A python reimplementation of A. Sierk's BARFIT

G. Henning

▶ **To cite this version:**

G. Henning. A python reimplementation of A. Sierk's BARFIT. 2021. hal-03132426v1

HAL Id: hal-03132426
https://hal.science/hal-03132426v1

Preprint submitted on 5 Feb 2021 (v1), last revised 29 Sep 2022 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A python reimplementation of A. Sierk's BARFIT

Greg Henning (Université de Strasbourg, CNRS, IPHC UMR 7178)

February 5th, 2021

Abstract: *Between 1984 and 1986, A. J. Sierk developped the `BARFIT` routine to compute fission barrier, ground state energy, maximum angular momentum and moments of inertia in the framework of the liquid drop model for a large range of nuclei. The routine used fitted value over a wide range of A, Z and L and is still available today from the RIPL-3 website[1]. In this paper, I will decribe the re-implementation of the routine in python. This work is motivated by the difficulty to compile 1986's FORTRAN code on modern computers, the advantage of easily deploying python code over a wide range of infrastucture (personnal computers, computing grid, containers, …) and the preservation of A. Sierk's work for future use.*

History and motivation

In the 1980's, A. Sierk did publish a FORTRAN routine to calculate, on demand, liquid drop fission barriers (B_f), along with ground state energy (E_{gs}), maximum angular momentum (L_{max}) and moments of inertia ($I_{min,mid,max}$, given as a multiple of the moment of inertia for a rigid sphere ($I_{\text{rigid sphere}} = 2/5 M_0 R_0^2$ with $R_0 = 1.16 A^{1/3}$ fm and $M_0 = 913.5016 \times A - 0.511004 \times Z$ MeV) for a wide range of nuclei ($19 < \text{Z} < 111$). The value were obtained by a multi dimensionnal fit from calculated values[2].

One has to remember that at the time, there were no solution to either re-calculate the values on demand (microprocessors were usually 16 bits with clock frequency rarely above 50 MHz), or store and retrieve a large amount of data (the *HD* floppy disk introduced in 1986 had a capacity of 1.44 MB). Therefore, A. Sierk went for an elegant an efficient solution: finding the parameters for a global fit and having the value recalculated when needed. The FORTRAN code if very clear and is an impressive routine. Today, one would be using a "machine learning" model using an external dependecy acting as a *black box*, but this routine lays the logic bare and clear. For that reason alone, it is important to preserve the work of A. Sierk and continue to make it avaiblable.

---

[1] https://www-nds.iaea.org/RIPL-3/

[2] Macroscopic model of rotating nuclei, Arnold J. Sierk, Phys. Rev. C 33, 2039 (https://doi.org/10.1103/PhysRevC.33.2039)

In addition, one has to note that old FORTRAN code is not easy to compile on modern 2020's computers. The unmodified code returns errors when trying to compile it and it uses a lot of function defined as *archaic* when looking them up in a documentation. In today's context when a code maybe executde on a personnal computer, on a computing grid or in a container instance, having a python code that can be executed without any modification whatever the Operating System or machine characteristic is an advantage.

Of course, the base calculation of the liquid drop values [3] could be performed easily and quickly by modern computers . However, the implementation of this routine would be tedious (with integration, multiple parameters, …) [4] to obtain values that are alray available via this routine.

Reimplementation into python

The FORTRAN code is easily translated into Python [5],[6]. The reimplementation is therefore an almost line for line transcription of the original code. This makes it by some respect *non-pythonic*[7], but that ensures the accurate re-implemantation. The reimplemtation is actually done on a 1996 version of the code by A. Sierk, with improved Lmax parameters and calculation of moments of inertia [8].

The global fit routine uses Legendre Polynomials [9] values to increase the number of parameters from 3 (Z, A, L) to 23 (via values up to the 7th, 7th and 9th order Legendre polynomials). The `lpoly` function from the FORTRAN code was the trickiest to transcribe as FORTRAN's array are indexed from 1 and not 0 and the recursive relation with Legendre

---

[3] Cohen, S., Plasil, F., & Swiatecki, W. J. (1974). Equilibrium configurations of rotating charged or gravitating liquid masses with surface tension. II. Annals of Physics, 82(2), 557–596. doi:10.1016/0003-4916(74)90126-2

[4] Modern implementations that I am not aware of probaly exist. Additionally, calculations beyond the liquid drop model are available too (for example see also (e.g. https://t2.lanl.gov/nis/data/astro/molnix96/molnix.html).

[5] Python Software Foundation. Python Language Reference, version 3.6. Available at http://www.python.org

[6] "G. van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995."

[7] Style Guide for Python Code https://www.python.org/dev/peps/pep-0008/

[8] T. L. Khoo, private communication.

[9] Legendre, A.-M. (1782). "Recherches sur l'attraction des sphéroïdes homogènes" (PDF). Mémoires de Mathématiques et de Physique, présentés à l'Académie Royale des Sciences, par divers savans, et lus dans ses Assemblées (in French). X. Paris. pp. 411–435.

polynomials include the order of the polynomials. Initially, the output of the Python `lpoly` were compared to the values calculated with the `numpy` library[10]. Once the values by `lpoly` were correct, the dependence to `numpy` was removed. The rest of the implementation was easy and straighforward. The choice of keeping multi-dimensionnal arrays *flatten* instead of folding them back on their dimensions has been made for the sake of transcription simplicity. Future versions may include reformatted arrays and more *pythonic* expressions once the accuracy of the transcribed code is established and improved.

Study of differences

With some little modification and compiler options[11] on the GNU FORTRAN compiler [12], one can get the code to compile and compare the results over a wide range of values. . For this purpose, in addition to the 3 nuclei $^{58}_{28}$Ni, $^{139}_{65}$Tb and $^{229}_{93}$Np for which A. Sierk gives reference values in the comments of is code, 37 additionnal nuclei with significant natural abundance or of interest when it comes to fission has been studied: $^{45}_{21}$Sc, $^{48}_{22}$Ti, $^{51}_{23}$V, $^{52}_{24}$Cr, $^{55}_{25}$Mn, $^{56}_{26}$Fe, $^{59}_{27}$Co, $^{58}_{28}$Ni, $^{60}_{28}$Ni, $^{75}_{33}$As, $^{79}_{35}$Br, $^{81}_{35}$Br, $^{89}_{39}$Y, $^{93}_{41}$Nb, $^{103}_{45}$Rh, $^{127}_{53}$I, $^{133}_{55}$Cs, $^{141}_{59}$Pr, $^{151}_{63}$Eu, $^{153}_{63}$Eu, $^{153}_{65}$Tb, $^{159}_{65}$Tb, $^{165}_{67}$Ho, $^{169}_{69}$Tm, $^{175}_{71}$Lu, $^{181}_{73}$Ta, $^{197}_{79}$Au, $^{208}_{82}$Pb, $^{209}_{83}$Bi, $^{232}_{90}$Th, $^{230}_{91}$Pa, $^{233}_{92}$U, $^{235}_{92}$U, $^{238}_{92}$U, $^{229}_{93}$Np, $^{235}_{93}$Np, $^{239}_{94}$Pu, $^{241}_{94}$Pu, $^{242}_{95}$Am, $^{254}_{102}$No [13].

For each of these isotopes, the original fortan code and the Python implemtation were used to computer all the properties (B_f, E_{gs}, L_{max}, $I_{min,mid,max}$) from angular momentum 0 $\hbar$ to L_{max}. The result are then compared in groups for $L = 0\ \hbar$ and $L > 0\ \hbar$ (using a Pandas data frame[14]). In each group, we look at the mean and maximum relative $\left(\left(V_{FORTRAN} - V_{python}\right)/V_{FORTRAN}\right)$ and absolute $(|V_{FORTRAN} - V_{python}|)$. Also, we look at the fraction of results that present differences above the stated uncertainties of the code given by A. Sierk in the FORTRAN code. In particular, $U_{Bf} = 0.5$ MeV, $U_{Egs} = 0.5$ MeV, $U_{Lmax} < 0.5\ \hbar$ and $U_{MoI} = 1$ %.

---

[10] Harris, Charles R., K. Jarrod Millman, Stefan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, et al. 2020. "Array Programming with NumPy." Nature 585 (7825): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

[11] Modification of some arrays definition and and use of -std=legacy option for the compiler.

[12] "GFORTRAN, Gnu compiler collection (gcc)" Version 9.3.0

[13] J.K. Tuli, Nuclear Wallet Cards (NNDC, Brookhaven National Laboratory), 2005.

[14] Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, … Mortada Mehyar. (2020, March 18). pandas-dev/pandas: Pandas 1.0.3 (Version v1.0.3). Zenodo. http://doi.org/10.5281/zenodo.3715232

		L_{max}	B_f	E_{gs}	Moments of Inertia		
					I_{min}	I_{mid}	I_{max}
Ground state ($L = 0\ \hbar$)	Mean relative difference	0.000	0.000		0.000	0.000	0.000
	Maximum relative difference	0.003	0.001		0.001	0.000	0.000
	Mean absolute difference	0.027 $\hbar$	0.000 MeV		0.000 I_{rs}	0.000 I_{rs}	0.000 I_{rs}
	Maximum absolute difference	0.141 $\hbar$	0.005 MeV		0.000 I_{rs}	0.000 I_{rs}	0.000 I_{rs}
	Fraction above expected uncertainty	0.000	0.000		0.000	0.000	0.000
High L ($L > 0\ \hbar$)	Mean relative difference		0.091	0.019	0.000	0.000	0.000
	Maximum relative difference		0.979	0.484	0.050	0.024	0.079
	Mean absolute difference		0.484 MeV	0.048 MeV	0.000 I_{rs}	0.001 I_{rs}	0.000 I_{rs}
	Maximum absolute difference		3.050 MeV	3.411 MeV	0.030 I_{rs}	0.041 I_{rs}	0.257 I_{rs}
	Fraction above expected uncertainty		32.0 %	8.57 %	0.037 %	0.074 %	0.037 %

(L_{max} does not depend on the angular momentum, E_{gs} at $L = 0\ \hbar$ is always 0, I_{rs} stands for the Moment of Inertia of a rigid sphere.)

We notice that B_f and E_{gs} has significant discrepancy at higher momentum. This is not directly clear why. They are the variables most dependant on complex calculations and therefore, any divergence in floating point precision will have an impact. Additionnal study showed that it's value for $L > L_{max}/2$ that are the most affected. To this day, there is no eveident way to resolve the differences and they should be attributed to being at the limits in Z, A and L of the fit region. However, they concern only a small fraction of outputs.

Publication and availability

The code is hosted as a git repository on gitlab.in2p3.fr and published on hal.archives-ouvertes.fr with a DOI number[15]. For convenience, the module is made available on Pypi, and can be installed using the command `pip install fisbar`.

Additionnaly, a simple way to run the code for a given Z, A, and *L* is available via a notebook hosted on `binder`: Click on the 

link and once the notebook is started, run the only cell at the top with the `Execute` ▶ button (or the `Cell` menu and `Execute All`). The interface to enter your input Z, A and L will appear. Put in your parameters of interest and click the **Go** button. The result will be displayed below. If the calculation failed, the shown values will be **0**, **NaN** or **"**"**

Conclusions

Althought the interest of a routine to compute liquid drop fission barriers (and some associated values) from a mutli-fit performed in 1986 is of limited impact, the continued availability of such a function is important. First, because this value is a good starting point for the study of a nucleus, and the `BARFIT` routine by A. Sierk performs such calculation for a very wide range of Z. Second, the transcription to Python make this routine work on any platform that support `python` (so… almost everywhere) without modification. This makes it more portable than a FORTRAN code. Finally, it allows the preservation of an elegant piece of code. The general agreement between Fortan and Python implementation is good, in particular for low angular momentum. In some cases, the deviation can be significant, without a clear idea why. In the future, the Python code might be upgraded to be more *pythonic* and easier to understand and maintain.

Finally, the author wants to add a general comment: there might be a prejudice against old code that does not compile easily on modern system. However, there is value in it, despite the difficulties to make it run. Preserving past work via transcription in a more modern language (such as, for example, Python, julia[16], go[17] or even modern implementation of long time language like C++: C++17 with large use of iterators and such is quite different from the C++98 standart) is a good thing.

---

[15] Python implementation of A. Sierk's BARFIT https://hal.archives-ouvertes.fr/hal-03052073

[16] Julia: A Fresh Approach to Numerical Computing. Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral B. Shah. (2017) SIAM Review, 59: 65–98. doi: 10.1137/141000671. pdf.

[17] The Go Authors. The Go Programming Language Specification. https://golang.org/ref/spec, November 2016