Fast approximations of exponential and logarithm functions combined with efficient storage/retrieval for combustion kinetics calculations

Federico Perini[a,*], Rolf D. Reitz[a]

[a] *University of Wisconsin–Madison. 1500 Engineering Drive, Madison, WI 53706*

Abstract

We developed two approaches to speed up combustion chemistry simulations by reducing the amount of time spent computing exponentials, logarithms, and complex temperature-dependent kinetics functions that heavily rely on them. The evaluation of these functions is very accurate in 64-bit arithmetic, but also slow. Since these functions span several orders of magnitude in temperature space, some of this accuracy can be traded with greater solution speed, provided that the governing ordinary differential equation (ODE) solver still grants user-defined solution convergence properties. The first approach tackled the `exp()` and `log()` functions, and replaced them with fast approximations which perform bit and integer operations on the exponential-based IEEE-754 floating point number machine representation. The second approach addresses complex temperature-dependent kinetics functions via storage/retrieval. We developed a function-independent piecewise polynomial approximation method with the following features: it minimizes table storage requirements, it is not subject to ill-conditioning over the whole variable range, it is of arbitrarily high order $n > 0$, and is fully vectorized. Formulations for both approaches are presented; and their performance assessed against zero-dimensional reactor simulations of hydrocarbon fuel ignition delay, with reaction mechanisms ranging from 10 to 10^4 species. The results show that, when used concurrently, both methods allow global speed-ups of about one order of magnitude even with an already highly-optimized sparse analytical Jacobian solver. The methods also demonstrate that global error is within the integrator's requested accuracy, and that the solver's performance is slightly positively affected, i.e., a slight reduction in the number of timesteps per integration is seen.

Keywords: exponential, logarithm, floating-point algebra, table interpolation, chemical kinetics

*Corresponding author
Email address: `perini@wisc.edu` (Federico Perini)

1. Introduction

The computational cost associated with the solution of stiff Ordinary Differential Equations (ODEs) describing chemical kinetics is still one of the major factors limiting the usage of detailed chemistry in multidimensional combustion simulations [1]. In a chemically reactive gas-phase environment, conservation equations for the closed system's mass and energy appear as rates of change of species mass fractions Y_i and temperature T:

$$\frac{dY_i}{dt} = \dot{Y}_i = \frac{W_i}{\rho} \sum_{j=1}^{n_r} \left[\left(\nu''_{j,i} - \nu'_{j,i} \right) q_j \right], \tag{1}$$

$$\frac{dT}{dt} = \dot{T} = -\frac{1}{\bar{c}_v} \sum_{i=1}^{n_s} \frac{U_i \dot{Y}_i}{W_i},$$

when the mixture of $i = 1, \ldots, n_s$ species M_i, is subject to a reaction mechanism, i.e., a network of $j = 1, \ldots, n_r$ chemical reactions:

$$\sum_{i=1}^{n_s} \nu'_{j,i} M_i \rightleftharpoons \sum_{k=1}^{n_s} \nu''_{j,k} M_k, \; j = 1, ..., n_r; \tag{2}$$

ν' and ν'' are sparse matrices containing stoichiometric reaction coefficients of reactants and products respectively [2]. The system usually exhibits very stiff behavior because of both the exponential form of the reaction rates, and the strongly nonlinear coupling between species concentrations caused by the law of mass action, as witnessed by the species' mutual excitation rate [2]:

$$\frac{\partial \dot{Y}_i}{\partial Y_j} = \frac{W_i}{\rho} \sum_{k=1}^{n_r} \left\{ \frac{\nu_{k,i}}{Y_j} \left[\nu'_{k,j} k_{f,k} \prod_{r=1}^{n_s} \left(\frac{\rho Y_r}{W_r} \right)^{\nu'_{k,r}} - \nu''_{k,j} k_{r,k} \prod_{s=1}^{n_s} \left(\frac{\rho Y_s}{W_s} \right)^{\nu''_{k,s}} \right] \right\}, \tag{3}$$

where ρ is the system's density, k_f and k_r the forward and reverse reaction rates, W the species' molecular weights. Because of its stiffness, time integration of the reactive system of Equation 1 is usually performed as an independent ODE system even in multidimensional simulations, where an operator splitting scheme is employed to relax the solver integration constraints towards the flow time scales (see [3]).

In recent years, several studies have addressed aspects of the chemical kinetics ODE system to increase its computational efficiency. Some researchers have focused on ODE solution methods for stiff systems [4–7] aimed at achieving time advancement of the solution with the least number of integrator timesteps. Perhaps the greatest CPU time savings have been achieved by adopting fine-tuned analytical formulations of the chemistry system and its Jacobian [2, 8, 9],

coupled with sparse matrix algebra [10, 11] to take advantage of the reaction mechanism's sparsity. Some recent efforts have also attempted to exploit graphics processing units (GPUs) to increase throughput of the kinetics calculations [12].

This study focuses on reducing the cost of evaluating reaction kinetics functions involving exponentials and logarithms. Two approaches are presented: first, several fast approximations of the `exp()` and `log()` functions exploiting the IEEE-754 floating-point number representation [13] were developed. Second, a storage/retrieval approach for costly temperature-dependent functions was introduced. A simulation matrix featuring 11 reaction mechanisms from 10 to 10^4 species, simulating ignition of fuel-air mixtures at conditions relevant to combustion devices was established, in order to assess the robustness, the accuracy and the speed of the proposed approaches. Combustion CFD simulations, including 2D and 3D cases, were validated as well. The results demonstrate significant speed-ups of almost an order of magnitude for the total CPU time even in presence of analytical Jacobian and sparse algebra; plus, a stabilizing effect of the smoothed function approximations on the ODE solvers' performance.
The major contributions of this work can be summarized as follows:

- A fast equally-spaced tabulation/polynomial interpolation approach for exponentially-varying functions which has limited storage needs, is continuous in both function and derivative evaluations up to an arbitrary order, is defined piecewise like a spline, but its accuracy is not affected by what happens far from the interpolation point;

- New, improved methods for fast evaluation of exponential and logarithm functions, that provide not only a continuous function, but also continuous derivatives. These methods improve on the fast exponential approach based on floating point representation manipulation methodology developed by [14] by using an arbitrary-order spline reconstruction of the mantissa, which compares favorably even to the most recent formulations [15].

2. Algorithm description

We developed methods to approximate the exponential and logarithm functions for 64-bit (`double precision`) floating-point numbers complying with the IEEE-745 standard [13]. This format represents a real number by subdividing the 64-bit space into three integer strings, as reported in Figure 1:

$$r = (-1)^s \, 2^{x-b} \, (1 + m) \,, \tag{4}$$

where:

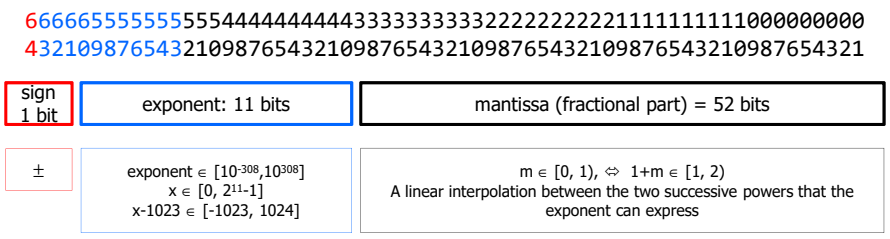

Figure 1: IEEE-754 representation of a 64-bit (`double`) real number.

- s (1 bit) is the sign bit;
- x (11 bits) $\in [0, 2047]$ is an integer exponent, shifted by a fixed bias $b = 1023$, such that both negative and positive integer powers can be represented: $2^{x-b} \in [2^{-1023}, 2^{+1024}] \approx [10^{-308}, 10^{+308}]$;
- m (52 bits) is the *mantissa* or a fractional part: $m \in [0, 1)$, or $[1 + m) \in [1, 2)$.

This model produces an exact represenation of any integer powers of 2, which have empty mantissa $m = 0$; the mantissa acts as a truncated linear interpolation between subsequent powers of two, hence allowing any real numbers r in the range to be represented within an accuracy of approximately 15 decimal digits.

2.1. Fast Exponential function

Approximations of the exponential function exploiting the IEEE-754 standard were developed by Schraudolph [14] and later extended by other researchers [16]. These formulations target 32-bit numbers, and are not suitable for double precision integration of highly stiff problems. However, a recent paper by Malossi et al. [15] targeted 64-bit numbers and, while coefficients are not given, employs a McLaurin series expansion, and is included here for completeness. The exponential function is naturally defined as a series:

$$\exp(x) = e^x = \sum_{n=1}^{+\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots \tag{5}$$

and this feature is exploited by accurate exponential evaluation methods [17]. However, in [14] it was demonstrated that a fast approximation of the exponential can be achieved by just manipulating the IEEE-754 number representation of Equation 4 using simple bit shift and integer algebra operations. First, the

exponential operation is reduced to a power-of-two operation with a change of basis:

$$e^x = 2^{x/\log 2} = 2^y; \tag{6}$$

if number y is represented as an integer and a fractional part, $y = y_i + y_f$, then its machine representation fits its power-of-two exponentiation well:

$$2^y = 2^{y_i} 2^{y_f} = (-1)^s 2^{x-b}(1+m). \tag{7}$$

Sign $s = 0$ is always positive. The integer term 2^{y_i} can be computed by simply fitting a suitable integer $x = 1023 + y_i$ into the exponent part of the floating point representation, i.e., by left-shifting integer x by 52 positions, and casting it into the real number representation:

$$2^{y_i}\ (\texttt{real64}) \leftarrow 2^{52} \cdot (1023 + y_i)\ (\texttt{int64}); \tag{8}$$

This operation produces exact results for any numbers being exponentiated that correspond to integer powers of two, i.e., when computing: $e^{\log(2)}, e^{2\log(2)}, e^{3\log(2)}, \ldots$. The remainder of the floating-point exponentiation deals with the fractional part y_f of Equation 7, and must comply with the following relationship:

$$2^{y_f} = 1 + m, \tag{9}$$

i.e., one should find what value for the mantissa $m \in [0, 1)$ best interpolates the power-of-two function in the $[2^0, 2^1)$ interval. If the mantissa was simply left unchanged from its value in the number y, one would get a linear interpolation even with no additional operations, such as employed in [14], just by observing that $m \equiv y_f$. However, a more general formulation is to use a Taylor series. The most convenient formulation is to cast Equation 9 as a correction term:

$$\Delta(y_f) = 1 + m - 2^{y_f} = 1 + y_f - 2^{y_f}, \tag{10}$$

as represented in Figure 2. Function Δ has small values compared to the power function in the whole interval: $\Delta < 0.1$, and is always subtracted from the current mantissa. Hence, it is legitimate to also subtract the value of Δ from the whole number y, since its exponent bits can not be affected.
The overall fast exponential calculation procedure can be summarized as a change of base and an integer-based assignment:

$$\begin{aligned} y &\leftarrow x/\log 2; \\ i(\texttt{int64}) &\leftarrow 2^{52}(y - \Delta(y_f)) + 1023 \cdot 2^{52}. \end{aligned} \tag{11}$$

As the long integer i now contains the IEEE-754 `real64` number representation of e^x, it only has to be cast to a real number representation (`EQUIVALENCE` statement in FORTRAN 77, `transfer` function in Modern Fortran, or `()` cast operator in C++).

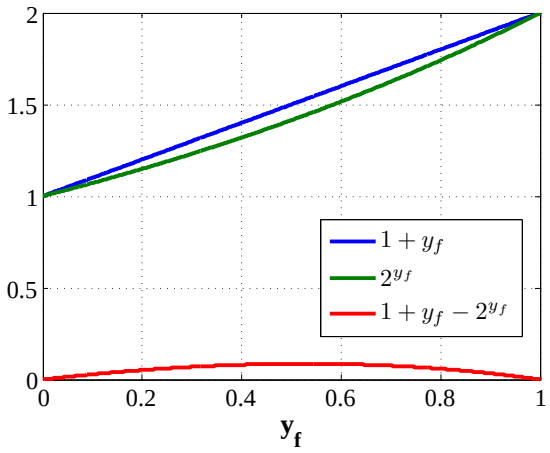

Figure 2: Fractional power computation: comparison between linear interpolation of the mantissa $(1 + y_f)$, exact desired value (2^{y_f}) and Delta function.

2.1.1. Fractional correction formulations.

A first formulation of Δ as proposed in [15] features a McLaurin series:

$$\Delta(y_f)|_{y_f \in [0,1)} = \sum_{n=0}^{+\infty} \frac{\Delta^{(n)}(0)}{n!} y_f^n = (1 - \log 2) y_f + \sum_{n=2}^{+\infty} \frac{-(\log 2)^n}{\Gamma(n+1)} y_f^n. \tag{12}$$

The series formulation allows Δ to be computed as an arbitrary degree polynomial, whose coefficients are the series parameters in Equation 12. Figure 3 reports the relative accuracy of $fexp(x) = e^x$ this formulation, with polynomial degrees from 1 to 5. All cases have worsening accuracy the farther downstream of the series' center, which creates an asymmetric error profile. One way to improve on this behavior is to adopt a more general Taylor series, centered in the midpoint of the mantissa range, i.e., $y_0 = 1/2$:

$$\begin{aligned} \Delta(y_f)|_{y_f \in [0,1)} &= \sum_{n=0}^{+\infty} \frac{\Delta^{(n)}(1/2)}{n!} \left(y_f - \frac{1}{2} \right)^n \\ &= 1 - \sqrt{2} + y_f + \sum_{n=1}^{+\infty} \frac{-\sqrt{2}(\log 2)^n}{\Gamma(n+1)} \left(y_f - \frac{1}{2} \right)^n. \end{aligned} \tag{13}$$

This more general series formulation comes at the price of a few additional floating point operations per evaluation, but provides a more reasonable error profile, as reported in Figure 4: the error curve is symmetric with respect to the interpolation interval, and the error peak per same polynomial degree is significantly smaller. For example, for a degree-3 polynomial, maximum error

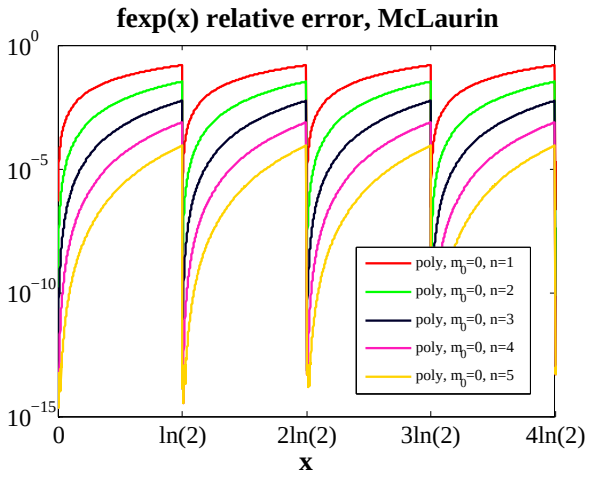

Figure 3: Relative accuracy of approximate exponential with McLaurin series-based fractional part reconstruction, Eq. 12, degree 1 to 5.

was of 0.56% for the McLaurin expansion, and 0.078% for the Taylor expansion centered in $y_0 = 1/2$.

Because none of the series-based formulation cares about smoothness of the exponential curve, i.e., continuity of its successive derivatives, we developed an additional piecewise polynomial formulation in third- and a fifth-order degree versions. The idea is that of reconstructing the exponential as a spline curve [18], where however a unique polynomial for the fractional mantissa correction applies to all pieces of the function. Modeling the correction funcion of Equation 10 as a piecewise polynomial in the interval $y \in [0, 1]$, one has conditions at the extremes on both the function and its derivatives:

$$\begin{aligned} \Delta(y) &= 1 + y - 2^y; \\ \Delta'(y) &= 1 - \log 2 \cdot 2^y; \\ \Delta''(y) &= -(\log 2)^2 \cdot 2^y. \end{aligned} \tag{14}$$

Coefficients for the polynomial reconstruction can be obtained by solving a linear system with conditions applied at the extreme points. The cubic polynomial is found by enforcing function value and the first derivative (4 total equations); the fifth-degree polynomial by also adding two equations for the second derivative. The resulting set of coefficients is reported in Table 2.1.1 for both piecewise polynomial formulations, with the polynomial expressed as:

$$\Delta(y) = \sum_{i=0}^{n} s_i y^i. \tag{15}$$

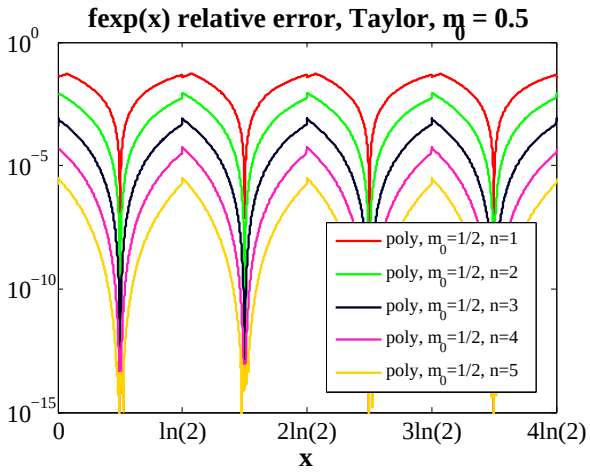

Figure 4: Relative accuracy of approximate exponential with Taylor series-based fractional part reconstruction, Eq. 13, centered in $m_0 = 0.5$ degree 1 to 5.

Approximation errors reported in Figure 5 show that smoother error behavior

Table 1: Coefficients for piecewise polynomial interpolation of the mantissa Δ correction function of Equation 10, $\Delta(x) \approx \sum_{i=0}^{n} s_i x^i$

Coefficient	Cubic ($n = 3$)	Quintic ($n = 5$)
s_5	-	-1.90188191959304e-3
s_4	-	-9.01146535969578e-3
s_3	-7.6167541742324804e-2	-5.57129652016652e-2
s_2	-0.23141283591588344	-2.40226506959101e-1
s_1	0.30758037765820823	3.06852819440055e-1
s_0	0	0

is achieved by the spline representations thanks to continuity being enforced not only at the function but also at its first (and second, if $n = 5$) derivative.
The computational performance of both the spline and the Taylor series-based formulations is reported in Figure 6. Here, CPU time monitoring was performed over a set of 10^8 statistically-sampled `fexp(x)` evaluations for $x \in [0, 1]$. In order to circumvent possible compiler optimizations, each exponential was also summed to a cumulative sum variable, whose additional overhead was also evaluated separately and then removed from the measure. The application was compiled with a `GCC/gfortran 6.2.0` compiler on a Microsoft Windows machine equipped with an Intel i7-4770K quad-core processor. CPU time measurements include advanced compiler optimization flags (`-O3`) as well as CPU architecture-

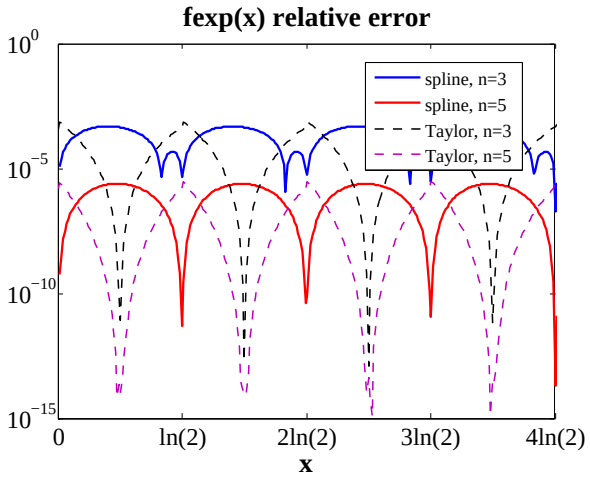

Figure 5: Relative accuracy of approximate exponential with third-degree and fifth-degree piecewise polynomial fractional part reconstruction, Equation 15, compared with the similar degree Taylor expansions of Equation 13.

specific tuning (`-march=core-avx2`). Only slight improvements were achieved by architecture-specific tuning against the 'standard' optimization flags; they allowed some CPU time reduction especially at the most demanding fifth-degree formulations, achieving computational time reductions always greater than 85% for any formulations, most likely thanks to the `avx2` set of instructions. These include `FMA`, or Fused Multiply-Add, where two floating-point operations (multiplication and addition) of the type `a+(b*c)` can be performed within the same CPU clock cycle, which well suit Horner's rule for polynomial evaluation adopted in the current implementation:

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_0 + x\left(a_1 + x\left(a_2 + x\left(a_3 + \ldots\right)\right)\right). \tag{16}$$

Furthermore, each Taylor series implementation was slower than its similar-degree spline version, due to the additional operations associated with the non-zero series center.

2.2. Fast Logarithm function

A fast approximate method for the natural logarithm function $\log(x)$ was developed based on the same template as for the exponential function; the interpolation method for the being restricted to the piecewise polynomial functions of the spline type because of the benefits highlighted in the previous section. In case of a natural logarithm operation, the opposite change of base happens:

$$\log_e(x) = \log_e(2) \cdot \log_2(x); \tag{17}$$

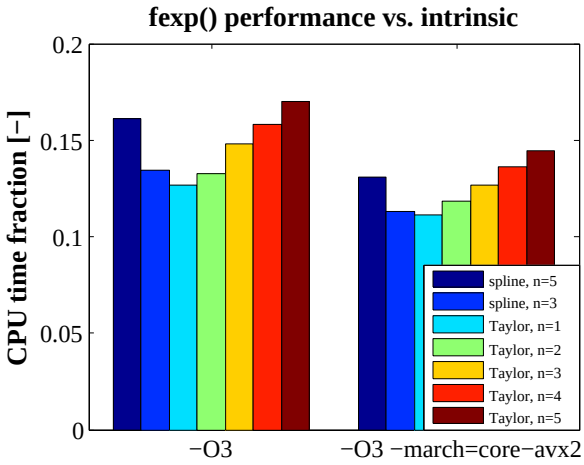

Figure 6: Relative performance of approximate exponential function `fexp(x)` with piecewise polynomial or Taylor series based fractional part reconstruction. Compiled with `GCC/gfortran 6.2.0` for an Intel i7-4770K CPU. Compiler optimization flags: (left) fast `-O3` only, (right) with CPU architecture tuning, `-O3 -march=core-avx2`.

hence, the object of the fast approximation is the base-2 logarithm of the input variable y. Using the same IEEE-754 representation trick as from Equation 7, the logarithm can be split into a sum of an integer part and a fractional part

$$\begin{aligned} log_2(y) &= log_2\left(2^{x-b}(1+m)\right) \\ &= (x-b) + log_2(1+m), \end{aligned} \tag{18}$$

i.e., the integer part $(x-b)$ is already stored in the real number as its exponent x, while the fractional part stored in the mantissa is again subject to modeling. A spline interpolation approach similar to that of Equation 14 was used, whose coefficients were found by solving a linear system with the following function and derivative conditions:

$$\begin{aligned} \Delta(m) &= log_2(1+m); \\ \Delta'(m) &= \left(\log 2 \cdot (1+m)\right)^{-1}; \\ \Delta''(m) &= -\left(\log 2 \cdot (1+m)^2\right)^{-1}. \end{aligned} \tag{19}$$

The polynomial coefficients for cubic and fifth-degree spline reconstruction of the Delta function in Equation 19 are reported in Table 2. Despite the lower computational complexity of the logarithm function, the amount of speed-up achieved by the fast implementation is more significant than for the exponential, as reported in Figure 7; also the CPU-optimized compilation showed a

Table 2: Coefficients for piecewise polynomial interpolation of the mantissa Δ correction function of Equation 19 (fast log), $\Delta(x) \approx \sum_{i=0}^{n} s_i x^i$

Coefficient	Cubic ($n = 3$)	Quintic ($n = 5$)
s_5	-	4.88829563330264e-2
s_4	-	-2.12375830888126e-1
s_3	0.164042561333445	4.42145354110618e-1
s_2	-0.606737602222408	-7.21347520444482e-1
s_1	1.442695040888963	1.44269504088896e+0
s_0	0	0

greater increase in performance, with maximum speed-ups of more than 95% for any of the formulations. Regarding the relative accuracy reported in Figure 8, the mantissa interpolation function now acts between periods of x which are integer powers of 2. The same smooth error pattern produced by the fast spline exponential was seen also for the spline logarithm function; relative errors being slightly smaller in the region close to $x = 1$ where the logarithm becomes zero; still consistently uniform across the whole variable range. Peak relative errors were of 1.4% for the cubic version, and 0.11% for the fifth-degree formulation. Modern Fortran implementation of the fifth-degree spline fast approximations of the exponential and logarithm functions are reported in Appendix A.

2.3. Fast interpolation with tabulated data

An alternate approach to speeding up the evaluation of derived functions involving exponentials and logarithms is to totally replace function evaluation with a storage/retrieval technique [2]. The function is sampled in a well-defined variable range at select, equally spaced points; the table is then queried whenever function values are requested. The number of table points needed per function evaluation depends on the desired interpolation accuracy. This approach has the advantage of replacing on-the-fly computation with stored data: the user can store more complex functions that involve multiple operations, hence saving further CPU time; furthermore, by sharing table range and sampling points among multiple functions, the procedure can be fully vectorized hence allowing further CPU time savings when multiple functions are involved – for instance, in the case of multiple reaction rates in a mechanism. However, this comes at the price of a non-negligible memory footprint; plus, the accuracy of the function reconstruction depends on the user's choice of a suitable sampling step.

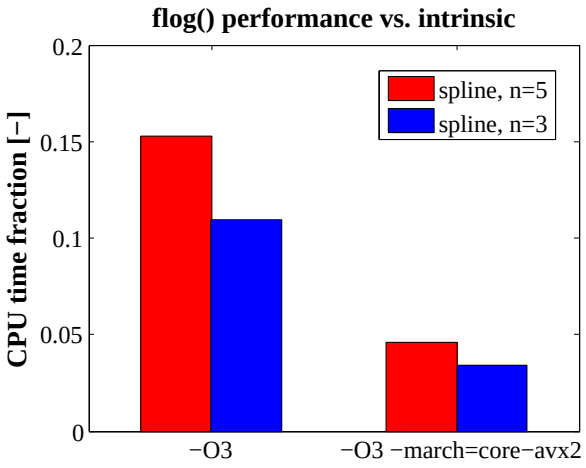

Figure 7: Relative performance of approximate natural logarithm function `flog(x)` with piecewise polynomial fractional part reconstruction. Compiled with `GCC/gfortran 6.2.0` for an Intel i7-4770K CPU. Compiler optimization flags: (left) fast `-O3` only, (right) with CPU architecture tuning, `-O3 -march=core-avx2`.

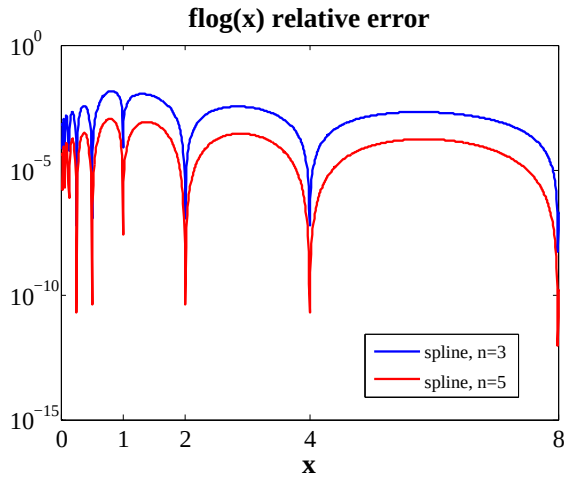

Figure 8: Relative accuracy of approximate logarithm with third- and fifth-degree piecewise polynomial fractional part reconstruction, Equation 19.

2.3.1. Fast Piecewise Interpolation.

Consider a vector function $\mathbf{f}(x) : \mathbb{R} \rightarrow \mathbb{R}^n$ that maps a single variable $x \in [x_{min}, x_{max}]$ to values in an $n-$ dimensional space. A table $\mathbb{T}$ is a set of sampled function vectors $\{\mathbf{f}(x_1 \equiv x_{min}), \mathbf{f}(x_2), \ldots, \mathbf{f}(x_{n_s} \equiv x_{max})\}$ at a number n_s of equally-spaced sampling points, with step-size $\Delta x = x_i - x_{i-1}$. A fast interpolation method was developed for tabulated vector functions such as defined. A schematic of this configuration is reproduced in Figure 9. Because all tabulated values are equally spaced in the unknown variable space, their actual range can be replaced with an integer range with a linear mapping function. So, first, the position of the unknown x is located within the tabulation interval using normalized coordinates:

$$\begin{aligned} \chi &= (x - x_{min}) / \Delta x; \\ I_0 &= \lfloor \chi \rfloor ; \\ r &= \{\chi\} = \chi - I_0; \end{aligned} \tag{20}$$

where χ is the normalized real coordinate, or number of steps from the first mapped point; I_0 is its floor function [19], or greatest integer smaller than χ; r is the fractional part assumed by χ within its containing interval. Point χ is hence included in the $[I_0, I_0 + 1)$ interval, identified by the left bound I_0, which can assume integer values betweeh 1 and $n_s - 1$. We developed a piecewise polynomial basis functions that shifts the left bound to the origin in the normalized coordinate axis, so the interpolation point is only identified by coordinate r and lies in the $[0, 1)$ range. The successive neighboring points in table $\mathbb{T}$ are identified by their relative position, i.e., by normalized positive $(2, 3, 4, \ldots)$ or negative $(-2, -3, \ldots)$ coordinates. We want the basis function to be a polynomial of arbitrary order n_P:

$$P(r) = \sum_{i=0}^{n_P} p_i r^i. \tag{21}$$

Hence, coefficients p_i are obtained by solving a linear system to equal the polynomial formulation of Eq. 21 with a necessary number of tabulated data points to the current interval. In the simple case of linear interpolation, $P_1(r) = p_1 r + p_0$, the two interval extremes will be enough:

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_0 \end{bmatrix} = \begin{bmatrix} y(0) \\ y(1) \end{bmatrix}. \tag{22}$$

In the general case of a higher-degree polynomial, $n_p + 1$ mapped data points are needed in order for the linear system to be solved. Hence, one must choose an appropriate number of neighboring data points on the left- and right-hand side of interval $[0, 1]$. A global set of $n_p + 1$ consecutive datapoint locations

$[l, l+1, \ldots, 0, 1, \ldots, l+n_p]$ is centered around the [0,1] interval for any piecewise polynomial degree if the leftmost neighbor index is chosen as $l = -[n_p/2-1] \in \mathbb{Z}$, where [] denotes rounding to the nearest integer. The arbitrary-degree linear system to be solved for coefficients p_i in Equation 21 has the following form:

$$\begin{bmatrix} l_p^n & l^{n_p-1} & \ldots & l & 1 \\ (l+1)_p^n & (l+1)^{n_p-1} & \ldots & (l+1) & 1 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & 1 \\ 1 & 1 & \ldots & 1 & 1 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ (l+n_p)_p^n & (l+n_p)^{n_p-1} & \ldots & (l+n_p) & 1 \end{bmatrix} \cdot \begin{bmatrix} p_{n_p} \\ p_{n_p-1} \\ p_{n_p-2} \\ \vdots \\ \vdots \\ \vdots \\ p_0 \end{bmatrix} = \begin{bmatrix} y(l) \\ y(l+1) \\ \vdots \\ y(0) \\ y(1) \\ \vdots \\ y(l+n_p) \end{bmatrix}. \quad (23)$$

Since the table usually contains large array of values $\mathbf{y} = \mathbf{f}(x)$ per tabulated data point, the polynomial calculation was grouped by tabulated data points $\mathbf{y}(l+i)$ instead of by coefficient p_i, to achieve a fully vectorized computation. For example, for a fourth-degree polynomial, one has

$$\begin{aligned} \mathbf{P_4}(r) = &\left(\frac{5}{120}r^4 + \frac{5}{60}r^3 - \frac{5}{120}r^2 + \frac{5}{60}r\right) \cdot \mathbf{y}(-2) \\ &+ \left(-\frac{1}{6}r^4 + \frac{1}{6}r^3 + \frac{2}{3}r^2 - \frac{2}{3}r\right) \cdot \mathbf{y}(-1) \\ &+ \left(\frac{1}{4}r^4 - \frac{5}{4}r^2 + 1\right) \cdot \mathbf{y}(0) \\ &+ \left(-\frac{1}{6}r^4 - \frac{1}{6}r^3 + \frac{2}{3}r^2 + \frac{2}{3}r\right) \cdot \mathbf{y}(+1) \\ &+ \left(\frac{5}{120}r^4 + \frac{5}{60}r^3 - \frac{5}{120}r^2 - \frac{5}{60}r\right) \cdot \mathbf{y}(+2); \end{aligned} \quad (24)$$

By differentiating the polynomial formulation of Equation 21, one has a fully vectorized method to estimate both polynomial function reconstruction and any of its derivatives with the same set of vectorized table retrievals:

$$\begin{aligned} \frac{\partial \mathbf{P}}{\partial x} &= \frac{\partial \mathbf{P}}{\partial r}\frac{\partial r}{\partial x} = \frac{1}{\Delta x}\frac{\partial \mathbf{P}}{\partial r}; \\ \frac{\partial^2 \mathbf{P}}{\partial x^2} &= \frac{\partial^2 \mathbf{P}}{\partial r^2}\left(\frac{\partial r}{\partial x}\right)^2 = \frac{1}{(\Delta x)^2}\frac{\partial^2 \mathbf{P}}{\partial r^2}, \end{aligned} \quad (25)$$

since $\partial r/\partial x = 1/\Delta x$ is a constant. By employing the proposed tabulation and polynomial interpolation approach, several advantages versus traditional piecewise polynomial reconstruction functions (spline, PCHIP) could be achieved:

- only one datum, i.e., the function value, is stored per sampling point: no derivatives or other pre-computed polynomial coefficients are needed. Hence, smallest memory footprint possible per sampling point is achieved;

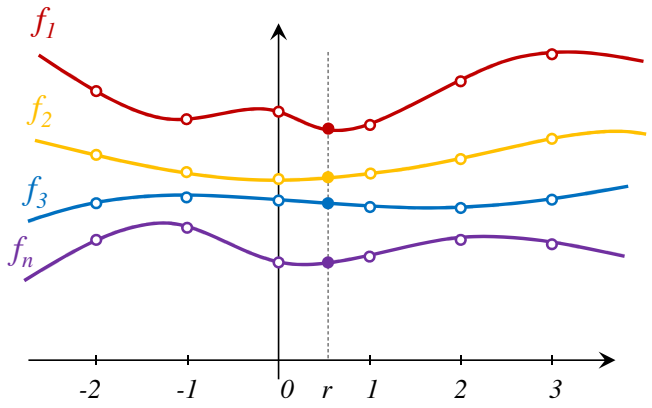

Figure 9: Vector function tabulation at equally-spaced points.

- Note that the same tabulated data are retrieved for both function and derivative. This increases the overall computational efficiency in case both function and its derivatives are needed at the same time, plus, it saves storage from the derivative formulations which would otherwise have to be tabulated as well;

- The linear system of Equation 23 produces universally-valid results per same polynomial degree, which can be hard-wired into the code as highly efficient routines. The implementation in the current work features piecewise polynomials and their derivatives up to degree $n_p = 6$.

- Both table retrievals and the interpolation functions are fully vectorized, which makes the method particularly suitable to take advantage of modern CPU architectures;

- The problem of oscillating interpolants is minimized when dealing with functions such as exponentials, which span several tens of orders of magnitude. In fact, when using piecewise polynomial methods based on solving linear systems such as spline and PCHIP, truncation errors occur, which can cause severe oscillations in the interpolant. This issue is avoided since the piecewise interpolant is computed based on a subset of neighbor tabulated points, not by solving systems over the whole sampled set.

3. Results and Discussion

The computational performance of the proposed methods to speed up the evaluation of exponential and logarithm functions was assessed for ignition delay

calculations of hydrocarbon fuels using the `SpeedCHEM` package [2, 11]. The package, written in modern Fortran, solves zero-dimensional conservation equations of mass and energy in homogeneous reactive gas-phase environments, reported in Equation 1. Computational performance of the `SpeedCHEM` package scales linearly with reaction mechanism size, thanks to vectorized property handling, a fully analytical Jacobian implementation of the chemical kinetics system [2], and usage of sparse matrix computation for both the Jacobian construction and its solution [11]; the package, available online open-source, has been successfully employed for zero-dimensional reactor modeling and for detailed multidimensional combustion simulations [20–24].

3.1. Homogeneous reactors.

The study featured 11 reaction mechanisms of sizes ranging from 10 to 7171 species, as reported in Figure 10. For each reaction mechanism, ignition delay time integration problems were simulated for 31 initial reactor configurations of interest to combustion modeling: initial air-fuel mixture equivalence ratio $\phi_0 = 0.5, 1.0, 2.0$; reactor pressure $p_0 = 10, 40 bar$; temperature $T_0 = 750, 925, 1100, 1275, 1450 K$. The initial charge was initialized as a fuel-air mixture of given equivalence ratio of standard air (`O2/N2` or `O2/N2/Ar` where available) with a mechanism-specific fuel surrogate, whose mass composition is reported together with other mechanism details in Table 3.1.

The study focused on the global integration performance of introducing approximate formulations for exponential- and logarithm-based kinetics functions; so, solver settings were maintained constant for all simulations. These featured usage of the sparse LSODE solver [25], with relative and absolute integration tolerances `RTOL = 1.0e-04` and `ATOL = 1.0e-15`, suitable for multidimensional simulations; sparse analytical Jacobian formulation of the chemical kinetics system, and linear system solution employing a direct sparse solver; i.e., neither iterative solutions nor preconditioning of the Jacobian matrix were activated.

The fast exponential/logarithm approximation applied to several temperature-dependent functions in the reaction mechanism, whose baseline computational cost is reported per element in Figure 11. Figure 11 highlights that costs associated with some of the most relevant temperature-dependent functions are essentially independent of mechanism size, suggesting that all dense vector functions that define the system's state and reactivity do not benefit from mechanism sparsity. $\mathbf{k}_\infty$ represents Arrhenius reaction rate at the high-pressure limit:

$$k_{\infty,j}(T) = A_j T^{b_j} e^{-\frac{E_j}{RT}}, \tag{26}$$

with size ($\mathtt{n_r}$), already subject to investigation for speed up in [1]; $\mathbf{c}_p$ is the ($\mathtt{n_s}$) species' constant-pressure specific heat ($J/mol/K$), defined according to

JANAF's 7-coefficient polynomial format:

$$c_{p,i}(T) = R\left(a_i + b_i T + c_i T^2 + d_i T^3 + e_i T^4\right); \tag{27}$$

$\mathbf{K}_{eq,c}$, of size ($\mathtt{n_{r,eq}}$) is the concentration-based equilibrium constant. This constant defines the ratio between forward and reverse reaction rates, and applies to all $n_{r,eq}$ equilibrium-based reversible reactions in the mechanisms. The equilibrium ratio applies to all reactions according to the theory, but it is common practice in mechanism development to override it with a user-defined reverse reaction rate in Arrhenius form (`REV`), to better match experimental data, so in general $n_{r,eq} < n_r$:

$$K_{eq,c,j}(T) = \exp\left(-\Delta g_j^0\right)\left[\frac{p_{atm}}{RT}\right]^{\sum_{i=1}^{n_s} \nu''_{j,i} - \nu'_{j,i}},$$

where $\Delta g_j^0 = \sum_{i=1,n_s}\left(\nu''_{j,i} - \nu'_{j,i}\right) g_i^0$ is the reaction's change in non-dimensional Gibbs free energy $g_i^0(T)$ between reactants and products. Troe's centering factor $log_{10}\mathbf{F}_{cent}$ [26], with size ($\mathtt{n_{TROE}}$), is the temperature-dependent part of the pressure weighting parameter in Troe-formulated pressure dependent reactions [2]

$$F_{cent,j} = (1 - a_j)\exp\left(-\frac{T}{T_{3,j}}\right) + a_j \exp\left(-\frac{T}{T_{1,j}}\right) + \exp\left(-\frac{T_{2,j}}{T}\right); \tag{28}$$

in Figure 11, these temperature-dependent quantities are also compared to bulk CPU times for the evaluation of species concentration rates of change $d\omega/dt$, the Jacobian matrix J ($\mathtt{n_s + 1\,x\,n_s + 1}$), and its sparse LU solve.

Mechanism	$\mathbf{n_s}$	$\mathbf{n_r}$	**fuel**	**composition**	**ref.**
Hydrogen	10	21	hydrogen	$[H_2 : 1.0]$	[27]
ERC n-heptane	29	52	n-heptane	$[nC_7H_{16} : 1.0]$	[28]
ERC PRF	47	142	PRF25	$[nC_7H_{16} : 0.75, iC_8H_{18} : 0.25]$	[29]
Wang PRF	86	392	PRF25	$[nC_7H_{16} : 0.75, iC_8H_{18} : 0.25]$	[30]
ERC multiChem	128	503	PRF25	$[nC_7H_{16} : 0.75, iC_8H_{18} : 0.25]$	[31]
LLNL n-heptane (red.)	160	1540	n-heptane	$[nC_7H_{16} : 1.0]$	[32]
LLNL n-heptane (det.)	654	2827	n-heptane	$[nC_7H_{16} : 1.0]$	[33]
LLNL PRF	1034	4236	PRF25	$[nC_7H_{16} : 0.75, iC_8H_{18} : 0.25]$	[33]
LLNL n-hexadecane	2837	10719	DPRF58	$[nC_{16}H_{34} : 0.42, iC_{16}H_{34} : 0.58]$	-
LLNL mehtyl-decanoate	2878	8555	methyl-decanoate	$[md : 1.0]$	[34]
LLNL n-alkanes	7171	31669	diesel surrogate	$[nC_7H_{16} : 0.4, nC_{16}H_{34} : 0.1, nC_{14}H_{30} : 0.5]$	[35]

Table 3: Overview of the reaction mechanisms used for testing in this study.

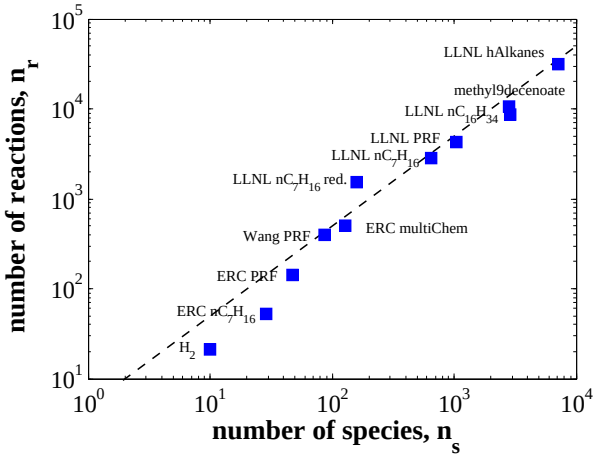

Figure 10: Range of reaction mechanism sizes employed in the current study.

The combined effects of the fast exponential/logarithm approximations of section 2.1 and 2.2 are reported in Figures 13 to 16 for classes of mechanism size: tiny (10-29 species), CFD-size (128-160 species), detailed (654-1034 species), comprehensive (2878-7171 species). In the plots, each line represents results achieved with a same exp/log formulation method, while temperature function tabulation/interpolation methods are identified along the x-axis. For the fast exponential approximation, Taylor series with $n = 1, 3, 5$ and spline interpolation with $n = 3, 5$ were tested. For tabulation/interpolation, the developed piecewise polynomial method was tested at all orders from 1 to 5, plus, an 'optimal-degree' version was added which chooses the requested interpolation order at each temperature interval based on a relative accuracy constraint $\varepsilon < 10^{-6}$ versus the exact value which is computed and stored during table initialization. Furthermore, standard global interpolation methods including cubic spline, piecewise cubic hermite polynomial (PCHIP), and Akima spline were included. These latter state-of-the-art methods still produce a piecewise representation of the function, but compute their coefficients a priori by solving a linear system over the whole tabulated dataset instead of inferring them on-the-fly from tabulated data only, such as in the current method.

All plots report relative quantities of the baseline case, which uses double precision intrinsic functions as well as no tabulation/interpolation approaches. Performance was evaluated by total CPU time, total number of integration steps, as well as average relative error on the predicted IDTs over the 31 conditions simulated per case. Since most CPU time during the integration is spent on sparse solver machinery [11], global advantage was expected to be not as large

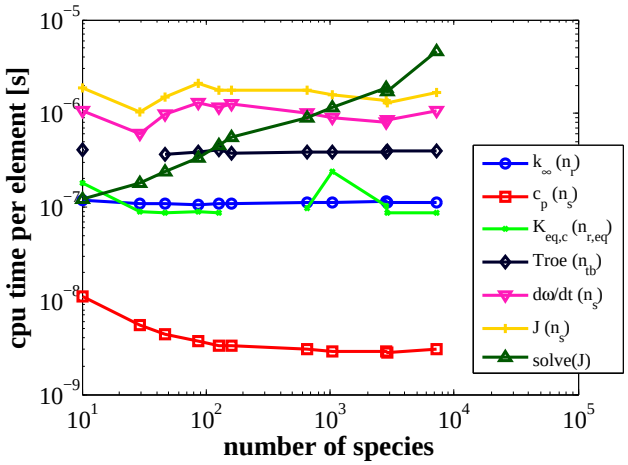

Figure 11: CPU time cost (per element) for several kinetics functions, as a function of reaction mechanism size.

as for the single exponential function evaluation. Thus, the focus of this study was to assess how much total speed-up is achieveable with the proposed approximate formulations, even with an already well-tuned platform.

All cases showed that, regardless of mechanism size, some time advantage was seen for any configuration that led to successful ODE integration. The largest CPU time reduction (down to 18% of the baseline) was seen for the 128-species multiChem mechanism. A still remarkable least amount of reduction, down to 37% was seen for the smallest 10-species mechanism where vectorization of CPU operations might not be well exploited due to the small size. The configurations that produced optimal CPU time results also showed a similar number of integration steps as the baseline case, sometimes even reducing it by a few percent points. This indicates that the smoothly interpolated functions can have a stabilizing effect on the solver by providing smoother temperature-dependent function/derivative relationships, since the system's stiffness is unchanged. Both CPU time and average error have clearly defined patterns with the methods' order. For example, it is seen that linear interpolation is insufficient and leads to several integrator failures. The same observation applies to standard interpolation methods (spline, PCHIP, Akima): they lead to failures or severe performance losses at any scenarios. For them, extreme function ranges that can span tens of orders of magnitude between 300K and 3000K, can cause severe ill-conditioning of the linear system being used to compute their coefficients, causing oscillatory behavior.

The amount of speed-up is maximized by combining fast exponential and data tabulation/interpolation even if good speed-ups can still be achieved as well also

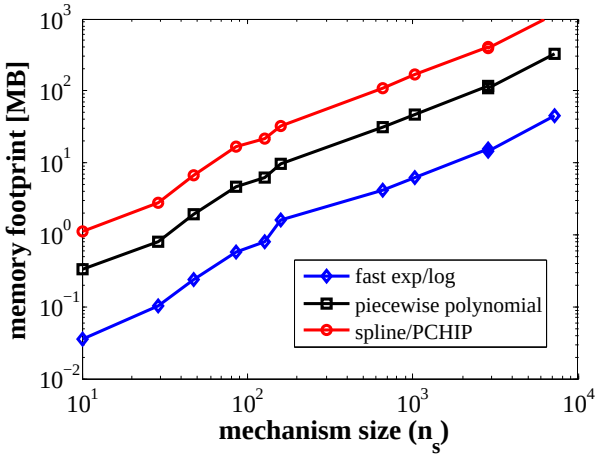

Figure 12: `SpeedCHEM` memory footprint for reactor calculations with the reaction mechanisms of Table 3.1. (blue) fast exponential and logarithm functions, no data tabulation; (black) piecewise polynomial method of section 2.3; (red) global piecewise polynomial interpolation (cubic spline/Hermite). All tabulated data have temperature-dependent function tabulation with $\Delta T = 10K$ in the $[10K, 300K]$ range.

when no tabulation is applied. In this latter case, some sensitivity to the selected approximated method appears. The quintic spline approach outperforming the others in accuracy, while the cubic spline is best in global CPU time.

When faster results are obtained adding tabulation/interpolation, a larger memory footprint is needed, as reported in Figure 12. Here, `SpeedCHEM`'s total RAM footprint is plotted versus mechanism size for the solver configuration employed in the stud,y i.e., it includes the reaction mechanism representation including storage for the sparse algebra manipulations, as well as the ODE solver's working arrays. No additional storage is required when employing fast exponentiation only. In this case, the total memory footprint is a negligible amount of RAM for mechanisms up to medium size ($\approx$ 1 to $10MB$), and has a modest 44MB for the 7171-species mechanism. Tabulation with $\Delta T = 10K$ has instead a noticeable effect on RAM used by the program, and can increase it by approximately one order of magnitude. For the piecewise polynomial interpolation developed, only tabulated data points are needed. For spline, PCHIP and Akima interpolation, additional coefficients are needed per tabulated datapoint, increasing RAM demand even further, up to more than one gigabyte for the largest mechanism. For mechanisms of about 100-200 species typically used in CFD, the piecewise polynomial approach developed in this study has modest RAM needs of approximately 10MB, which makes it suitable for combustion applications in combination with fast exponentiation to maximize CPU time performance.

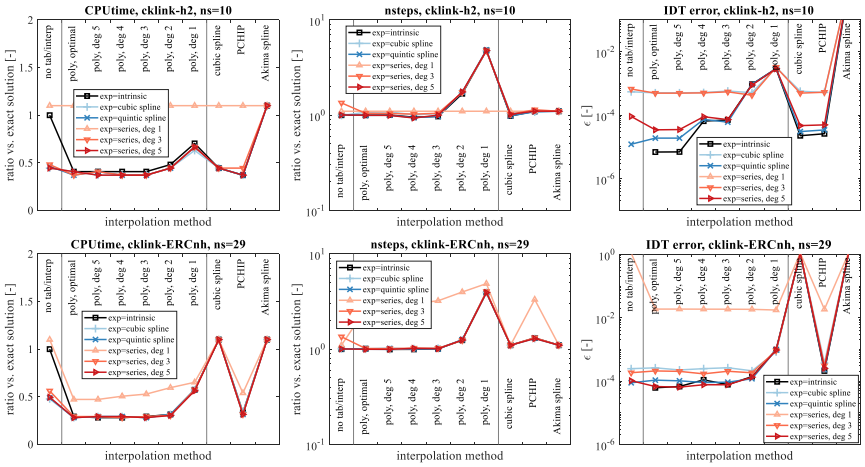

Figure 13: Performance and accuracy comparison of mechanisms of tiny size against `fexp`/`flog` (intrinsic, spline, Taylor series) and tabulation/interpolation formulations (piecewise formulation, global cubic spline/Hermite). (left) global integration CPU time ratio versus baseline case; (center) number of integration steps ratio; (right) average relative error on ignition delay time.

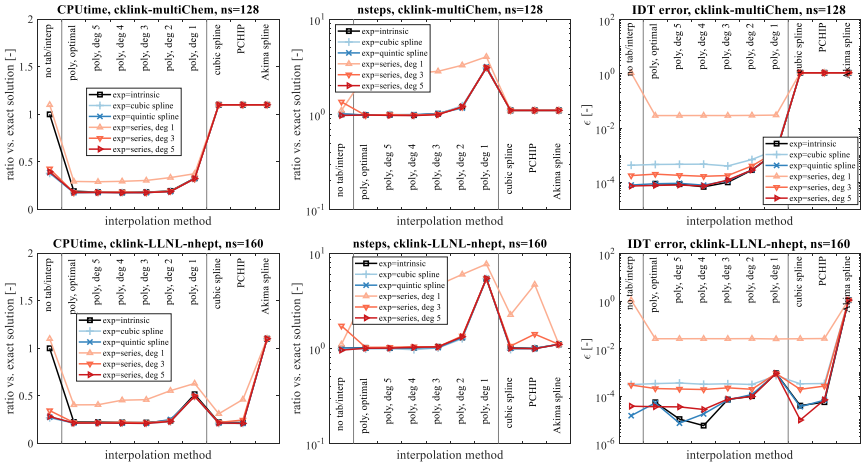

Figure 14: Performance and accuracy comparison of mechanisms of size tailored for CFD calculations against `fexp`/`flog` (intrinsic, spline, Taylor series) and tabulation/interpolation formulations (piecewise formulation, global cubic spline/Hermite). (left) global integration CPU time ratio versus baseline case; (center) number of integration steps ratio; (right) average relative error on ignition delay time.

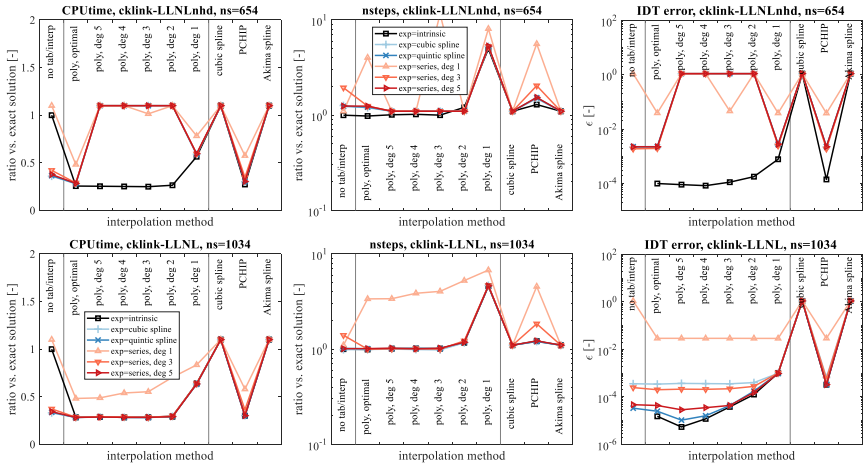

Figure 15: Performance and accuracy comparison of detailed mechanisms against `fexp/flog` (intrinsic, spline, Taylor series) and tabulation/interpolation formulations (piecewise formulation, global cubic spline/Hermite). (left) global integration CPU time ratio versus baseline case; (center) number of integration steps ratio; (right) average relative error on ignition delay time.

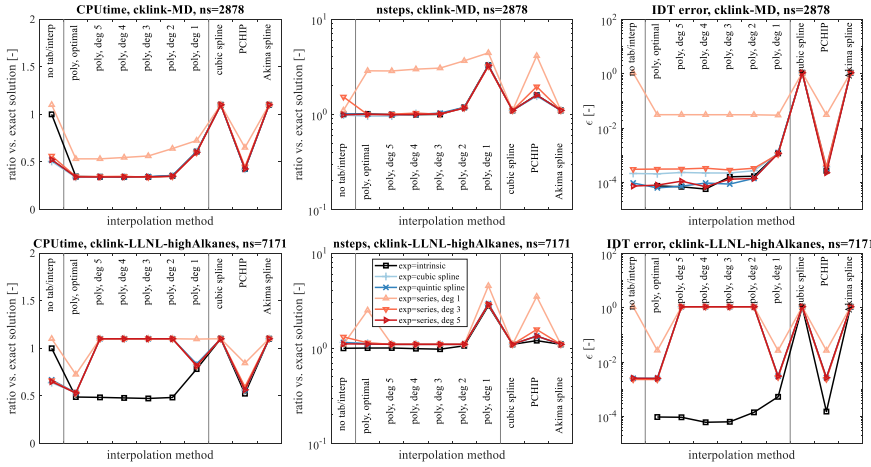

Figure 16: Performance and accuracy comparison of large comprehensive mechanisms against `fexp/flog` (intrinsic, spline, Taylor series) and tabulation/interpolation formulations (piecewise formulation, global cubic spline/Hermite). (left) global integration CPU time ratio versus baseline case; (center) number of integration steps ratio; (right) average relative error on ignition delay time.

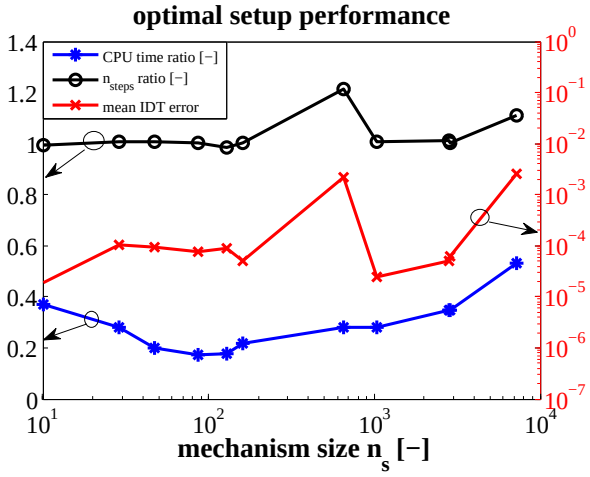

Figure 17: Performance and accuracy of the optimal setup featuring storage/retrieval with piecewise polynomial reconstruction of optimal degree, and fast `exp()/log()` functions with quintic spline formulation. Performance expressed as ratios of corresponding quantities in the baseline setup (intrinsic `exp()/log()`, no storage/retrieval.

Figure 17 summarizes performance and accuracy with an optimal choice of methods. This setup features fast `exp()` and `log()` functions with quintic spline formulation, and storage/retrieval with optimal-degree piecewise polynomial interpolation at $\epsilon = 10^{-6}$. This configuration succeeded at all IDT cases and for all mechanisms. By introducing an approximation in the exponential functions one would expect a trade-off between improved CPU efficiency due to faster evaluation of those functions, and additional overhead due to lesser simulation stability, perhaps leading to an increase in number of integration timesteps. However, compared with the baseline case, which employs intrinsic functions and no storage/retrieval, consistent ODE solver performance was observed, with needed integration steps ranging from -1.6% to $+1.0\%$, except for two mechanisms ([33],[35]) whose integration used up to $+21.5\%$ more steps than the baseline. Overall integration times were smoothly lower than the baseline case for all mechanism, ranging from a speed-up of -46.9% for the largest mechanism, to a peak of -82.3% CPU time for the multiChem mechanism [31]. Performance improvement worsens for very large mechanisms where sparse linear algebra overhead dominates over the whole integration time. Integral relative error on predicted ignition delays was well within acceptable accuracy limits set by the ODE solver tolerance, `RTOL = 1.0e-4`, as also highlighted in Figure 18: instantaneous temperature and species mass fraction profiles computed using either temperature function tabulation/retrieval or fast `exp()/log()` calculation are virtually undistinguishable from the exact ones.

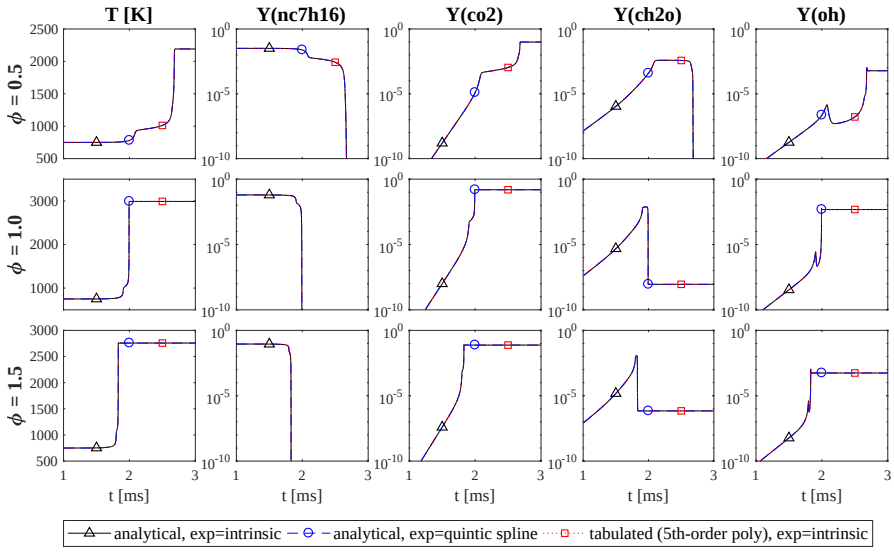

Figure 18: Effect of fast exponential functions and tabulation ($\Delta T = 10K$) on predicted temperature and mass fraction profiles for relevant species (n-heptane, carbon dioxide, formaldehyde and hydroxyl radiacal) with the reaction mechanism of [32]. Initial PRF25-air mixture at pressure $p_0 = 40bar$, temperature $T_0 = 750K$, and equivalence ratios $\phi = 0.5, 1.0, 1.5$.

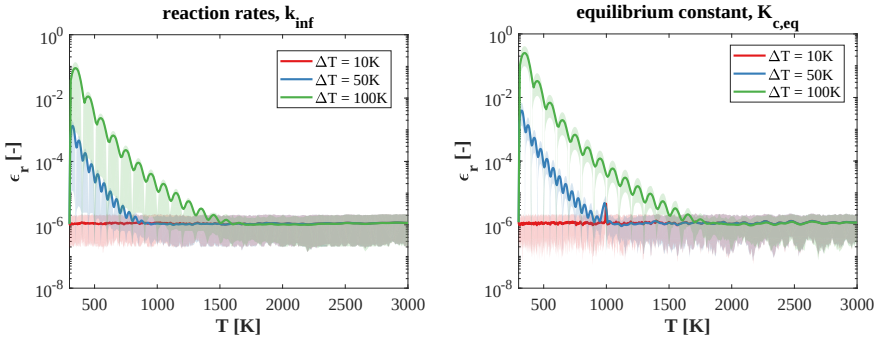

Figure 19: Relative error from tabulation/5^{th}-degree interpolation procedure for (left) forward reaction rates, (right) equilibrium constants.

3.2. Variable range sampling.

The proposed piecewise interpolation algorithm only stores function values at the sampled points, and does not require to pre-compute and store additional parameters per point. Hence, the accuracy of the resulting interpolating functions can only be adjusted by choosing a suitable sampling interval. For combustion chemical kinetics applications, a reasonable total temperature range covers approximately $300K$ to $3000K$. Within this range, functions describing thermodynamic properties (c_p, c_v, etc.) typically vary less than a few times; relative errors due to interpolation were observed to be well within double-precision accuracy for several choices of the temperature sampling interval. Kinetics functions involving exponentials vary by several tens of orders of magnitude instead: making the choice of a suitable sampling step critical to their accuracy. Figure 19 reports the relative error of interpolated kinetics functions (forward reaction rates, k_{inf}, and equilibrium constant, $K_{c,eq}$) for three sampling step values of 10, 50 and $100K$, and 5^{th}-degree polynomial interpolation. Each line represents the average error over the whole set of reactions, while each shaded colored band represents the range between 25^{th} and 75^{th} percentiles of the relative error band at that sampling step size.

3.3. ODE solver convergence features.

Tabulation was not seen to destabilize ODE integration convergence at different solver tolerances, as reported in Figure 20, where the set of homogeneous reactor conditions tested was employed to measure the ratio between the number of integration steps needed when employing tabulated temperature-dependent functions versus their exact computation:

- With a temperature sampling stepsize of $10K$ or less, essentially the same number of steps as from the exact solution was observed. When larger

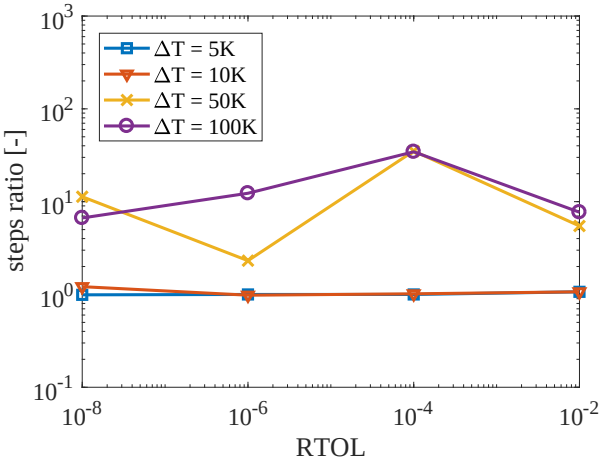

Figure 20: Ratio between number of integration steps using tabulation/interpolation versus exact analytical computations, at different ODE solver tolerances: `RTOL=1e-2, 1e-4, 1e-6, 1e-8`, and `ATOL=1e-10, 1e-15, 1e-18, 1e-20`, respectively.

sampling stepsizes (50 or $100K$) were employed, the number of integration steps increased by about an order of magnitude even for very large tolerances.

- In either scenario, tabulation accuracy did not dramatically affect the number of steps across the tolerance range, which had more or less always the same ratio with the number of steps when using the exact functions.

3.4. Engine calculations.

Further validation of the proposed approach was carried out against combustion CFD calculations in internal combustion engines, where a broad range of composition-pressure-temperature variables is experienced, which, if not appropriately accurate, could potentially lead to stability issues. A 2D HCCI engine combustion simulation with a two-dimensional sector mesh was performed employing the FRESCO platform, a combustion CFD code being developed at the University of Wisconsin.

The HCCI experiments by Dempsey et al. [36] were modeled, using a PRF50 fuel, employing a 2D computational mesh with 3096 cells at bottom dead center as described in[21]. Three PRF mechanisms from Table 3.1 were employed, from 47 to 1034 species. Chemistry was integrated using `RTOL=1e-4` and `ATOL=1e-15`. The comparison between exact functions and tabulation with fast exponentiation in Figure 21 highlighted very good consistency for both thermodynamic

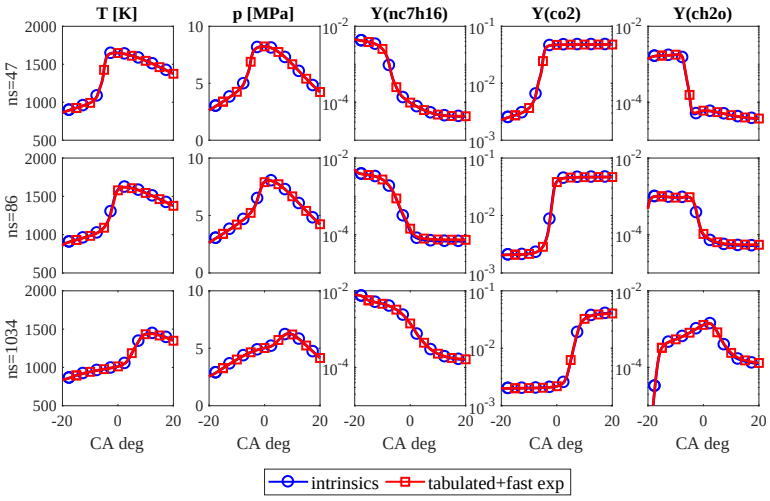

Figure 21: Comparison between analytical kinetics functions and tabulation/polynomial interpolation with fast `exp()/log()`: 2D HCCI combustion, with reaction mechanism sizes $n_s = 47, 86, 1034$ (top to bottom).

quantities and instantaneous species mass fractions, regardless of reaction mechanism size. The speedup achieved by tabulation and fast exponentiation was of 1.70× for $n_s = 47$, 1.81× for $n_s = 86$, and 1.33× for $n_s = 1034$, respectively.

A three-dimensional spray combustion simulation in a diesel engine was run as well. The simulation modeled diesel combustion in the Sandia National Laboratories 1.9L, light-duty optical diesel engine, operating a single-pulse injection, conventional diesel combustion condition (CDC9) at 9bar IMEP load, and 19.7% O_2 molar fraction in the intake charge. Details of the engine and its operating conditions can be found in [37]. The injected fuel was a binary Diesel Primary Reference Fuel mixture DPRF58, made of 58% heptamethylnonane and 42% n-hexadecane, injected mass 26.7 mg. Its combustion was simulated using the ERC multiChem mechanism (ns=229 species) [38], using direct mapping of the liquid-phase to the gas-phase species. Figure 22 reports a comparison of predicted bulk in-cylinder quantities, while local temperature and NO_x mass fraction contours are reproduced in Figure 23, at a vertical cut-plane through the injection axis. The comparison shows satisfactory reproducibility of the results even in a complex CFD framework, not only at the scale of bulk quantities, but also locally within the domain. The tabulation approach reduced the amount of time spent on chemistry calculations in the simulation from 47.9 to 29.2 hours on 16 CPUs, corresponding to a speed up of 1.65×.

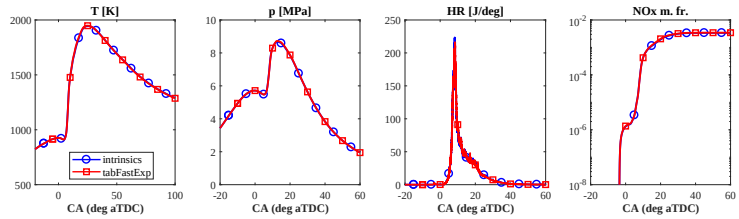

Figure 22: Predicted in-cylinder temperature, pressure, heat released and NOx mass fraction in the SNL 1.9L engine, CDC9 operating condition. Comparison between analytical kinetics functions with `exp()` intrinsics (red), or tabulation ($\Delta T = 10K$) with 5^{th}-degree polynomial interpolation and quantic spline fast `exp()`/`log()` (blue).

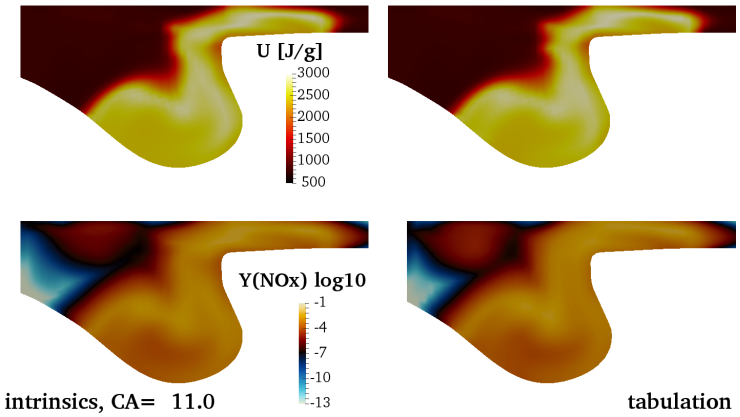

Figure 23: Predicted (top) internal energy U and (bottom) NO_x mass fr. fields in the SNL 1.9L engine, CDC9 operating condition, $CA = 11 degaTDC$, at vertical cut-planes through the injection axis within the combustion chamber. Comparison between analytical kinetics functions with `exp()` intrinsics (left), or tabulation ($\Delta T = 10K$) with 5^{th}-degree polynomial interpolation and quantic spline fast `exp()`/`log()`.

4. Concluding remarks

We developed two fast approximation approaches for costly combustion kinetics quantities involving the exponential and logarithm functions. The first approach developed fast `exp(x)` and `log(x)` functions in 64-bit precision exploiting low-level operations in the number's bit representation. The second approach developed a piecewise polynomial formulation for equally-spaced tabulated points, which achieves speed up by combined storage/retrieval of tabulated data with a fast and data-independent interpolation method of arbitrary degree. The two approaches were tested, both separately and combined, for solving a set of ignition delay time integration problems with the `SpeedCHEM` package and their performance was assessed in terms of global speed-up, accuracy and solver stability. Based on the analysis, the following conclusions could be drawn:

- All fast exponential and logarithm functions provided approximately one-order-of-magnitude speed-ups versus the respective accurate intrinsics. An interpolation order of at least $n = 3$ was needed for robust chemistry ODE integration. The spline formulations outperformed the Taylor series ones in both evaluation time and chemistry ODE performance.

- Fast exponentiation could alone achieve global chemistry solution speed-ups larger than -60% for CFD-sized mechanisms where sparse linear algebra is already demanding. However, the best performance was always achieved when fast exponentiation was coupled with tabulation of temperature-dependent functions with piecewise polynomial reconstruction of degree $n \geq 5$; here, speed-ups greater than -80% were achieved for CFD-sized mechanisms.

- The tabulation/interpolation approach for costly temperature dependent functions allowed significant speed-ups thanks to storage/retrieval; plus use of a fully vectorized and function-independent piecewise polynomial formulation that applies the same few scalar coefficients to vector functions of arbitrarily large size.

- The piecewise polynomial method developed in this study outperformed standard piecewise interpolation methods in two aspects: 1) smaller memory footprint – no derivative data has to be stored in addition to the tabulated function values; 2) better error performance: the polynomial coefficients are not solved from a global linear system, such as in other piecewise polynomial methods, hence avoiding ill-conditioning when the function spans several orders of magnitude, such as with exponentials.

A modern Fortran implementation of the spline versions of the fast Exponential and Logarithm functions developed in this study is reported in Appendix A.

5. Acknowledgments

The authors wish to acknowledge support for part of this research provided through the Sandia National Laboratories by the U.S. Department of Energy, Office of Vehicle Technologies, program managers Leo Breton, Gupreet Singh.

Appendix A. Modern Fortran implementations

```
! Fast exp(x) - fifth-degree spline
elemental real(real64) function fexp_quintic(x) result(exp_x)
  use iso_fortran_env, only: real64,int64
  implicit none
real(real64), intent(in) :: x

  ! Local variables and parameters
  integer(int64), parameter :: mantissa = 2_int64**52
  integer(int64), parameter :: bias = 1023_int64
  integer(int64), parameter :: ishift = mantissa*bias
  real(real64), parameter :: log2 = log(2.0_real64)
  real(real64), parameter :: s5(5)= [-1.90188191959304e-3_real64,&
         -9.01146535969578e-3_real64,-5.57129652016652e-2_real64,&
         -2.40226506959101e-1_real64, 3.06852819440055e-1_real64]
  real(real64) :: y,yf
  integer(int64) :: i8

  y  = x*log2      ! Change of base: e^x -> 2^y
  yf = y-floor(y)  ! Compute fractional part
  y  = y-((((s5(1)*yf+s5(2))*yf+s5(3))*yf+s5(4))*yf+s5(5))*yf ! Add Delta
  i8 = mantissa*y + ishift   ! Perform Integer operation
  exp_x = transfer(i8,exp_x) ! Cast to real
end function fexp_quintic

! Fast log(x) - fifth-degree spline
elemental real(real64) function flog_quintic(x) result(log_x)
  use iso_fortran_env, only: real64,int64
  implicit none
  real(real64), intent(in) :: x

  ! Local variables and parameters
  real(real64) :: yi,yf
  integer(int64) :: i8
  integer(int64), parameter :: mantissa = not(shiftl(2047_int64,52))
```

```
  integer(int64), parameter :: mantissa_left = 2_int64**52
  integer(int64), parameter :: bias     = 1023_int64
  integer(int64), parameter :: ishift   = mantissa_left*bias
  real(real64), parameter :: log2 = log(2.0_real64)
  real(real64), parameter :: s5(5)= [ 1.44269504088896e+0_real64,&
        -7.21347520444482e-1_real64, 4.42145354110618e-1_real64,&
        -2.12375830888126e-1_real64, 4.88829563330264e-2_real64]

  ! Extract exponent
  i8 = transfer(x,i8)
  yi = shiftr(i8,52)-bias
  ! Extract mantissa
  yf = transfer(iand(i8,mantissa)+ishift,yf)-1.0_real64
  ! Apply quintic polynomial
  yf = yf*(s5(1)+yf*(s5(2)+yf*(s5(3)+yf*(s5(4)+yf*s5(5)))))
  ! Change of base: log_2(x) -> log_e(x) = log2*log_2(x)
  log_x = (yf+yi)*log2

end function flog_quintic
```

References

[1] T. Lu, C. K. Law, Toward accommodating realistic fuel chemistry in large-scale computations, Progress in Energy and Combustion Science 35 (2) (2009) 192 – 215. doi:10.1016/j.pecs.2008.10.002.
URL http://www.sciencedirect.com/science/article/pii/S036012850800066X

[2] F. Perini, E. Galligani, R. Reitz, An analytical Jacobian approach to sparse reaction kinetics for computationally efficient combustion modelling with large reaction mechanisms, Energy & Fuels 26 (8) (2012) 4804–4822. doi:10.1021/ef300747n.
URL http://pubs.acs.org/doi/abs/10.1021/ef300747n

[3] D. J. Torres, M. F. Trujillo, KIVA-4: An unstructured ALE code for compressible gas flow with sprays, Journal of Computational Physics 219 (2) (2006) 943 – 975. doi:http://dx.doi.org/10.1016/j.jcp.2006.07.006.
URL http://www.sciencedirect.com/science/article/pii/S002199910600338X

[4] A. Sandu, J. Verwer, J. Blom, E. Spee, G. Carmichael, F. Potra, Benchmarking stiff ODE solvers for atmospheric chemistry problems II: Rosenbrock solvers, Atmospheric Environment 31 (20) (1997) 3459 – 3472. doi:http://dx.doi.org/10.1016/S1352-2310(97)83212-8.
URL http://www.sciencedirect.com/science/article/pii/S1352231097832128

[5] V. Damian, A. Sandu, M. Damian, F. Potra, G. R. Carmichael, The kinetic preprocessor KPP-a software environment for solving chemical kinetics, Computers and Chemical Engineering 26 (11) (2002) 1567 – 1579. doi:http://dx.doi.org/10.1016/S0098-1354(02)

00128-X.
URL http://www.sciencedirect.com/science/article/pii/S009813540200128X

[6] F. Bisetti, Integration of large chemical kinetic mechanisms via exponential methods with krylov approximations to jacobian matrix functions, Combustion Theory and Modelling 16 (3) (2012) 387–418. arXiv:http://dx.doi.org/10.1080/13647830.2011.631032, doi:10.1080/13647830.2011.631032.
URL http://dx.doi.org/10.1080/13647830.2011.631032

[7] B. Savard, Y. Xuan, B. Bobbitt, G. Blanquart, A computationally-efficient, semi-implicit, iterative method for the time-integration of reacting flows with stiff chemistry, Journal of Computational Physics 295 (2015) 740 – 769. doi:http://dx.doi.org/10.1016/j.jcp.2015.04.018.
URL http://www.sciencedirect.com/science/article/pii/S0021999115002648

[8] M. J. McNenly, R. A. Whitesides, D. L. Flowers, Faster solvers for large kinetic mechanisms using adaptive preconditioners, Proceedings of the Combustion Institute 35 (1) (2015) 581 – 587. doi:http://dx.doi.org/10.1016/j.proci.2014.05.113.
URL http://www.sciencedirect.com/science/article/pii/S1540748914001163

[9] A. Cuoci, A. Frassoldati, T. Faravelli, E. Ranzi, OpenSMOKE++: An object-oriented framework for the numerical modeling of reactive systems with detailed kinetic mechanisms, Computer Physics Communications 192 (2015) 237 – 264. doi:http://dx.doi.org/10.1016/j.cpc.2015.02.014.
URL http://www.sciencedirect.com/science/article/pii/S0010465515000715

[10] D. A. Schwer, J. E. Tolsma, W. H. Green, P. I. Barton, On upgrading the numerics in combustion chemistry codes, Combustion and Flame 128 (3) (2002) 270 – 291. doi:10.1016/S0010-2180(01)00352-2.
URL http://www.sciencedirect.com/science/article/pii/S0010218001003522

[11] F. Perini, E. Galligani, R. D. Reitz, A study of direct and Krylov iterative sparse solver techniques to approach linear scaling of the integration of chemical kinetics with detailed combustion mechanisms, Combustion and Flame 161 (5) (2014) 1180 – 1195. doi:http://dx.doi.org/10.1016/j.combustflame.2013.11.017.
URL http://www.sciencedirect.com/science/article/pii/S0010218013004306

[12] K. E. Niemeyer, C.-J. Sung, Accelerating moderately stiff chemical kinetics in reactive-flow simulations using gpus, Journal of Computational Physics 256 (2014) 854 – 871. doi:http://dx.doi.org/10.1016/j.jcp.2013.09.025.
URL http://www.sciencedirect.com/science/article/pii/S0021999113006396

[13] D. Zuras, M. Cowlishaw, A. Aiken, M. Applegate, D. Bailey, S. Bass, D. Bhandarkar, M. Bhat, D. Bindel, S. Boldo, S. Canon, S. R. Carlough, M. Cornea, M. Cowlishaw, J. H. Crawford, J. D. Darcy, D. Das Sarma, M. Daumas, B. Davis, M. Davis, D. Delp, J. Demmel, M. A. Erle, H. A. H. Fahmy, J. P. Fasano, R. Fateman, E. Feng, W. E. Ferguson, A. Fit-Florea, L. Fournier, C. Freitag, I. Godard, R. A. Golliver, D. Gustafson, M. Hack, J. R. Harrison, J. Hauser, Y. Hida, C. N. Hinds, G. Hoare, D. G. Hough, J. Huck, J. Hull, M. Ingrassia, D. V. James, R. James, W. Kahan, J. Kapernick, R. Karpinski, J. Kidder, P. Koev, R.-C. Li, Z. A. Liu, R. Mak, P. Markstein, D. Matula, G. Melquiond, N. Mori, R. Morin, N. Nedialkov, C. Nelson, S. Oberman, J. Okada, I. Ollmann, M. Parks, T. Pittman, E. Postpischil, J. Riedy, E. M. Schwarz, D. Scott, D. Senzig, I. Sharapov, J. Shearer, M. Siu, R. Smith, C. Stevens, P. Tang, P. J. Taylor, J. W. Thomas, B. Thompson, W. Thrash, N. Toda, S. D. Trong, L. Tsai, C. Tsen,

F. Tydeman, L. K. Wang, S. Westbrook, S. Winkler, A. Wood, U. Yalcinalp, F. Zemke, P. Zimmermann, IEEE standard for Floating-Point arithmetic, Tech. rep., IEEE (Aug. 2008). doi:10.1109/ieeestd.2008.4610935.
URL http://mfkp.org/INRMM/article/4001400

[14] N. N. Schraudolph, A fast, compact approximation of the exponential function, Neural Computation 11 (4) (1999) 853–862. arXiv:http://dx.doi.org/10.1162/089976699300016467, doi:10.1162/089976699300016467.
URL http://dx.doi.org/10.1162/089976699300016467

[15] A. C. I. Malossi, Y. Ineichen, C. Bekas, A. Curioni, Fast exponential computation on simd architectures, in: HiPEAC 2015 - 1st Workshop On Approximate Computing (WAPCO), 2015, pp. 1–6. doi:10.13140/2.1.4362.3207.

[16] G. C. Cawley, On a fast, compact approximation of the exponential function, Neural Computation 12 (9) (2000) 2009–2012. arXiv:http://dx.doi.org/10.1162/089976600300015033, doi:10.1162/089976600300015033.
URL http://dx.doi.org/10.1162/089976600300015033

[17] A. Ralston, H. Wilf, Mathematical methods for digital computers, no. v. 1 in Mathematical Methods for Digital Computers, Wiley, 1960.

[18] L. Piegl, W. Tiller, The NURBS Book (2Nd Ed.), Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[19] K. E. Iverson, A Programming Language, John Wiley & Sons, Inc., New York, NY, USA, 1962.

[20] F. Perini, A. Krishnasamy, Y. Ra, R. D. Reitz, Computationally efficient simulation of multicomponent fuel combustion using a sparse analytical jacobian chemistry solver and high-dimensional clustering, Journal of Engineering for Gas Turbines and Power 136 (9) (2014) 091515–091515. doi:10.1115/icef2013-19039.
URL http://dx.doi.org/10.1115/1.4027280

[21] F. Perini, B. D. Adhikary, J. H. Lim, X. Su, Y. Ra, H. Wang, R. Reitz, Improved chemical kinetics numerics for the efficient simulation of advanced combustion strategies, SAE International Journal of Engines 7 (1) (2014) 243–255. arXiv:http://saeeng.saejournals.org/content/7/1/243.full.pdf+html, doi:10.4271/2014-01-1113.
URL http://saeeng.saejournals.org/content/7/1/243.abstract

[22] F. Perini, D. Sahoo, P. C. Miles, R. D. Reitz, Modeling the ignitability of a pilot injection for a diesel primary reference fuel: Impact of injection pressure, ambient temperature and injected mass, SAE Int. J. Fuels Lubr. 7 (2014) 48–64. doi:10.4271/2014-01-1258.
URL http://doi.org/10.4271/2014-01-1258

[23] F. Perini, R. D. Reitz, Computationally efficient dimension reduction of combustion chemistry via principal components analysis based domain partitioning, in: 2nd Frontiers in Computational Physics - Energy Sciences conference, Zurich, 2015, pp. 1–30.

[24] F. Perini, Y. Ra, K. Hiraoka, K. Nomura, A. Yuuki, Y. Oda, C. Rutland, R. Reitz, An efficient level-set flame propagation model for hybrid unstructured grids using the g-equation, SAE Int. J. Engines 9 (2016) 1409–1424. doi:10.4271/2016-01-0582.
URL http://doi.org/10.4271/2016-01-0582

[25] A. C. Hindmarsh, LSODE and LSODI, two new initial value ordinary differnetial equation solvers, SIGNUM Newsletter 15 (4) (1980) 10–11. doi:10.1145/1218052.1218054.
URL http://doi.acm.org/10.1145/1218052.1218054

[26] J. Troe, Fall-off Curves of Unimolecular Reactions, Berichte der Bunsengesellschaft fur physikalische Chemie 78 (5) (1974) 478–488. doi:10.1002/bbpc.19740780510.
URL http://dx.doi.org/10.1002/bbpc.19740780510

[27] M. O Conaire, H. J. Curran, J. M. Simmie, W. J. Pitz, C. K. Westbrook, A comprehensive modeling study of hydrogen oxidation, International Journal of Chemical Kinetics 36 (11) (2004) 603–622. doi:10.1002/kin.20036.
URL http://dx.doi.org/10.1002/kin.20036

[28] A. Patel, S.-C. Kong, R. D. Reitz, Development and Validation of a Reduced Reaction Mechanism for HCCI Engine Simulations, SAE Technical Paper 2004-01-0558doi:10.4271/2004-01-0558.
URL http://dx.doi.org/10.4271/2004-01-0558

[29] Y. Ra, R. D. Reitz, A reduced chemical kinetic model for IC engine combustion simulations with primary reference fuels, Combustion and Flame 155 (4) (2008) 713 – 738. doi:http://dx.doi.org/10.1016/j.combustflame.2008.05.002.
URL http://www.sciencedirect.com/science/article/pii/S0010218008001351

[30] H. Wang, M. Yao, R. D. Reitz, Development of a reduced primary reference fuel mechanism for internal combustion engine combustion simulations, Energy & Fuels 27 (12) (2013) 7843–7853. arXiv:http://dx.doi.org/10.1021/ef401992e, doi:10.1021/ef401992e.
URL http://dx.doi.org/10.1021/ef401992e

[31] Y. Ra, R. D. Reitz, A combustion model for IC engine combustion simulations with multi-component fuels, Combustion and Flame 158 (1) (2011) 69 – 90. doi:http://dx.doi.org/10.1016/j.combustflame.2010.07.019.
URL http://www.sciencedirect.com/science/article/pii/S0010218010002075

[32] R. Seiser, H. Pitsch, K. Seshadri, W. Pitz, H. Curran, Extinction and autoignition of n-heptane in counterflow configuration, Proceedings of the Combustion Institute 28 (2) (2000) 2029 – 2037. doi:http://dx.doi.org/10.1016/S0082-0784(00)80610-4.
URL http://www.sciencedirect.com/science/article/pii/S0082078400806104

[33] H. Curran, P. Gaffuri, W. Pitz, C. Westbrook, A comprehensive modeling study of iso-octane oxidation, Combustion and Flame 129 (3) (2002) 253 – 280. doi:http://dx.doi.org/10.1016/S0010-2180(01)00373-X.
URL http://www.sciencedirect.com/science/article/pii/S001021800100373X

[34] O. Herbinet, W. J. Pitz, C. K. Westbrook, Detailed chemical kinetic oxidation mechanism for a biodiesel surrogate, Combustion and Flame 154 (3) (2008) 507 – 528. doi:http://dx.doi.org/10.1016/j.combustflame.2008.03.003.
URL http://www.sciencedirect.com/science/article/pii/S0010218008000771

[35] C. K. Westbrook, W. J. Pitz, O. Herbinet, H. J. Curran, E. J. Silke, A comprehensive detailed chemical kinetic reaction mechanism for combustion of n-alkane hydrocarbons from n-octane to n-hexadecane, Combustion and Flame 156 (1) (2009) 181 – 199. doi:http://dx.doi.org/10.1016/j.combustflame.2008.07.014.
URL http://www.sciencedirect.com/science/article/pii/S0010218008002125

[36] A. B. Dempsey, N. R. Walker, E. Gingrich, R. D. Reitz, Comparison of low temperature combustion strategies for advanced compression ignition engines with a focus on controllability, Combustion Science and Technology 186 (2) (2014) 210–241. arXiv:https://doi.org/10.1080/00102202.2013.858137, doi:10.1080/00102202.2013.858137.
URL https://doi.org/10.1080/00102202.2013.858137

[37] S. Busch, K. Zha, P. C. Miles, Investigations of closely coupled pilot and main injections as a means to reduce combustion noise in a small-bore direct injection diesel engine, International Journal of Engine Research 16 (1) (2015) 13–22. arXiv:https://doi.org/10.1177/1468087414560776, doi:10.1177/1468087414560776.
URL https://doi.org/10.1177/1468087414560776

[38] Y. Ra, R. D. Reitz, A combustion model for multi-component fuels using a physical surrogate group chemistry representation (psgcr), Combustion and Flame 162 (10) (2015) 3456 – 3481. doi:http://dx.doi.org/10.1016/j.combustflame.2015.05.014.
URL http://www.sciencedirect.com/science/article/pii/S0010218015001522