How Fast is AI in Pharo?
Benchmarking Linear Regression

Oleksandr Zaitsev[1,2,*,†], Sebastian Jordan-Montaño[2,*,†] and Stéphane Ducasse[2]

[1]*Arolla, 25 Rue du Louvre, 75001 Paris, France*

[2]*Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, Park Plaza, Parc scientifique de la Haute-Borne, 40 Av. Halley Bât A, 59650 Villeneuve-d'Ascq, France*

Abstract

As many other modern programming languages, Pharo spreads its applications into computationally demanding fields such as machine learning, big data, cryptocurrency, etc. This raises a need for fast numerical computation libraries. In this work, we propose to speed up the low-level computations by calling the routines from highly optimized external libraries, *e.g.,* LAPACK or BLAS through the foreign function interface (FFI). As a proof of concept, we build a prototype implementation of linear regression based on the DGELSD routine of LAPACK. Using three benchmark datasets of different sizes, we compare the execution time of our algorithm agains pure Pharo implementation and scikit-learn — a popular Python library for machine learning. We show that LAPACK&Pharo is up to 2103 times faster than pure Pharo. We also show that scikit-learn is 8-5 times faster than our prototype, depending on the size of the data. Finally, we demonstrate that pure Pharo is up to 15 times faster than the equivalent implementation in pure Python. Those findings can lay the foundation for the future work in building fast numerical libraries for Pharo and further using them in higher-level libraries such as pharo-ai.

Keywords

pharo, benchmarking, machine learning, foreign function interface, lapack, linear regression

1. Introduction

Pharo (https://pharo.org/) is an open-source dynamically-typed reflective, object-oriented programming language inspired by Smalltalk [1]. Pharo community is growing and spreading its applications into various domains of modern computer science: big data [2, 3], machine learning [4],[1] cryptocurrency,[2][3] internet of things (IoT) [5]. Those domains require fast numerical algorithms capable of working with large datasets. Modern programming languages provide such libraries and use them as backend for more high-level ones such as scikit-learn[4], a popular

[1]https://github.com/pharo-ai/wiki
[2]https://github.com/smartanvil/Fog
[3]https://github.com/oliveiraallex/Pharocks-Cryptobot
[4]https://scikit-learn.org/

machine learning library. Pharo also provides a library for numerical computations — PolyMath. However, as we will see in this paper, the current implementation of certain algorithms in Pharo is significantly slower than the ones provided by similar libraries in Python and R that delegate the low-lewel computations to the fast routines compiled in Fortran or C.

In this paper, we propose to speed up the numerical computations in Pharo by calling low-lewel routines from the highly optimized external libraries such as LAPACK or BLAS. As a proof of concept, we propose a prototype implementation of the linear regression algorithm in Pharo based on the DGELSD routine of LAPACK.[5] We benchmark our algorithm against both an alternative implementation in pure Pharo and scikit-learn — a widely used Python library for machine learning.

The Pharo & LAPACK implementation is 1820 times faster than pure Pharo on a small-size dataset (200K rows) and 2103 times faster on a medium-size dataset (1M rows). This serves as a proof of concept that the speed of numerical algorithms, and more specifically machine learning in Pharo can be significantly improved by calling highly optimized external libraries through FFI. We also show that scikit-learn is still 8-5 times faster than our prototype implementation in Pharo (depending on the size of data), which might be caused by data preprocessing and other optimizations. We propose to further explore this difference in future work. Finally, we show that pure Pharo implementation of linear regression is about 5-15 times faster than the equivalent implementation in pure Python. This can be explained by the just-in-time compilation performed by Pharo virtual machine.

The rest of this paper is structured in the following way. In Section 2, we discuss the possibility of implementing a machine learning library in Pharo and the challenges associated with it. In Section 4, we briefly explain the foreign function interface (FFI) and how it can be used to call C and Fortran routines from Pharo. In Section 3, we explain different implementations of the linear regression algorithm that we have selected for benchmarking.

2. Can We Do Machine Learning in Pharo?

Modern machine learning is resource demanding. It requires fast algorithms capable of processing large volumes of data. Many algorithms in machine learning are based on matrix algebra and numerical optimization. This raises the need for fast libraries that implement the basic algebraic operations such as matrix-vector multiplication and also highly optimized algebraic algorithms as singular value decomposition, Cholesky decomposition, eigenvector and eigenvalues calculation, etc. for different types of matrices, i.e., symmetric, orthogonal, hermitian, etc.

In Pharo, there is PolyMath — a library for numerical computations and scientific computing. With its 60 packages and 903 green tests[6] PolyMath provides a wide range of algorithms that cover various aspects of mathematics: differential equations, complex numbers, fuzzy algorithms, etc. It also provides matrix algebra, statistical distributions, and numerical optimization methods that can be particularly useful for machine learning. That being said, some algorithms that are

---

[5]All the code, the datasets, and the instructions on how to reproduce our experiment are available at https://anonymous.4open.science/r/lapack-experiment-55FB.
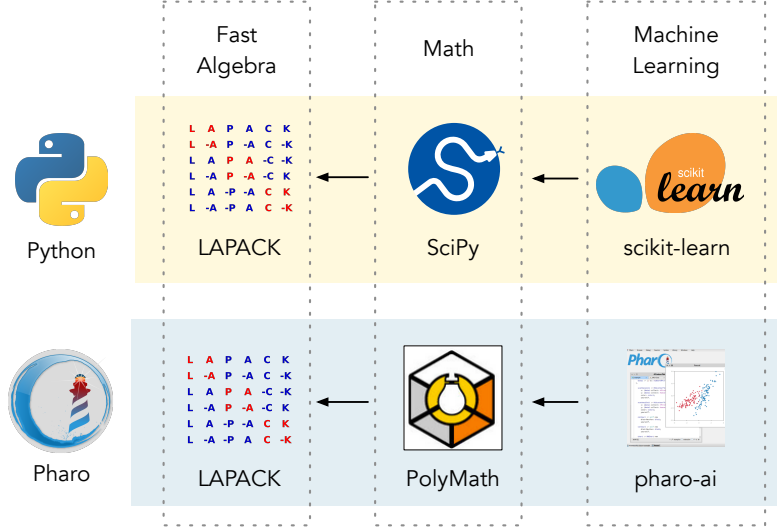[6]Measured at https://github.com/PolyMathOrg/PolyMath on June 1, 2022.

Figure 1: Machine learning libraries depend on mathematical libraries that in their turn depend on Lapack - a highly-optimized library providing fast algebraic operations.

implemented in PolyMath are very slow, compared to the similar numerical libraries in other languages. For example, as we will see in Section 6, the linear regression implementation that is based on the singular value decomposition (SVD) provided by PolyMath takes almost 5 hours to converge on a dataset with 5,000 rows and 5 columns.

There is also an ongoing effort in Pharo community to implement the tools for data science, machine learning, artificial intelligence, data mining, etc.[7] One such library is *pharo-ai* — a collection of different machine learning algorithms implemented in Pharo: linear and logistic regression, support vector machines, K-means, naive bayes, etc. It is inspired by *scikit-learn*, a similar machine learning library in Python. scikit-learn is a popular library that has many industrial applications. Many of its algorithms are very fast because internally scikit-learn depends on a mathematical library *scipy*, which in turn depends on *LAPACK* — an efficient low-level library implemented in Fortran that provides fast algorithms for linear algebra.

We propose to improve the speed of PolyMath and pharo-ai by calling LAPACK routines through foreign function interface — a technique for calling external functions which is already implemented by Pharo (see Section 4). In Figure 1, we show how LAPACK is used by scikit-learn in Python and propose a way how it could be used by pharo-ai library. In this scheme, PolyMath implements various high-level mathematical algorithms by delegating low-level algebraic operations to LAPACK. Then pharo-ai simply depends on PolyMath to implement fast machine learning algorithms.

As a proof of concept, in this paper we present a prototype implementation of linear regression in Pharo based on LAPACK. We benchmark the training time of our model and show that it is almost 500 times faster than the pure Pharo implementation.

---

[7]For the non-extensive list of different machine learning libraries and tools in Pharo, see https://github.com/pharo-ai/awesome-pharo-ml.

3. Linear regression and How it is Implemented

Linear Regression is a machine learning algorithms that models a relationship between the input matrix X and the output vector y [6].

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix} ; \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} ; \quad \theta = \begin{pmatrix} \theta_0 \\ \vdots \\ \theta_n \end{pmatrix}$$

The task of linear regression is to find a set of parameters $\theta = (\theta_0, \ldots, \theta_n)$ such that the predictions $\hat{y}_i = h_\theta(x_i)$ are as close as possible to the real output values y_i. The function h_θ (also known as hypothesis function) is defined as:

$$h_\theta(x_i) = \theta_0 + \theta_1 x_{i1} + \cdots + \theta_n x_{in}$$

This is an optimization problem: we need to minimize the objective function that is often defined as *mean squared error*:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (y_i - h_\theta(X_{i:}))^2$$

We have selected linear regression for benchmarking because: (1) it is one of the most well known and commonly used machine learning algorithms; (2) it can be solved by finding a minimum norm solution to the linear least squares problem — an algorithm of linear algebra that is implemented in LAPACK.

Gradient Descent Solution   Gradient descent is the iterative numerical optimization technique that is used by many machine learning algorithms, including linear regression.

It is based on the fact that derivative of the function in a given point is positive if the function is increasing and negative if it is decreasing. This means that by subtracting the value of derivative $\frac{\partial}{\partial \theta_j} J(\theta)$, scaled by a parameter α (called the learning rate), from the current value of parameter θ_j, we decrease the cost $J(\theta)$:

$$\theta_j^{(new)} = \theta_j^{(old)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Least Squares Solution   Alternatively, the problem of linear regression can be expressed in terms of linear least squares. In this case, $X\theta = y$ is viewed as a system of linear equation where θ is the unknown. Unless all points of the dataset lie on a straight line, this system has no solutions. However, it is possible to find the closest approximation to the solution by solving the system $X\theta = \hat{y}$ where $\hat{y}$ is the orthogonal projection of y onto the columnspace of X (thus the distance between true output y and predicted output $\hat{y}$ is the smallest). This is the same as finding the optimal parameters $\hat{\theta}$ by minimizing the norm:

$$\hat{\theta} = \operatorname{argmin} \left|\left| y - X\theta \right|\right|_2$$

In this work, we compare the following implementations of linear regression:

1. Iterative gradient descent implementation in pure Pharo and pure Python.
2. Least squares implementation in Pharo (our prototype) and Python (scikit-learn) that are both based on a routine from LAPACK library.

4. Foreign Function Interface in Pharo

As many modern programming languages, Pharo supports the *Foreign Function Interface* (FFI) — the mechanism that allows the call of routines from another programming language [7]. One of the common uses is to speed up computations in interpreted or virtual machine languages by calling routines from natively compiled libraries (*e.g., *.dll* files on Windows or **.so* files on Linux and Mac). Here is an example of calling a C function uint clock() from Pharo using FFI:

```
self
    ffiCall: #(uint clock())
    library: 'libc.so.6'.
```

The method ffiCall:library: is understood by every object in Pharo. It accepts two arguments which specify the shared library and the signature of a method that should be called. The value returned by this method will be the same as the return value of the external function that is being called.

Calling LAPACK routine   The Linear Algebra PACKage (LAPACK) is a Fortran library that provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems, matrix factorizations, etc. In this work, we use one specific routine of LAPACK that computes the minimum-norm solution to a real linear least squares problem — DGELSD.[8] It accepts as input matrix X and vector y and finds the solution vector $\hat{\theta}$ which minimizes the norm $||y - X\theta||_2$ (see Section 3).

5. Experiment Setup

Research Questions   In our study, we answer the following research questions:

- *RQ.1 - Measuring LAPACK speedup.* How much time improvement can we achieve by calling LAPACK from Pharo?
- *RQ.2 - Comparing to scikit-learn.* How does Pharo & LAPACK implementation compare to the one provided by scikit-learn?
- *RQ.3 - Comparing pure Pharo with Python.* How does pure Pharo implementation of linear regression compare to equivalent pure Python implementation?

---

[8]https://www.netlib.org/lapack/explore-html/d7/d3b/group__double_g_esolve_ga94bd4a63a6dacf523e25ff617719f752.html

Datasets for benchmarking We generated three datasets of different sizes using the make_regression() method of scikit-learn with a random linear regression model with a fixed seed. The sizes of those datasets can be seen in Table 1. The number of columns was fixed at 20 and the number of rows was gradually increased. We did this because the height (number of rows) and width (number of columns) of a dataset have different effect on the training time [6] and we did not want to mix them.

The first 19 columns represent the input matrix X. They contain floating point numbers that are normally distributed around 0 with standard deviation 1. The last column of each dataset represents the output vector y generated by applying a non-biased linear regression model with gaussian noise.

Table 1
Three datasets that we generated for this study

Name	Columns	Rows	Size
Small	200,000	20	82 Mb
Medium	1,000,000	20	411 Mb
Large	5,000,000	20	2.06 Gb

Measuring the execution time In this study, we only measure that time that it takes to train the linear regression model as it is generally the most consuming part of the machine learning workflow. In Pharo, we measure time using the timeToRun method:

```
time := [ "... some code..." ] timeToRun.
```

In Python, we use the time library:

```
import time

start_time = time.time()
# ... some code...
time = time.time() - start.time()
```

6. Results

In this section, we answer the research questions and discuss the results of our experiments. The benchmarks were run in a MacBook Pro 2015 with a Intel i7 processor of 4 cores and 2.2G Hz and 16 GB of RAM memory. The operating system is Monterrey v12.4. To run the benchmarks, we closed all the applications that could be closed. In addition, we disconnected the Internet and did not use the computer while experiment was in progress. The computer was plugged into the charger for the duration of the experiment.

RQ.1 - Measuring LAPACK speedup

To answer the first research question, we measured the execution time of two implementations of linear regression on the three datasets that were discussed in Section 5. The first implementation

was implemented in pure Pharo (no external libraries) and based on gradient descent. The second implementation was based on least squares, implemented by calling the DGELSD LAPACK routine from Pharo. As can be seen in Table 2, the LAPACK implementation was 1820 times faster on small dataset and 2103 times faster on medium dataset. We could not measure the execution time of pure Pharo implementation on large dataset because it was too long and we had to stop it after 5 hours. The LAPACK implementation took about 15 seconds to converge on the same dataset.

Table 2
The speedup achieved by calling LAPACK routines fom Pharo.

Dataset	Pharo	Pharo & LAPACK	Diff
Small	00:19:39.090	00:00:00.648	1820×
Medium	01:43:59.000	00:00:02.967	2103×
Large	∞	00:00:15.676	—

> **Summary:** Calling LAPACK routines from Pharo can provide a significant speedup. The linear regression implemented with Pharo&LAPACK is 462 times faster than the pure Pharo implementation when measured on small dataset and 284 times faster on medium dataset.

RQ.2 - Comparing to scikit-learn

We compared the execution time of LAPACK-based implementation of linear regression in Pharo to the one provided by LinearRegression class in scikit-learn, which is also based on the DGELSD routine of LAPACK. As can be seen in Table 3, scikit-learn is 8 times faster than our implementation on small dataset, 8 times faster on medium dataset, and 5 times faster on large dataset. Considering that scikit-learn is well designed library with many industrial users and our implementation is a prototype, we hypothesize that this difference is due to various optimizations that are performed in scikit-learn. For example, it uses a highly optimized data structure NumPy[9]. A further study is needed to explain this phenomenon.

Table 3
Comparison with scikit-learn (both implementations use LAPACK).

Dataset	Pharo & LAPACK	scikit-learn	Diff
Small	00:00:00.648	00:00:00.079	8×
Medium	00:00:02.967	00:00:00.790	8×
Large	00:00:15.676	00:00:03.499	5×

> **Summary:** scikit-learn implementation is 8 times faster than our prototype implementation in Pharo. It is 5 times faster on a large dataset.

---

[9]https://numpy.org/

RQ.3 - Comparing pure Pharo with pure Python

We also compare pure Pharo and pure Python implementations of linear regression based on gradient descent. As can be seen in Table 4, Pharo is 5 times faster than Python on small dataset and 15 times faster on medium dataset. We could not measure the execution time on large dataset, because in both cases it was too long and we had to stop the experiment.

Table 4
Comparison between pure Pharo and pure Python

Dataset	Pharo	Python	Diff
Small	00:19:39.090	01:39:54.275	5×
Medium	01:43:59.000	08:25:00.000	15×
Large	∞	∞	—

> **Summary:** The implementation of gradient descent-based linear regression in pure Pharo (no external libraries) is about 5-15 times faster than the equivalent implementation in pure Python, depending on the size of the dataset.

7. Threats to Validity

We consider the four types of validity threats that were presented by Wohlin *et al.,* [8]: internal, external, construct, and conclusion validity. Below, we discuss the threats to internal and external validity. We did not find any threats to conclusion and construct validity.

Internal Validity

- The implementation of linear regression with LAPACK was based on least squares while the pure Pharo implementation was based on gradient descent. This poses a threat to internal validity because the time difference may be affected by the choice of algorithm.
- The implementation that we propose for Pharo is a prototype, while scikit-learn is the industry standard which contains many optimizations. We tried to study the source code of scikit-learn as much as possible, but this still constitutes a validity threat.
- Measuring the execution time is always prone to the noise introduced by different processes that are run on the computer. To reduce this noise, we have closed all unnecessary windows, disconnected the Internet, plugged the computer to the charger, and killed all processes that could be stopped.

External Validity

- In this work, we demonstrated how the training time of linear regression can be reduced using LAPACK. The same technique can be used for many other machine learning algorithms: logistic regression, neural networks, etc. Nonetheless, there are also algorithms that can not benefit from it because, unlike linear regression, they do not require heavy matrix-vector calculations. Some examples include k-means or k-nearest neighbours.

Those algorithms can still be boosted with optimized C code called through FFI, but this requires a separate study.

8. Related Work

Over the years, Pharo has been successfully applied in various domains of scientific computing and numerical optimization. Many of those applications use the algorithms provided by PolyMath [9] — a library for scientific computing in Pharo. Other applications depend on the algorithms of artificial intelligence and machine learning [4]. Bergel *et al.,* [10] use neural networks and a visual environment to verify various properties related to the source code. Cota *et al.,* [11] use genetic algorithms for automatic tests generation. Pharo is also used for agent-based modeling by CORMAS (Common Pool Resources and Multi-Agent Systems) — a platform dedicated to natural and common-pool resources management [12, 13]. All those applications could benefit from speeding up numerical computations in Pharo.

In this work, we benchmark our prototype implementation of linear regression against scikit-learn — a Python machine learning library that is widely used in both research and industry [14]. Pedregosa *et al.,* [15] report that scikit-learn outperforms other machine learning libraries in Python (MLP, PyBrain, pymvpa, MDP, and Shogun) in 4 out of 6 classical machine learning algorithms.

To the best of our knowledge, this is the first attempt to implement a fast numerical library in Pharo by calling an external library such as LAPACK. It is also the first study that benchmarks the machine learning algorithms in Pharo.

9. Conclusion

In this work we presented a proof of concept that demonstrates how numerical computations in Pharo can be boosted by calling the routines highly optimized external libraries through FFI. We built a prototype implementation of linear regression algorithm in Pharo based on the DGELSD routine from LAPACK. We measured the execution time on three benchmark datasets and compared it to the time that is needed to train the analogous models in scikit-learn and in pure Pharo. We show that LAPACK-based implementation is up to 2103 times faster than pure Pharo but still 8-5 times slower than scikit-learn implementation, which also uses LAPACK underneeth. We also show that pure Pharo implementation is up to 15 times faster than the equivalent model implemented in pure Python. This interesting finding can be explained by just-in-time compilation that is performed by Pharo virtual machine.

10. Acknowledgements

References

[1] A. P. Black, S. Ducasse, O. Nierstrasz, D. Pollet, D. Cassou, M. Denker, Pharo by Example, Square Bracket Associates, Kehrsatz, Switzerland, 2009. URL: http://books.pharo.org.

[2] M. Marra, G. Polito, E. Gonzalez Boix, A debugging approach for live big data applications, Science of Computer Programming 194 (2020) 102460. doi:`10.1016/j.scico.2020.102460`.

[3] M. Marra, A Live Debugging Approach for Big Data Processing Applications, Ph.D. thesis, Vrije Universiteit Brussel, 2022.

[4] A. Bergel, Agile Artificial Intelligence in Pharo, Apress, 2020.

[5] A. Oliveira, S. Costiou, S. Ducasse, S. Stinckwich, A pharothings tutorial, in: A PharoThings Tutorial, Square Bracket Associates, 2021. URL: https://github.com/SquareBracketAssociates/Booklet-APharoThingsTutorial.

[6] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems, " O'Reilly Media, Inc.", 2019.

[7] G. Polito, S. Ducasse, P. Tesone, T. Brunzie, Unified ffi - calling foreign functions from pharo, 2020. URL: http://books.pharo.org/booklet-uffi/.

[8] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering: an introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[9] D. H. Besset, Object-Oriented Implementation of Numerical Methods An Introduction with Pharo, Square Bracket Associates, 2016.

[10] A. Bergel, P. Melatagia, S. Stinckwich, An api and visual environment to use neural network to reason about source code, in: Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, 2018, pp. 117–120.

[11] A. Cota Vidaure, E. Cusi Lopez, J. P. Sandoval Alcocer, A. Bergel, Testevoviz: Visual introspection for genetically-based test coverage evolution., in: VISSOFT, 2020, pp. 1–11.

[12] P. Bommel, N. Becu, C. Le Page, F. Bousquet, Cormas: an agent-based simulation platform for coupling human decisions with computerized dynamics, in: Simulation and gaming in the network society, Springer, 2016, pp. 387–410.

[13] C. Le Page, N. Becu, P. Bommel, F. Bousquet, Participatory agent-based simulation for renewable resource management: the role of the cormas simulation platform to nurture a community of practice, Journal of Artificial Societies and Social Simulation 15(1) (2012) 16.

[14] M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, B. Bischl, mlr3: A modern object-oriented machine learning framework in r, Journal of Open Source Software 4 (2019) 1903.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, the Journal of machine Learning research 12 (2011) 2825–2830.