**A Python implementation of a Steady-state Real Time Optimization (SRTO) and Realtime optimization with persistent adaptation (ROPA)**

Carlos C. Sanza, Galo A.C. Le Rouxa

a Department of Chemical Engineering, Polytechnic School, University of São Paulo Av. Prof. Lineu Prestes, 580, Bloco 18, São Paulo, Brasil, galoroux@usp.br

Abstract

In recent years, the chemical/petrochemical industry has been under increasing pressure to optimize its processes. Economic, environmental, Industry 4.0, and circular economy considerations are some of the main drivers for this optimization. However, most companies rely on expensive software for process simulation and optimization algorithms. For leading chemical/petrochemical companies, this type of investment is not a problem. In some companies, an entire team is dedicated to this task. However, for medium-sized or small companies that are subject to the same constraints, such as environmental compliance or the most profitable operating point, the burden can be very high. Unfortunately, there are not many low-cost solutions available. In addition, in some cases, optimization should be performed in a small portion of the process, such as just a reactor or a distillation train. And, often, small companies tend to oversize the problem and the involved costs.

**Keywords**: real-time optimization, python, real-time optimization persistente adptation

* 1. Introduction

SRTO requires the detection whether the system is at steady-state, it a complex task. One way to overcome the limitations of steady-state detection, is to use an alternative RTO method named Real-time Optimization with Persistent Adaptation (ROPA).Unlike standard RTO, which waits for a confirmed steady-state before updating model parameters, ROPA continually adjusts the model parameters using real-time online estimation methods, treating transient measurements as if they represent a steady-state condition. The goal is not to achieve continuous optimization per se, but to improve set-points over time until they align with the actual steady-state optimum, MATIAS, J. O. A , 2018

ROPA serves as an intermediary between static RTO and dynamic optimization approaches like Dynamic Real Time Optimization (DRTO) and Economic Model Predictive Control (EMPC), which are theoretically appealing but seldom implemented on a large scale due to the lack of comprehensive models for complex processes. By decoupling the estimation problem, ROPA allows for the use of well-established stationary economic optimization software and literature, while enabling asynchronous updates of the plant-wide model, tailored to sections with varying parameter update frequencies. This decoupled approach facilitates solving the steady-state optimization problem at any desired frequency, J. Matias, et al., 2022.

However, we aim at using simple and readably available Python libraries to check the possibilities of implementing such scheme in a smaller scale trying to bring optimization to small companies at a low cost. As far as we could find in the open literature, all Python libraries used in this paper have a permissive license, such as the Apache License 2.0, MIT License, BSD 3-Clause License. There were no changes or modifications on any library, the libraries were used “as it is”.

* 1. Cases

We used two different problems, a simple binary column system and the well-known Williams-Otto reactor, which is part of the Willliams-Otto plant model, Williams, T. J., and Otto, R. E. 1960. The process for the Williams-Otto reactor system can be seen on Figure 1a. It is a CSTR with two pure inlet products A and B performing 3 simultaneous reactions, equations (1), (2) and (3).

r1 : A + B 🡪 C (1)

r2 : B + C 🡪 P + E (2)

r3 : C + P 🡪 G (3)

It was assumed that the temperature Tr, of the CSTR could be changed instantly at any given time. Also, the flow rate of feed stream B can be controlled. On the other hand, the flow rate of A has only an indication and can have a step change at any time. Species E and P are measured at the

|  |
| --- |
| Figure 1a Williams-Otto reactor diagram |

reactor’s outlet. There are no measurements for species A, B, C and G. The reactor system can be modeled by a set of 6 differential equations and one algebraic equation, as shown in Figure 1b. Table 1 shows the values of the constants for that system.

|  |
| --- |
| Figure 1b – Model of the Williams-Otto reactor |

|  |
| --- |
| Table 1 – Reaction rate constants |
|  | ***A***i | ***E***i |
| **r1** | 1.6599 e6 | 6666.7 |
| **r2** | 7.2177e8 | 8333.3 |
| **r3** | 2.6745e12 | 11111 |

The binary distillation column follows what was described by S. Skogestad, 1997. The binary column system can be modeled by a set of differential equations. An extensive and comprehensive explanation may be found in S. Skogestad, 1997 and at his site. It is an excellent reference for this problem. Another reference may be found in W. L. Luyben, 1986, the author presents all material and energy balances required for a multicomponent distillation column.

A few considerations about the distillation column problem : a) It was assumed that the relative volatility of the two components remains constant throughout the column, b) no energy balance was performed, c) the vapor flow of the column was assumed constant. We must reinforce that the aim of this work was to verify if it was possible to simulate SRTO and ROPA with available python libraries, at this time the complexity of the model was not a concern. Clearly, for a more precise work, the model would have to be as complex as required, with a good physical properties package, energy balances, multicomponent, tray efficiencies, and so on.

|  |
| --- |
| Figure 2: Binary Distillation Diagram |

 The main purpose for the column problem was to assess the feasibility of using python to simulate a model and to verify cpu time with two different approaches (with and without jax library)

**3-The optimization problem**

|  |
| --- |
| Figure 3: SRTO and ROPA Diagram |

Both ROPA and SRTO are well-known control/optimization schemes and have been extensively described in literature, such as in J.O.A. Matias (2018). Figure 3 provides a clear schematic representation of these schemes. In this work, we simplified the scheme by assuming a perfect controller and that the steady state could be attained quickly, with the total simulation spanning an 8-hour period. For the Williams-Otto reactor, we developed a function to evaluate the SRTO – MPA (Model Parameter Adaptation) for the unmeasured parameter, specifically the stream flow rate of component A.

**4-Implementation**

In addition to code the model of the Williams-Otto reactor and the binary distillation column, we had to code for the ROPA case: a) an Extended Kalman Filter function, which is the state and parameter estimator for the ROPA measured variables prediction (xe and xp) and unmeasured variables evaluation(xa,xb,xc,xg and Fa; b) the profit optimization function and, for the SRTO case; c) evaluation of the unmeasured parameters, Fa (MPA). Basically we used the following Python libraries: jax.jacrev function that calculates the Jacobian of a system of equations by the reverse mode; jax.jit a function that compiles specific parts of the code increasing the cpu speed ; numpy a multipurpose library of the most important functions; scipy.integrate.solve\_ivp a function to integrate initial value differential equations; matplotlib.pyplot a plotting dedicated library; scipy.optimize library for optimization of nonlinear problems; jax.experimental.ode (odeint) a function to integrate initial value differential equations. It would be very helpful if an easy installation and powerful orthogonal collocation library for python was readably available. There are some libraries for instance, SUNDIALS from Lawrence Livermore National Security and Southern Methodist University. However, its installation in a python environment is not straightforward. As python is a collaborative and open environment its almost certain that other libraries will follow.

|  |
| --- |
| Figure 4: model output and output plus the gaussian noise |

To resolve the steady-state problem, we utilized our implementation (NRDA.py) of the Newton-Raphson algorithm, integrating JAX for automatic differentiation and JIT compilation. While Python code is typically interpreted, JAX's JIT feature compiles parts of the code, thereby substantially increasing execution speed and reducing computation time. Additionally, we developed Python code for an Extended Kalman Filter. The SRTO implementation includes a subroutine (model.py) containing the system of differential equations. This subroutine is used for steady-state calculations with the Newton-Raphson method (NRDA.py), setting the time-dependent derivative to zero, and for dynamic behavior using scipy.integrate.solve\_ivp and jax.experimental.ode (odeint)

|  |
| --- |
| Figure 5: Flow rate estimated by EKF and MPA |

The process measurement was simulated using the process model (model.py) by adding Gaussian white noise (with zero mean) to the dynamic or steady-state simulation, as applicable. For example, Figure 4 for the Williams-Otto reactor displays both the model output and the output with added Gaussian noise. The plot illustrates the mass fractions of components E (xe) and P (xp), with Figure 4 depicting a step changes at 2.0 hours (to 2.3 kg/s), 4.0 hours (to 1.7 kg/s), and 6.0 hours (to 2.0 kg/s) over an 8-hour time horizon. In this plot, the solid line represents the model’s prediction, while the dots indicate measured values. Figure 5 presents the model's behavior for the flow rate of component A alongside estimates from the implemented Model Parameter Estimation and the Extended Kalman Filter. The lag observed for MPA is minimal, though it would likely be longer in a real-world scenario. Notably, the estimates align closely with the model, even when the magnitude of the white noise increases. The reactor temperature and inlet flow rate of component B are setpoints for the controller, calculated by optimizing the objective function (equation 4), subject to constraints of the steady-state model and bounds for both variables.

$Profit=1143.38 ×xp×Fr+25.92×xe×Fr-72.63×Fa+114.34×Fb$ (4)

The objective function is subject to the system of equations constraints (shown in Figure 1b) and, it is required to satisfy the following bounds: Lower Bound temperature = 25°C, Lower Bound for B flow rate = 2 kg/s, Upper Bound temperature = 150°C, and Upper Bound for B flow rate = 8 kg/s.

The ROPA implementation scheme started with the same set of measured variables.

|  |
| --- |
| Figure 6: Instantaneous Profit. |

The first step was to initialize the Extended Kalman Filter. For the Process noise covariance (Q) we use 0.5 for both variables (E and P). For the Measurement noise covariance (R) we used 0.001 for the Gaussian white noise. Figure 6 shows de final result, with MPA and the “traditional RTO” and with ROPA and the EKF filter.

**5 – Conclusion**

Simulating SRTO and ROPA processes using Python is indeed feasible. However, this approach presents several challenges that must be addressed. Utilizing JAX offers significant benefits, such as automatic differentiation and the capability for JIT compilation, resulting in faster execution. It is important to note, however, that JAX is specifically designed for 'pure' functions.

As a result, the use of certain Python libraries, like scipy.optimize and solve\_ivp, is not recommended within a pure JAX environment. The jax.experimental.odeint function for integration, while available, is still in the experimental phase. Similarly, JAX's experimental optimization function lacks straightforward parameter passing, complicating its use. Python provides various packages for optimization and integrating systems of differential equations, yet some of the most powerful ones can be difficult to install. In terms of CPU time, functions utilizing JAX and JIT demonstrate exceptional performance, with improvements ranging from 5 to 15 times faster than non-compiled code. Implementing MPA or the Extended Kalman Filter did not pose significant issues, and the optimization of profit functions or parameter estimation for MPA proved to be effective.

Future work will involve integrating freeware software to simulate unit operation models and physical properties packages. This integration is crucial for obtaining suitable models if one intends to implement control systems effectively.

**References**

* Williams, T. J., and Otto, R. E. 1960. “A Generalized Chemical Processing Model for investigation of Computer Control.” AIEE Trans 79: 458-68.
* S. Skogestad, 1997. Dynamics and control of distillation columns: A tutorial introduction. Chemical Engineering Research and Design 75 (6), 539-562.
* W. L. Luyben, 1986. Process Modeling, Simulation, and Control for Chemical Engineers”, McGraw-Hill Book Company.
* Chun Y Chen, 1987. On-line Optimization Using a Two-Phase Approach: An Application Study. Ind. Eng. Chem. Res 26.
* Amrit R., Rawlings, J. Biegler, 2014. Optimizing Process Economics Online Using Model Predictive Control. Computers and Chemical Engineering 58
* MATIAS, J. O. A , 2018: Real-time Optimization with persistent parameter adaptation using online parameter estimation. PhD Thesis – Escola Politectica da USP
* CARNEIRO, A. A. B., 2018:Application of Real-time Optimization with Persistent Parameter Adaptation (ROPA) to Processes using Online Parameter Estimation.
* J. Matias, G. A. Le Roux, 2018. Real-time optimization with persistent parameter adaptation using online estimation parameters. Journal of Process Control 68, 195-204
* J. Matias, et al., 2022. Steady-state real-time optimization using transient measurements on an experimental rig. Journal of Process Control 115, 181-196.
* J. Matias, et al., 2022. State and Parameter Estimation in Dynamic Real-time Optimization with Closed-loop Prediction. ESCAPE 2022 – Athens