Systems & Control Transactions

Original Research Article
Peer Reviewed Conference Proceeding
ESCAPE 36 - European Symposium on Computer Aided Process Engineering
Sheffield, UK. 21-24 June 2026

PSE PRESS

A Unified Python/JAX Framework for Thermodynamic Modeling, Nonlinear Solvers, and DAE Solution of Hydrocarbon Systems

Carlos C. Sanz[a] and Galo Le Roux[a]*

[a] Universidade de São Paulo, Chemical Engineering Department, São Paulo, Brazil
* Corresponding Author: galoroux@usp.br

ABSTRACT

Dynamic simulation of distillation columns and chemical reactors remains essential for plant design, controllability analysis, and economic optimization. High-purity separations of close-boiling mixtures present significant computational challenges due to nonlinear thermodynamic behavior and stiff differential-algebraic equation (DAE) systems. This work presents a unified Python/JAX framework integrating four computational modules: (1) Peng-Robinson thermodynamics with complex-step differentiation, (2) nonlinear solvers (Newton, Broyden, Newton-Krylov) with automatic Curtis-Reid scaling, (3) DAE solver with Radau IIA collocation and intelligent auto-selection, and (4) constrained optimization using the Augmented Lagrangian Method with JAX automatic differentiation. The framework leverages JAX's just-in-time compilation (JIT), vectorization (vmap), and automatic differentiation (AD) to achieve near-compiled-language performance. Validation includes: nonlinear solver benchmarks with Newton-Krylov achieving 100% success across seven problems (n=2 to 5000), Williams-Otto reactor optimization with 0.06% deviation from published literature and 1325× real-time speedup, and a 180-stage propylene-propane splitter with 18× real-time performance under three concurrent disturbances. The framework shows that is possible to have an open-source alternative suitable for real-time optimization, operator training, and advanced process control applications.

Keywords: Python, JAX, Process Simulation, Nonlinear Solvers, DAE Systems, Optimization, Distillation

1. INTRODUCTION

Dynamic modeling of chemical processes supports applications from conceptual plant design to real-time optimization and operator training. However, several technical challenges complicate simulation of hydrocarbon separation processes. High-purity product specifications (>99%) demand rigorous thermodynamic models that accurately capture vapor-liquid equilibrium. The resulting mathematical models are differential-algebraic equations (DAE) systems in which algebraic constraints (phase equilibrium, summation) couple with differential mass balances. Large industrial distillation columns (50-200 stages) produce systems with thousands of coupled equations, necessitating efficient sparse linear algebra.

This work addresses these challenges using Python with JAX [1, 2], providing: (1) automatic differentiation for exact Jacobians and gradients, (2) just-in-time compilation via XLA [2], achieving near-C++ performance, (3) vectorization through vmap for batch processing, and (4) functional purity ensuring reliable transformations.

The framework comprises four modules: **thermodynamics** (Peng-Robinson with complex-step differentiation), **nonlinear solvers** (Newton/Broyden/Newton-Krylov with Curtis-Reid scaling), **DAE solver** (Radau IIA with auto-selection), and **optimization** (Augmented Lagrangian with JAX AD).

Sections 2–5 present methods, Section 6 validates via benchmarks and applications, and Section 7 concludes.

Table 1: JAX Commands Reference.

Command	Description	Application	Example
@jit	**JIT compilation to machine code via XLA**	**All computational kernels**	**@jit def f(x): return x**2**
vmap(f, in_axes)	Vectorize function over batch dimension	Multi-stage VLE, collocation	vmap(stage_func)(stages)
jacfwd(f)(x)	Forward-mode AD: Jacobian via n forward passes	Newton solver, constraints (n_out ≥ n_in)	J = jax.jacfwd(F)(x)
grad(f)(x)	Reverse-mode AD: gradient via 1 forward + 1 backward	Scalar optimization objectives	g = jax.grad(f)(x)
jvp(f, (x,), (v,))	Jacobian-vector product J·v	Newton-Krylov matrix-free solver	_, Jv = jax.jvp(F, (x,), (v,))
jnp.*	JAX NumPy functions (GPU/TPU compatible)	All array operations	jnp.sum(x), jnp.linalg.solve(A, b)

2. THERMODYNAMIC MODEL

2.1 Peng-Robinson Equation of State

The Peng-Robinson equation of state [3] relates pressure P, temperature T, and molar volume V. Parameters a(T) and b are computed from critical properties (Tc, Pc) and acentric factor ω [4]. For mixtures, van der Waals mixing rules employ binary interaction parameters kij fitted to experimental VLE data. Fugacity coefficients φi enable K-value calculations: $K_i = \varphi_i^L / \varphi_i^V$. Complete derivation in [3]; property calculations following [4].

2.2 Complex-Step Differentiation

Computing derivatives for Newton-Raphson VLE solvers requires careful consideration of JAX transformation composition. Using JAX automatic differentiation inside PREOS functions would create nested JIT/vmap contexts when called from DAE solvers, causing exponential memory growth and compilation overhead.

Complex-step differentiation [12] computes derivatives via $f'(x) = \mathrm{Im}[f(x + ih)]/h$ with $h \approx 10^{-20}$, achieving machine precision (10^{-16} accuracy). This approach operates as pure arithmetic without additional JAX transformations, avoiding nested issues while maintaining exact derivatives. Care was taken to contain complex dtype within thermodynamic kernels through dtype-agnostic implementations. Complete mathematical treatment in [12].

2.3 JAX Implementation

All thermodynamic kernels use @jit for compilation (10–50× speedup). Vectorization via vmap enables batch VLE calculations across stages (8–15× speedup). Precision enforced at 64-bit: jax.config.update("jax_enable_x64", True).

3. NONLINEAR EQUATION SOLVERS

3.1 Problem Formulation

General nonlinear system: $F(x) = 0$ where $F: \mathbb{R}^n \to \mathbb{R}^n$. Jacobian computed via forward-mode AD: J = jax.jacfwd(F)(x) evaluates n forward passes for exact derivatives.

3.2 Newton Method

Newton iteration: $J(x_k)\Delta x_k = -F(x_k)$, $x_{k+1} = x_k + \alpha_k \Delta x_k$. Three variants implemented: pure (α=1), line search (backtracking Armijo condition), dogleg (trust-region [10]). Cost: $O(n^3)$ due to LU decomposition.

3.3 Broyden Quasi-Newton

Broyden's method [10] uses rank-1 Jacobian approximation where $s_k = x_{k+1} - x_k$, $y_k = F(x_{k+1}) - F(x_k)$. Periodic Jacobian refresh (every 10–20 iterations) improves convergence. Cost: $O(n^2)$ per iteration.

3.4 Newton-Krylov Method

For large sparse systems, Newton-Krylov [6] solves the Newton equation using GMRES (Generalized Minimal Residual method for linear systems [6]) without forming the explicit Jacobian. Matrix-vector products computed efficiently via JVP. For n=5000, jax.jvp evaluates J·v in O(n) time with single forward pass, versus $O(n^2)$ to form explicit Jacobian. Eisenstat-Walker forcing terms [7] adapt GMRES tolerance dynamically.

Parameter sensitivity: Requires tuning GMRES restart (10–40), linear solver tolerance (1e-4 to 1e-6), and preconditioner choice. Framework uses conservative defaults (restart=40, tolerance=1e-4, no preconditioner) ensuring reliability without manual tuning. Cost: O(n) per iteration for sparse systems.

3.5 Curtis-Reid Scaling

Ill-conditioned systems with variables spanning multiple orders of magnitude cause convergence failure. Curtis-Reid scaling [5] addresses this through diagonal scaling. Activated when conditioning ratio $\kappa = \max(Dx)/\min(Dx) > 1000$. Scaled system: $\tilde{F}(\tilde{x}) = F(Dx \cdot \tilde{x})/Df = 0$ is solved iteratively.

4. DAE SOLVER

4.1 DAE Formulation

We solve an index-1 DAE in residual form

$$\mathbf{F}(t,\ \mathbf{y},\ \dot{\mathbf{y}},\ \mathbf{z}) = \mathbf{0},$$

with differential states $\mathbf{y} \in \mathbb{R}^{n_y}$ and algebraic variables $\mathbf{z} \in \mathbb{R}^{n_z}$. For example, for a distillation column, $\mathbf{y}(t)$ stacks stage component holdups (and controller integrators, if any), while $\mathbf{z}(t)$ stacks stage temperatures and algebraic quantities needed to enforce VLE + summation constraints. The model is semi-rigorous: it advances component material balances dynamically and enforces phase equilibrium algebraically, without explicit tray-by-tray energy balances. Temperatures are therefore algebraic unknowns solved from equilibrium closure.

4.2 Radau IIA Collocation

Time integration uses a stiffly accurate Radau IIA implicit collocation method with $K = 3$ nodes

$$\tau_1 = \frac{4-\sqrt{6}}{10}, \qquad \tau_2 = \frac{4+\sqrt{6}}{10}, \qquad \tau_3 = 1,$$

chosen for L-stability (critical for stiff DAEs).

For each time step $[t_n, t_{n+1}]$ with $h = t_{n+1} - t_n$, introduce collocation unknowns

$$\mathbf{Y}_i \approx \mathbf{y}(t_n + h\tau_i), \qquad \mathbf{Z}_i \approx \mathbf{z}(t_n + h\tau_i), \qquad i = 1, \dots, K,$$

with $\mathbf{Y}_0 = \mathbf{y}(t_n)$ known.

Differentiation matrix.

Collocation reconstructs $\dot{\mathbf{y}}_i$ from nodal values using a differentiation matrix $\mathbf{A}$:

$$\dot{\mathbf{Y}}_i \ = \ \sum_{j=0}^{K} A_{ij}\, \mathbf{Y}_j, \qquad i = 1, \dots, K.$$

In the implementation, $\mathbf{A}$ is computed once from the Lagrange basis polynomials on $\{0, \tau_1, \tau_2, \tau_3\}$, and its derivatives are obtained via JAX AD (no numerical differencing for $\mathbf{A}$).

Collocation residuals.

At each collocation node, enforce the DAE residual:

$$\mathbf{F}\left(t_n + h\tau_i,\ \mathbf{Y}_i,\ \sum_{j=0}^{K} A_{ij}\, \mathbf{Y}_j,\ \mathbf{Z}_i\right) = \mathbf{0}, \qquad i = 1, \dots, K.$$

This produces a single nonlinear system per step with

$$K\left(n_y + n_z\right) \text{ unknowns} \quad \leftrightarrow \quad K\left(n_y + n_z\right) \text{ equations},$$

over the stacked vector

$$\mathbf{w} = \begin{pmatrix} \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_K \\ \mathbf{Z}_1 \\ \vdots \\ \mathbf{Z}_K \end{pmatrix}.$$

So, "time integration" is dominated by repeatedly solving $\mathbf{R}(\mathbf{w}) = \mathbf{0}$ for each step.

4.3 Residual Structure and Vectorization

The DAE residual $\mathbf{R}(\mathbf{w})$ decomposes naturally into:

Differential residuals: stage component material balance mismatches, using $\dot{\mathbf{Y}}_i$ from the collocation reconstruction.

Algebraic residuals: VLE relations, e.g.,

$$y_j = K_j(T, P, \mathbf{x})\, x_j,$$

and summation constraints ($\sum_j x_j = 1,\ \sum_j y_j = 1$, etc.).

Residual evaluation is organized stage-wise and then vectorized across stages via a single vmap call. This avoids Python loops over trays and exposes SIMD parallelism to XLA. Practically, one batched evaluation replaces $\mathcal{O}(10^2)$ sequential stage evaluations, which is a major speed driver in stiff dynamic runs.

4.4 Dense Newton vs. Matrix-Free Newton–Krylov

Each step solves $\mathbf{R}(\mathbf{w}) = \mathbf{0}$. Two solvers are supported:

Dense Newton (explicit Jacobian).

Compute the Jacobian $\mathbf{J} = \partial\mathbf{R}/\,\partial\mathbf{w}$ with JAX forward-mode AD (jacfwd) and solve the Newton system

$$\mathbf{J}\Delta\mathbf{w} = -\mathbf{R}(\mathbf{w})$$

via dense linear algebra (LU). This is robust and low-tuning, but the linear solve cost grows rapidly with n.

Newton–Krylov (matrix-free).

Avoid forming $\mathbf{J}$ explicitly. Use JVP to evaluate Jacobian–vector products $\mathbf{Jv}$ inside GMRES, i.e.,

$$\mathbf{Jv} = \mathrm{jvp}(\mathbf{R},\ \mathbf{w};\ \mathbf{v}).$$

This reduces memory pressure and can be faster for large n, especially when the effective Jacobian is sparse/structured.

4.5 Auto-Selection: Size/Sparsity-Aware Method Choice

Because the "best" method depends on system size and effective sparsity, the solver estimates Jacobian density ρ by sampling a small number of columns using JVP and computing

$$\rho = \frac{\text{nnz}}{n^2}.$$

A simple cost model compares:

Dense Newton: Jacobian build $\mathcal{O}(n^2)$ + factorization $\mathcal{O}(n^3)$

Newton–Krylov: residual $\mathcal{O}(n)$ + GMRES $\mathcal{O}(k \cdot n)$ per Newton step

Decision rules:

- *Very large systems:* force matrix-free (memory-bound)
- *Small systems:* prefer dense (robust, minimal tuning)
- *Intermediate:* choose lower predicted cost based on ρ and n

When Newton–Krylov is chosen, conservative defaults (tolerances/restarts) are used to preserve convergence without case-specific parameter tuning. However, for ill-conditioned or highly nonlinear systems, some tuning may still be necessary.

5. CONSTRAINED OPTIMIZATION

5.1 Augmented Lagrangian Method

General constrained problem: minimize f(x) subject to equality constraints h(x) = 0 and inequality constraints g(x) ≤ 0. Augmented Lagrangian formulation [11] uses penalty parameter μ > 0 and Lagrange multiplier estimates λh, λg. Algorithm alternates between solving unconstrained subproblem for x and updating multipliers. Penalty parameter μ increases if constraint violations persist, gradually enforcing feasibility.

5.2 Gradient Computation via JAX AD

JAX automatic differentiation provides exact derivatives with minimal computational overhead. For Williams-Otto reactor (n=2 decision variables), jax.grad returns exact 2D gradient in one forward plus one backward pass. Forward-mode AD is optimal when n_outputs > n_inputs [10]. The framework includes complex-step and finite-difference options for comparison.

BFGS quasi-Newton [10] solves unconstrained subproblems using rank-2 Hessian approximation updates with line search satisfying Wolfe conditions.

6. RESULTS AND VALIDATION

6.1 Nonlinear Solver Benchmarks

Seven Moré-Garbow-Hillstrom benchmark problems [13] spanning dimensions n=2 (Rosenbrock) to n=5000 (Brusselator 2D). Convergence tolerance 10^{-6}, maximum 100 iterations, default parameters.

Table 2: Success Rates on Standard Benchmarks.

Solver	Successful	Failed	Success Rate
Newton-Pure	6/7	1/7	85.7%
Newton-LineSearch	5/7	2/7	71.4%
Broyden-Pure	6/7	1/7	85.7%
Newton-Krylov-Pure	7/7	0/7	100.0%
Newton-Krylov-LineSearch	6/7	1/7	85.7%

Newton-Pure and Broyden-Pure achieved 85.7% success using fixed default parameters. Newton-Krylov-Pure achieved 100% success across all seven problems, notably solving ill-conditioned Brown Almost Linear (n=100) where all other methods failed due to near-singular Jacobians. This demonstrates the fundamental advantage of iterative linear solvers for poorly conditioned systems [6].

For Brusselator 2D (n=5000): Newton-Pure 15 iterations/19.5s versus Newton-Krylov-Pure 16 iterations/18.6s, validating theoretical O(n) scaling advantage of matrix-free methods.

Parameter sensitivity analysis: Newton-Krylov required problem-specific tuning of GMRES restart (10–40), linear tolerance (1e-4 to 1e-6), and preconditioner to achieve optimal performance. Newton-Pure converged reliably with fixed defaults, representing the fundamental trade-off between asymptotic complexity and configuration effort [6, 7].

6.2 Williams-Otto Reactor

Reaction System

The Williams-Otto reactor [14] consists of three series-parallel reactions occurring in a continuous stirred-tank reactor (CSTR) with constant holdup W = 2105 kg. Six components participate: reactants A and B, intermediate C, desired products P and E, and by-product G.

Reaction Scheme

$$A + B \rightarrow C, r_1 = k_1 C_A C_B$$

$$B + C \rightarrow P + E, r_2 = k_2 C_B C_C$$

$$C + P \rightarrow G, r_3 = k_3 C_C C_P$$

where Ci denotes the mass fraction of component i in the reactor (dimensionless), and ri is the reaction rate (s^{-1}).

Arrhenius Kinetics

The temperature-dependent rate constants follow the Arrhenius law:

$$k_i = k_i^0 \, exp\left(-\frac{E_{a,i}}{RT_R}\right), i = 1,2,3$$

where k_i^0 is the pre-exponential factor, $E_{a,i}$ is the activation energy, R = 8.314 J/(mol·K) is the universal gas constant, and T_R is the reactor temperature (K).

Table 3. Arrhenius kinetic parameters for the Williams-Otto reactor [14].

Reaction	Stoichiometry	k_0 (s^{-1})	Ea/R (K)	Ea (kJ/mol)
1	A + B → C	5.9755×10^9	12000	99.77
2	B + C → P + E	2.5962×10^{12}	15000	124.71
3	C + P → G	9.6283×10^{15}	20000	166.28

Component Mass Balances

For a CSTR with constant holdup W (kg), the dynamic mass balances for the six components are:

$$\frac{dC_A}{dt} = \frac{F_A C_{A,in} - FC_A}{W} - r_1$$

$$\frac{dC_B}{dt} = \frac{F_B C_{B,in} - FC_B}{W} - r_1 - r_2$$

$$\frac{dC_C}{dt} = -\frac{FC_C}{W} + r_1 - r_2 - r_3$$

$$\frac{dC_P}{dt} = -\frac{FC_P}{W} + r_2 - r_3$$

$$\frac{dC_E}{dt} = -\frac{FC_E}{W} + r_2$$

$$\frac{dC_G}{dt} = -\frac{FC_G}{W} + r_3$$

where F = $F_A + F_B$ is the total inlet mass flow rate (kg/s), F_A = 1.8275 kg/s is the fixed feed of pure A ($C_{A,in}$ = 1), and F_B is the adjustable feed of pure B ($C_{B,in}$ = 1). No C, P, E, or G enter with the feed.

Steady-State Condition

At steady state, all time derivatives vanish. The system reduces to six nonlinear algebraic equations:

$$F(C, T_R, F_B) = 0, where\ C = (C_A, C_B, C_C, C_P, C_E, C_G)$$

Given the reactor temperature TR and feed flow FB, the steady-state concentrations C are implicitly determined by solving the nonlinear system (11) via Newton's method.

Operating Conditions and Economic Parameters

Table 4. Reactor operating parameters and economic data.

Parameter	Symbol	Value	Units
Reactor holdup	W	2105	Kg
Feed flow of A (fixed)	F_a	1.8275	kg/s
Feed flow of B (adjustable)	F_E	4.0–6.0	kg/s
Reactor temperature range	T_r	70–100	°C
Parameter	**Symbol**	**Value**	**Units**
Base case temperature	T_r^0	85.0	°C
Base case feed flow B	F_E^0	4.50	kg/s
Price of product P	p_P	80	$/kg
Price of product E	p_E	20	$/kg
Cost of reactant A	c_A	40	$/kg
Cost of reactant B	$c_Æ$	25	$/kg

Constrained Optimization Problem

The economic optimization seeks to maximize the instantaneous profit by selecting the reactor temperature and feed flow of B. The problem is formulated as a constrained Nonlinear Programming (NLP) problem, where the decision variables are $x = (T_R, F_B)^T \in \mathbb{R}^2$

Objective Function

The profit function Π ($/h) accounts for product revenues minus raw material costs: The complete optimization problem is defined as:

$$max_{T_R, F_B}\ \Pi = (p_P X_P + p_E X_E) F - (c_A F_A + c_B F_B)$$

subject to: $F(C, T_R, F_B) = 0$

$$T_R \le T_R^{max} = 100\ °C$$

$$F_B \le F_B^{max} = 6\ kg/s$$

$$T_R^{min} \le T_R \le T_R^{max}, F_B^{min} \le F_B \le F_B^{max}$$

Inequality constraints (operational limits):

$$g_1(x): T_R \le 100\ °C\ (thermal\ safety\ limit)$$

$$g_2(x): F_B \le 6\ kg/s\ (pump\ capacity\ limit)$$

Equality constraints (steady-state process model):

$$h(x): F(C, T_R, F_B) = 0\ (6\ nonlinear\ algebraic\ equations$$

Variable bounds:

$$70\ °C \le T_R \le 100\ °C, 2\ kg/s \le F_B \le 6\ kg/s$$

where X_P and X_E are the mass fractions of products P and E at steady state (functions of x through the implicit steady-state solution), and F = $F_A + F_B$ is the total outlet flow.

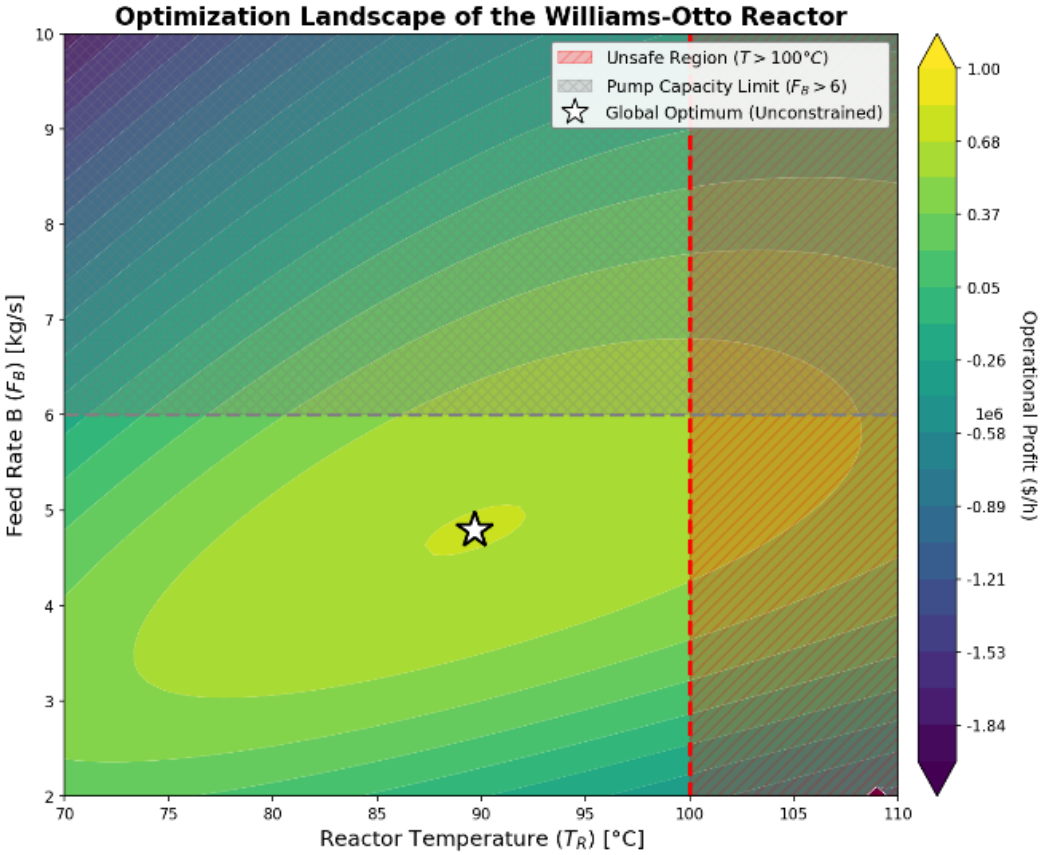

Figure 1. Optimization landscape over $T^R \in [70, 110]$°C and $F^B \in [2, 10]$ kg/s. Constraint regions indicated by red hatching (T>100°C) and gray hatching (F^B>6 kg/s). White star marks constrained global optimum.

The steady-state equations couple the decision variables to the component mass fractions, making the objective function implicitly dependent on the process model solution. The problem is solved using the Augmented Lagrangian Method with JAX automatic differentiation for exact gradient computation.

This is a two-dimensional NLP with 6 implicit equality constraints (steady-state model), 2 inequality constraints (safety and capacity), and box bounds on the decision variables. The steady-state equations couple the decision variables to the component mass fractions through the nonlinear kinetic model, making the objective function implicitly dependent on the process model solution.

The problem is solved using the Augmented Lagrangian Method with JAX automatic differentiation for exact gradient computation (see Section 5).

Steady-state calculation at base operating point converged in 5 Newton iterations to machine precision (residual 9.96×10^{-16}). Dynamic simulation over 1500s horizon using K=3 Radau IIA collocation completed in 1.132s wall-clock time, demonstrating 1325× real-time capability suitable for operator training simulators.

Constrained optimization via Augmented Lagrangian Method with JAX automatic differentiation converged in 0.894s (12 outer iterations) to global optimum: profit increased 1.9% from base case ($687, 529/h versus $674, 580/h). Both constraints remain inactive at optimum (temperature slack 10.3°C, flow slack 1.21 kg/s). Solution exhibits strong sensitivity to temperature as evidenced by tight contour spacing near optimum, justifying gradient-based optimization.

Comparison with Roberts (1979) published results [15] shows excellent agreement: temperature deviation +0.057°C (0.06%), flow deviation +0.004 kg/s (0.08%), profit deviation <0.01%. This validates all four framework modules against established literature benchmarks.

6.3 Propylene-Propane Splitter

High-purity separation of propylene from propane represents a major challenge in petrochemical industry due to close boiling points. At typical operating conditions (15–16 bar), these C3 hydrocarbons exhibit relative volatility α≈1.1 as computed from rigorous thermodynamics, requiring tall columns (150–200 stages) to achieve commercial purity specifications.

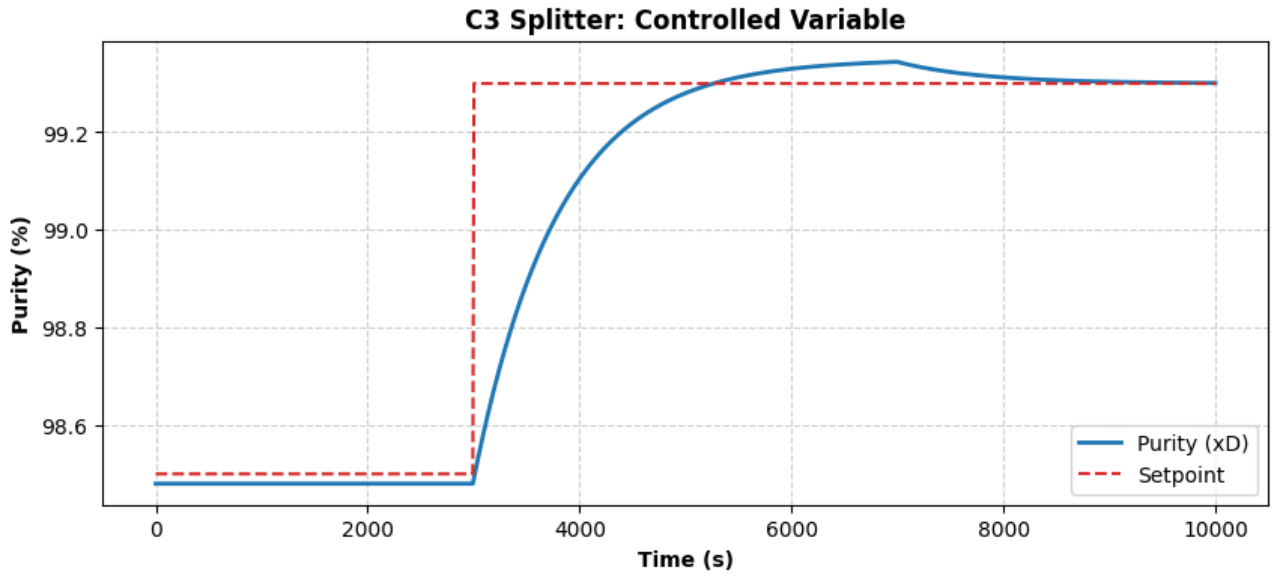

Figure 2. Dynamic response: propylene purity tracking

Table 5: Williams-Otto Comprehensive Performance.

Parameter	Base Case	Optimum	Units
Reactor Temperature TR	**85.00**	**89.70**	**°C**
Feed Rate FB	4.50	4.79	kg/s
Profit Π	674, 580	687, 529	$/h
Profit Increase	—	+1.9	%
Product P mass fraction XP	0.1098	0.1095	—
Product E mass fraction XE	0.2840	0.2906	—
Newton iterations	5	8	—
Residual norm	9.96×10^{-16}	1.2×10^{-15}	—
Optimization time	—	0.894	s
Dynamic simulation time (1500s)	1.132	—	s
Real-time speedup	1325×	—	—
Temperature deviation	—	+0.057	°C
Flow deviation	—	+0.004	kg/s
Profit deviation	—	<0.01	%

Table 6: C3 Splitter Complete Specification and Results.

Parameter	Value	Units
Number of stages	180	—
Feed stage	90	—
Top pressure (Ptop)	15.91	bar
Stage pressure drop (ΔP)	50	Pa
Feed flow rate (F)	100	mol/s
Feed temperature (Tfeed)	315	K
Propylene composition (z_C3H6)	0.55	mol frac
Binary interaction (k_{12})	0.0063	—
Controller gain (Kc)	85	—
Integral time constant (τI)	150	s
t=3000s: Setpoint step	98.5→99.3	%
Parameter	**Value**	**Units**
t=3000s: Feed flow ramp	100→120	mol/s
t=6000s: Composition ramp	0.55→0.60	—
Differential states (holdups)	360	—
Algebraic states (T, compositions)	180	—
Total DAE variables	540	—
Physical time simulated	10, 000	s
Wall-clock computation time	544.2	s
Real-time speedup factor	18.4×	—
Auto-selected DAE method	Dense Newton	—

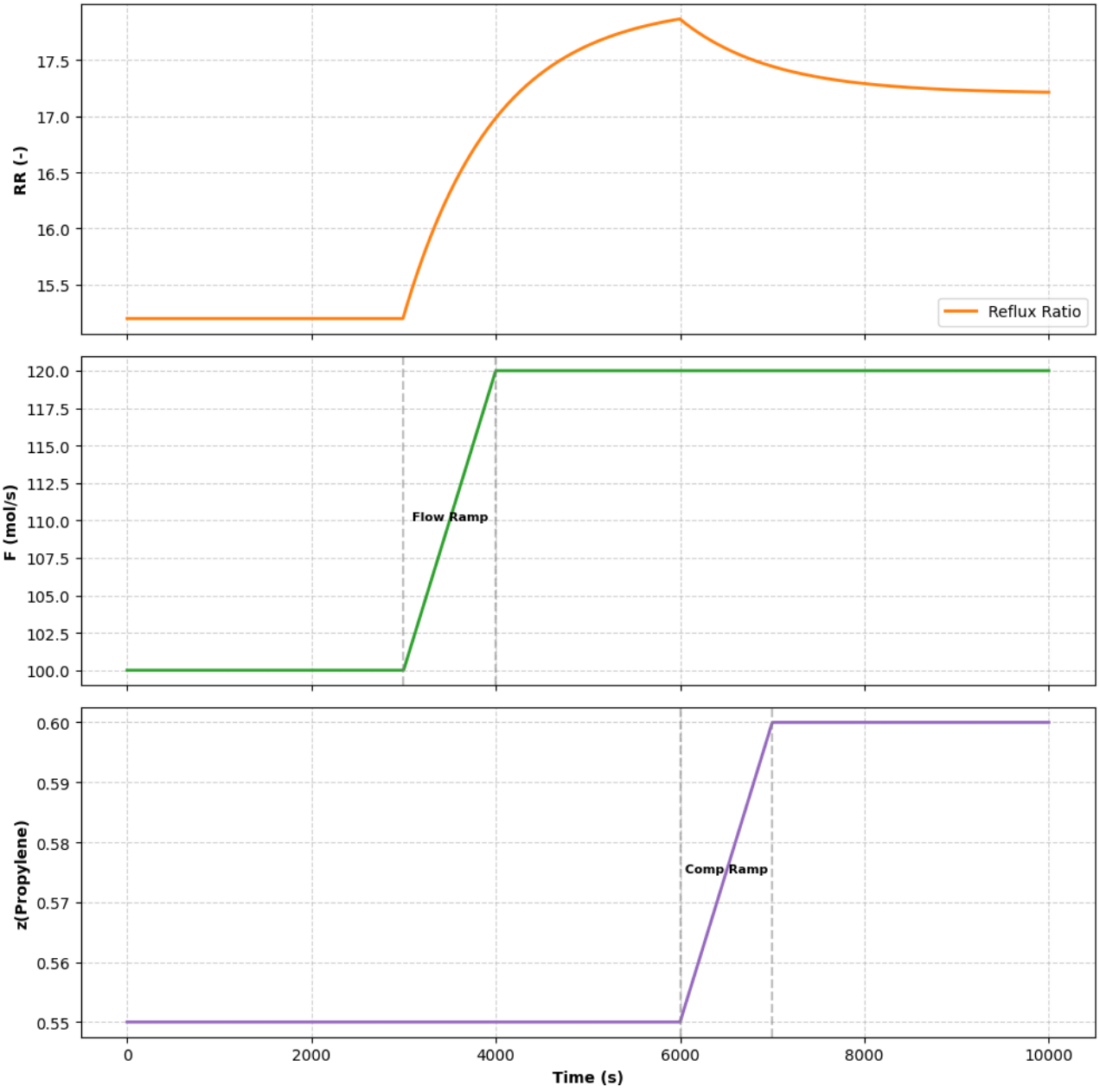

Figure 3 Dynamic response: top plot, reflux ratio manipulation (15.3 to peak 19.2); middle plot, feed flow ramp (+20%); bottom plot, feed composition evolution. Vertical dashed lines mark disturbance events.

The framework employs stage-specific rigorous VLE calculations rather than simplified constant relative volatility correlations. Pressure profile construction: Given top pressure Ptop = 15.91 bar and stage pressure drop ΔP = 50 Pa/stage, yielding pressures from 1591 kPa (top) to 1600 kPa (bottom). Rigorous VLE via Peng-Robinson EOS at each stage with binary interaction parameter k_{12}=0.0063 fitted to experimental C3–C3 VLE data. This rigorous approach captures pressure-dependent thermodynamic behavior during large flow disturbances.

Three sequential disturbances stress-test the controller: (1) t=3000s: concurrent setpoint step from 98.5% to 99.3% propylene purity synchronized with severe feed flow ramp from 100 to 120 mol/s over 1000s (+20% throughput increase); (2) t=6000s: feed composition ramp from 55% to 60% propylene over 1000s (+9% relative increase).

The controller successfully tracked the 99.3% setpoint with 0.4% overshoot and ~600s rise time. During the concurrent feed flow ramp, the controller increased reflux ratio from 15.3 to 19.2 (+25%) to maintain separation, demonstrating excellent disturbance rejection with

a maximum purity deviation of only −0.1%. The delayed composition disturbance at t=6000s was smoothly compensated, with the system stabilizing by t=8000s.

The DAE solver auto-selected dense Newton method based on system size (540 collocation variables < 800 threshold), completing 10, 000s physical time in 544.2s wall-clock time (18.4× real-time capability). With 540, 000 rigorous VLE calculations using Peng-Robinson thermodynamics with complex-step differentiation, this performance demonstrates suitability for control system design, operator training simulators, and multi-scenario optimization studies.

Additional validation: A 15-stage benzene-toluene-o-xylene (BTX) ternary column was simulated to validate multi-component capabilities beyond binary mixtures. The BTX column achieved 49× real-time performance under multiple disturbances, confirming that the framework's complex-step differentiation strategy and JAX vectorization scale effectively to systems with three or more components.

7. CONCLUSIONS

This work presents a Python/JAX framework for hydrocarbon process simulation validated through three applications: (1) nonlinear solver benchmarks with Newton-Krylov achieving 100% success on seven standard problems (n=2 to 5000), (2) Williams-Otto reactor optimization showing 0.06% deviation from published literature with 1325× real-time dynamic simulation capability, (3) 180-stage propylene-propane splitter achieving 18× real-time performance with 540, 000 rigorous VLE calculations under three concurrent disturbances.

Key implementations: Peng-Robinson thermodynamics with complex-step differentiation [12], avoiding nested JAX transformation issues, Newton/Broyden/Newton-Krylov solvers with Curtis-Reid auto-scaling [5], Radau IIA DAE solver with auto-selection between dense and matrix-free methods [8, 9], Augmented Lagrangian optimization with JAX automatic differentiation [11].

Current limitations: Semi-rigorous distillation formulation without tray-by-tray energy equations, Peng-Robinson equation of state limited to nonpolar hydrocarbon systems, Newton-Krylov methods may require parameter tuning for optimal performance.

Why a JAX-friendly DAE solver? While several Python libraries exist for DAE systems (Assimulo [17], GEKKO [18], CasADi [19]), they typically lack one or more of: (1) native automatic differentiation for exact Jacobians, (2) JIT compilation for near-C++ performance, (3) vectorization primitives (vmap) for batch stage-wise calculations, or (4) transparent pure-Python implementations enabling straightforward debugging and customization.

Future directions: (1) Rigorous distillation models implementing MESH equations and/or the DDPA model of Matias et al (2018) [16], (2) extended thermodynamics including CPA and NRTL/UNIQUAC activity models, (3) state estimation integration (EKF, UKF, MHE) for real-time applications, (4) GPU acceleration evaluation for large-scale simulations, (5) Sparse matrix handling JAX-friendly, as an alternative for Newton-Krylov

ACKNOWLEDGEMENTS

The authors acknowledge the support of the Chemical Engineering Department at Universidade de São Paulo.

AUTHOR IDENTIFIERS

Author ORCIDs:
Sanz CC: 0009-0008-4267-3410
Le Roux G: 0000-0001-7227-5271

REFERENCES

1. J. Bradbury et al., "JAX: composable transformations of Python+NumPy programs, " 2018. http://github.com/google/jax
2. OpenXLA Project, "XLA: Optimizing Compiler for Machine Learning." https://openxla.org/xla
3. Peng D-Y, Robinson DB. A New Two-Constant Equation of State. Ind Eng Chem Fundam 15(1):59-64 (1976)
4. Cox KR, Chapman WG. The properties of gases and liquids, 5th edition by bruce E. poling (university of toledo), john M. prausnitz (university of california at berkeley), and john P. o'connell (university of virginia). mcgraw-hill: new york. 2001. 768 pp. $115.00. ISBN 0-07-011682-2.. J. Am. Chem. Soc. 123:6745-6745 (2001). https://doi.org/10.1021/ja0048634
5. CURTIS AR, REID JK. On the automatic scaling of matrices for gaussian elimination. IMA J Appl Math 10:118-124 (1972). https://doi.org/10.1093/imamat/10.1.118
6. Knoll DA, Keyes DE. Jacobian-free newton–krylov methods: a survey of approaches and applications. Journal of Computational Physics 193:357-397 (2004). https://doi.org/10.1016/j.jcp.2003.08.010
7. Eisenstat SC, Walker HF. Choosing the forcing terms in an inexact newton method. SIAM J. Sci. Comput. 17:16-32 (1996). https://doi.org/10.1137/0917003
8. Hairer E, Wanner G. Order conditions for index 2 DAE. Springer Series in Computational Mathematics :506-518 (1996). https://doi.org/10.1007/978-3-642-05221-7_35

9. Ascher UM, Petzold LR. Computer Methods for ODE and DAE, SIAM (1998)
10. Nocedal J, Wright SJ. Numerical Optimization, 2nd Ed., Springer (2006)
11. Conn AR, Gould NIM, Toint PhL. Trust-Region Methods, SIAM (2000)
12. Martins JRRA, Sturdza P, Alonso JJ. The complex-step derivative approximation. ACM Trans. Math. Softw. 29:245-262 (2003). https://doi.org/10.1145/838250.838251
13. Moré JJ, Garbow BS, Hillstrom KE. Testing unconstrained optimization software. ACM Trans. Math. Softw. 7:17-41 (1981). https://doi.org/10.1145/355934.355936
14. Williams TJ, Otto RE. A generalized chemical processing model for the investigation of computer control. Trans. AIEE, Part I: Comm. Electron. 79:458-473 (1960). https://doi.org/10.1109/tce.1960.6367296
15. Roberts PD. An algorithm for steady-state system optimization. Int J Syst Sci 10(7):719-734 (1979)
16. Matias JOA, Ebrahimpour M, Le Roux GAC. Development and validation of a reduced-index dynamic model of an industrial high-purity column. Ind. Eng. Chem. Res. 57:1531-1546 (2018). https://doi.org/10.1021/acs.iecr.7b02789
17. Andersson C, Führer C, Åkesson J. Assimulo: a unified framework for ODE solvers. Mathematics and Computers in Simulation 116:26-43 (2015). https://doi.org/10.1016/j.matcom.2015.04.007
18. Beal LDR, Hill DC, Martin RA, Hedengren JD. GEKKO optimization suite. Processes 6:106 (2018). https://doi.org/10.3390/pr6080106
19. Andersson JAE, Gillis J, Horn G, Rawlings JB, Diehl M. Casadi: a software framework for nonlinear optimization and optimal control. Math. Prog. Comp. 11:1-36 (2018). https://doi.org/10.1007/s12532-018-0139-4