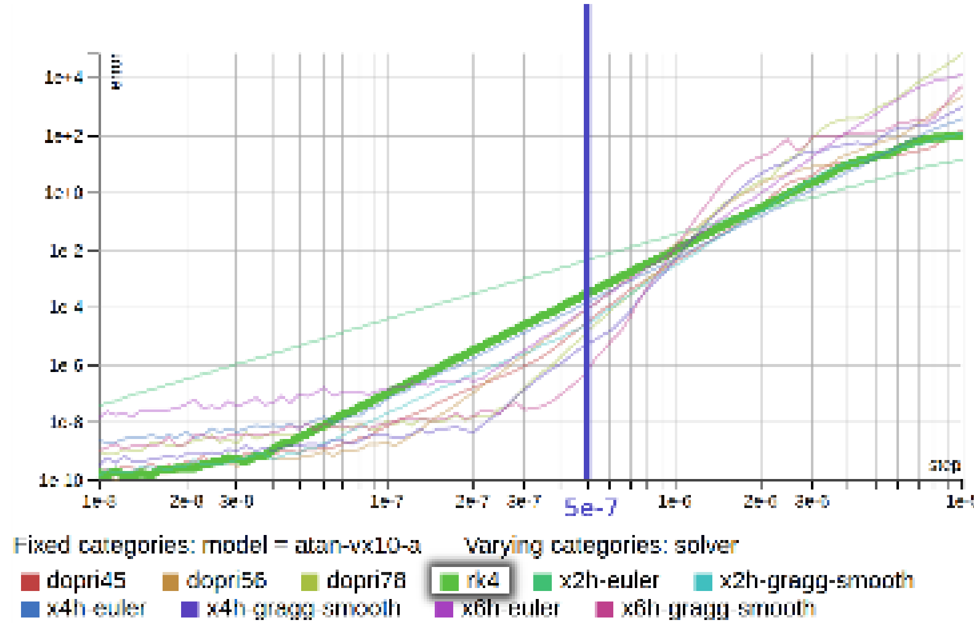
Exploring numerical methods

- Explicit methods
  - Easily implemented
  - Step size limited by stability requirements
  - But stability region can be extended...
- Semi-implicit methods
  - Require system Jacobian or its approximation
  - Linear system(s) at time step
- Completely implicit methods
  - Require system Jacobian or its approximation
  - Nonlinear system(s) at time step

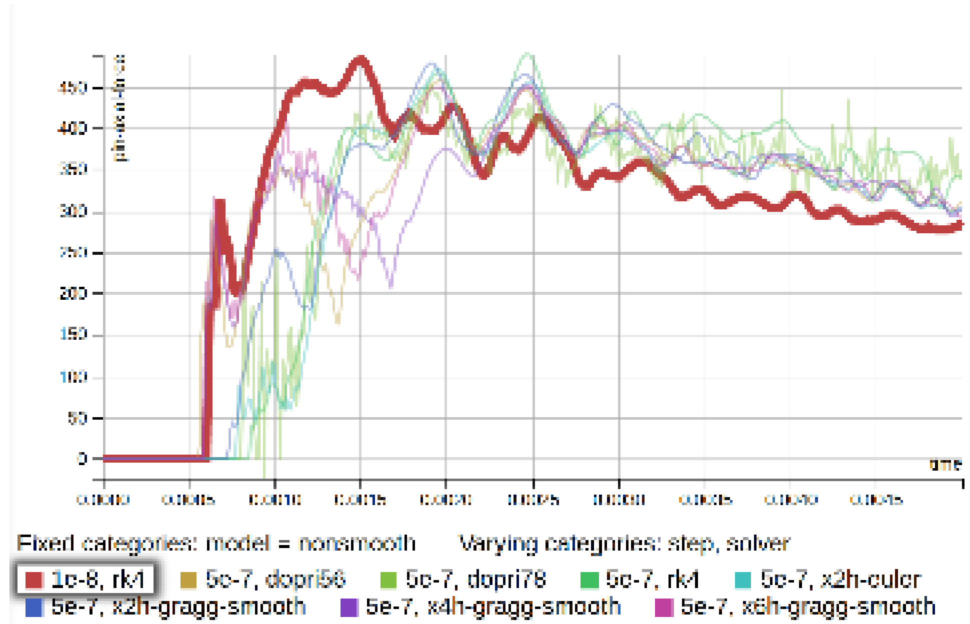
Explicit methods

Stability problems

- Common RK schemes
  - RK4
  - DOPRI45
  - DOPRI56
  - DOPRI78
  - GBS (smoothed)
  - Extrapolated Euler



Local error

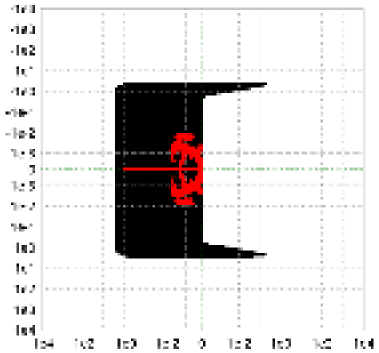


Sample curve

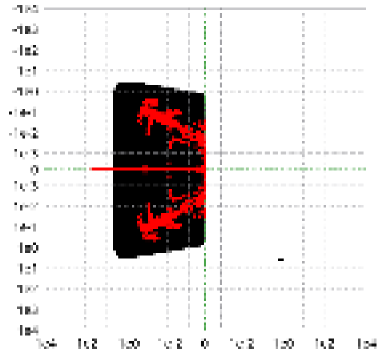
Explicit methods

Stability problems

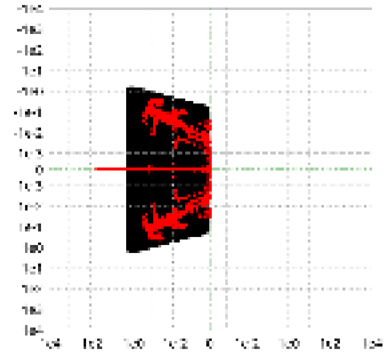
RK4 @ 1e-8



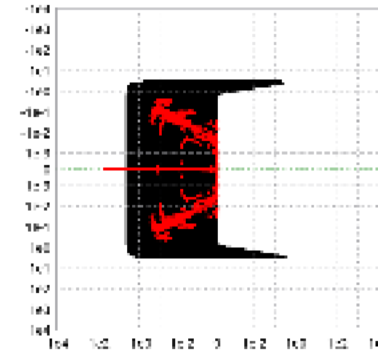
DOPRI45 @ 5e-7



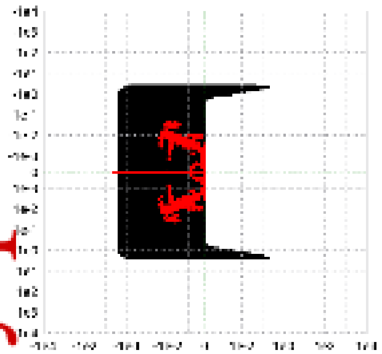
x2h-euler @ 5e-7



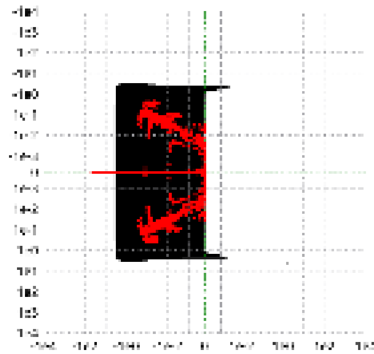
GBS-2h @ 5e-7



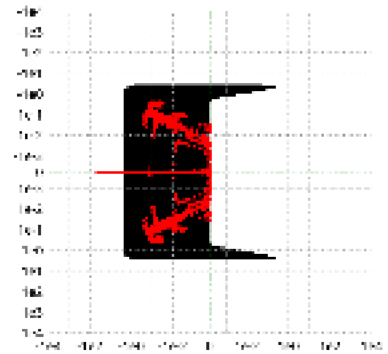
RK4 @ 5e-8



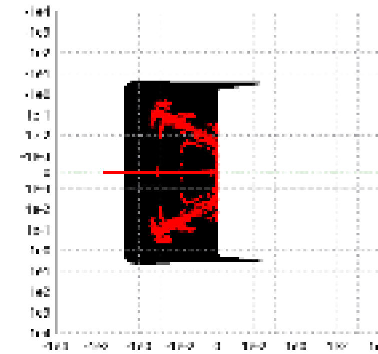
DOPRI56 @ 5e-7



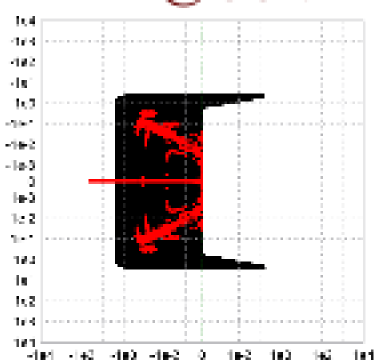
x4h-euler @ 5e-7



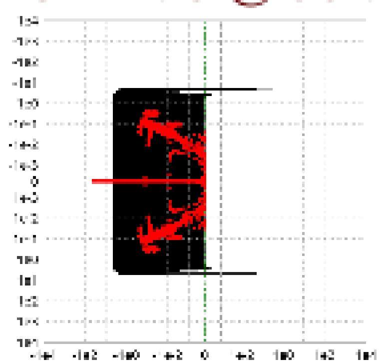
GBS-4h @ 5e-7



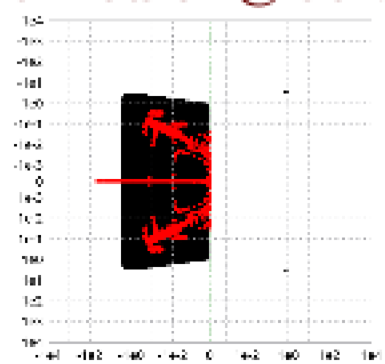
RK4 @ 5e-7



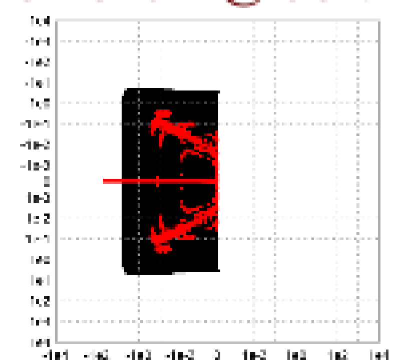
DOPRI78 @ 5e-7



x6h-euler @ 5e-7

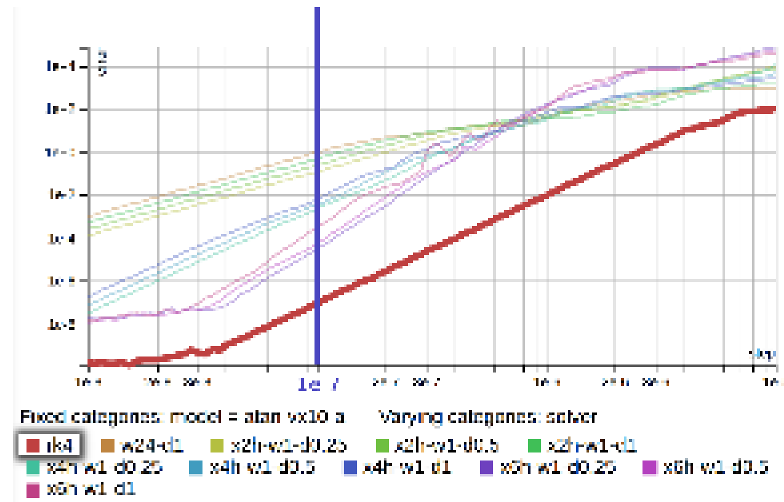


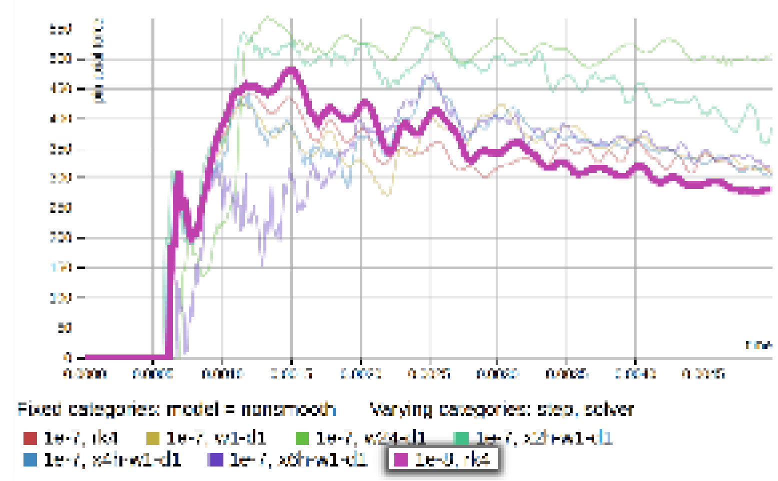
GBS-6h @ 5e-7



Semi-implicit methods

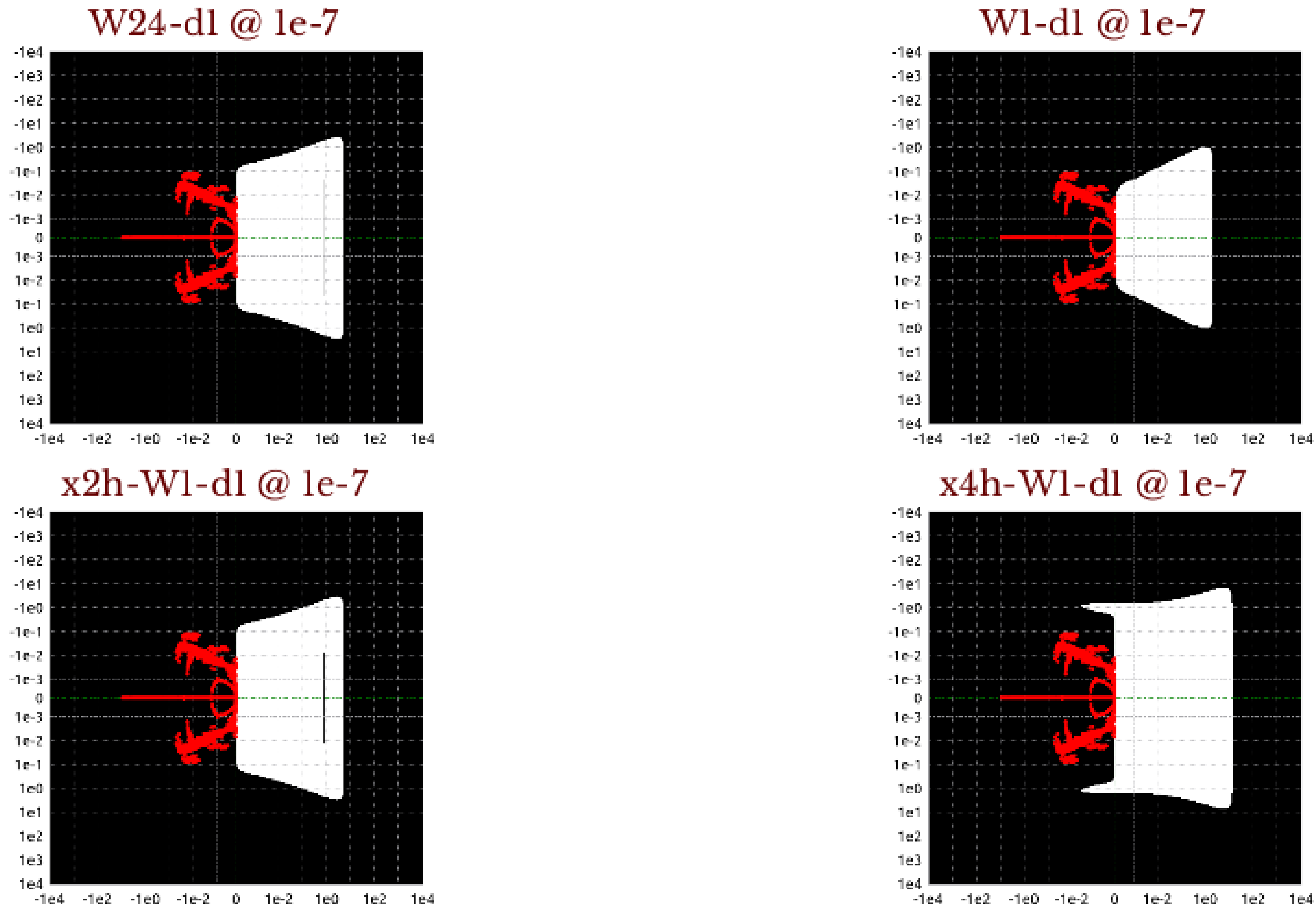
- Rosenbrock
  - Requires ODE RHS Jacobian
  - Jacobian is expensive
  - too slow (?)
- W-methods
  - Reuse Jacobin across steps
  - Could work quite fast
  - Schemes
    - W1, SW2-4, X-SW1
  - Accuracy problems





Semi-implicit methods

Accuracy problems?



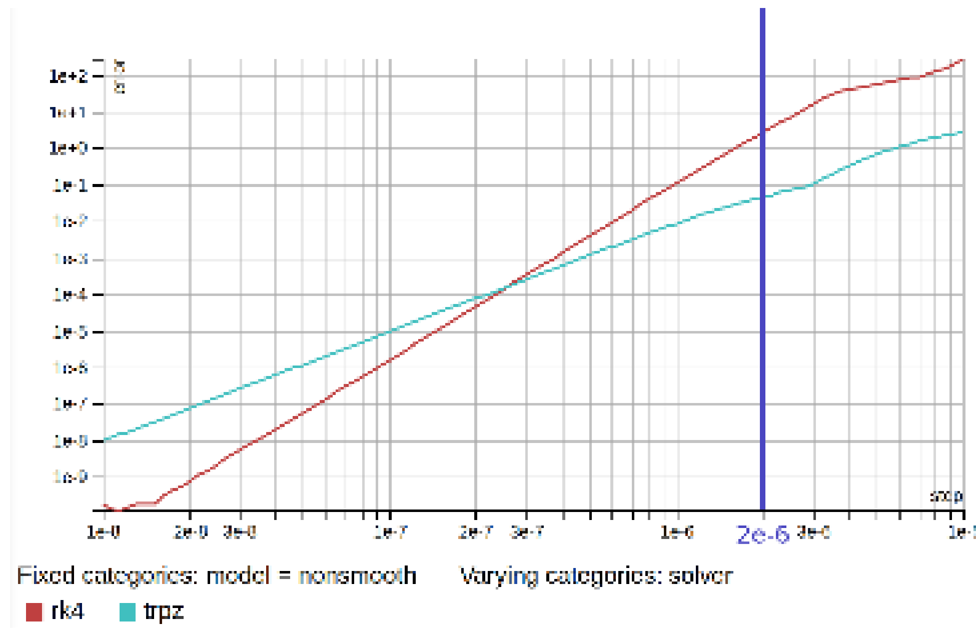
Maybe stability diagrams for W-methods are not representative

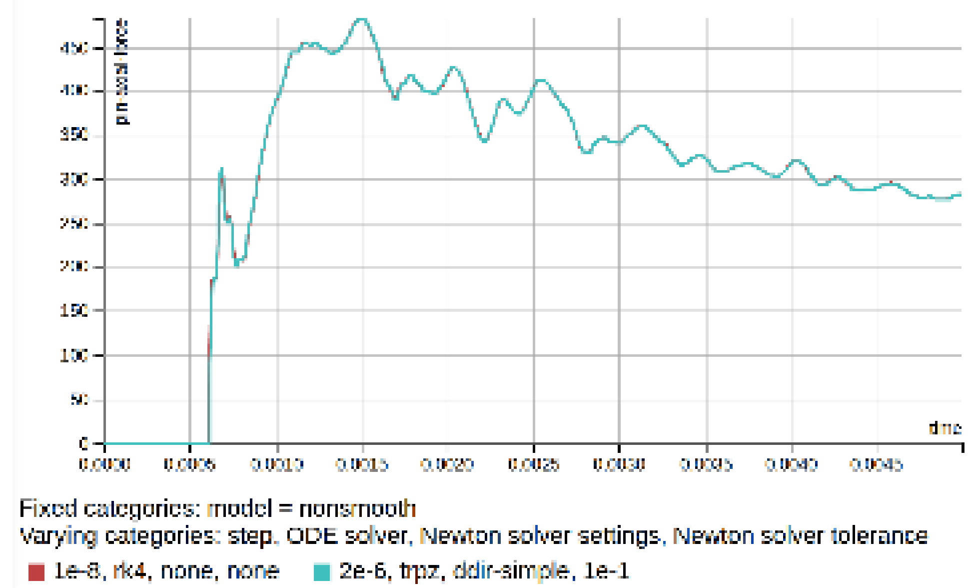
Trapezoidal rule

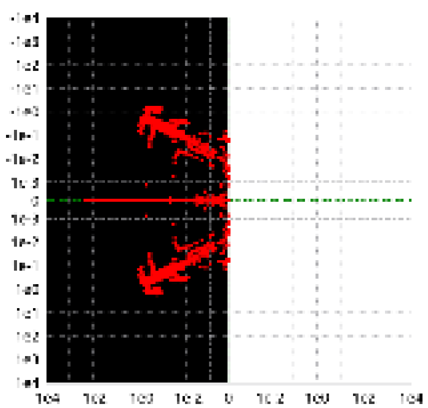
- Excellent results at $h = 2 \cdot 10^{-6}$
- Convergence problems at larger steps
- Lots of things to tweak in nonlinear solver
  - How to compute Jacobian
    - Recompute rarely
    - Update to have superlinear convergence
  - How to do linear search
  - How to predict initial guess
  - How to regularize equation
- Still too slow w/o specialized code for Jacobian

Trapezoidal rule

- Sample curve at $h = 2 \cdot 10^{-6}$ is the same as the "exact" solution (RK4, $h = 2 \cdot 10^{-8}$)
- Potentially, h could be greater, up to 10^{-5}
  - But this requires step size control

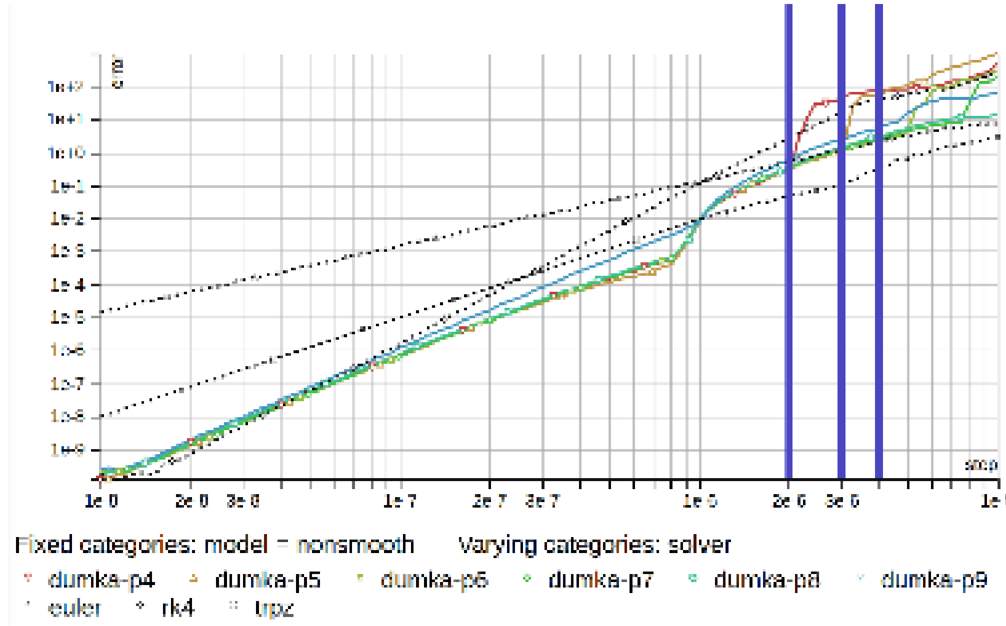


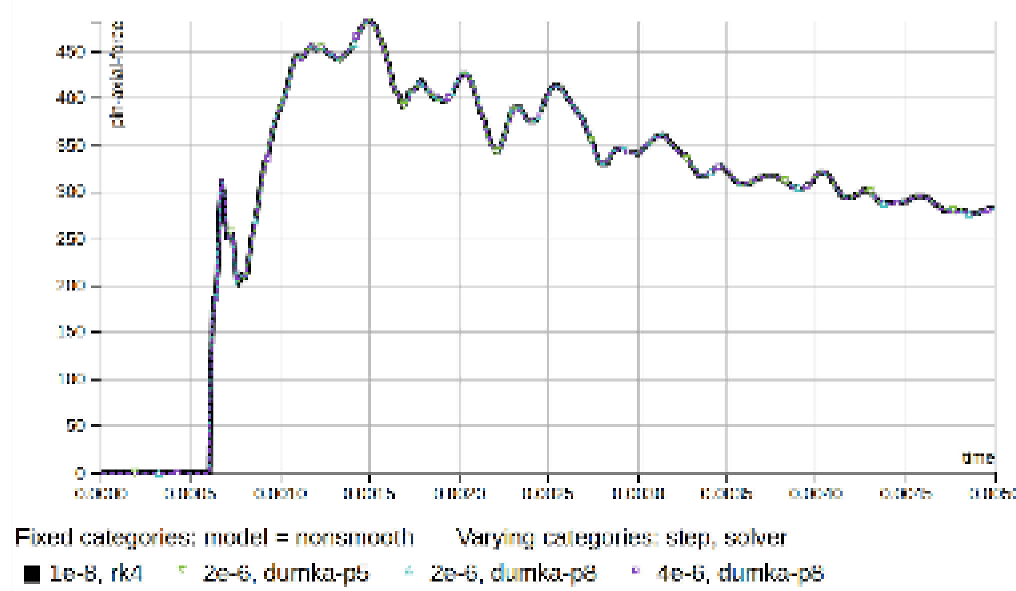




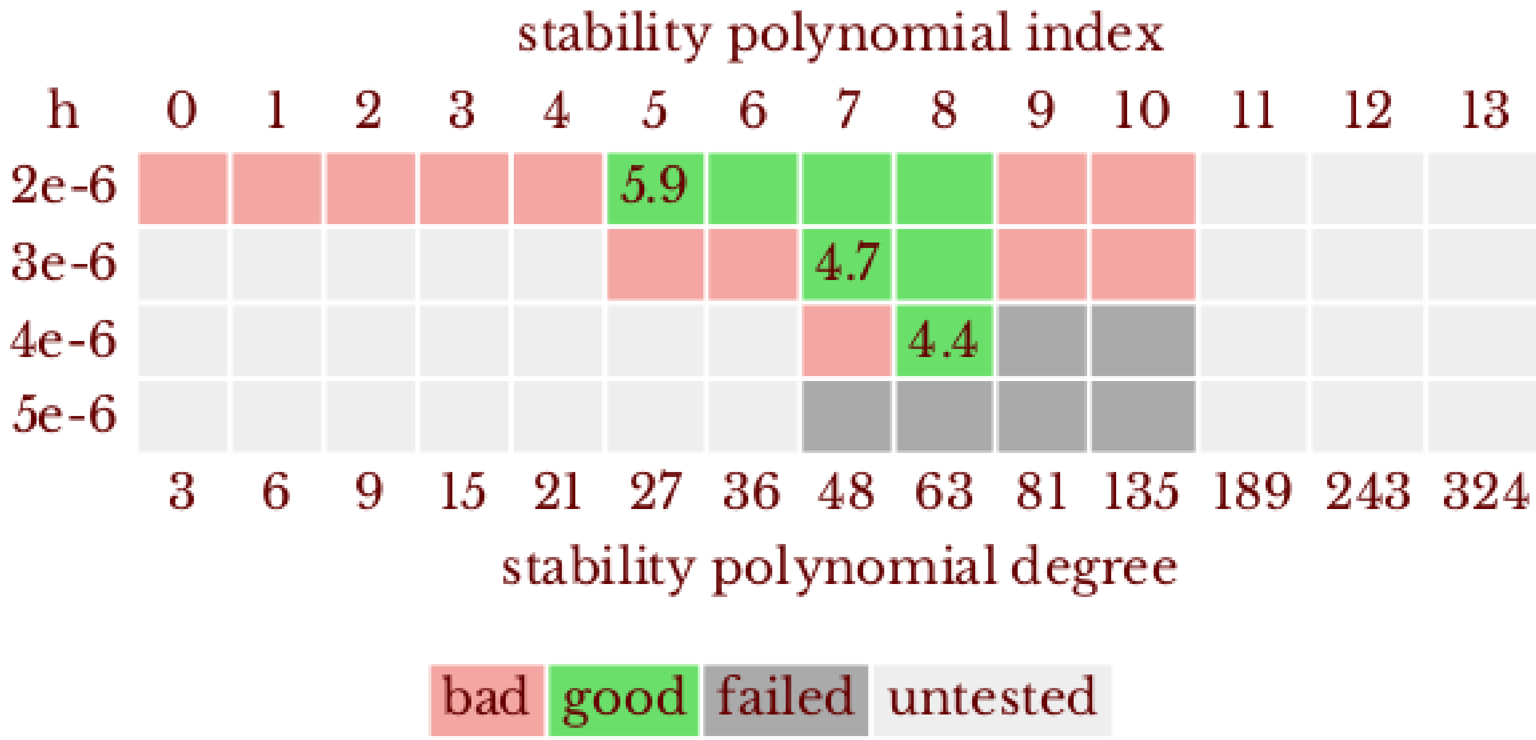
Stabilized explicit RK: DUMKA3

- Excellent results at h up to $4 \cdot 10^{-6}$ (sample curve same as the "exact" solution)
- 5.9x practical speedup (DUMKA-p5 @2e-6 vs RK4 @5e-8)
- Had to disable original step size & polynomial order control
  - Not ready for production

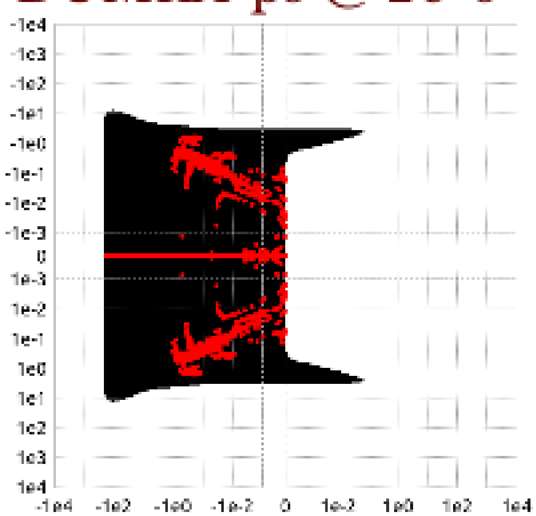




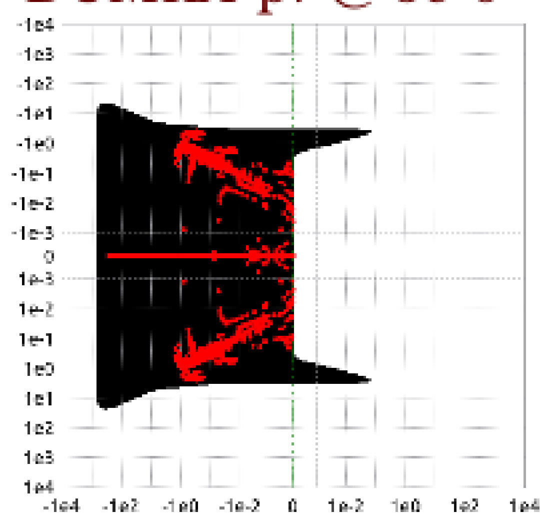
Stabilized explicit RK: DUMKA3



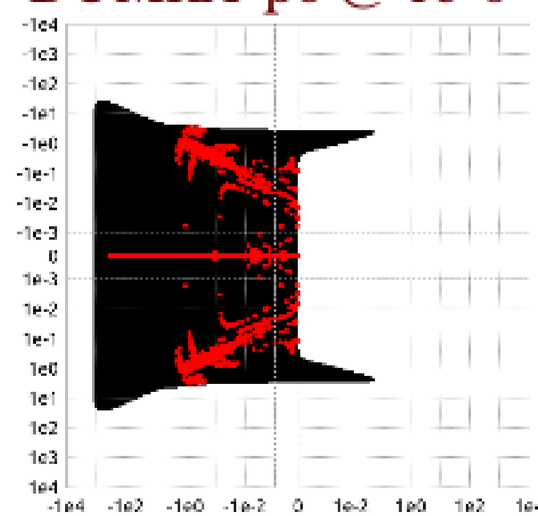
DUMKA-p5 @ 2e-6



DUMKA-p7 @ 3e-6



DUMKA-p8 @ 4e-6



Future work

- Parallelization
  - Optimize & parallelize inertia matrix decomposition
  - Improve scalability of forces calculation
- Numerical integration
  - Maybe try multistep methods
  - Develop code to evaluate ODE RHS Jacobian faster
- Both
  - Parallelize numerical integration algorithms, if possible

Conclusions

- Parallelization
  - Chain forces scale better within one CPU
  - There are more things to do (Amdahl's law is still here)
  - Total speedup 2.8x (6 threads), 3.3x (12 threads)
- Numerical methods
  - Only DUMKA3 is faster @ given accuracy than RK4
  - W-methods didn't work at all :(
  - Implicit will be faster when J is computed faster
  - There are more methods to try
  - Total speedup 5.9x with DUMKA3
- Both
  - ~19x cumulative speedup (estimated)
  - There are things to do

Thank you

Questions?