A Benchmark of Selected Algorithmic Differentiation Tools on Some Problems in Machine Learning and Computer Vision

Filip Srajer[1*] Zuzana Kukelova[2] Andrew Fitzgibbon[2]

[1]Czech Technical University in Prague
srajefil@fel.cvut.cz

[2]Microsoft Research Ltd
{zukuke,awf}@microsoft.com

1 Introduction

We look at three objectives from machine learning and computer vision, and fifteen ways of computing their derivatives (including hand-derived and finite differences). These objectives are for the most part simple, in the sense that no iterative loops are involved, and conditional statements are encapsulated in functions such as `abs` or `logsumexp`. However, it is important for the success of algorithmic differentiation that such "simple" objective functions are handled efficiently, as so many problems in these fields are of this form.

The objective functions considered are: log-likelihood of a Gaussian mixture model, bundle adjustment, and hand tracking. These functions include features such as sparse Jacobians, matrix expressions, and domain-specific special functions such as `logsumexp(x: vector)`, defined stably as `log(sum(exp(v - max(v)))) + max(v)`.

We first describe the objective functions used for benchmarking. Next, we give an overview of the selected AD tools. Then, we present the results and finally give our conclusions, foremost among which is that even with reasonable care devoted to efficiency in each of the input languages, the runtimes vary through four orders of magnitude.

Objective GMM: Gaussian Mixture Model Fitting

The Gaussian mixture model with Wishart prior has log-posterior function $\log\left(p(\boldsymbol{x};\boldsymbol{w},\boldsymbol{\mu},\boldsymbol{\Sigma})\right) =$

$$\log\left(\prod_{i=1}^{N}\sum_{k=1}^{K} w_k \det\left(2\pi\Sigma_k\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)\right)\prod_{k=1}^{K} C(D,m)|\Sigma_k|^m \exp\left(-\frac{1}{2}\operatorname{trace}(\Sigma_k)\right)\right) \quad \text{s.t.} \sum_{k=1}^{K} w_k = 1 \text{ and } \Sigma_k \text{ is positive-semidefinite } \forall k \in \{1,\ldots,K\} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^{D\times N}$ are data points, $\boldsymbol{w} \in \mathbb{R}^{K}$ weights, $\boldsymbol{\mu} \in \mathbb{R}^{D\times K}$ means, $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D\times K}$ covariance matrices, m is a Wishart hyperparameter and C is a function not dependent on independent variables. To integrate the constraints on weights and covariances into the objective function, we reparametrize the GMM function (1). After simplification, the final function to be optimized looks like

$$\log(p(\boldsymbol{x};\boldsymbol{\alpha},\boldsymbol{\mu},\boldsymbol{q},\boldsymbol{l})) = \sum_{i=1}^{n}\operatorname{logsumexp}\left(\left[\alpha_k + \operatorname{sum}(\boldsymbol{q}_k) - \frac{1}{2}||Q(\boldsymbol{q}_k,\boldsymbol{l}_k)(\boldsymbol{x}_i-\boldsymbol{\mu}_k)||^2\right]_{k=1}^{K}\right) - n\operatorname{logsumexp}(\boldsymbol{\alpha}) + \frac{1}{2}\sum_{k=1}^{K}\left(||\exp\left(\boldsymbol{q}_k\right)||^2 + ||\boldsymbol{l}_k||^2\right) - m\operatorname{sum}(\boldsymbol{q}_k) + C'(D,m) \tag{2}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^{K}$ corresponds to weights, $\boldsymbol{q} \in \mathbb{R}^{D\times K}$ and $\boldsymbol{l} \in \mathbb{R}^{\frac{d(d-1)}{2}\times K}$ to covariance matrices, C' does not depend on independent variables and Q assembles a $D \times D$ lower triangular matrix.

We benchmark the AD tools on gradient computation of Eq. (2). The size of the gradient changes with D and K, while $\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{q}$ and $\boldsymbol{l}$ are independent variables.

*Corresponding Author

Objective BA: Bundle Adjustment

Consider a weight $w \in \mathbb{R}$, a 3D point $\boldsymbol{X} \in \mathbb{R}^3$ and a camera with parameters $\boldsymbol{p} = [\boldsymbol{r}; \boldsymbol{C}; f; \boldsymbol{x}_0; \boldsymbol{\kappa}] \in \mathbb{R}^{11}$, i.e., rotation, camera center, focal length, principal point and radial distortion. The point $\boldsymbol{X}$ can be projected by the camera as

$$\text{project}(\boldsymbol{r}, \boldsymbol{C}, f, \boldsymbol{x}_0, \boldsymbol{\kappa}, \boldsymbol{X}) = \text{distort}(\boldsymbol{\kappa}, \text{p2e}(\text{rodriguez}(\boldsymbol{r}, \boldsymbol{X} - \boldsymbol{C})))f + \boldsymbol{x}_0 \tag{3}$$

$$\text{distort}(\boldsymbol{\kappa}, \boldsymbol{x}) = \boldsymbol{x}(1 + \kappa_1||\boldsymbol{x}||^2 + \kappa_2||\boldsymbol{x}||^4) \tag{4}$$

$$\text{p2e}(\boldsymbol{X}) = \boldsymbol{X}_{1..2}/X_3 \tag{5}$$

$$\text{rodriguez}(\boldsymbol{r}, \boldsymbol{X}) = \boldsymbol{X}\cos\theta + (\boldsymbol{v} \times \boldsymbol{X})\sin\theta + \boldsymbol{v}(\boldsymbol{v}^T\boldsymbol{X})(1 - \cos\theta), \quad \theta = ||\boldsymbol{r}||, \boldsymbol{v} = \frac{\boldsymbol{r}}{||\boldsymbol{r}||} \tag{6}$$

The observed image point is $\boldsymbol{m} \in \mathbb{R}^2$ and the residual $\boldsymbol{e}$ concatenates its reprojection error and w's regularizer:

$$\boldsymbol{e} = [w(\boldsymbol{m} - \text{project}(\boldsymbol{r}, \boldsymbol{C}, f, \boldsymbol{x}_0, \boldsymbol{\kappa}, \boldsymbol{X}))^\top; 1 - w^2]^\top \tag{7}$$

The goal of BA is to optimize 3D points, camera parameters and weights simultaneously in a system with multiple cameras and points [1, 2]. The Jacobian has only 15 non-zero entries in every reprojection-error row and one non-zero in every weight-term row. This is typically [2] exploited by computing only small independent Jacobians and then inserting them into the large sparse one.

Objective HT: Hand Tracking

The hand tracking problem [3] has independent variable $\boldsymbol{p} \in \mathbb{R}^{26}$ parameterizing motion. A hand is modelled by a set of points $\boldsymbol{X} \in \mathbb{R}^{3\times M}$ and their triangulation, i.e., the model is a collection of adjacent triangles, which make up a surface. Then, there are N correspondences between observed data points $\boldsymbol{Y} \in \mathbb{R}^{3\times N}$ and the triangles. The q^{th} correspondence has independent variable $\boldsymbol{u}_q \in \mathbb{R}^2$ defining an exact spot inside a triangle to which $\boldsymbol{Y}_q$ corresponds.

The procedure for computing the error for all measurements is the following:

1. Compute $\boldsymbol{Z} \in \mathbb{R}^{3\times M}$ by transforming $\boldsymbol{X}$ according to $\boldsymbol{p}$. This includes constructing transformation matrices (similar approach as Eq. (6)) and multiplying the matrices with the points.
2. For q-th measurement corresponding to the triangle (i, j, k), compute $\boldsymbol{W}_q = u_{q,1}\boldsymbol{Z}_i + u_{q,2}\boldsymbol{Z}_j + (1 - u_{q,1} - u_{q,2})\boldsymbol{Z}_k$.
3. For q-th measurement, compute error $\boldsymbol{e}_q = \boldsymbol{Y}_q - \boldsymbol{W}_q$.

The Jacobian has a dense part composed of columns of $\boldsymbol{p}$ and a sparse part corresponding to $\boldsymbol{u}$, where every row has two non-zero entries. AD tools supporting sparsity compute the whole sparse part with only two passes. Several tools are not run as they do not support dynamic sparsity (in the tested version [2]).

2 Tools

We have chosen several well-known or promising AD tools (see Tab. 1). The selection covers various languages and AD approaches as well as symbolic differentiation. The newest version of all the tools that was available in the period July-August 2015 was used. In addition, we give results for *finite differences* and *manual*, i.e., a hand-derived optimized implementation.

From the chosen tools, we did not benchmark clad as it does not have support for arrays, ADiGator because it generated syntactically incorrect code for GMM and ADIC2 as our attempts to compile it were unsuccessful.

For tools that have both forward and reverse mode, they are called with the one that is more suitable for the given objective. Diffsharp in particular runs significantly slower in default mode so it is called in its special forward and reverse modes for first-order derivatives. Tapenade offers differentiation of both clean C and Fortran code but we use it only with C. Unfortunately, its source transformation occasionally produces non-compiling output, so the user has to fix a few errors. MuPAD optimizes code using common subexpression elimination and compiles it via C++ to MEX. Theano is written in a modified Python and compiles either into optimized Python or C++. Theano is always ran in CPU mode to allow a fair comparison since all the tools use only CPU.

3 Experiments

To benchmark the AD tools, we first ran pre-processing routines (e.g. source transformation, symbolic differentiation, taping). All of the routines that need to be run only once for different data are not included in the runtimes that we provide. This is justified since a user of AD tools would typically run it only once on the objective before calling the differentiated function many times to optimize parameters.

The benchmarking is done on random data. The resulting runtimes are averaged over 1000 runs if one run is less than 5 seconds, over 100 runs if 5-30 seconds and over 10 runs if 30-120 seconds. Otherwise, the runtimes are not

Language	Tool	Approach	Mode	BA	HT small	HT big
C++		Manual (by hand)		20	1	98
C++		Finite differences		43	18	390
C++	Adept [5]	OO	F, R	68	30	760
C++	ADIC2 [6]	ST	F, R	•	•	•
C++	ADOL-C [7]	OO	F, R	850	23	760
C++	Ceres Solver [2]	OO	F	230	•	•
C++	clad [8]	ST via compiler	F	•	•	•
C/Fortran	Tapenade [9]	ST	F, R	24	•	•
F#	DiffSharp [10]	OO	F, R	540	720	72,000
MATLAB	ADiGator [11]	ST via OO	F	•	•	•
MATLAB	ADiMat [12]	OO via ST	F, R	550,000	310	75,000
MATLAB	MuPAD [13]	Symbolic		27	•	•
Julia	ForwardDiff.jl [14]	OO	F	1,300	•	•
Python	Autograd [15]	OO	F	170,000	•	•
Python	Theano [16]	Symbolic		18,000	59,000	timeout

Table 1: List of tools. OO: operator overloading, ST: source transformation: F: forward, R: reverse, •: not run (see text). For BA and HT, runtimes are shown in milliseconds, to 2 significant digits. Theano's AD-like R-op mode is used for HT. Standard symbolic mode would not handle sparsity.

averaged. A single machine with a processor Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz, memory 32GB and OS Windows 10 64-bit was used for all the experiments.

Fig. 1 shows gradient computation runtimes for GMM. We have noticed that some of the tools do not handle bigger instances. The biggest instance size ($D = 64$, $K = 200$) was taken from [4]. We help out some of the tools by manually splitting the gradient computation

$$\nabla \log(p(\boldsymbol{x}; \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{q}, \boldsymbol{l})) = \sum_{i=1}^{N} \nabla f(\boldsymbol{x}_i; \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{q}, \boldsymbol{l}) + \nabla g(\boldsymbol{\alpha}, \boldsymbol{q}, \boldsymbol{l}) \tag{8}$$

which is symbolized by *(split)* in the figures. Moreover, GMM allows for a vectorized implementation (denoted by *(vector)*), where most necessary computations are done in one huge matrix multiplication. We show this *(vector)* version with languages that are able to utilize it.

Next, Tab. 1 shows runtimes for BA. We have tried running BA on various problem sizes ranging from 21 cameras, 11k 3D points and 36k observations to 14k cameras, 4M 3D points, 29M observations. The runtimes are, as expected (see above), linearly dependent on the number of observations only. Thus, we show only runtimes for the smallest problem size.

Tab. 1 also gives results for HT. We show results for a small model suitable for a real-time application and a big one which would be run offline. The small instance has 544 3D points and 192 correspondences whereas the big one has 10k 3D points and 100k correspondences.

4 Conclusion

First, we have described three real-world objective functions from areas of machine learning and computer vision. Second, we have chosen several approaches and tools for computing derivatives to be benchmarked. Then, we have provided runtimes for computing derivatives.

We have seen that the runtimes of derivative computation range through four orders of magnitude. This is partially dependent on a programing language. It will also depend on programmer skill, and familiarity with the tools, so we have made open source all our materials[1], in order that others may improve on our efforts. However, we contend that this paper presents an important datapoint: a skilled programmer devoting roughly a week to each tool produced the timings above. For many projects, these will represent typical results achieved before a tool is selected.

We conclude that there are useful tools in most languages but there is also still some space for improvement. Availability of various features proves to be crucial for the success and efficiency of algorithmic differentiation. Important features for our objectives range from sparsity support (at least manually specified) and support of matrix libraries to memory optimizations for big problem instances and the option to choose from both forward and reverse mode. Importantly, note that we benchmarked only computation of the first-order derivatives and some tools do not support higher-order derivatives.

[1] https://github.com/awf/autodiff

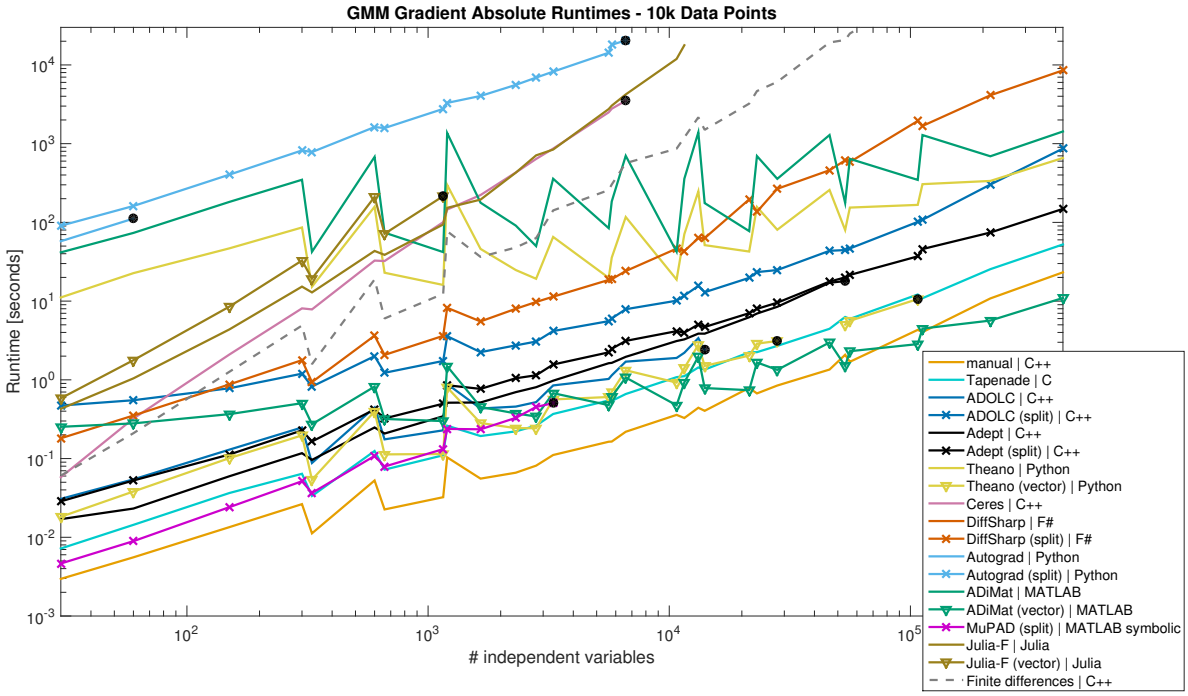

Figure 1: Runtimes in seconds for GMM. Some of the tools were run with (split) or (vector) implementations (see Sec. 3). The black dots emphasize the end of a curve symbolizing that the tools crashed on bigger problem sizes. Note that both axes are log-scaled. Best viewed in color.

5 Acknowledgements

This work was done during an internship in Microsoft Research Ltd. We thank Jonathan Taylor for an example implementation of a hand tracking function in Python.

References

[1] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proc. of the Intl. Workshop on Vision Algorithms: Theory and Practice*, ICCV, pages 298–372, 2000.

[2] S. Agarwal, K. Mierle, and Others. Ceres Solver. `http://ceres-solver.org`.

[3] J. Taylor, R. Stebbing, V. Ramakrishna, C. Keskin, J. Shotton, S. Izadi, A. Hertzmann, and A. Fitzgibbon. User-specific hand modeling from monocular depth sequences. In *CVPR*, 2014.

[4] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *ICCV*, pages 479–486, Nov 2011.

[5] R. J. Hogan. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM TOMS*, 40(4):26:1–26:24, 2014.

[6] S. H. K. Narayanan, B. Norris, and B. Winnicka. ADIC2: Development of a component source transformation system for differentiating c and c++. *Procedia Computer Science*, 1(1):1845 – 1853, 2010.

[7] A. Walther and A. Griewank. Getting started with ADOL-C. In *Combinatorial Scientific Computing*, chapter 7, pages 181–202. Chapman-Hall CRC Computational Science, 2012.

[8] V. Vassilev, V. Ilieva, L. Moneta, A. Penev, and M. Vassilev. clad. `https://github.com/vgvassilev/clad`.

[9] L. Hascoët and V. Pascual. The Tapenade automatic differentiation tool: Principles, model, and specification. *ACM TOMS*, 39(3):20:1–20:43, 2013.

[10] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *arXiv preprint arXiv:1502.05767*, 2015.

[11] A. V. Rao and M. J. Weinstein. ADiGator. `https://sourceforge.net/projects/adigator/`.

[12] C. H. Bischof, H. M. Bücker, B. Lang, A. Rasch, and A. Vehreschild. Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs. In *SCAM*, pages 65–72, 2002.

[13] The MathWorks, Inc. Mupad. MATLAB Symbolic Math Toolbox.

[14] J. Revels, T. Papamarkou, M. Lubin, and Others. JuliaDiff: ForwardDiff.jl. `https://github.com/JuliaDiff/ForwardDiff.jl`.

[15] D. Maclaurin, D. Duvenaud, and M. Johnson. Autograd. `https://github.com/HIPS/autograd`.

[16] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. DLUFL NIPS Workshop, 2012.