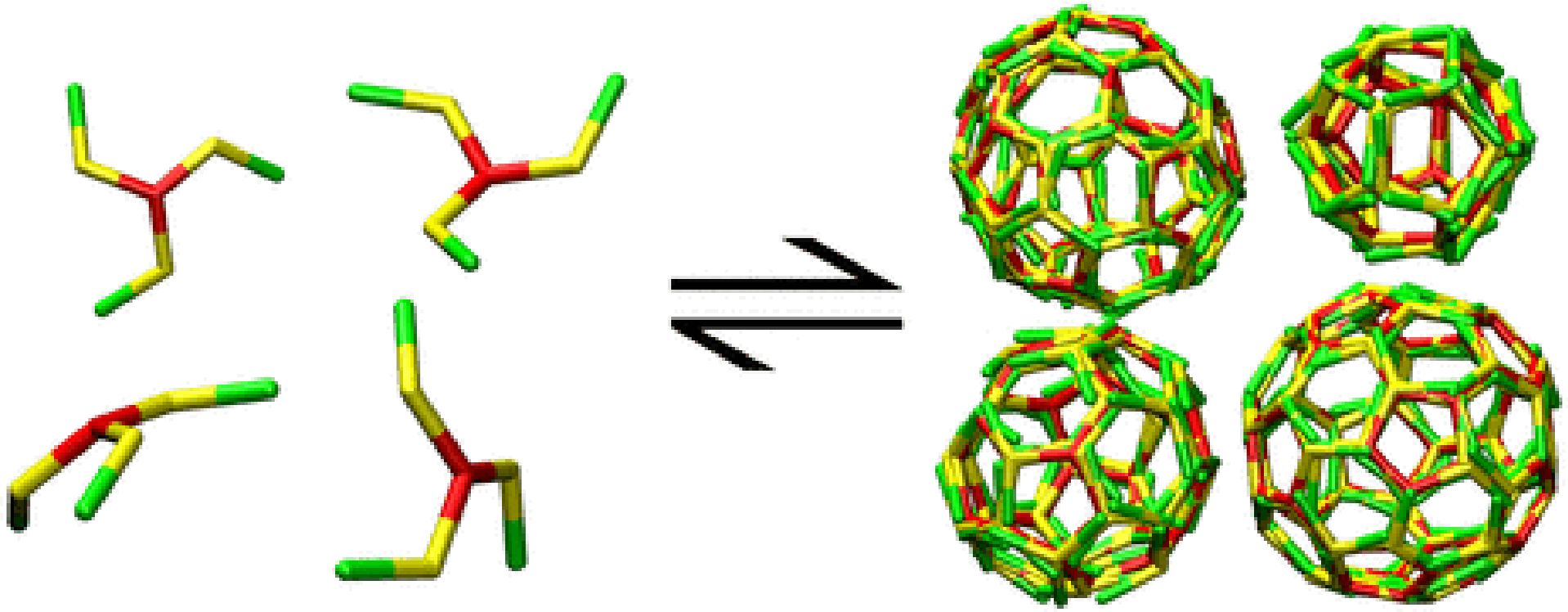
Optimization of Kinetic Self-Assembly Rate Constants

Sho Takeshita

Johnson Lab

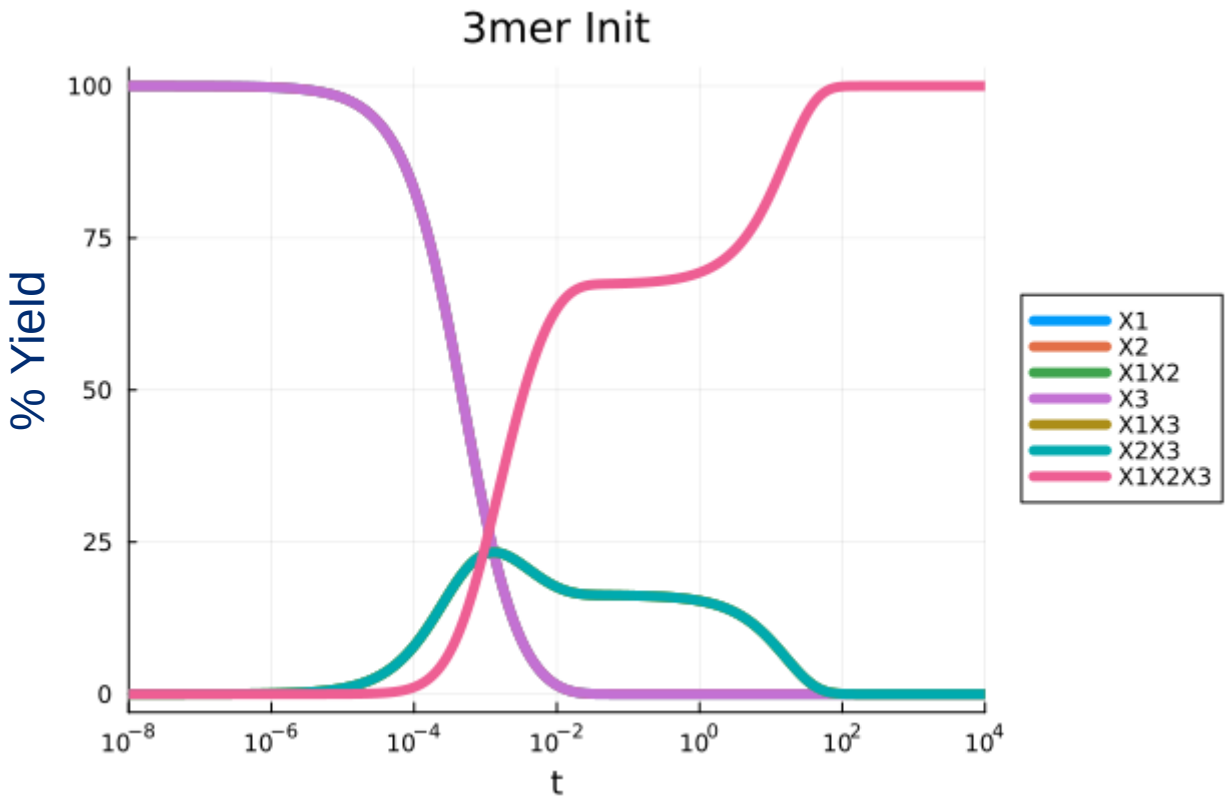
JOHNS HOPKINS UNIVERSITY

Self-assembly is ubiquitous for living systems



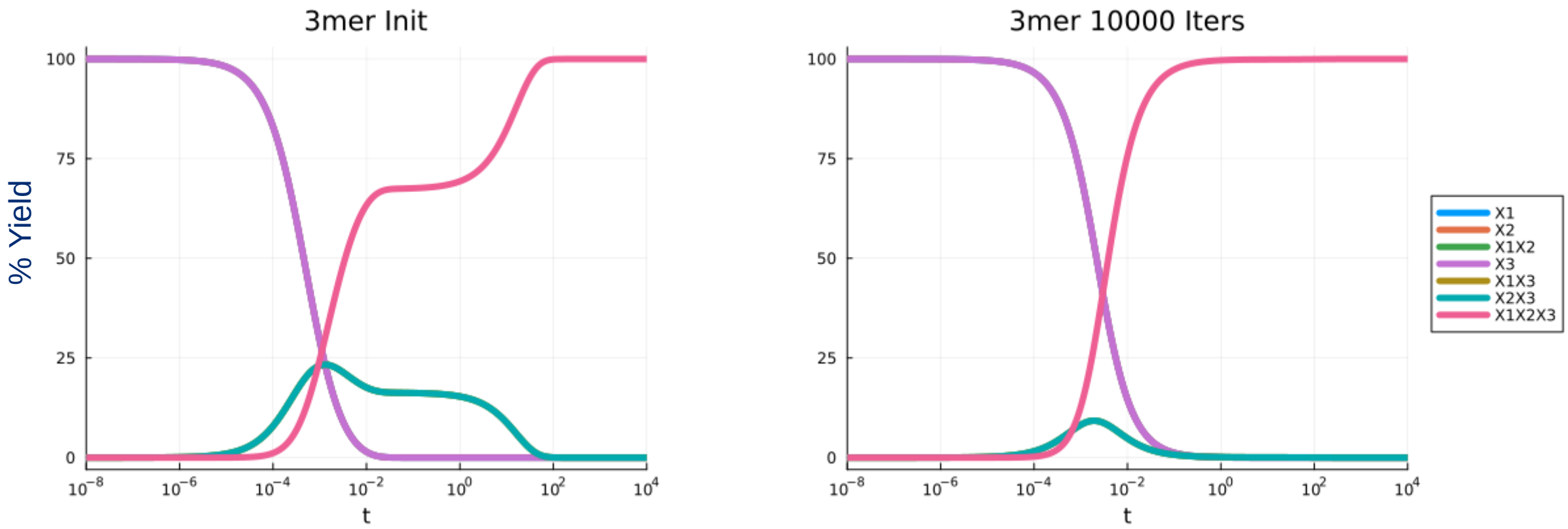
W.J. Briels (2011)

Kinetic Trapping



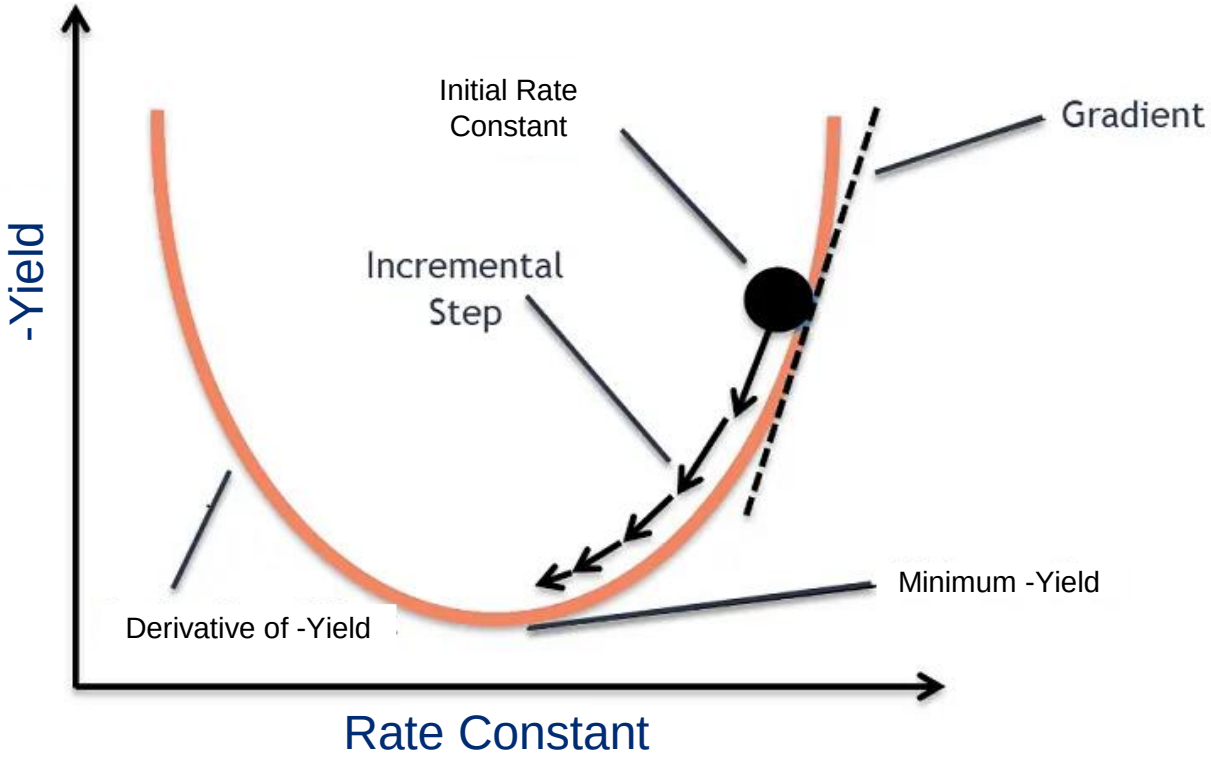
- Experiments and protein design
- Trapping leads to max yield taking too long

Goal is optimization



Calculating gradients is challenging

- Need to calculate gradient w.r.t rates to navigate large parameter space
- There is no explicit function of the yield – can't get analytical gradients
- Automatic differentiation!
  - Uses chain rule
  - Iteratively updates the rate constants until solution obtained

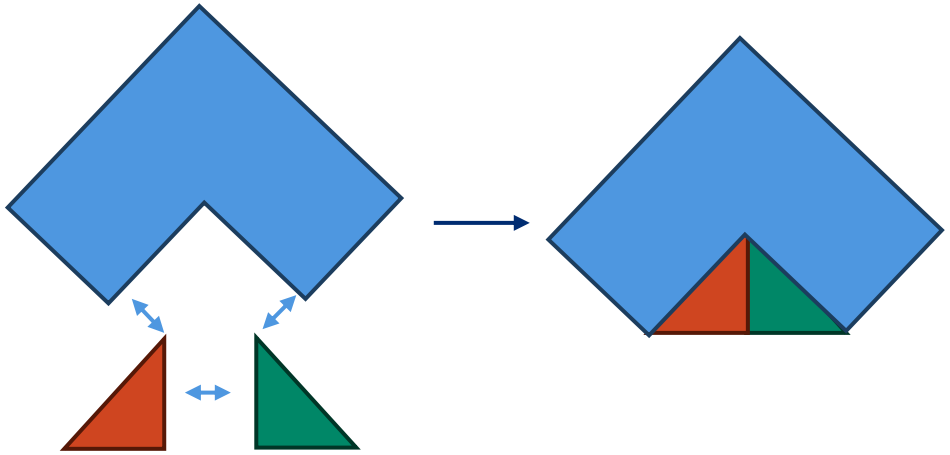


Past Work

- Previously done in Python using AutoDiff, but inefficient for large systems (A. Jhaveri, S. Loggia, Y. Qian, M.E. Johnson (2024))
- Limited to 7mers with 256GB
- Need to study bigger systems – Jhaveri et al. showed it works well, but limited by computational resources

Fully connected hetero-n-mers

Fully Connected



Rate Growth

$$\mathrm{X1} + \mathrm{X2} \underset{\mathrm{k2}}{\overset{\mathrm{k1}}{\rightleftharpoons}} \mathrm{X1X2}$$

$$\mathrm{X1} + \mathrm{X3} \underset{\mathrm{k2}}{\overset{\mathrm{k1}}{\rightleftharpoons}} \mathrm{X1X3}$$

$$\mathrm{X2} + \mathrm{X3} \underset{\mathrm{k2}}{\overset{\mathrm{k1}}{\rightleftharpoons}} \mathrm{X2X3}$$

$$\mathrm{X1} + \mathrm{X2X3} \underset{\mathrm{k4}}{\overset{\mathrm{k3}}{\rightleftharpoons}} \mathrm{X1X2X3}$$

$$\mathrm{X2} + \mathrm{X1X3} \underset{\mathrm{k4}}{\overset{\mathrm{k3}}{\rightleftharpoons}} \mathrm{X1X2X3}$$

$$\mathrm{X3} + \mathrm{X1X2} \underset{\mathrm{k4}}{\overset{\mathrm{k3}}{\rightleftharpoons}} \mathrm{X1X2X3}$$

- Subunits of the same size n share association and dissociation rates

julia

- Designed for performance for scientific applications
- Python is slow and costly in memory
- Has convenient libraries – Catalyst!

How it works

- Give program an nmer and initial rates and concentrations
- Generates reaction network
- Constructs differential equations
- Optimizes using AutoDiff

$$\mathrm{X1} + \mathrm{X2} \underset{\mathrm{k2}}{\overset{\mathrm{k1}}{\rightleftharpoons}} \mathrm{X1X2}$$

$$\mathrm{X1} + \mathrm{X3} \underset{\mathrm{k2}}{\overset{\mathrm{k1}}{\rightleftharpoons}} \mathrm{X1X3}$$

$$\mathrm{X2} + \mathrm{X3} \underset{\mathrm{k2}}{\overset{\mathrm{k1}}{\rightleftharpoons}} \mathrm{X2X3}$$

$$\mathrm{X1} + \mathrm{X2X3} \underset{\mathrm{k4}}{\overset{\mathrm{k3}}{\rightleftharpoons}} \mathrm{X1X2X3}$$

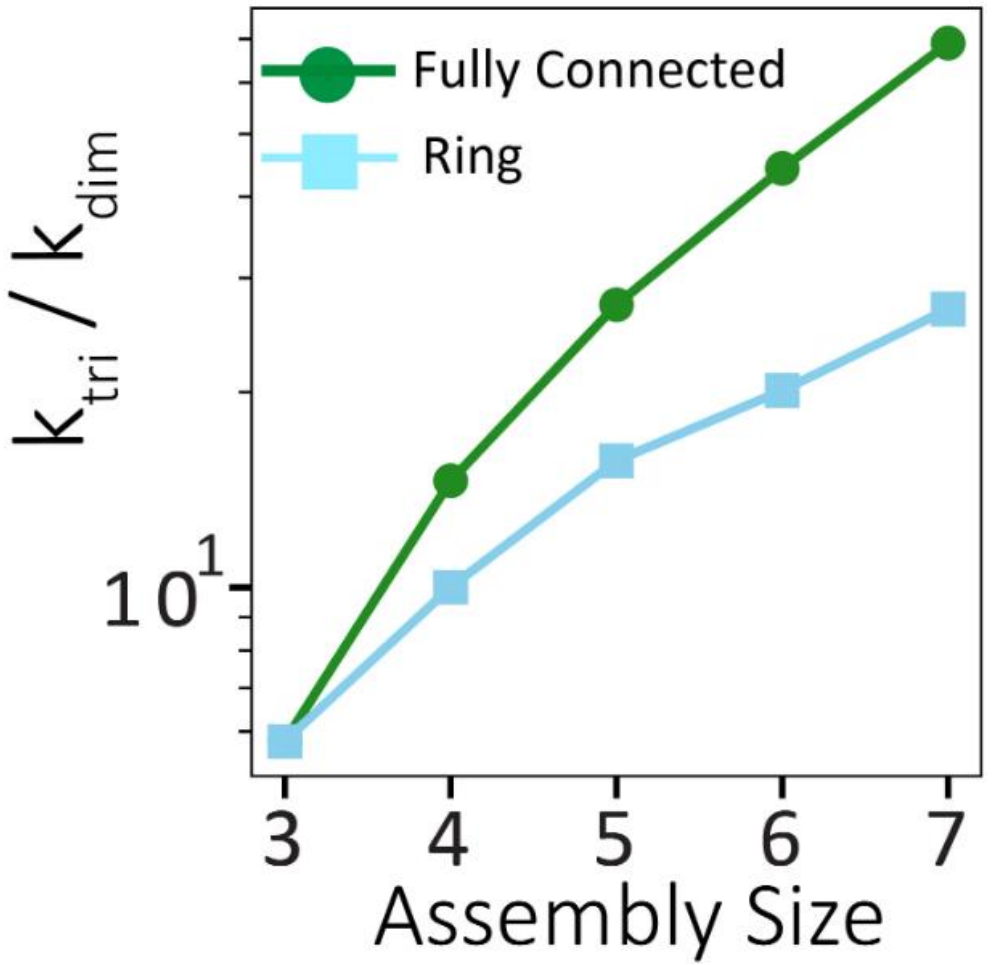
$$\mathrm{X2} + \mathrm{X1X3} \underset{\mathrm{k4}}{\overset{\mathrm{k3}}{\rightleftharpoons}} \mathrm{X1X2X3}$$

$$\mathrm{X3} + \mathrm{X1X2} \underset{\mathrm{k4}}{\overset{\mathrm{k3}}{\rightleftharpoons}} \mathrm{X1X2X3}$$

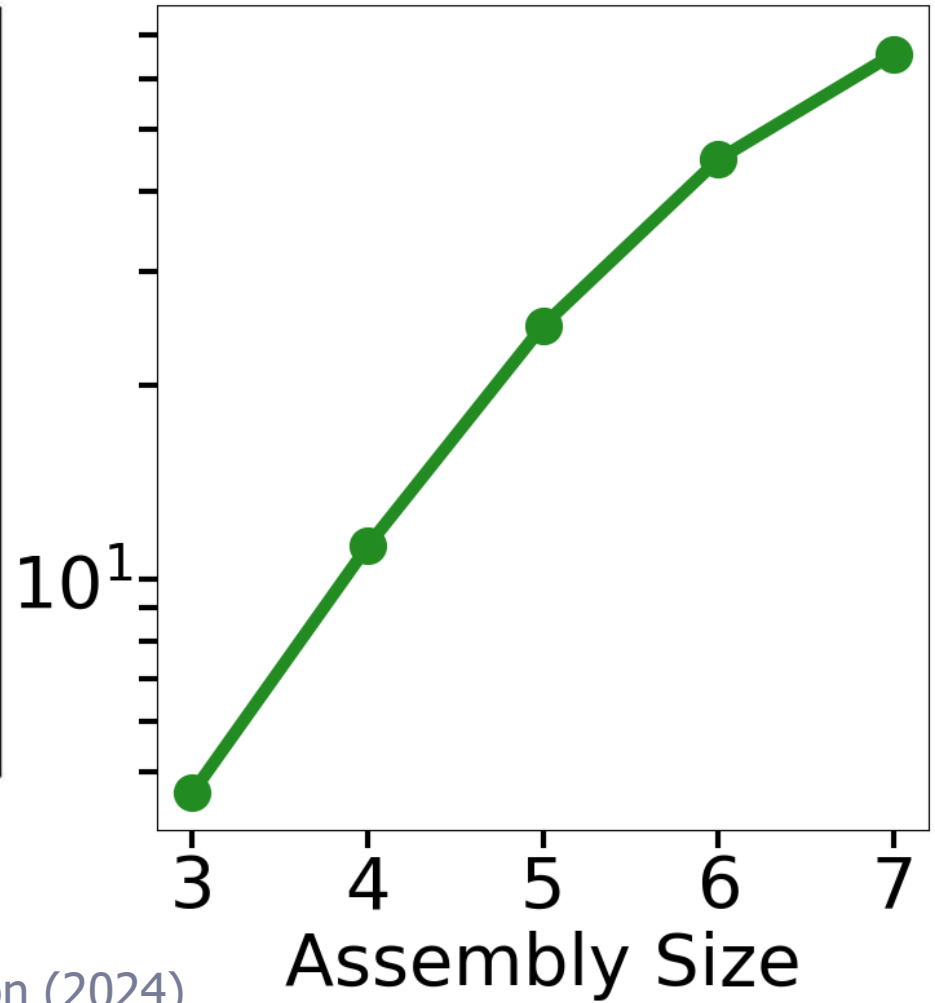
Solution Similarity

Optimal ratio over assembly size

Python

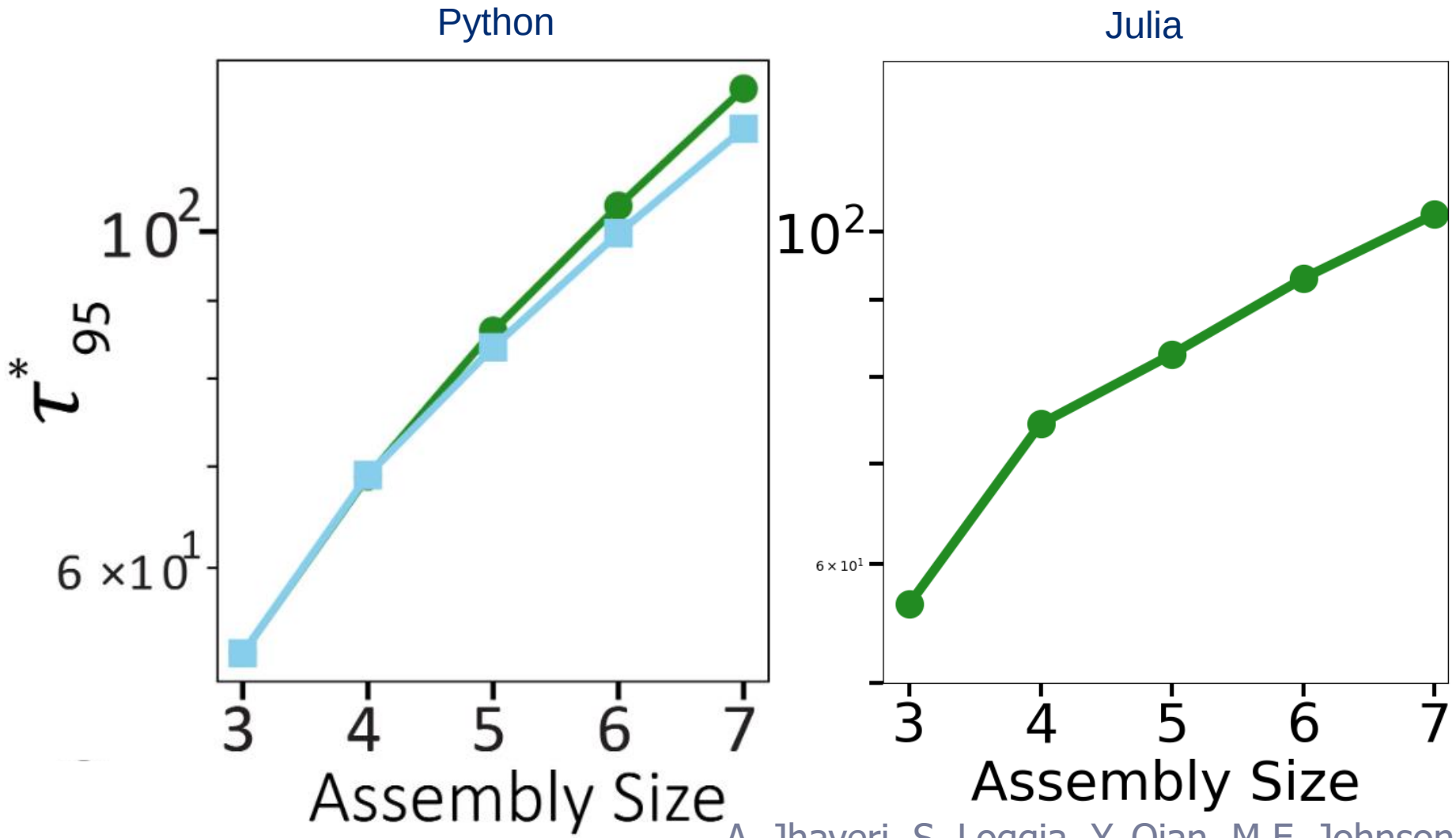


Julia



A. Jhaveri, S. Loggia, Y. Qian, M.E. Johnson (2024)

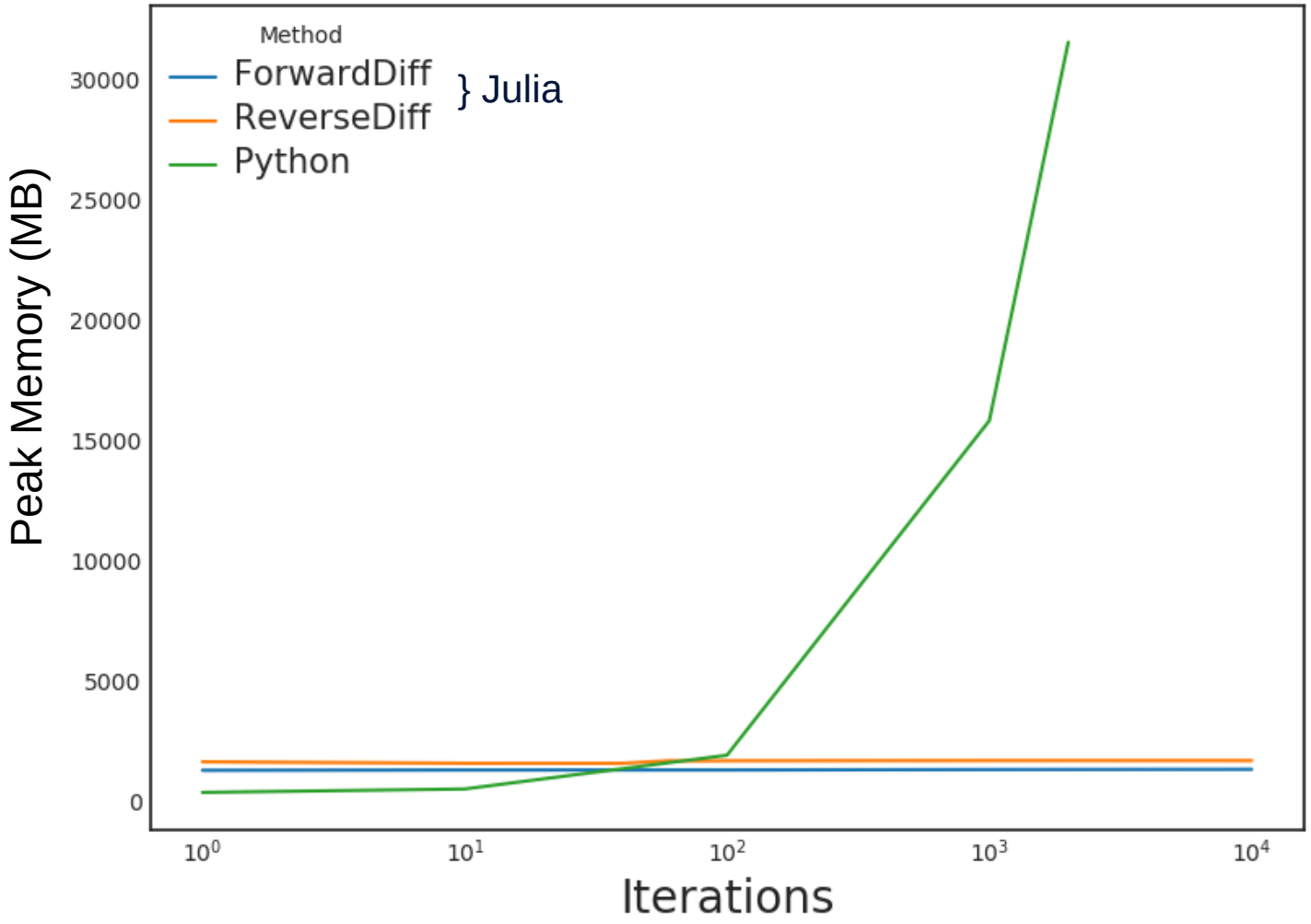
Time to reach 95% yield over assembly size



A. Jhaveri, S. Loggia, Y. Qian, M.E. Johnson (2024)

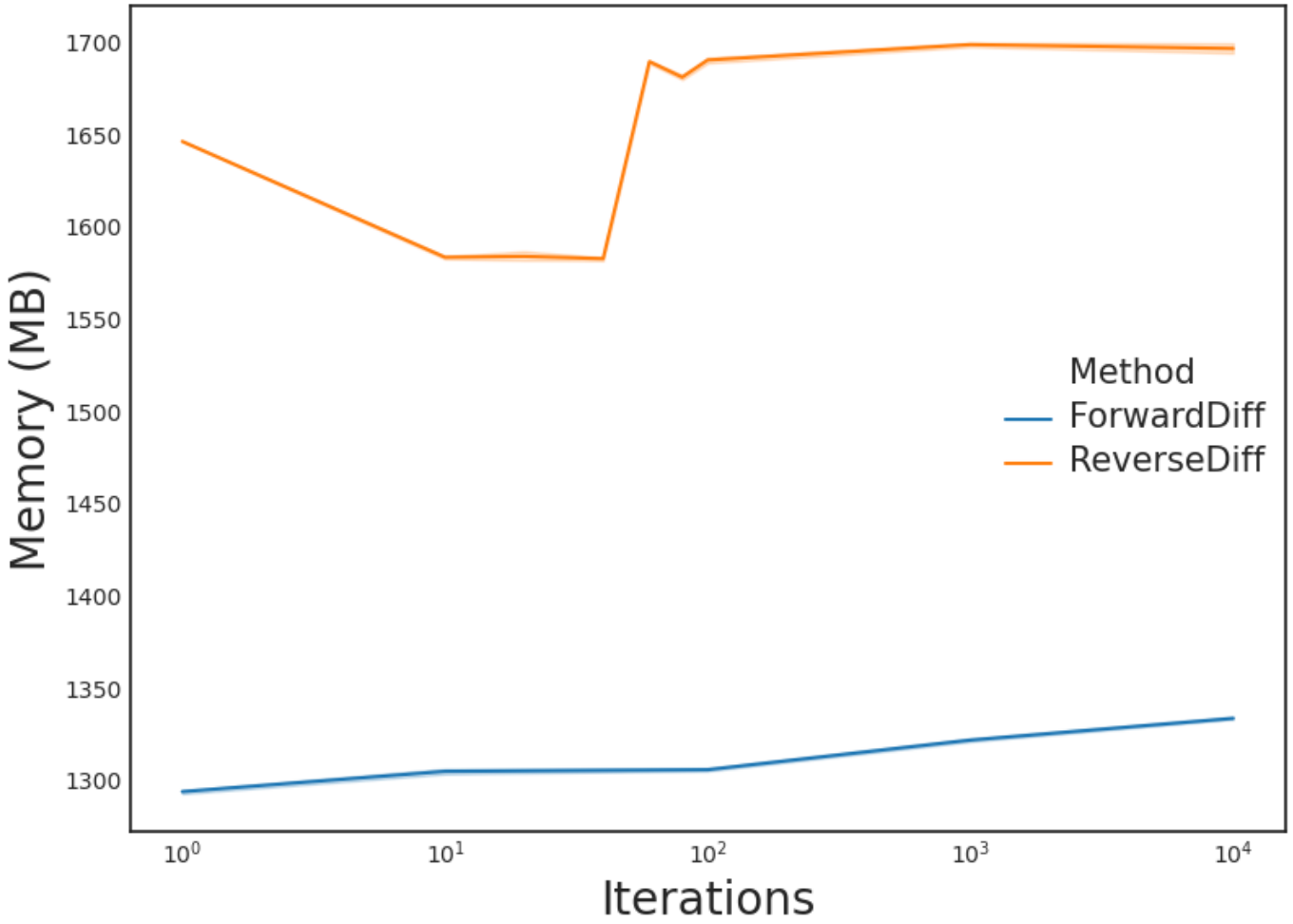
Performance

Memory Performance Over Iterations

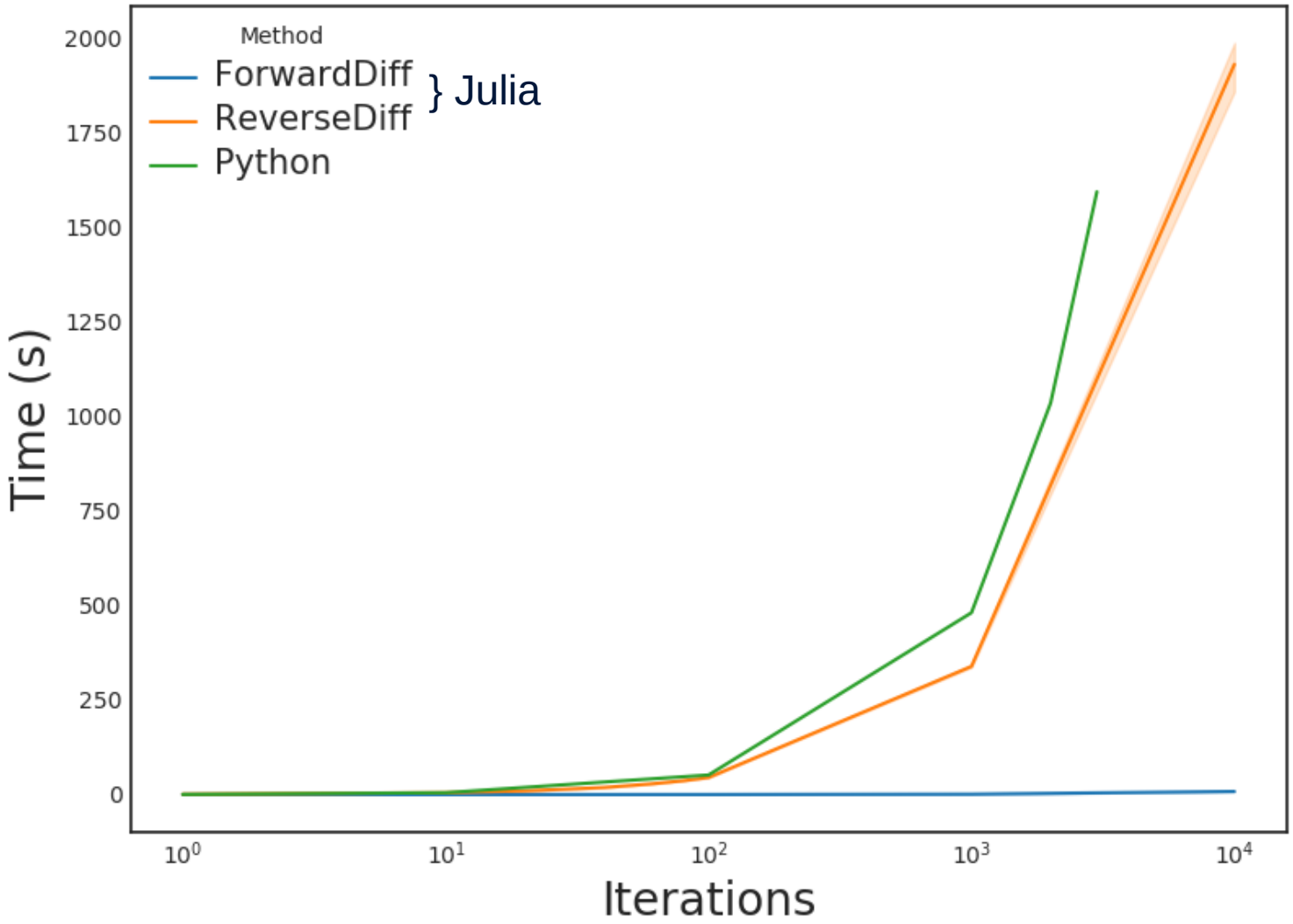


Memory Performance Over Iterations

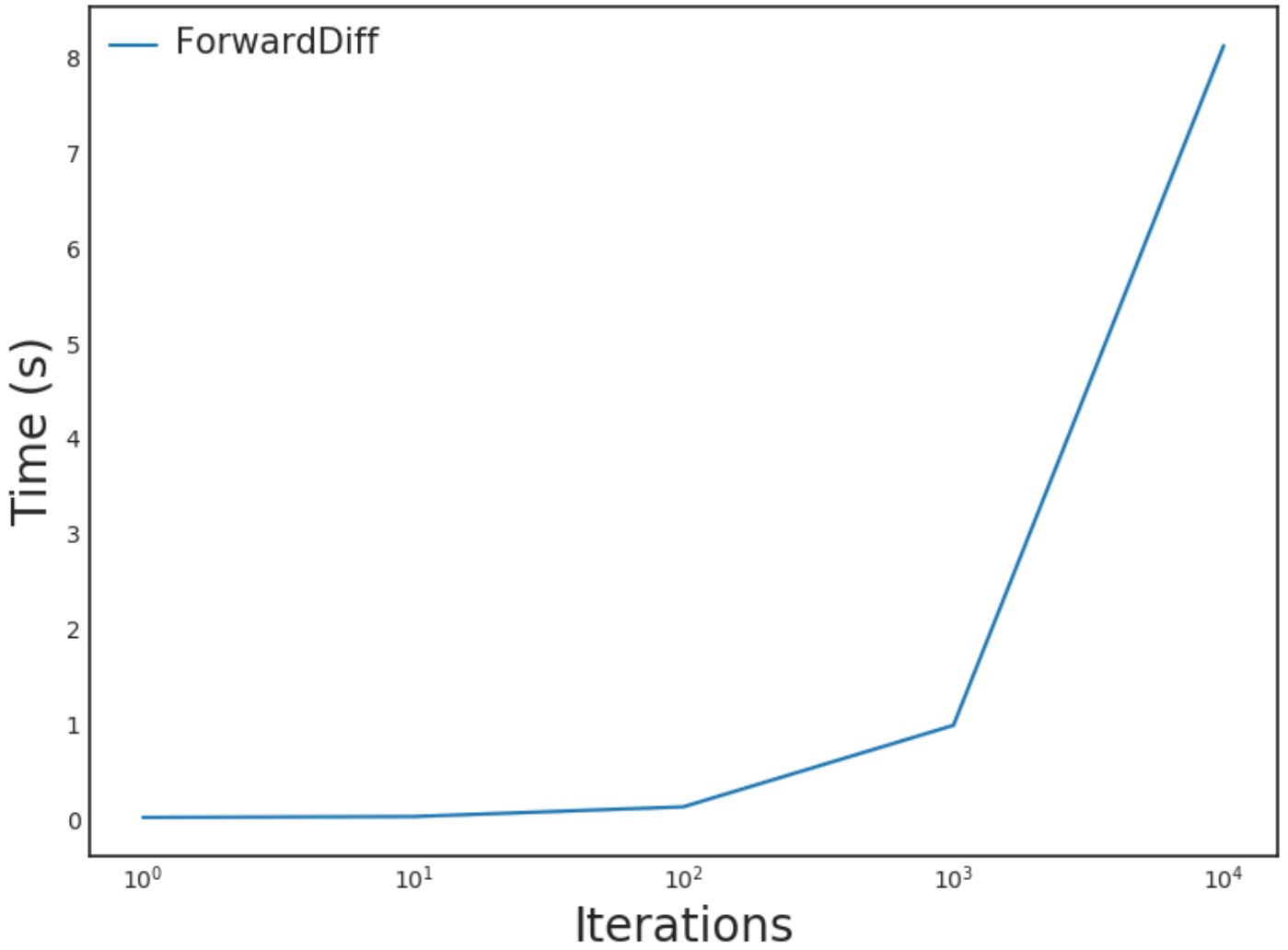
Julia Only



Time Performance Over Iterations

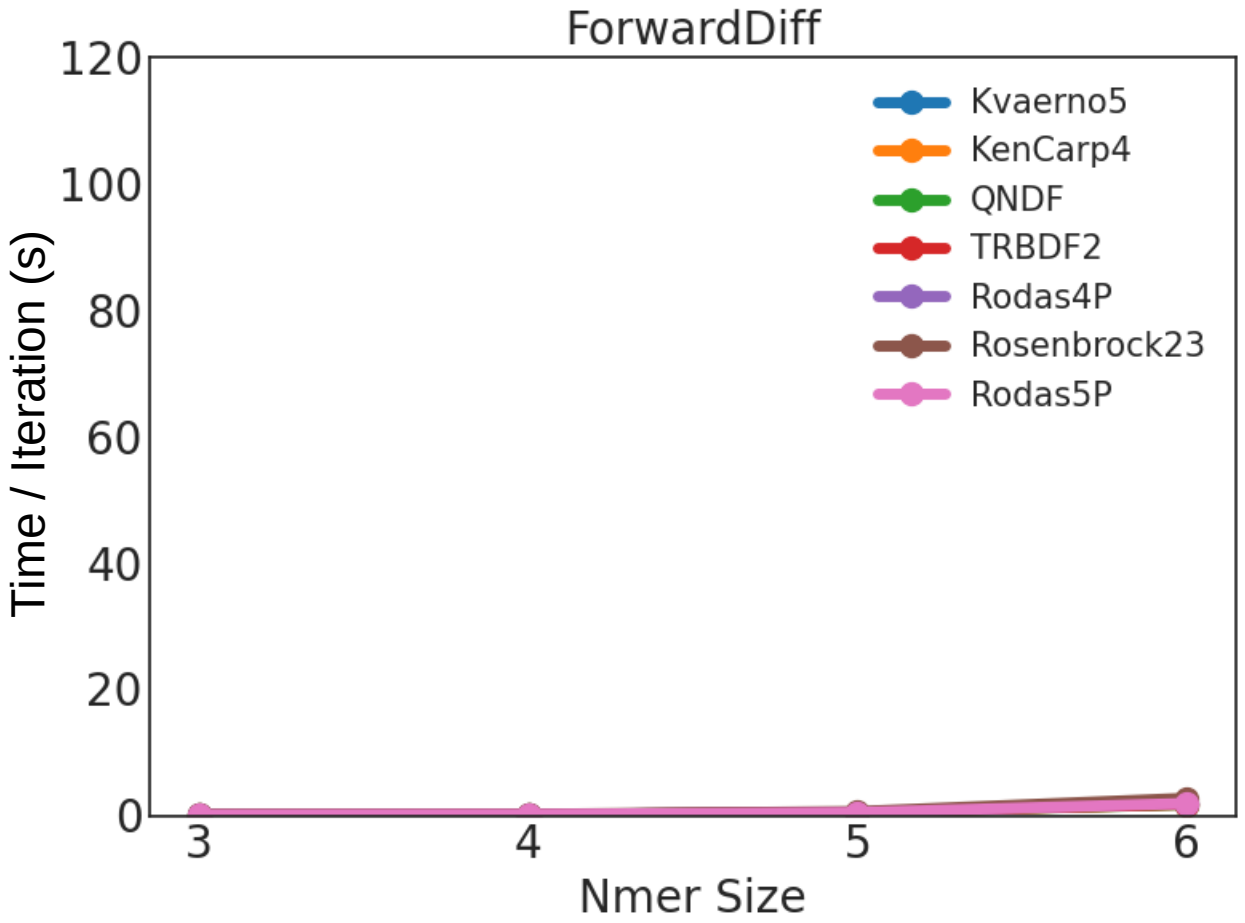


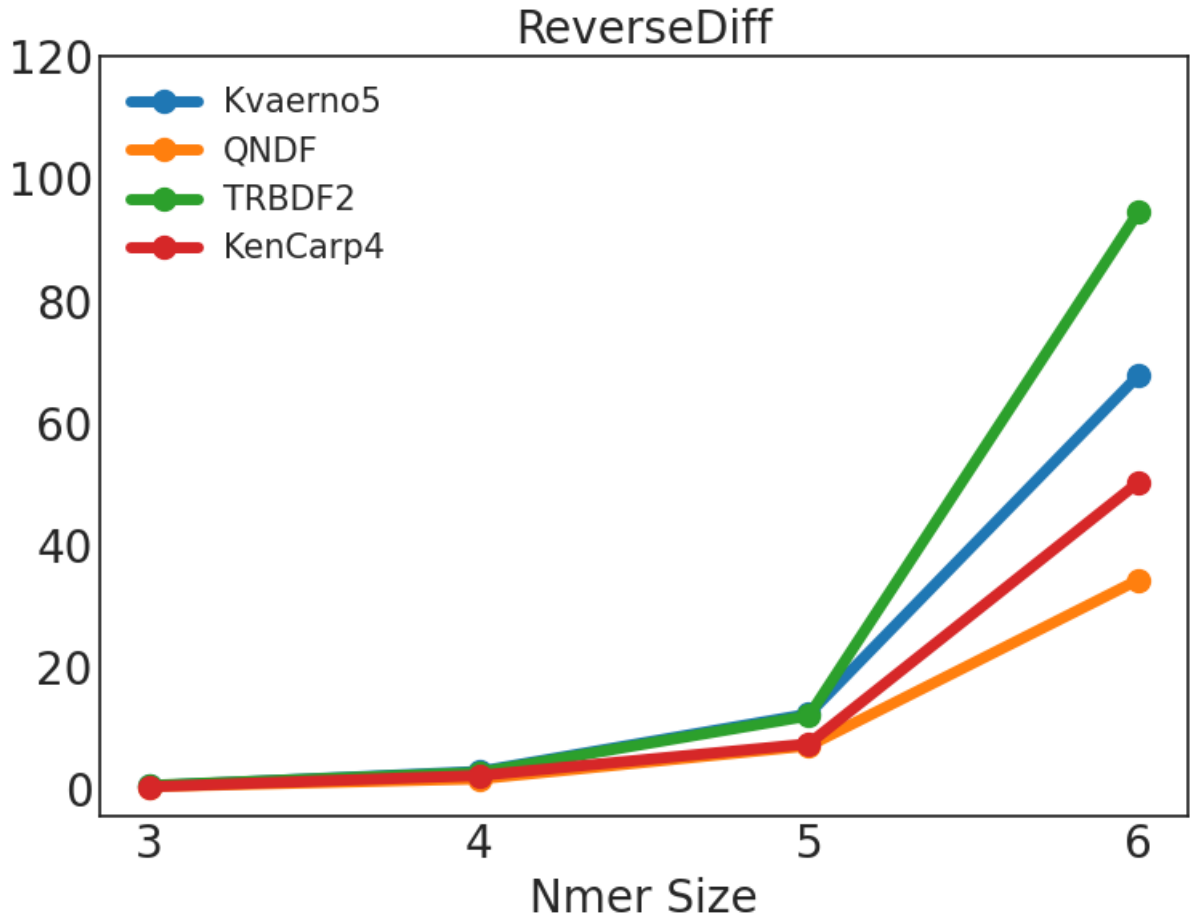
Time Performance Over Iterations



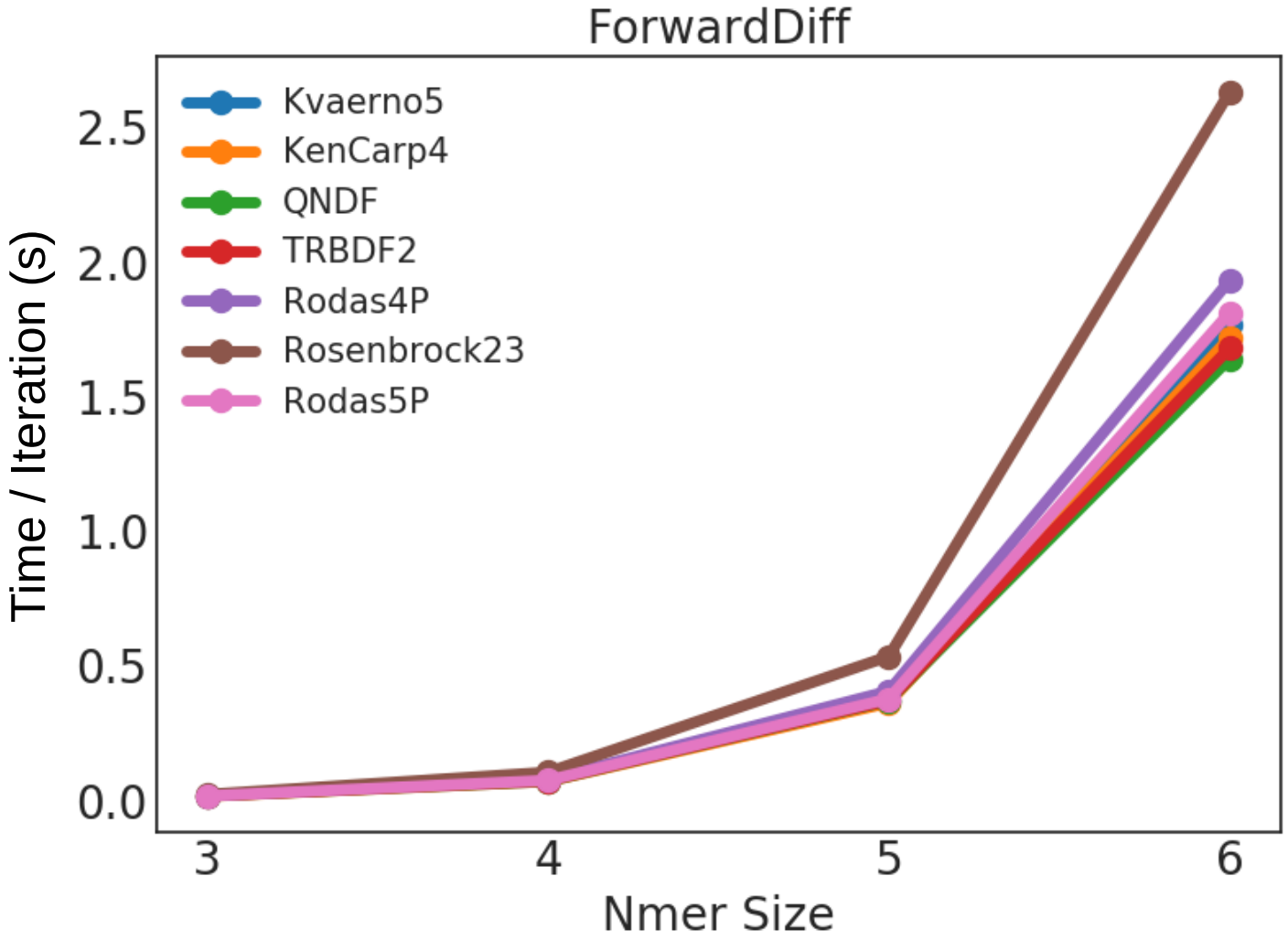
Integrator Performance

Time Performance of Integrators

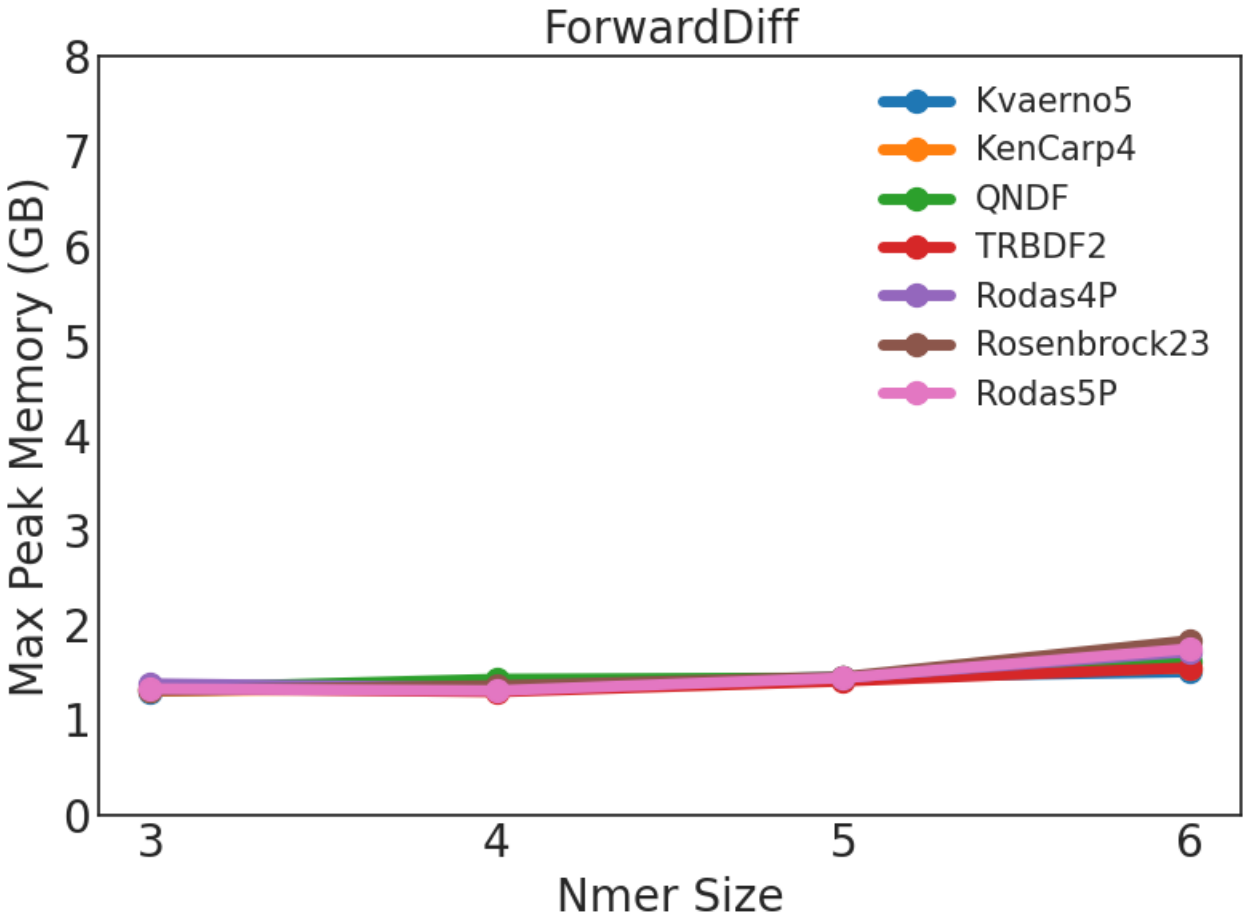


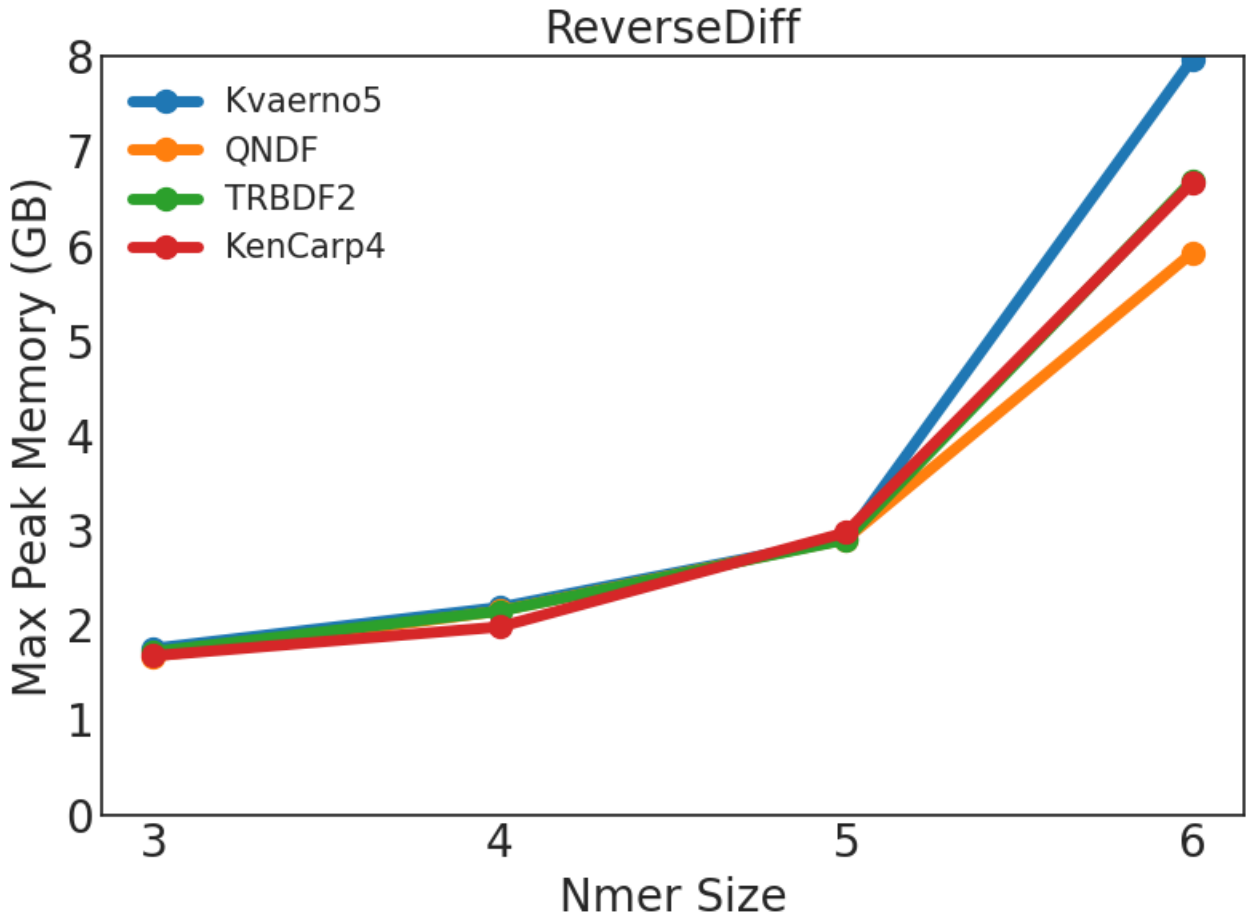


Time Performance of Integrators

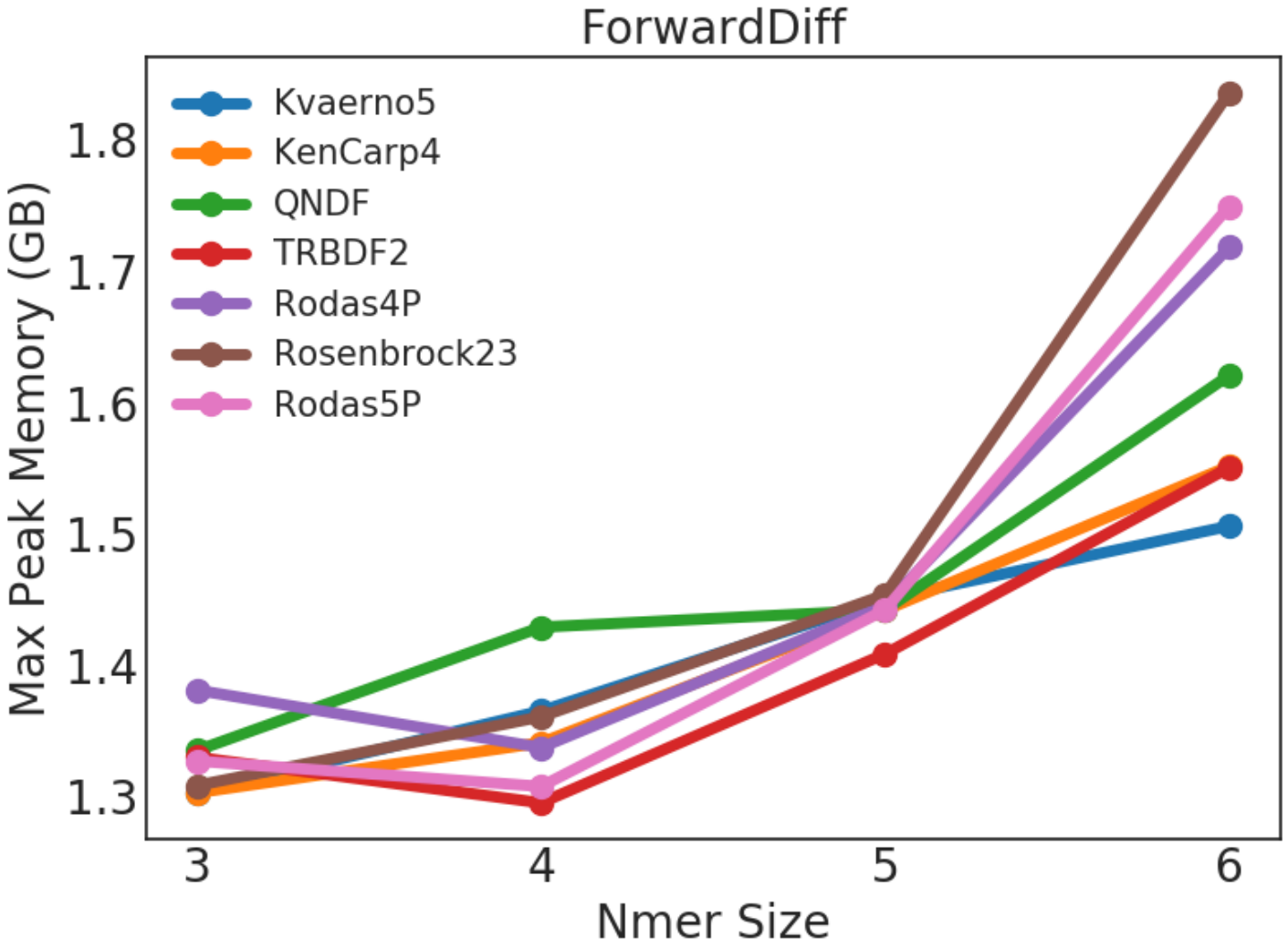


Memory Performance of Integrators





Memory Performance of Integrators



Summary

- Solutions are reasonable and similar to the Python version
- Julia performs better than Python
- ForwardDiff is more efficient than ReverseDiff
- QNDF is optimal for ReverseDiff
- More testing for ForwardDiff integrators

Acknowledgements

Johnson Lab

Dr. Margaret Johnson

Adip Jhaveri

Dr. Sam Foley

Jonathan Fischer

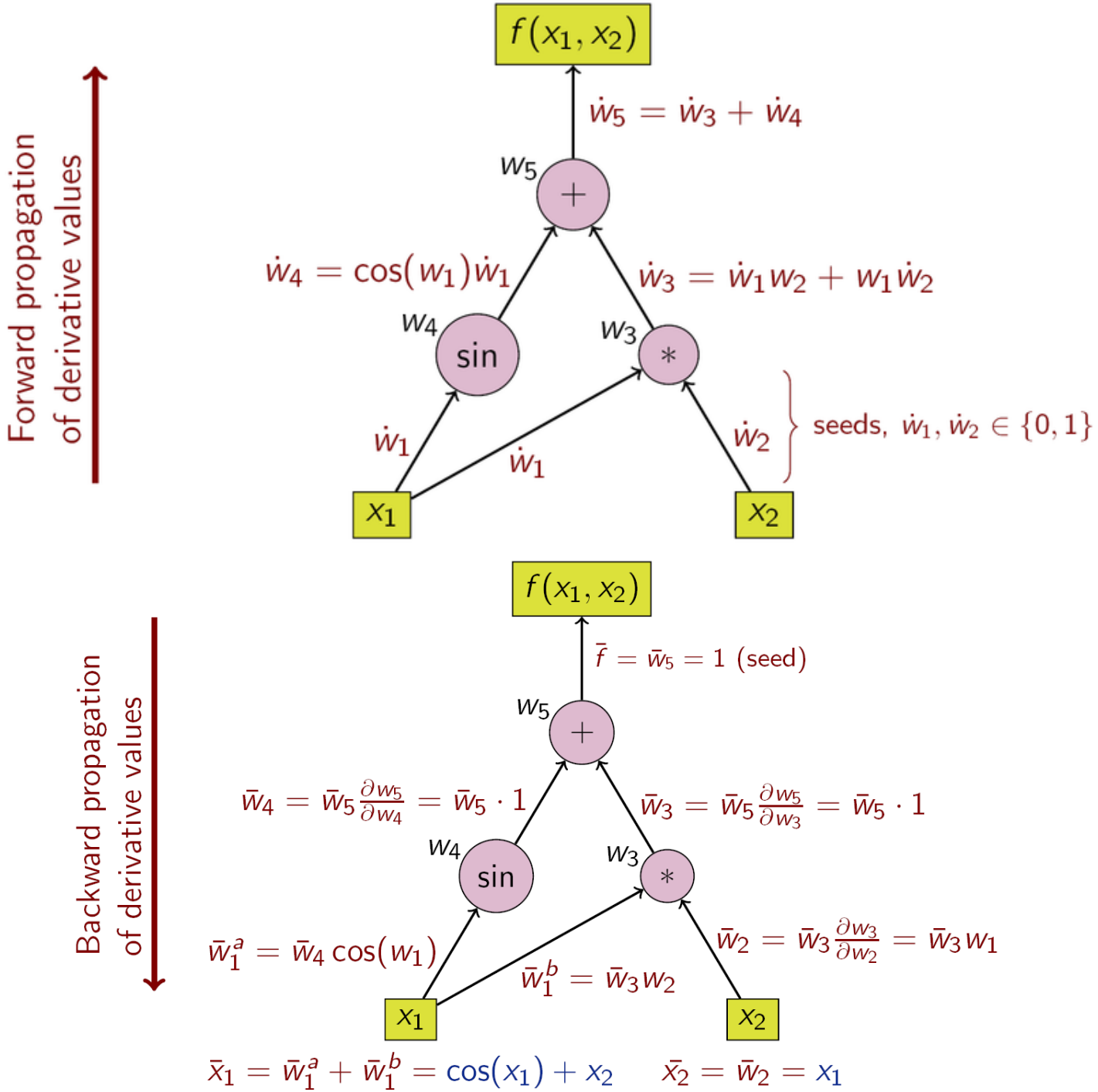
Dr. Sikao Guo

Mankun Sang

Moon Ying

Smriti Chhibber

Questions


Forward propagation of derivative values
$f(x_1, x_2)$
$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$
w_5
+
$\dot{w}_4 = \cos(w_1)\dot{w}_1$
$\dot{w}_3 = \dot{w}_1 w_2 + w_1 \dot{w}_2$
w_4
sin
w_3
*
$\dot{w}_1$
$\dot{w}_1$
$\dot{w}_2$
seeds, $\dot{w}_1, \dot{w}_2 \in \{0, 1\}$
x_1
x_2
Backward propagation of derivative values
$f(x_1, x_2)$
$\bar{f} = \bar{w}_5 = 1$ (seed)
w_5
+
$\bar{w}_4 = \bar{w}_5 \frac{\partial w_5}{\partial w_4} = \bar{w}_5 \cdot 1$
$\bar{w}_3 = \bar{w}_5 \frac{\partial w_5}{\partial w_3} = \bar{w}_5 \cdot 1$
w_4
sin
w_3
*
$\bar{w}_1^a = \bar{w}_4 \cos(w_1)$
$\bar{w}_1^b = \bar{w}_3 w_2$
$\bar{w}_2 = \bar{w}_3 \frac{\partial w_3}{\partial w_2} = \bar{w}_3 w_1$
x_1
x_2
$\bar{x}_1 = \bar{w}_1^a + \bar{w}_1^b = \cos(x_1) + x_2$
$\bar{x}_2 = \bar{w}_2 = x_1$