Checkpoint Reverse Automatic Differentiation

(plus some extra stuff I hope you're interested in)

Barak A. Pearlmutter <barak@pearlmutter.net>
Maynooth University

Jeff Siskind (Purdue)

(Long-term project: PLT for AD)

$\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$

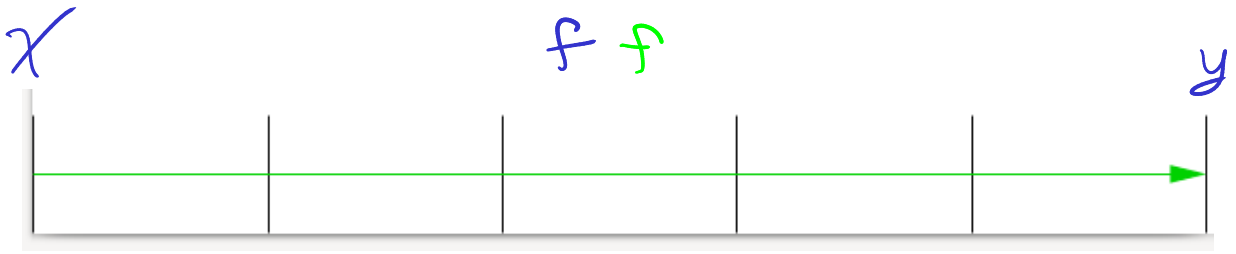
~~Prog → Diff Prog~~ (AD)

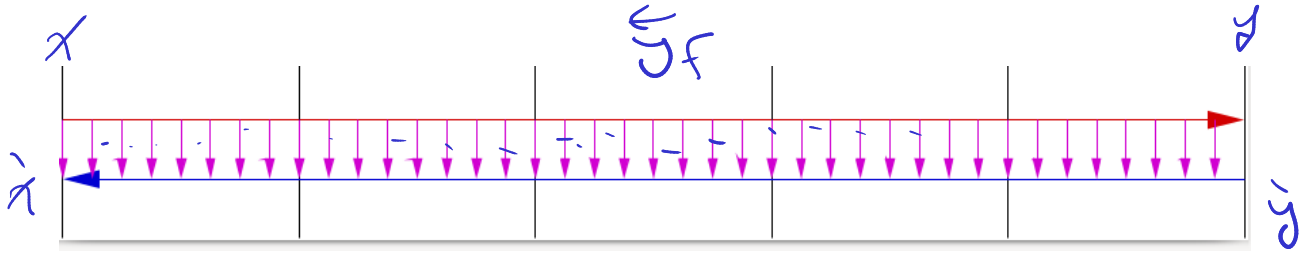
PL ∪ {$\overrightarrow{\mathcal{J}}$, $\overleftarrow{\mathcal{J}}$} = PL′

$\overrightarrow{\mathcal{J}} : (\mathbb{R}^n \to \mathbb{R}^m) \to ((\mathbb{R}^n \times \mathbb{R}^n) \to (\mathbb{R}^m \times \mathbb{R}^m))$

$\overleftarrow{\mathcal{J}} : (\mathbb{R}^n \xrightarrow{f} \mathbb{R}^m) \to \mathbb{R}^n \to (\mathbb{R}^m \times (\mathbb{R}^m \to \mathbb{R}^n))$

$\overleftarrow{\mathcal{J}}$ **Treeverse**





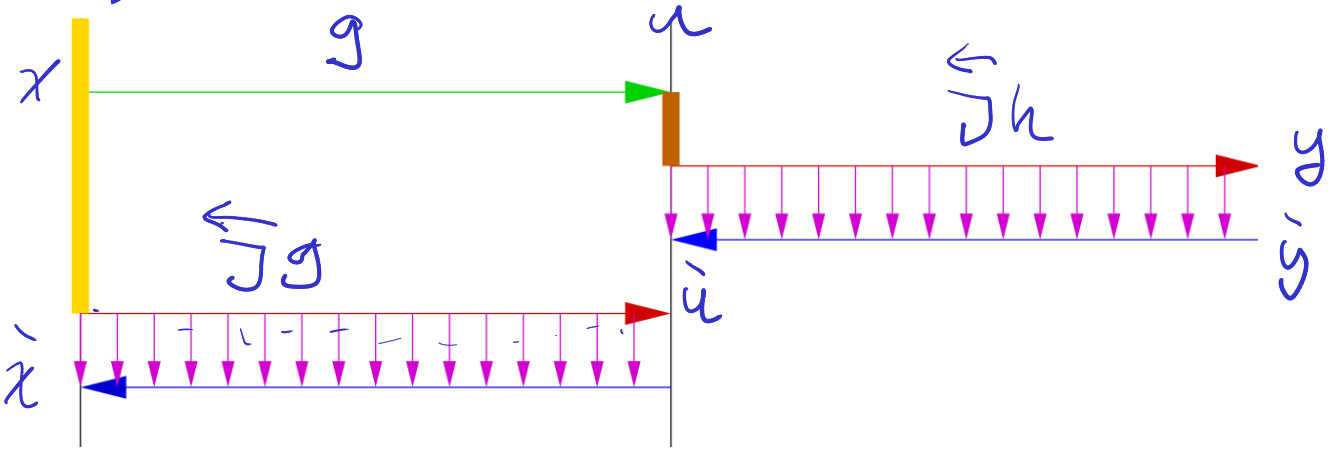
time overhead: const

space overhead: linear in time

let $(x, y) = z$

let $h \circ g = f$

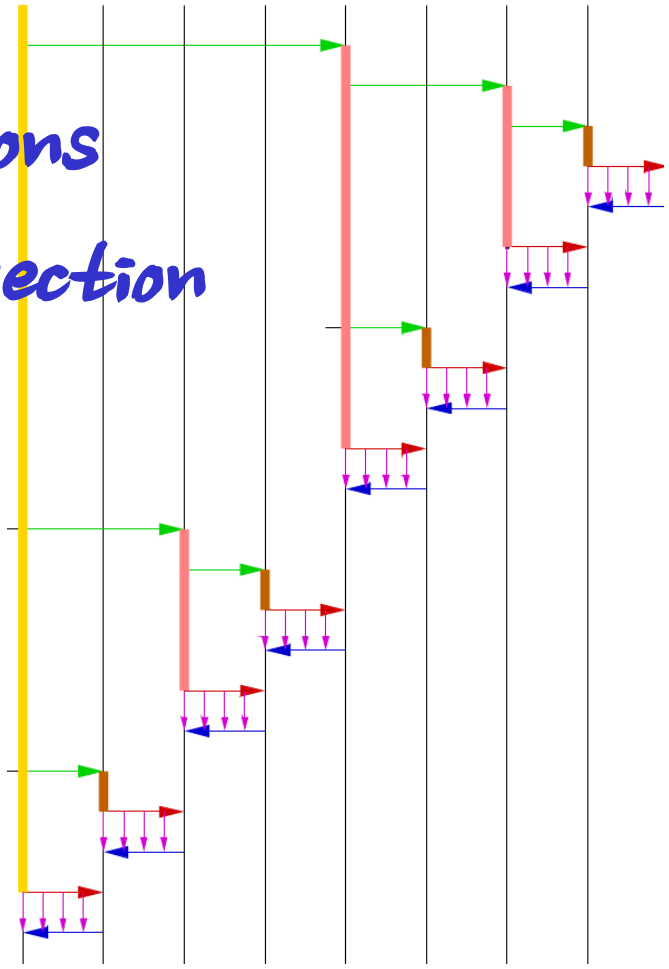
diagramatic representation



composition of 8 functions

checkpoint schedule: bisection

overhead: logarithmic
(time & space)



Implementation:

- old - program is a nice "for loop"
- Tapenade - checkpoint at calls
- here - PLT to decompose

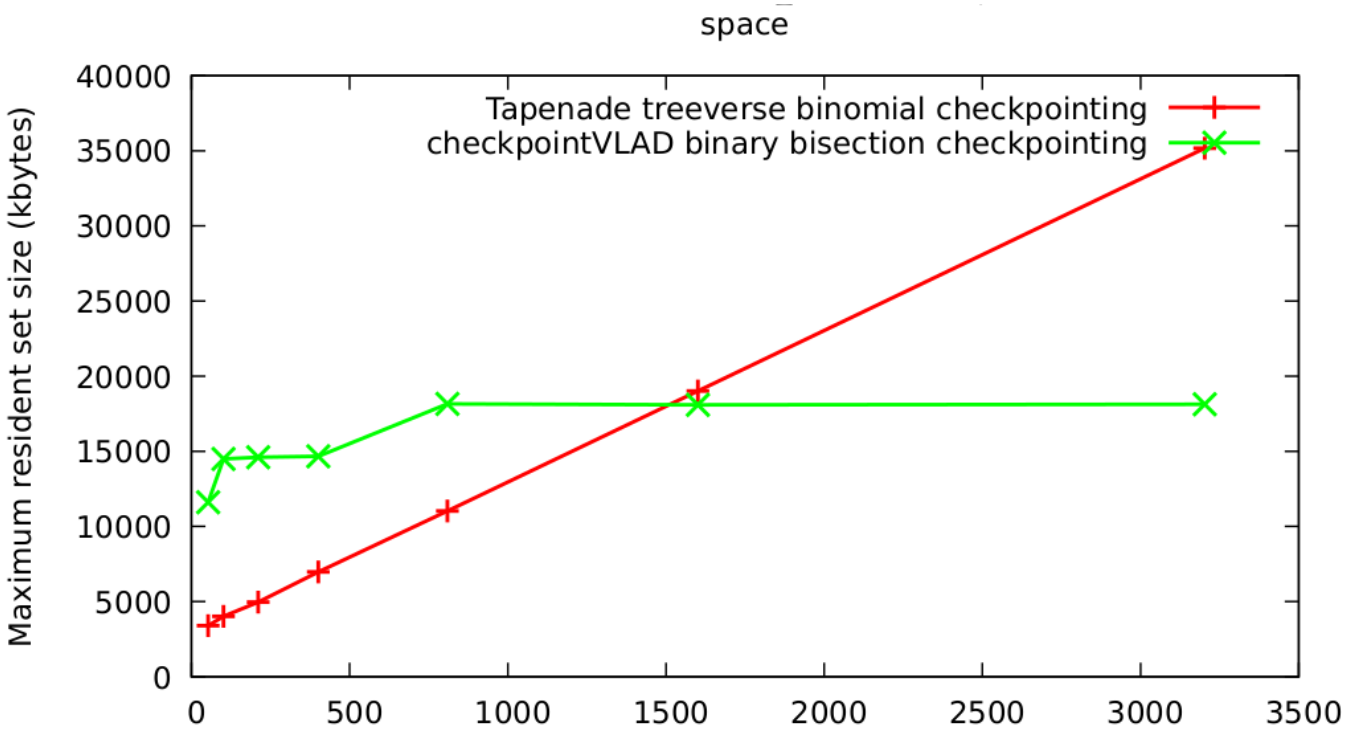
To compute $(y, \dot{x}) = \overrightarrow{\mathcal{J}}\ f\ x\ \dot{y}$:

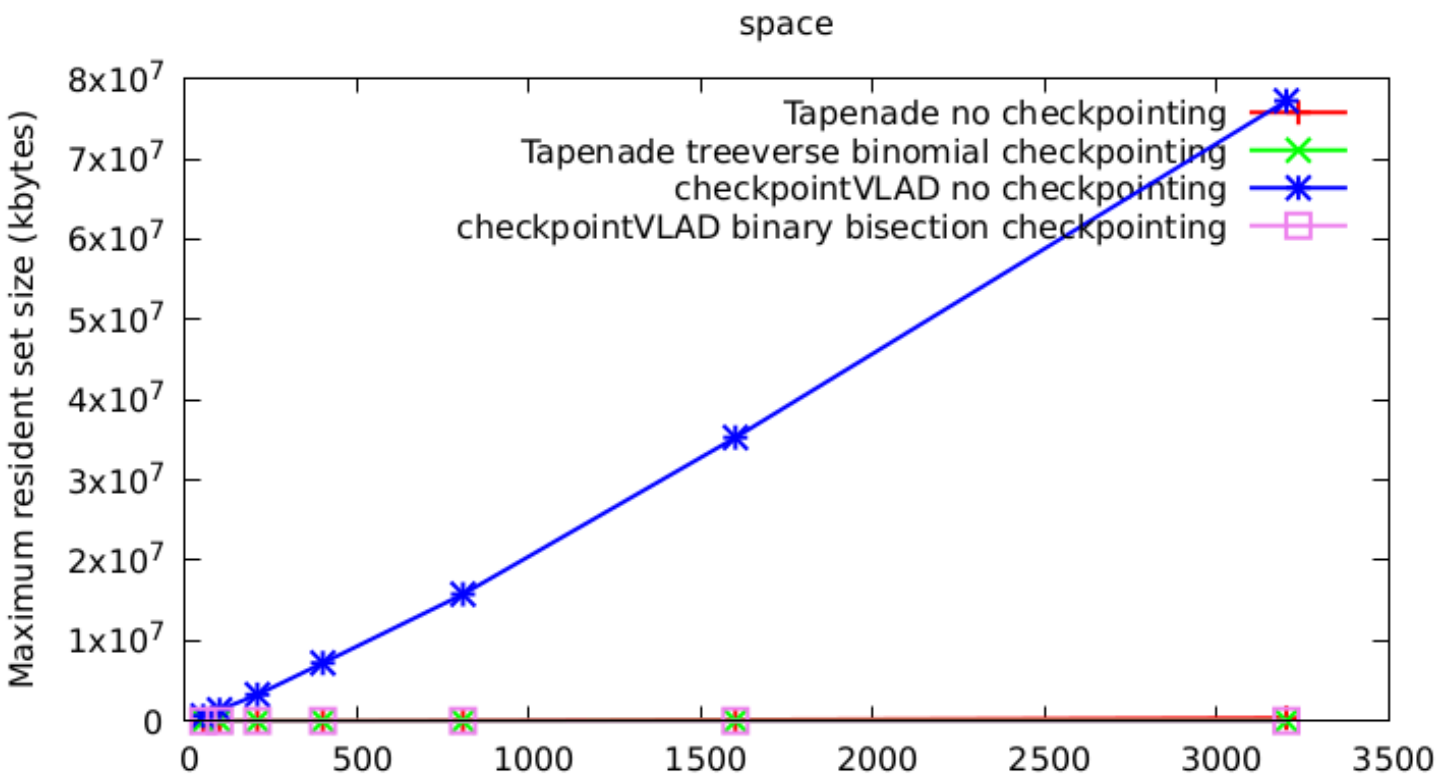
base case ($f\ x$ fast)**:**	$(y, \dot{x}) = \overleftarrow{\mathcal{J}}\ f\ x\ \dot{y}$	(step 0)
inductive case:	$h \circ g = f$	(step 1)
	$z = g\ x$	(step 2)
	$(y, \dot{z}) = \overrightarrow{\mathcal{J}}\ h\ z\ \dot{y}$	(step 3)
	$(z, \dot{x}) = \overrightarrow{\mathcal{J}}\ g\ x\ \dot{z}$	(step 4)

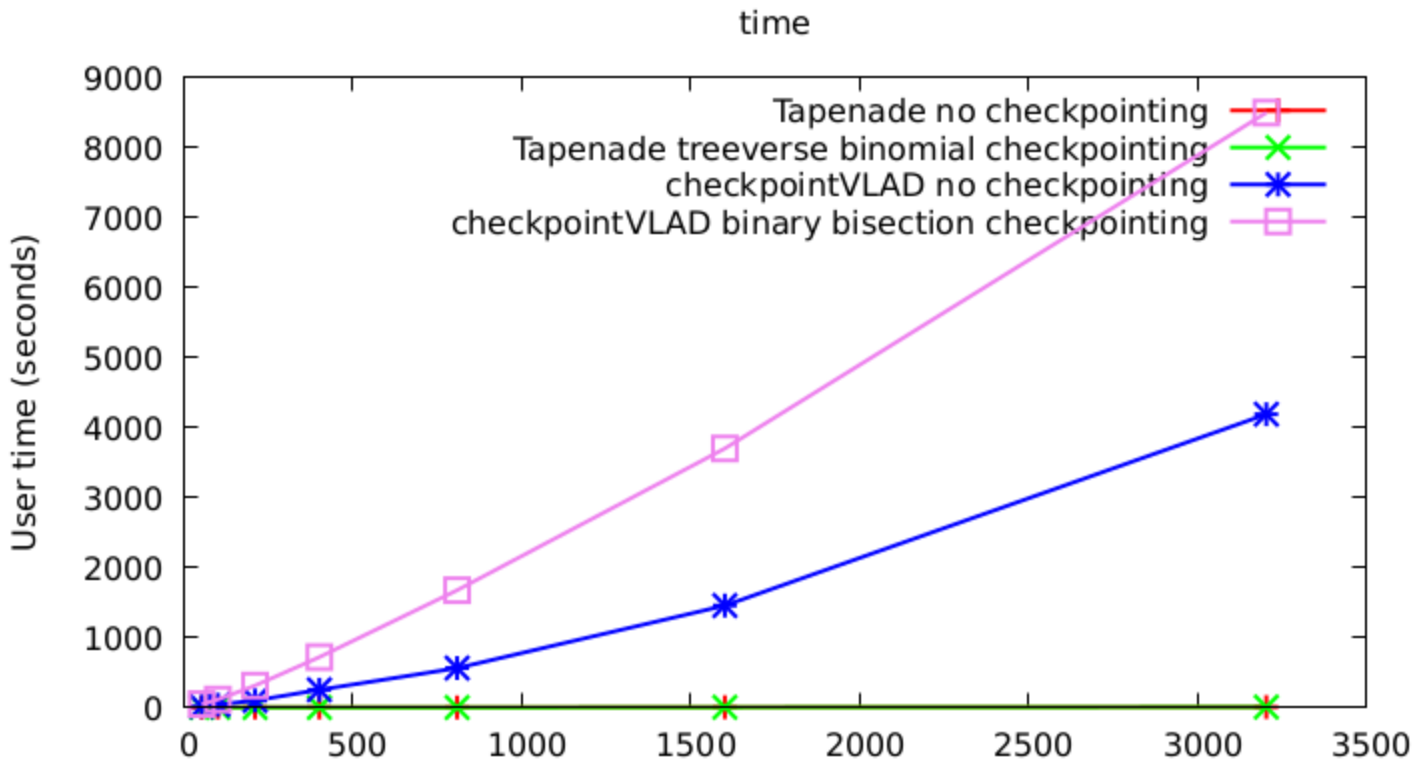
PRIMOPS $f\ x \mapsto l$	Return the number l of evaluation steps needed to compute $y = f(x)$.
INTERRUPT $f\ x\ l \mapsto z$	Run the first l steps of the computation of $f(x)$ and return a capsule z.
RESUME $z \mapsto y$	If $z =$ (INTERRUPT $f\ x\ l$), return $y = f(x)$.

'CheckpointVLAD'

Aggressive implementation, see paper.







CheckpointVLAD : scalar/CPU

:: Scorch : tensor/GPU

Bonus Material

- nesting of AD operators

ε be "fresh"

$$\mathcal{D}\ f\ c \triangleq \mathcal{E}\ \varepsilon\ (f\ (c + \varepsilon))$$

$$\mathcal{E}\ \varepsilon\ (x + x'\varepsilon) \triangleq x'$$

$$\frac{d}{dx}\left(x\left(\frac{d}{dy}\,xy\,\Big|_{y=2}\right)\right)\Big|_{x=1}$$

$$= \mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x \times y)\ 2))\ 1$$

$$= \mathcal{E}\ \varepsilon_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x \times y)\ 2))\ (1 + \varepsilon_a))$$

$$= \mathcal{E}\ \varepsilon_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) \times y)\ 2))$$

$$= \mathcal{E}\ \varepsilon_a\ ((1 + \varepsilon_a) \times (\mathcal{E}\ \varepsilon_b\ ((\lambda y\ .\ (1 + \varepsilon_a) \times y)\ (2 + \varepsilon_b))))$$

$$= \mathcal{E}\ \varepsilon_a\ ((1 + \varepsilon_a) \times (\mathcal{E}\ \varepsilon_b\ ((1 + \varepsilon_a) \times (2 + \varepsilon_b))))$$

$$= \mathcal{E}\ \varepsilon_a\ ((1 + \varepsilon_a) \times (\mathcal{E}\ \varepsilon_b\ ((2 + 2\varepsilon_a) + (1 + \varepsilon_a)\varepsilon_b)))$$

$$= \mathcal{E}\ \varepsilon_a\ ((1 + \varepsilon_a) \times (1 + \varepsilon_a))$$

$$= \mathcal{E}\ \varepsilon_a\ (1 + 2\varepsilon_a)$$

$$= 2$$

- Optimization API

- Hv

- AD of higher-order functions?

$$\vec{\mathcal{J}} : (\alpha \to \beta) \to (\alpha \times T_* \alpha) \to (\beta \times T_* \beta)$$

$$?: T_* (t_1 \to t_2) = t_1 \to T_* t_2$$

$$D_s 0 f x = D f x$$

$$g = D_s 0 \qquad s_c f x = f(x + c)$$

- AD of higher-order functions

→ nesting!

- Lots of room at the bottom

Tiark Rompf
de-scaffold levels

Benchmarks

		particle				saddle				probabilistic-lambda-calculus		probabilistic-prolog		backprop		
		FF	FR	RF	RR	FF	FR	RF	RR	F	R	F	R	F	Fv	R
VLAD	STALIN∇	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	■	1.00
FORTRAN	ADIFOR	2.05	■	■	■	5.44	■	■	■	■	■	■	■	15.51	3.35	■
	TAPENADE	5.51	■	■	■	8.09	■	■	■	■	■	■	■	14.97	5.97	6.86
C	ADIC	■	■	■	■	■	■	■	■	■	■	■	■	22.75	5.61	■
C++	ADOL-C	■	■	■	■	■	■	■	■	■	■	■	■	12.16	5.79	32.77
	CPPAD	■	■	■	■	■	■	■	■	■	■	■	■	54.74	■	29.24
	FADBAD++	93.32	■	■	■	60.67	■	■	■	■	■	■	■	132.31	46.01	60.71
ML	MLTON	78.13	111.27	45.95	32.57	114.07	146.28	12.27	10.58	129.11	114.88	848.45	507.21	95.20	■	39.90
	OCAML	217.03	415.64	352.06	261.38	291.26	407.67	42.39	50.21	249.40	499.43	1260.83	1542.47	202.01	■	156.93
	SML/NJ	153.01	226.84	270.63	192.13	271.84	299.76	25.66	23.89	234.62	258.53	2505.59	1501.17	181.93	■	102.89
HASKELL	GHC	209.44	■	■	■	247.57	■	■	■	■	■	■	■	■	■	■
SCHEME	BIGLOO	627.78	855.70	275.63	187.39	1004.85	1076.73	105.24	89.23	983.12	1016.50	12832.92	7918.21	743.26	■	360.07
	CHICKEN	1453.06	2501.07	821.37	1360.00	2276.69	2964.02	225.73	252.87	2324.54	3040.44	44891.04	24634.44	1626.73	■	1125.24
	GAMBIT	578.94	879.39	356.47	260.98	958.73	1112.70	89.99	89.23	1033.46	1107.26	26077.48	14262.70	671.54	■	379.63
	IKARUS	266.54	386.21	158.63	116.85	424.75	527.57	41.27	42.34	497.48	517.89	8474.57	4845.10	279.59	■	165.16
	LARCENY	964.18	1308.68	360.68	272.96	1565.53	1508.39	126.44	112.82	1658.27	1606.44	25411.62	14386.61	1203.34	■	511.54
	MIT SCHEME	2025.23	3074.30	790.99	609.63	3501.21	3896.88	315.17	295.67	4130.88	3817.57	87772.39	49814.12	2446.33	■	1113.09
	MZC	1243.08	1944.00	740.31	557.45	2135.92	2434.05	194.49	187.53	2294.93	2346.13	57472.76	31784.38	1318.60	■	754.47
	MZSCHEME	1309.82	1926.77	712.97	555.28	2371.35	2690.64	224.61	219.29	2721.35	2625.21	60269.37	33135.06	1364.14	■	772.10
	SCHEME->C	582.20	743.00	270.83	208.38	910.19	913.66	82.93	69.87	811.37	803.22	10605.32	5935.56	597.67	■	280.93
	SCMUTILS	4462.83	■	■	■	7651.69	■	■	■	7699.14	■	83656.17	■	5889.26	■	■
	STALIN	364.08	547.73	399.39	295.00	543.68	690.64	63.96	52.93	956.47	1994.44	15048.42	16939.28	435.82	■	281.27

Comparative benchmark results for the `particle` and `saddle` examples (Siskind and Pearlmutter, 2008a), the `probabilistic-lambda-calculus` and `probabilistic-prolog` examples (Siskind, 2008) and an implementation of backpropagation in neural networks using AD. Column labels are for AD modes and nesting: F for forward, Fv for forward-vector aka stacked tangents, RF for reverse-over-forward, etc. All run times normalized relative to a unit run time for STALIN∇ on the corresponding example except that run times for `backprop-Fv` are normalized relative to a unit run time for STALIN∇ on `backprop-F`. Pre-existing AD tools are named in blue, others are custom implementations. Key: ■ not implemented but could implement, including FORTRAN, C, and C++; ■ not implemented in pre-existing AD tool; ■ problematic to implement. All

- finish with WARNING:

AD & approximation DO NOT COMMUTE!