SClib, a hack for straightforward embedded C functions in (I)Python

Esteban Fuentes [*] Hector E. Martinez [†]

Abstract

We present SClib, a simple hack that allows easy and straightforward evaluation of C functions within python code, boosting flexibility for better trade-off between computation power and feature availability, such as visualization and existing computation routines in SciPy. We also present two use cases for SClib.
In the first set of applications we use SClib to implement a Python script that solves efficiently the Schrödinger equation for bound-states in the context of particle physics. We present an introduction to the situations where this script have been used. We also describe the solution to the related problem of solving a set of coupled Schrödinger-like equations where SClib is used to implement the speed-critical parts of the code. We argue that when using SClib within IPython we can use NumPy and Matplotlib for the manipulation and visualization of the solutions in an interactive environment with no performance conceading.
The second case is an engineering application. We use SClib to evaluate the control and system derivatives in a feedback control loop for electrical motors. With this and the integration routines available in SciPy, we can run simulations of the control loop a la Simulink. The use of C code not only boosts the speed of the simulations, but also enables to test the exact same code that we use in the test rig to get experimental results. Again, integration with (I)Python gives us the flexibility to analyze and visualize the data.

1 SClib

At the core of SClib[1] is `ctypes` [1], which actually does the whole work: it maps Python data to C compatible data and provides a way to call functions in DLLs or shared libraries. SClib acts as glue: it puts things together for the user, to provide him with an easy to use interface.

The requirements for SClib are very simple: call a function on an array of numbers of arbitrary type and size and return the output of the function, again of arbitrary type and size.

The resulting interface is also very simple: A *library* is initialized in the python side with the path to the DLL (or shared library) and a list with the names of the functions to be called:

```
In [1]: import SClib as sc
In [2]: lib = sc.Clib('test.so', ['fun'])
```

The functions are then available as a members of the *library* and can be called with the appropriate number of arguments, which are one dimensional arrays of numbers. The function returns a list containing the output arrays of the function:

```
In [3]: out, = lib.fun([0])
```

In the C counterpart, the function declaration must be accompanied with specifications of the inputs and outputs lengths and types. This is accomplished with the helper macros defined in `sclib.h`:

```
#include <sclib.h>
      PYO(fun, 1,   1);
PYO_TYPES(fun, 1, INT);
      PYI(fun, 1,   1);
PYI_TYPES(fun, 1, INT);
void fun(int * out, int * in) {
    *out = 42;
}
```

An arbitrary number of inputs or outputs can be specified, for example:

```
#include <math.h>
#include <sclib.h>
      PYO(fun, 2,   1,     2);
PYO_TYPES(fun, 2, INT, FLOAT);
      PYI(fun, 2,   1,     2);
PYI_TYPES(fun, 2, INT, FLOAT);
void fun(int * out0, float * out1,
         int * in0, float * in1) {
    *out0 = 42*in0[0];
    out1[0] = in1[0]*in1[1];
    out1[1] = powf(in1[0], in1[1]);
}
```

In the function declaration, all the outputs must precede the inputs and must be placed in the same order as in the `PY` macros.

These specifications are processed during compilation time, but only the number of inputs and outputs is static, the lengths of each component can be overridden at run time:

[*]E. Fuentes is with the Electric Drives and Power Electronics group at the Technical University of Munich. 80333 Munich, Germany (email: esteban.fuentes@tum.de).

[†]H. Martinez is with the Physics Department T30F for Theory on Particles and Nuclear Physics at the Technical University of Munich. (email: hector.martinez@tum.de)

[1] The code for SClib and example use are availible at **https://github.com/drestebon/SClib**

```
In [4]: lib.INPUT_LEN['fun'] = [10, 1]
In [5]: lib.retype()
```

In these use cases the length of the arguments should be given to the function through an extra integer argument.

In the function body, both inputs and outputs should be treated as one dimensional arrays.

2 Applications quarkonium physics

2.1 Motivation

The Schrödinger equation is the one the fundamental equations in physics for describing non-relativistic quantum mechanical dynamics. For the applications we will present in this section we will focus on the time-independent version which in natural units is given by

$$\left(-\frac{\nabla_{\mathbf{r}}^2}{2m}+V(\mathbf{r})\right)\psi(\mathbf{r})=E\psi(\mathbf{r}). \tag{1}$$

It corresponds to an eigenvalue equation where the term inside the parenthesis in l.h.s. is called the Hamiltonian operator, the value E, its eigenvalue, is the measurable quantity (the energy) associated with it, m is the mass (or the reduced mass if the system formed by more than one particle) and the wavefunction, $\psi(\mathbf{r})$, is the entity containing all the information about the system, since its modulus squared correspond to the probability density of a given measurement, it has to be normalized to unity. The term $V(\mathbf{r})$ in the Hamiltonian is called the potential.
Since its discovery, the Schrödinger equation has played an important role in our understanding of nature and it is present in almost every aspect of modern physics. In this section we will review some cases where SClib has been used to implement solutions of the computing problems associated with eq. (1) that arise in the study of heavy quarkonia[2].
Quarkonium is a bound-state composed by a quark and its corresponding antiquark. By heavy we mean states composed by the charm and bottom quarks, called charmonium and bottomonium respectively. Due to its large mass, the top quark decays before forming a bound state. For heavy quarkonium the relative velocity between the quark and antiquark inside of the bound-system is believed to be small enough for the system to be considered, at least in a first approximation, non-relativistic, making it suitable for being described by eq. (1). Considering the equal mass case with a spherically symmetric potential, the angular part can be neglected (it correspond to the spherical harmonics) and the relevant part of eq. (1) reduces to the one-dimensional equation given by

$$\left[-\frac{1}{m}\frac{d^2}{dr^2}+\frac{l(l+1)}{r^2}+V(r)\right]y_{n,l}(r)=E_{n,l}y_{n,l}(r), \tag{2}$$

where r is the relative distance between the quark and the antiquark, l is the angular momentum quantum number, m is the (anti)quark mass, $y_{n,l}$ is called the reduced wavefunction and the eigenvalue $E_{n,l}$ is interpreted as the binding energy of the bound-system, where $n=0,1,2,\ldots$ accounts for the number of nodes (radial excitations) of the wavefunction. The total mass of the quarkonium is then given by

$$M=2m+E_{n,l}. \tag{3}$$

The potential $V(r)$ describes the quark-antiquark interaction, it is a function of r and $\Lambda_{\rm QCD}$, the typical hadronic scale ($\sim 200\,\text{MeV}$). For $r\Lambda_{\rm QCD}\ll 1$ (short-distance regime) the potential may be evaluated perturbatively, but for $r\Lambda_{\rm QCD}\gtrsim 1$ (long-distance regime) it cannot. To overcome this issue, models based on non-relativistic reductions of phenomenological observations have been used to describe heavy quarkonia, one these being the so-called Cornell potential [3–5])

$$V(r)=\frac{a}{r}+kr, \tag{4}$$

where a and k are unknown parameters which need to be fixed by experimental (or lattice) data of some observable. This potential incorporates two of the main observed characteristics of the quark-antiquark interaction: at short distances it exhibits a Coulombic behavior and in the long-distance regime the interaction is dominated by a confinement phase.
Since the beginning of the last decade, non-relativistic effective field theories (EFT), in particular non-relativistic QCD (NRQCD) [6,7] and potential NRQCD (pNRQCD) [9], have become the state-of-the-art tools for the study of heavy quarkonia (for review see [8]). NRQCD is obtained from QCD integrating out modes that scale like m, while pNRQCD is obtained from NRQCD integrating out modes that scale like the quark momentum[3]. The physics of the modes that have been integrated out is encoded in Wilson coefficients that must be calculated comparing at the same scale the results (observables, Green functions) of the EFT, with the ones of QCD (for NRQCD) or NRQCD (for pNRQCD). A key feature of pNRQCD is that it allows the relativistic corrections to the quark-antiquark potential to be organized as an expansion in powers of $1/m$, up to second order $V(r)$ can be written as

$$V(r)=V^{(0)}(r)+\frac{V^{(1/m)}(r)}{m}+\frac{V^{(1/m^2)}(r)}{m^2}, \tag{5}$$

where $V^{(1/m)}$ and $V^{(1/m^2)}$ are derived from QCD (through the matching procedure with NRQCD). The details about $V^{(1/m)}$ and $V^{(1/m^2)}$ and how they are obtained are beyond the scope of this document, however, we can list some of their features

- They correspond to Green functions that in the short-distance regime can be computed in perturbation theory.
- In the long-distance regime they can be computed in in lattice QCD, however only some of these Green functions have been already calculated.

[2]For a comprehensive review of the status and perspectives of the research in heavy quarkonia we refer the reader to chapter four of [2].

[3]These EFT exploit the hierarchy of energy scales present in the bound-system. If the relative velocity of between the quark and the antiquark, v, is small, we have that $mv^2(\sim E)\ll mv(\sim p)\ll m$, where p is the momentum of the particles and E its kinetic energy. If one is interested in studying a process that happen at the scale E (like the binding) it is more suitable to integrate out degrees of freedoms with energies that scale like the other two higher scales, this is the motivation behind pNRQCD. For a detailed analysis of the scales in heavy quarkonia we refer the reader to [8].

- Eq. (4) correspond, at least qualitatively, to the leading order $V^{(0)}$ in eq. (5)

For the details about the derivation of the terms present in eq. (5) we refer the reader to refs. [10] and [11]. It is important to recall that, although it can not be evaluated analytically in the whole range of r, eq. (5) represents a definite model-independent expression for the quark-antiquark potential, contrary to models like the one presented in eq. (4). Including the relativistic corrections to the potential the expression for the bound-state mass reads

$$\begin{aligned} M &= 2m + E^{(0)}_{n,l} + \frac{\langle nl|V^{(1/m)}(r)|nl\rangle}{m} \\ &+ \frac{\langle nl|V^{(1/m^2)}(r)|nl\rangle}{m^2} + \frac{1}{m^2}\sum_{m\neq n}^{\infty}\frac{|\langle nl|V^{(1/m)}|ml\rangle|^2}{E^{(0)}_{n,l}-E^{(0)}_{ml}}, \end{aligned} \tag{6}$$

where $E^{(0)}_{il}$ correspond to solve eq. (2) with $V(r) = V^{(0)}(r)$ and

$$\langle nl|f(r)|n'l'\rangle \propto \int_0^\infty dr\, y_{n,l}(r)f(r)y_{n'l'}(r), \tag{7}$$

where the proportionality factor will depended on the corresponding quantum numbers of the operators appearing in $V^{(1/m)}$ and $V^{(1/m^2)}$.

2.2 Applications of SClib

The simplest computational problem related to eq. (2) is to find $E_{n,l}$ for a given n and l. Methods to solve this problem have been implemented since long ago (see for instance [12]), in a nutshell, the standard method consist in to apply two known constraints to the reduced wavefunction $y_{n,l}$:

1. The number of nodes of $y_{n,l}(r)$ must be equal to n.
2. $y_{n,l}(r)$ has to be normalizable

$$\int_0^\infty dr[y_{n,l}(r)]^2 = 1. \tag{8}$$

In general $y_{n,l}(r)$ will diverge except when $E_{n,l}$ correspond to an eigenvalue. The procedure to find the eigenvalue consists in to perform a scan of values of $E_{n,l}$ until $y_{n,l}(r)$ has n nodes and converges for a large enough value of r (see fig. (1)). This implies that for each test value of $E_{n,l}$ eq. (2) must to be (numerically) solved. A popular[4] Mathematica [14] implementation of this method to solve eq. (2) has been available in [13]. This script has the advantage that the user can profit from the Mathematica built-in functions to plot, integrate or store the resulting wavefunctions, however, it has a very poor performance. With the goal of mimic the advantages of this script but without compromising in speed we have developed SChroe.py[5], a Python script that uses SClib to implement the speed-critical parts of the algorithm. In this script the wavefunctions are stored as NumPy arrays [18] so when the script is

[4]The paper describing the script ranks fifth among the most cited papers (91 citations) of the International Journal of Modern Physics C with the last citation from July 2014.

[5]Code available in `https://github.com/heedmane/schroepy/`

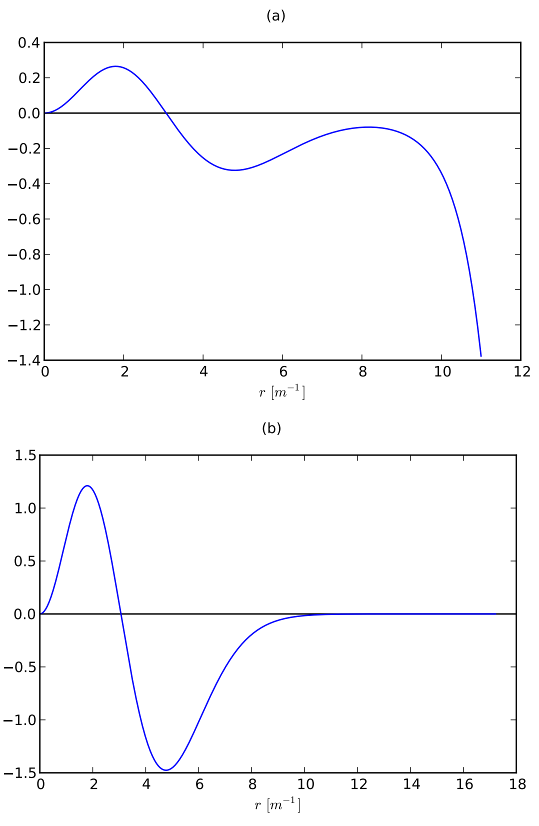

Figure 1: Reduced wavefunctions $y_{n,l}(r)$ for two steps in the search of the eigenvalue $E_{n=1\,l=1}$. For each step in the process to find the eigenvalue the nodes of the $y_{n,l}$ are counted, only when the value of $E_{n,l}$ correspond to an eigenvalue $y_{n,l}$ is not divergent. In the plot (a) $E_{n,l} = 3.1\,m$ and $y_{n,l}$ fulfills the condition of have one node, however, the accuracy in the value of $E_{n,l}$ is too low and the function diverges. In the plot (b) $E_{n,l} = 3.10952\,m$ so $y_{n,l} \to 0$ for larger values of r. We have used the Cornell potential eq. (4) with parameters $m = 1$ $a = 0.1$, $k = 0.5m^2$, all dimensions defined in terms of the mass.

run within IPython [22] together with SciPy [17], NumPy and Matplotlib [21] the user can profit of the same or more flexibility as with the Mathematica script plus a boosted speed. In table 1 we compare the performance of SChroe.py against other implementations of the same algorithm.

In [15] SChroe.py has been used to evaluate the relativistic corrections to the mass spectrum of quarkonium in the long-distance regime. In that paper the relativistic corrections $V^{(1/m)}$ and $V^{(1/m^2)}$ appearing in (6) were evaluated assuming the hypothesis that in the long-distance regime the interaction between the quark and the antiquark can be described by a string. In fig. (2) we show some of the energy levels (masses) corresponding to the string spectrum. It is noteworthy to mention that all the numerical calculations and plots of that paper were done with (I)Python using the SciPy library.
An application in which the speed of SChroe.py plays an important role is fixing the parameters of the potential given some experimental input. For instance, consider the problem of finding the parameters a and k of eq. (4) together with m, given the experimental values of the masses of three

n	$E_{n,l=1}$ $[m]$	schroe.nb [13]	Python	SChroe.py
0	2.15789	98.88	25.46	11.11
1	3.10952	124.14	30.95	12.65
2	3.93850	135.68	35.32	14.93
20	13.5995	370.0	88.04	32.13

Table 1: Time in seconds taken to compute the eigenvalues and reduced wavefunctions for the Cornell potential eq. (4). The column Python correspond to the implementation of the algorithm in Python without using SClib. The parameters of the potential are the same as in fig. (1). All the scripts were tested in the same machine, a notebook with a 2.4 Ghz core i5 processor (dual core) and 8 GB of RAM.

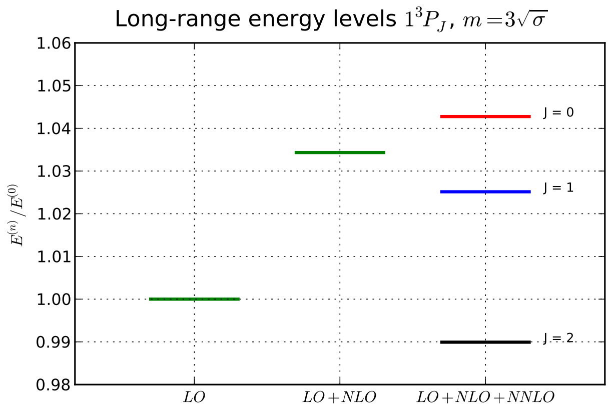

Figure 2: Long-range energy levels of the first triplet quarkonium state. The lines are calculated from eq. (6) using the relativistic corrections derived from the string hypothesis [15]. The leading order (LO) correspond to eq. (4) setting $a = 0$ and $k = 1$ (in the plot labeled σ) and $m = 3\sqrt{k}$. This plot shows the relative size of the next-to-leading-order (NLO) correction (the term proportional to $1/m$ in the r.h.s. of eq. (6)) and the newly computed next-to-next-to-leading-order (NNLO) corrections (the terms proportional $1/m^2$). For more details see [15].

different quarkonium states. If relativistic corrections are included, in order to find the parameters we must solve a system of three equations like eq. (6). For each probe value of (a, k, m) we have to find the eigenvalues and reduced wavefunctions of eq. (2) and then with these values evaluate the sums and integrals in (6). A parameter fixing of this type was necessary to implement in [16]. The implementation has been carried out using SChroe.py together with a mixture of C and SciPy functions using SClib to link both environments[6].
Another related computational problem that arises from the study of heavy quarkonium hybrids, bound-states composed by a quark-antiquark pair plus an exited gluon, is to solve a system of N Schrödinger-like coupled equations. Explicitly the system to solve reads

$$\left(-\frac{\delta_{ij}}{m}\frac{d^2}{dr^2} + V_{ij}(r,l)\right) u_{j,(n,l)}(r) = E_{n,l}\, u_{i,(n,l)}(r), \qquad (9)$$

[6]Some of the code will be available once the paper appear online

where $i = 1, 2, ..N$ and the angular momentum dependence has been included in the potential matrix. A method to solve this equation for the case $N = 2$ has been implemented in [19]. The method relies on an extension of the nodal theorem [20] and convergence conditions for the components of the vector wavefunction $u_{j,(n,l)}(r)$. The extension of the nodal theorem states that the number of nodes of the determinant of the matrix $U_{n,l}(r)$, whose columns are N lineal-independent solutions of eq. (9), is equal to n. The procedure then consist in a scan of values $E_{n,l}$; in each step the set of equations (9) is solved and the nodes of $|U_{n,l}(r)|$ are counted for a large enough interval of r. As in the one-dimensional case, if $E_{n,l}$ approached to an eigenvalue the components of $u_{j,(n,l)}$ converge for large r. In the solution presented in [19] the performance-intensive parts of the implementation rely on C functions linked to the (I)Python interface trough SClib.
As an example of the application of the method implemented in [19], in fig. (3) we show the results for the search of the first two eigenvalues and wavefunctions with the matrix potential given by

$$V_{ij}(r,l) = \begin{pmatrix} \frac{l(l+1)+2}{mr^2} + F_0(r) & -\frac{2\sqrt{l(l+1)}}{mr^2} \\ -\frac{2\sqrt{l(l+1)}}{mr^2} & \frac{l(l+1)}{mr^2} + F_1(r) \end{pmatrix} \qquad (10)$$

where

$$F_i(r) = \ln(a_i + b_i r). \qquad (11)$$

In all the applications described in this section the combination of SClib and the SciPy library within IPython provided a powerful interactive environment based entirely on open source software for solving problems that require a high performance and visualization tools.

3 Application in Control Engineering

Most control systems have the structure depicted in Fig 4. G is the plant, it represent the natural phenomena we wish to control. We usually describe it using ordinary differential equations:

$$G: \begin{cases} \frac{dx}{dt} & = f(x, u, d) \\ y & = c(x, u, d). \end{cases} \qquad (12)$$

x represents the internal state of the plant and y its output (the measurements). d is an independent variable, usually not measurable, named the perturbation and u is the actuation: the degree of freedom used by the controller C to achieve the control goal r. In general the controller is a function of the measurements and the reference r:

$$C: \ u = \pi(y, r),$$

but it also may comprise internal states. They are commonly used to reconstruct the state x out of the history of y and u. The latter systems are called state observers and the whole is called feedback control.

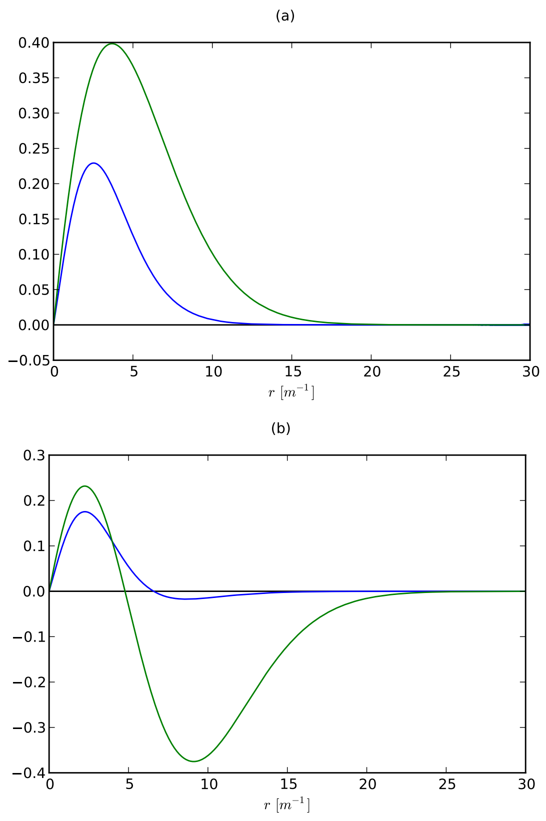

Figure 3: Solutions for the components of the vector wave-function $u_{n,l}(r)$ for the first two eigenvalues ($l = 1$) of eq. (9) with the matrix potential given in (10). We have used $m = 1$, $a_0 = 1$, $b_0 = 0.5$, $a_1 = 2$ and $b_1 = 0.1$. The eigenvalues are $E_{n=0,l=1} = 1.01727\,m$ for fig. (a) and $E_{n=1,l=1} = 1.18789\,m$ for fig. (b).

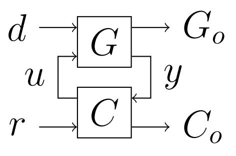

Figure 4: General scheme of a control system.

We use SClib to put together a simulator for these kind of systems. Both the system derivatives $f(\cdot)$ and the control $\pi(\cdot)$ are written in C and are evaluated using SClib. As stated before, the system state represents a natural phenomena, therefore it is natural to describe it as a continuous time variable, as eq. (12) suggests. To calculate the system state we have to solve this equation. In our simulator this is achieved using numerical methods, namely the integration routines available in `scipy.integrate`. On the other hand, the controller is usually implemented in a real-time computer, which can only sample y at a fixed interval (called h): it is a discrete-time system. This means, that the simulator only needs to evaluate $\pi(\cdot)$ at given times.

Traditional controllers took the form of linear filters, which could even be implemented using analog circuitry. As control techniques and requirements advance, more complex controllers are devised. Many modern control techniques are based on optimization methods. Time-optimal controllers, for example, require the solution of an usually very complex optimization problem, to find a control u that leads the system state x towards its target r in minimum time [23]:

$$u^* = \pi^*(x) = \underset{\pi\in U,\, x\in X}{\arg\min} \left\{T_x(u)\right\}. \qquad (13)$$

Here $T_x(u)$ is the time required to lead x towards its target and X and U are the regions where we want x and u to be confined, they constitute the constraints for the control problem. These kind of controllers require exhaustive computation and it is natural to implement them in C.

For motivation, we present the results for a minimum-time control strategy for a relatively simple and well known problem, the double integrator [24]:

$$\frac{d}{dt}\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{pmatrix} u/\tau_0 \\ x_0/\tau_1 \end{pmatrix}. \qquad (14)$$

The relevance of this system lays in that it models many mechanical systems: u, x_0 and x_1 may represent acceleration, speed and position, for example.

Fig. 5 presents a minimum time control strategy for this system. The form of $\pi(x)$ for this case reveals its non-linear nature.

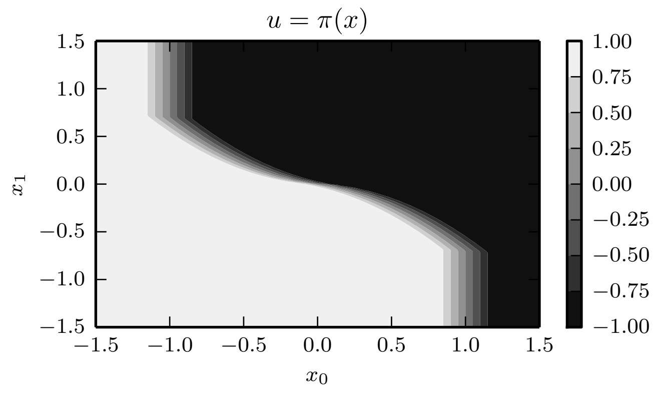

Figure 5: Time optimal control for the double integrator considering $\tau_0 = \tau_1 = 5$, $u \in [-1, 1]$, $h = 1$ and $x \in [-1, 1] \times \mathbb{R}$.

Fig. 6 presents the trajectory developed by the state using this control strategy and random initial conditions.

These results were obtained using SClib and the devised simulator. The example code is ready to reproduce them.

The main advantage we obtained from this work was that, since we were using a Linux based real time system in our test rig, we could use exactly the same code for the simulations and the experimental tests. Another feature of this work is that it effectively replaces Simulink in all of our use cases using only free software.

We hope the applications of SClib scope beyond the ones listed in this paper since we believe it provides a simple but powerful way to boost (I)Python performance.

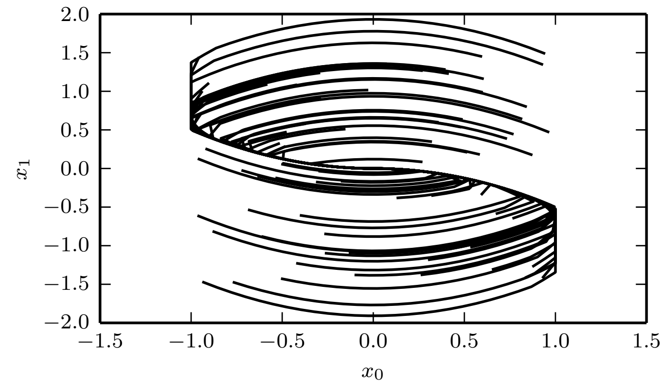

Figure 6: Time optimal trajectories for the double integrator, with random initial conditions.

Acknowledgments

H.M. acknowledges financial support from DAAD and the TUM Graduate School during the realization of this work.

References

[1] Heller. The `ctypes` module. `https://docs.python.org/3.4/library/ctypes.html#module-ctypes`

[2] N. Brambilla, S. Eidelman, P. Foka, S. Gardner, A. S. Kronfeld, M. G. Alford, R. Alkofer and M. Butenschoen *et al.*, arXiv:1404.3723 [hep-ph].

[3] E. Eichten, K. Gottfried, T. Kinoshita, J. B. Kogut, K. D. Lane and T. M. Yan, Phys. Rev. Lett. **34**, 369 (1975) [Erratum-ibid. **36**, 1276 (1976)].

[4] E. Eichten, K. Gottfried, T. Kinoshita, K. D. Lane and T. M. Yan, Phys. Rev. D **17**, 3090 (1978) [Erratum-ibid. D **21**, 313 (1980)].

[5] E. Eichten, K. Gottfried, T. Kinoshita, K. D. Lane and T. M. Yan, Phys. Rev. D **21**, 203 (1980).

[6] W. E. Caswell and G. P. Lepage, Phys. Lett. B **167**, 437 (1986).

[7] G. T. Bodwin, E. Braaten and G. P. Lepage, Phys. Rev. D **51**, 1125 (1995) [Erratum-ibid. D **55**, 5853 (1997)] [hep-ph/9407339].

[8] N. Brambilla, A. Pineda, J. Soto and A. Vairo, Rev. Mod. Phys. **77**, 1423 (2005) [hep-ph/0410047].

[9] N. Brambilla, A. Pineda, J. Soto and A. Vairo, Nucl. Phys. B **566**, 275 (2000) [hep-ph/9907240].

[10] N. Brambilla, A. Pineda, J. Soto and A. Vairo, Phys. Rev. D **63**, 014023 (2001) [hep-ph/0002250].

[11] A. Pineda and A. Vairo, Phys. Rev. D **63**, 054007 (2001) [Erratum-ibid. D **64**, 039902 (2001)] [hep-ph/0009145].

[12] P. Falkensteiner and H. Grosse and F. Schöberl and P. Hertel Comput. Phys. Comm. **34**, 287 (1985)

[13] W. Lucha and F. F. Schoberl, Int. J. Mod. Phys. C **10**, 607 (1999) [hep-ph/9811453].

[14] Wolfram Research, Inc. Mathematica Version 9.0 (2012)

[15] N. Brambilla, M. Groher, H. E. Martinez and A. Vairo, arXiv:1407.7761 [hep-ph].

[16] N. Brambilla, H. E. Martinez and A. Vairo, TUM-EFT 40/13, In preparation.

[17] Eric Jones and Travis Oliphant and Pearu Peterson and others `http://www.scipy.org/` (2001–)

[18] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, **13**, 22-30 (2011)

[19] M. Berwein and H. E. Martinez, TUM-EFT 48/14, In preparation.

[20] H. Amann and P. Quittner, Journal of Mathematical Physics **36**, 4553 (1995), doi:10.1063/1.530907.

[21] John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, **9**, 90-95 (2007)

[22] Fernando Pérez and Brian E. Granger. IPython: A System for Interactive Scientific Computing, Computing in Science & Engineering, **9**, 21-29 (2007)

[23] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms* . Springer-Verlag, 2011.

[24] E. Fuentes, D. Kalise, J. Rodriguez, and R. Kennel, “Cascade-free predictive speed control for electrical drives,” *Industrial Electronics, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.