Rose-Hulman Undergraduate Mathematics Journal

Volume 8
Issue 1

Article 8

Design and Optimization of Explicit Runge-Kutta Formulas

Stephen Dupal
Rose-Hulman Institute of Technology, stephen.dupal@rose-hulman.edu

Michael Yoshizawa
Pomona College, michael.yoshizawa@pomona.edu

REU NUMERICAL ANALYSIS PROJECT: DESIGN AND OPTIMIZATION OF EXPLICIT RUNGE-KUTTA FORMULAS*

STEPHEN DUPAL AND MICHAEL YOSHIZAWA

ABSTRACT. Explicit Runge-Kutta methods have been studied for over a century and have applications in the sciences as well as mathematical software such as MATLAB's `ode45` solver. We have taken a new look at fourth- and fifth-order Runge-Kutta methods by utilizing techniques based on Gröbner bases to design explicit fourth-order Runge-Kutta formulas with step doubling and a family of (4,5) formula pairs that minimize the higher-order truncation error. Gröbner bases, useful tools for eliminating variables, also helped to reveal patterns among the error terms. A MATLAB program based on step doubling was then developed to compare the accuracy and efficiency of fourth-order Runge-Kutta formulas with that of `ode45`.

1. INTRODUCTION

1.1. **Explicit Runge-Kutta Formulas.** Runge-Kutta methods are a family of methods which produce a sequence $\{x_n, y_n\}_{n=0}^{N}$ of approximating points along the solution curve of the system of ordinary differential equations represented by

$$y'(x) = f(x, y),\ y(0) = y_0, \tag{1}$$

where $f : \mathbf{R} \times \mathbf{R}^m \to \mathbf{R}^m$ is a differentiable vector field and $y_0 \in \mathbf{R}^m$ is the initial-value vector.

An explicit Runge-Kutta formula uses quadrature to approximate the value of (x_{n+1}, y_{n+1}) from (x_n, y_n). As described by Lambert [17], explicit Runge-Kutta formulas take sample derivatives in the solution space to help determine the new solution space for the next step. The actual formula for the s-stage explicit Runge-Kutta method with step

Date: March 18, 2007.

Key words and phrases. Runge-Kutta formula, Gröbner basis.

*This work was supported by NSF grant DMS-0353880.

size h is given by

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i, \tag{2}$$

where

$$k_i = f\left(x_n + c_i h, y_n + h \sum_{j=1}^{i-1} a_{ij} k_j\right), \ i = 1, 2, ...s. \tag{3}$$

The coefficients a_{ij} and c_i are related by

$$c_i = \sum_{j=1}^{i-1} a_{ij}. \tag{4}$$

These coefficients are customarily displayed in a Butcher tableau, which is written shorthand as $\begin{array}{c|c} c & A \\ \hline & \\ & b^{\mathrm{T}} \end{array}$. Figure 1.1 shows the tableau for a four-stage fourth-order formula.

$$\begin{array}{c|cccc} c_1 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ c_4 & a_{41} & a_{42} & a_{43} & \\ \hline & b_1 & b_2 & b_3 & b_4 \end{array}$$

Figure 1.1: Four-stage fourth-order Butcher tableau.

For an elementary introduction to Runge-Kutta formulas, consult Conte and deBoor [5]. Lambert [17] and Hairer and Wanner [9] provide a more advanced treatment.

It is also important to note that often polynomial interpolation is used with Runge-Kutta formulas to find solutions between Runge-Kutta steps. For example, MATLAB's `ode45` solver by default uses interpolation to quadruple the number of solution points to provide a smoother-looking graph. Ideally, this polynomial interpolation will make use of the derivative evaluations already performed by the Runge-Kutta formulas to limit the additional work required.

Besides being found in ODE software such as MATLAB's `ode45`, Runge-Kutta methods have been recently assessed for their effectiveness on stiff ODE problems [14] and partial differential equations [28]. Specialized Runge-Kutta methods are also being developed with applications in the sciences, such as computational acoustics [11], colored noise [10], Hamiltonian waves [21], and Navier-Stokes equations (which

are used in chemical reactions, for instance) [15]. Thus, Runge-Kutta methods continue to be a developing area of research.

1.2. **Butcher Order Conditions and Rooted Trees.** A Runge-Kutta formula has order p if the Taylor series of the computed solution and the exact local solution agree up to terms of order p. Butcher [4] found that the terms of the Taylor series of the computed solution can be represented by rooted trees. Specifically, a Runge-Kutta formula has order p if for every rooted tree τ with p or fewer vertices,

$$b^{\mathrm{T}} A^{(\tau)} = \frac{1}{\tau!}, \tag{5}$$

where $\tau!$ is a certain integer and the components of the vector $A^{(\tau)}$ are certain polynomials in the a_{ij} and c_i, each determined by the structure of τ.

If the formula has order p, but not order $p+1$, we define the leading-order truncation error coefficients by

$$\alpha_\tau \Big(b^{\mathrm{T}} A^{(\tau)} - \frac{1}{\tau!} \Big) \tag{6}$$

for all rooted trees τ having $p+1$ vertices. Each coefficient α_τ is the reciprocal of an integer that is determined by the structure of τ.

Thus, each rooted tree has a corresponding order condition expressed by a polynomial equation in the coefficients b_i, c_i, and a_{ij}. For an s-stage formula there are a total of $s(s+1)/2$ unknowns. Any set of coefficients a_{ij}, b_i, c_i satisfying the polynomial equations for all rooted trees with up to p vertices gives a Runge-Kutta formula of order p.

1.3. **ODE Software and the Control of Local Error.** The accuracy of a Runge-Kutta formula is usually judged by its *local error*. For equation (1), in the step from (x_n, y_n) to (x_{n+1}, y_{n+1}), we define the *local solution* $u_n(x)$ by

$$u_n'(x) = f(x, u_n(x)), \quad u_n(x_n) = y_n.$$

When y_{n+1} is computed using a Runge-Kutta formula of order p, the *local error* is defined to be $u_n(x_{n+1}) - y_{n+1}$ and has an expansion in powers of h of

$$u_n(x_{n+1}) - y_{n+1} = h^{p+1}\varphi_{p+1}(x_n, y_n) + h^{p+2}\varphi_{p+2}(x_n, y_n) + O(h^{p+3}). \tag{7}$$

The coefficient φ_{p+1} is known as the *principal error function* and is expressed in terms of the truncation error coefficients in Equation (6) by

$$\varphi_{p+1} = \sum_{\#\tau = p+1} \alpha_\tau \Big(b^{\mathrm{T}} A^{(\tau)} - \frac{1}{\tau!} \Big) D^{(\tau)} f,$$

where $\#r$ refers to the order of a tree τ and $D^{(\tau)}f$ represents the *elementary differential* corresponding to this tree [4].

For ODE initial value problems, modern software such as MATLAB estimates the local error and controls its step size to keep the local error estimate smaller than a user-supplied tolerance (`Tol`).

For example, on an integration step from x_n to $x_n + h$, the software computes $y_{n+1} \approx y(x_n + h)$ as well as an estimate of the local error `Est`. If $\texttt{Est} \leq \texttt{Tol}$, the step is accepted and the program proceeds to the next step. Otherwise, if $\texttt{Est} > \texttt{Tol}$, the step is rejected and retried with a smaller h. The software adjusts the step by using Equation (7) for the local error, assuming that the principal term $h^{p+1}\varphi_{p+1}$ is dominant.

There are two main strategies for error estimation in ODE software.

1.3.1. *Fehlberg embedding.* Fehlberg embedding is the favored approach by modern software to estimate the local error. It involves finding a set of weights $\hat{b}_i$ that correspond to an equation of lower order than what the weights b_i in Equation (2) provide. Thus, if the weights b yield y_{n+1} of order p, then the weights $\hat{b}$ give $\hat{y}_{n+1}$ of order $p-1$. The error estimate is hence given by

$$\texttt{Est} = \|y_{n+1} - \hat{y}_{n+1}\|.$$

The combination of formulas of orders $p-1$ and p is called a $(p-1, p)$ pair. In Section 3 we derive a family of (4,5) pairs.

1.3.2. *Step doubling.* Step doubling is a form of extrapolation that provides a local error estimate for one-step methods. It compares the solution after a single step of size h with the solution after two half steps of size $h/2$ to achieve an estimate of the local error.

Hence, by using this method, two estimates for the value of y_{n+1} are compared to yield an estimate for the error. Thus,

$$\texttt{Est} = \frac{1}{2^{p+1} - 2}\|y_{n+1} - \hat{y}_{n+1}\|, \tag{8}$$

where $y_n \to y_{n+\frac{1}{2}} \to y_{n+1}$ by two steps of size $\frac{h}{2}$ and $y_n \to \hat{y}_{n+1}$ by one step of size h.

While doubling is not as popular as Fehlberg's embedding method, Shampine [24] supports step doubling as a viable alternative. In fact, the two methods are conceptually very similar.

It is well known that an s-stage Runge-Kutta formula with doubling may be regarded as a $(3s-1)$-stage formula [23]. In Section 2 we adopt this point of view for a new approach to optimizing four-stage fourth-order formulas.

1.4. **Previous work on Explicit Runge-Kutta Formulas.**

1.4.1. *Four-stage fourth-order formulas.* Runge-Kutta formulas can be identified by their respective Butcher tableaus (Figure 1.1 on page 2). However, if there are more a_{ij}, b_i, and c_i coefficients than order conditions, there may be free parameters.

In fact, the constants for a four-stage fourth-order formula (Figure 1.1 on page 2) can all be written as rational functions of c_2 and c_3. Hence, a fourth-order Runge-Kutta formula can be identified just by the values for those two parameters. Early values for these constants were chosen to help with hand computation or to minimize round-off error [13]. With the help of computers, attention later turned toward finding parameters that yield the most accurate results.

Previous results include the classic formula, which is an attractive option for hand computation since the constants a_{31}, a_{41}, and a_{42} are all zero. The Kutta formula was later developed to have improved accuracy. Ralston [20] determined an optimum formula by assuming bounds on the partial derivatives of the vector field. Kuntzmann [16] developed his own optimum formula that eliminates four of the nine fifth-order truncation error coefficients. Hull and Johnston [13] analyzed the error of the fourth-order Runge-Kutta formulas using three different measures of the truncation error; they found that in all cases the optimum values for c_2 and c_3 were approximately 0.35 and 0.45 respectively. These results as well as a more in-depth analysis and bibliography can be found in Lapidus [17].

These notable formulas are presented here.

Name	c_2	c_3
Classic	$\frac{1}{2}$	$\frac{1}{2}$
Kutta	$\frac{1}{3}$	$\frac{2}{3}$
Ralston	$\frac{2}{5}$	$\frac{7}{8} - \frac{3\sqrt{5}}{16}$
Kuntzmann	$\frac{2}{5}$	$\frac{3}{5}$
Hull and Johnston	$\frac{7}{20}$	$\frac{9}{20}$

1.4.2. *(4,5) Runge-Kutta formula pairs.* Extensive research and development of (4,5) formula pairs has been conducted, such as the family of formulas developed by Dormand and Prince [6] and used in the `ode45` solver in MATLAB. Sharp and Verner showed that (4,5) formula pairs provide an efficient method to solving nonstiff initial value problems [27]. Papakostas and Papageorgiou [19] later constructed a new family

of formulas using simplifying conditions similar to those used in Section 3; they claim this family has a higher efficiency than the Dormand and Prince formulas do.

1.5. **Gröbner Bases and Normal Forms.** We use a Gröbner basis of the ideal generated by the Butcher order conditions to efficiently analyze the solutions to these conditions and the higher-order truncation error.

Recall from Equation (5) that the polynomial order conditions [4] are of the form

$$P^{(\tau)}(A,b,c) = b^{\mathrm{T}}A^{(\tau)} - \frac{1}{\tau!} = 0, \quad \#\tau \leq p,$$

where P is a polynomial and (A,b,c) represents the set of coefficients a_{ij}, b_i, and c_i.

Then (A,b,c) distinguishes a Runge-Kutta formula of order p if and only if

$$(A,b,c) \in V = \{(A,b,c) | P^{(\tau)}(A,b,c) = 0 \quad \forall\ \tau, \#\tau \leq p\}.$$

By letting J be the ideal generated by $\{P^{(\tau)} | \#\tau \leq p\}$, the Runge-Kutta formula derived by (A,b,c) has order p if and only if

$$(A,b,c) \in V(J),$$

where $V(J)$ is the *variety* of J, the set of common zeros of all polynomials in J. We then use a Gröbner basis of this ideal J to find solutions to the polynomial equations that generate J and simplify the truncation error terms.

In short, a *Gröbner* basis is a generating basis for an ideal where the leading terms of the elements in a Gröbner basis also generate the leading terms of all the elements in the ideal. For more information on Gröbner bases, consult the basic theory in Rotman [22] and Adams and Loustaunau [1].

When using an elimination order such as pure lexicographical order, reducing by a Gröbner basis eliminates variables and thus allows solutions to be easily written in parametric form. This type of reduction can be thought of as the nonlinear analogue of the reduced row echelon form of linear systems of equations. Boege, Gebauer,and Kredel were the first to suggest this application of Gröbner bases [2]. A second advantage with using a Gröbner basis of J is that the truncation error coefficients

$$\alpha_\tau\Big(b^{\mathrm{T}}A^{(\tau)} - \frac{1}{\tau!}\Big), \quad \text{for } \tau \text{ with } \#\tau > p,$$

can be reduced to a normal form. Thus, the coefficients are simplified, making it easier to distinguish patterns (see Sections 2.2.1 and 2.2.2).

2. Four-stage Fourth-Order Runge-Kutta Formula with Step Doubling

2.1. **Creation of an Eleven-stage Fifth-order Formula Using a Double Step.** As mentioned in Section 1.3.2, an ODE solver based on step doubling takes two steps and then compares the result with the solution computed after a single step of double length, allowing it to gain an extra order of accuracy. A proof of this can be found in Shampine [24]. By then picturing this process itself as a Runge-Kutta formula, we can get an idea of the error associated with each iteration of the step-doubling process. Furthermore, if we used an s-stage pth-order Runge-Kutta formula in the original solver, then the Runge-Kutta formula representing the double step is in fact of order $p+1$ with $3s-1$ stages.

In this case we are treating a four-stage fourth-order formula with step doubling as an eleven-stage fifth-order formula. This perspective is very helpful since it allows us to analyze the sixth-order truncation error associated with each double step.

2.2. **Using Gröbner Bases to Find Normal Forms of Order Conditions.** A four-stage Runge-Kutta formula has order four if and only if the following eight Butcher order conditions vanish:

$$
\begin{aligned}
&\text{Order 1:} && b_1+b_2+b_3+b_4-1\\
&\text{Order 2:} && b_2c_2+b_3c_3+b_4c_4-\frac{1}{2}\\
&\text{Order 3:} && b_2c_2^2+b_3c_3^2+b_4c_4^2-\frac{1}{3}\\
& && c_2(b_3a_{32}+b_4a_{42})+c_3b_4a_{43}-\frac{1}{6}\\
&\text{Order 4:} && b_2c_2^3+b_3c_3^3+b_4c_4^3-\frac{1}{4}\\
& && c_2(c_3b_3a_{32}+b_4c_4a_{42})+c_3c_4b_4a_{43}-\frac{1}{8}\\
& && c_2^2(b_3a_{32}+b_4a_{42})+c_3^2b_4a_{43}-\frac{1}{12}\\
& && c_2b_4a_{32}a_{43}-\frac{1}{24}
\end{aligned}
\tag{9}
$$

We used Maple to compute a Gröbner basis of these eight terms with a pure lexicographical term order.

Pure lexicographic order was chosen so that variables could be eliminated more easily before choosing values to optimize the Runge-Kutta

formula. We also chose a term ordering that omitted the c_2 and c_3 values in order for the basis to be calculated in a reasonable length of time. While this conveniently allowed us to solve for all other parameters in terms of c_2 and c_3, it did create problems due to artificial poles (see Section 2.3.1). Thus, the resulting Gröbner basis was a set of eight polynomials, where each can be solved for an individual parameter in terms of c_2 and c_3.

Not only did Gröbner bases allow us to avoid lengthy calculations to solve for parameters, but they also gave us an efficient method for simplifying truncation error terms. Our main concern was the fifth- and sixth-order error terms for the four-stage fourth-order Runge-Kutta formula, as well as the sixth-order error terms for the eleven-stage fifth-order formula. Reducing these three sets of error terms by our Gröbner basis led them to all be constants or, with a few exceptions, linear or quadratic functions of c_2 and c_3. This simpler form also revealed some interesting patterns.

2.2.1. *Pairing of rooted trees.* After reducing the truncation error coefficients into their normal forms, it became evident that a majority of the error coefficients of the fifth order for the four-stage fourth-order formula (see Appendix A) and of the sixth-order for the eleven-stage fifth-order formula could be sorted into pairs, where the equations were additive inverses of each other. Furthermore, the corresponding rooted trees of these pairs were related in that the child node connected by a single branch to the root of one tree served as the root of the other. Lastly, for a Runge-Kutta formula of order p and truncation error coefficients of order $p+1$, the rooted trees associated with these paired error coefficients had corresponding $\tau!$ values that were different by a factor of p. By definition, $\tau!$ is equal to the order of the tree multiplied by the order of all subtrees that are produced by systematically eliminating roots (see Butcher [4] for a more detailed explanation).

These observations can be explained by the condition that

$$b^{\mathrm{T}}(A + C - I) = 0. \tag{10}$$

Indeed, if the parameters of a Runge-Kutta formula of order p met this condition, then

$$b^{\mathrm{T}}A = b^{\mathrm{T}}(I - C).$$

Any tree that began with only one branch from its root would have the corresponding coefficient $b^{\mathrm{T}}A(...)e$, where $(...)$ refers to some combination of A's and C's that would make $b^{\mathrm{T}}A(...)e$ of order $p+1$. Then for

such trees,

$$b^{\mathrm{T}}A(...)e - \frac{1}{\tau!} = b^{\mathrm{T}}(...)e - b^{\mathrm{T}}C(...)e - \frac{1}{\tau!}.$$

As $b^{\mathrm{T}}(...)$ is of order p, its Butcher order condition must be satisfied since our Runge-Kutta formula has order p. Then by Equation (5), $b^{\mathrm{T}}(...)e = 1/\tau^*!$, where τ^* corresponds to the tree represented by $b^{\mathrm{T}}(...)e$. Therefore, as $b^{\mathrm{T}}A(...)e$ is $b^{\mathrm{T}}(...)e$ with a root and single stem added to its bottom, the recursive definition of $\tau!$ [3] implies that $\tau! = (p+1)\tau^*!$. Hence,

$$\begin{aligned} b^{\mathrm{T}}A(...)e - \frac{1}{\tau!} &= \frac{1}{\tau^*!} - \frac{1}{\tau!} - b^{\mathrm{T}}C(...)e \\ &= \frac{p+1}{(p+1)\tau^*!} - \frac{1}{(p+1)\tau^*!} - b^{\mathrm{T}}C(...)e \\ &= -\Big(b^{\mathrm{T}}C(...)e - \frac{p}{\tau!}\Big), \end{aligned}$$

which corresponds to all three of the relationships observed.

When the four-stage fourth-order and eleven-stage fifth-order Runge-Kutta parameters were tested, they both satisfied Equation (10).

Equation (10) was not met by the conditions for a third-order Runge-Kutta formula and hence this pairing was not observed. However, setting an additional constraint of $c_3 = 1$ on the order conditions would allow the third-order formula to satisfy Equation (10). It is hypothesized that for any explicit Runge-Kutta formula of s stages, setting $c_s = 1$ would imply Equation (10), but this has yet to be verified.

2.2.2. *Structure of T_{p+2}.* During the calculation of the fifth- and sixth-order terms for the fourth-order formula and the sixth-order terms for the double-step formula, a correlation among these error coefficients was noticed. Further investigation revealed the following theorem. This theorem shows how the truncation error coefficients of the extrapolated formula are related to those of the basic formula.

Theorem 1. *Consider an s-stage Runge-Kutta formula of order p. Let A and b represent the parameters for this formula. Similarly, let $\bar{A}$ and $\bar{b}$ represent the parameters for the $(3s-1)$-stage Runge-Kutta formula of order $(p+1)$ created via a double step. A single step is considered to be of size $h/2$, while a double step is of size h. Then for every τ with*

$\#\tau = p+2$,

$$\alpha_\tau\Big(b^T A^{(\tau)} - \frac{1}{\tau!}\Big) = -\frac{1}{2(2^p-1)}\alpha_\tau\Big(\bar{b}^T \bar{A}^{(\tau)} - \frac{1}{\tau!}\Big) + \frac{1}{4(2^p-1)} \sum_{\#\beta=p+1} g(\beta,\tau)\alpha_\beta\Big(\bar{b}^T \bar{A}^{(\beta)} - \frac{1}{\beta!}\Big), \tag{11}$$

where $g(\beta,\tau)$ is the number of times τ is produced by adding a leaf to a terminal vertex of β or a new root and a single stem to the bottom of β.

Proof. Suppose an s-stage Runge-Kutta method of order p is being applied to the autonomous differential equation $y' = f(y)$. We consider full steps to be of size h where $x_{n+i} = x_n + ih$. We then define the result of a single half step of size $h/2$ from the point (x_n, y_n) to be $(x_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})$, while two half steps result in (x_{n+1}, y_{n+1}). A full step of size h from (x_n, y_n) yields $(x_{n+1}, \hat{y}_{n+1})$.

The extrapolated solution $\bar{y}_{n+1}$ at x_{n+1} is the result of two half steps adjusted by the result of the single full-length step and is defined as

$$\bar{y}_{n+1} = y_{n+1} + \frac{2}{2^{p+1}-2}(y_{n+1} - \hat{y}_{n+1}). \tag{12}$$

This combination is chosen to eliminate truncation error terms of order $p+1$.

The order-$(p+2)$ error is the difference between the extrapolated solution and the local solution at x_{n+1} and is represented by the equation

$$u_n(x_{n+1}) - \bar{y}_{n+1} = h^{p+2} \sum_{\#\tau=p+2} \alpha_\tau\Big(\bar{b}^{\mathrm{T}} \bar{A}^{(\tau)} - \frac{1}{\tau!}\Big) D^{(\tau)} f + \mathrm{O}(h^{p+3}). \tag{13}$$

Recall that u_n is the local solution as defined in Section 1.3, where $u_n'(x) = f(u_n(x))$ with the initial condition that $u_n(x_n) = y_n$. We can also find $u_n(x_{n+1}) - \bar{y}_{n+1}$ by using Equation (12) to yield

$$u_n(x_{n+1}) - \bar{y}_{n+1} = u_n(x_{n+1}) - y_{n+1} - \frac{2}{2^{p+1}-2}(y_{n+1} - \hat{y}_{n+1}). \tag{14}$$

Thus, to find the local sixth-order error, we need to evaluate both $(u_n(x_{n+1}) - y_{n+1})$ and $(y_{n+1} - \hat{y}_{n+1})$. We rewrite the first expression as

$$u_n(x_{n+1}) - y_{n+1} = \big(u_n(x_{n+1}) - u_{n+\frac{1}{2}}(x_{n+1})\big) + \big(u_{n+\frac{1}{2}}(x_{n+1}) - y_{n+1}\big). \tag{15}$$

We define $z(x) = u_n(x) - u_{n+\frac{1}{2}}(x)$ to eventually get an expression for $u_n(x_{n+1}) - u_{n+\frac{1}{2}}(x_{n+1})$. Then

$$u_n(x) = u_{n+\frac{1}{2}}(x) + z(x).$$

Differentiating and using the differential equation gives on the one hand

$$u_n'(x) = u_{n+\frac{1}{2}}'(x) + z'(x)$$

and on the other

$$\begin{aligned} u_n'(x) &= f(u_n(x)) \\ &= f(u_{n+\frac{1}{2}}(x) + z(x)) \\ &= f(u_{n+\frac{1}{2}}(x)) + f'(u_{n+\frac{1}{2}}(x))z(x) + \mathrm{O}(|z|^2). \end{aligned}$$

Hence $u_{n+\frac{1}{2}}'(x) = f(u_{n+\frac{1}{2}}(x))$ implies that

$$z'(x) = f'(u_{n+\frac{1}{2}}(x))z(x) + \mathrm{O}(|z|^2). \tag{16}$$

As z satisfies the initial condition

$$\begin{aligned} z(x_{n+\frac{1}{2}}) &= u_n(x_{n+\frac{1}{2}}) - u_{n+\frac{1}{2}}(x_{n+\frac{1}{2}}) \\ &= u_n(x_{n+\frac{1}{2}}) - y_{n+\frac{1}{2}} \\ &= \left(\frac{h}{2}\right)^{p+1} \varphi_{p+1}(y_n) + \left(\frac{h}{2}\right)^{p+2} \varphi_{p+2}(y_n) + \mathrm{O}(h^{p+3}) \\ &= O(h^{p+1}), \end{aligned} \tag{17}$$

we get that $\mathrm{O}(|z|^2)$ is at least $\mathrm{O}(h^{p+3})$ (since $p \geq 1$). Evaluating z at x_{n+1} and taking the Taylor expansion then gives

$$\begin{aligned} u_n(x_{n+1}) - u_{n+\frac{1}{2}}(x_{n+1}) &= z(x_{n+1}) \\ &= z(x_{n+\frac{1}{2}}) + \frac{h}{2} z'(x_{n+\frac{1}{2}}) + \mathrm{O}(h^{p+3}). \end{aligned}$$

We can then substitute for z' with Equation (16) to get

$$= z(x_{n+\frac{1}{2}}) + \frac{h}{2} f'(u_{n+\frac{1}{2}}(x))z(x_{n+\frac{1}{2}}) + \mathrm{O}(h^{p+3}).$$

Substituting for $z(x_{n+\frac{1}{2}})$ with Equation (17) then gives

$$\begin{aligned} &= \left(\frac{h}{2}\right)^{p+1} \varphi_{p+1}(y_n) + \left(\frac{h}{2}\right)^{p+2} \varphi_{p+2}(y_n) \\ &\quad + \left(\frac{h}{2}\right)^{p+2} f'(u_{n+\frac{1}{2}}(x))\varphi_{p+1}(y_n) + \mathrm{O}(h^{p+3}) \\ &= \left(\frac{h}{2}\right)^{p+1} \varphi_{p+1}(y_n) \\ &\quad + \left(\frac{h}{2}\right)^{p+2} (\varphi_{p+2}(y_n) + f'(y_{n+\frac{1}{2}})\varphi_{p+1}(y_n)) + \mathrm{O}(h^{p+3}). \end{aligned}$$

Writing $f'(y_{n+\frac{1}{2}}) = f'(y_n) + \mathrm{O}(h)$, we get

$$\begin{aligned} &= \Big(\frac{h}{2}\Big)^{p+1} \varphi_{p+1}(y_n) \\ &\quad + \Big(\frac{h}{2}\Big)^{p+2} (\varphi_{p+2}(y_n) + f'(y_n)\varphi_{p+1}(y_n)) + \mathrm{O}(h^{p+3}). \end{aligned}$$

The second term in Equation (15) is just the local error in the step from $x_{n+\frac{1}{2}}$ to x_{n+1}. It is equal to

$$u_{n+\frac{1}{2}}(x_{n+1}) - y_{n+1} = \Big(\frac{h}{2}\Big)^{p+1} \varphi_{p+1}(y_{n+\frac{1}{2}}) + \Big(\frac{h}{2}\Big)^{p+2} \varphi_{p+2}(y_{n+\frac{1}{2}}) + \mathrm{O}(h^{p+3}).$$

Substituting in the Taylor expansion of $y_{n+\frac{1}{2}}$ then gives

$$= \Big(\frac{h}{2}\Big)^{p+1} \varphi_{p+1}\Big(y_n + \frac{h}{2} f(y_n)\Big) + \Big(\frac{h}{2}\Big)^{p+2} \varphi_{p+2}(y_n) + \mathrm{O}(h^{p+3}).$$

Again we substitute in the Taylor expansion, this time with φ_{p+1}, to get

$$= \Big(\frac{h}{2}\Big)^{p+1} \varphi_{p+1}(y_n) + \Big(\frac{h}{2}\Big)^{p+2} (\varphi_{p+2}(y_n) + \varphi'_{p+1}(y_n) f(y_n) + \mathrm{O}(h^{p+3}).$$

We can now evaluate Equation (15) to get

(18)

$$\begin{aligned} u_n(x_{n+1}) - y_{n+1} &= \Big(\Big(\frac{h}{2}\Big)^{p+1} \varphi_{p+1} + \Big(\frac{h}{2}\Big)^{p+2} (\varphi_{p+2} + \varphi'_{p+1} f)\Big) \\ &\quad + \Big(\Big(\frac{h}{2}\Big)^{p+1} \varphi_{p+1} + \Big(\frac{h}{2}\Big)^{p+2} (\varphi_{p+2} + f'\varphi_{p+1})\Big) + \mathrm{O}(h^{p+3}) \\ &= 2\Big(\frac{h}{2}\Big)^{p+1} \varphi_{p+1} + \Big(\frac{h}{2}\Big)^{p+2} (2\varphi_{p+2} + \varphi'_{p+1} f + f'\varphi_{p+1}) + \mathrm{O}(h^{p+3}). \end{aligned}$$

To evaluate Equation (14) we now just need to find $y_{n+1} - \hat{y}_{n+1}$. We can rewrite this as

$$y_{n+1} - \hat{y}_{n+1} = y_{n+1} - u_n(x_{n+1}) + u_n(x_{n+1}) - \hat{y}_{n+1}, \tag{19}$$

which takes advantage of the fact that we already have Equation (18).

The expression $u_n(x_{n+1}) - \hat{y}_{n+1}$ is simply the local error after a full step of size h. Thus, we can write

$$\begin{aligned} u_n(x_{n+1}) - \hat{y}_{n+1} &= (h)^{p+1}\varphi_{p+1}(y_n) + (h)^{p+2}\varphi_{p+2}(y_n) + \mathrm{O}(h^{p+3}) \\ &= 2^{p+1}\Big(\frac{h}{2}\Big)^{p+1}\varphi_{p+1}(y_n) + 2^{p+2}\Big(\frac{h}{2}\Big)^{p+2}\varphi_{p+2}(y_n) + \mathrm{O}(h^{p+3}). \end{aligned}$$

Substituting this expression and Equation (18) back into Equation (19) then gives

(20)

$$y_{n+1} - \hat{y}_{n+1} = (2^{p+1}-2)\Big(\frac{h}{2}\Big)^{p+1}\varphi_{p+1} + \Big(\frac{h}{2}\Big)^{p+2}((2^{p+2}-2)\varphi_{p+2} - \varphi'_{p+1}f - f'\varphi_{p+1}) + \mathrm{O}(h^{p+3}).$$

Inserting Equations (15) and (18) into (14) now yields

(21)

$$\begin{aligned} u_n(x_{n+1}) - \bar{y}_{n+1} &= u_n(x_{n+1}) - y_{n+1} - \frac{2}{2^{p+1}-2}(y_{n+2} - \hat{y}_{n+1}) \\ &= 2\Big(\frac{h}{2}\Big)^{p+1}\varphi_{p+1} + \Big(\frac{h}{2}\Big)^{p+2}\varphi_{p+2}(2\varphi_{p+2} + \varphi'_{p+1}f + f'\varphi_{p+1}) \\ &\quad - 2\Big(\frac{h}{2}\Big)^{p+1}\varphi_{p+1} - \frac{2}{2^{p+1}-2}\Big(\frac{h}{2}\Big)^{p+2}((2^{p+2}-2)\varphi_{p+2} - \varphi'_{p+1}f - f'\varphi_{p+1}) \\ &= h^{p+2}\Big(\frac{-1}{2(2^{p+1}-2)}\varphi_{p+2} + \left(\frac{1}{4(2^{p+1}-2)}\right)(\varphi'_{p+1}f + f'\varphi_{p+1})\Big). \end{aligned}$$

Consider now any rooted tree τ of order $p+2$. We want to show that the coefficient of $D^{(\tau)}f$ in $u_n(x_{n+1}) - \bar{y}_{n+1}$ is as stated in Theorem 1.

By definition, the coefficient of $D^{(\tau)}f$ in φ_{p+2} is

$$\alpha_\tau\left(b^{\mathrm{T}}A^{(\tau)} - \frac{1}{\tau!}\right),$$

so the first term in (21) agrees with (11).

Thus, it only remains to show that the coefficient of $D^{(\tau)}f$ in $(\varphi'_{p+1}f + f'\varphi_{p+1})$ is

$$\sum_{\#\beta=p+1} g(\beta,\tau)\alpha_\beta\left(b^{\mathrm{T}}A^{(\beta)} - \frac{1}{\beta!}\right).$$

We first consider $\varphi'_{p+1}f$. By definition, we have

$$\varphi'_{p+1}f = \sum_{\#\beta=p+1}\alpha_\beta\left(b^{\mathrm{T}}A^{(\beta)} - \frac{1}{\beta!}\right)\frac{\partial}{\partial y}D^{(\beta)}f.$$

We can write out

$$D^{(\beta)}f = f^{(\bar{p})}(D^{(\beta_1)}f, D^{(\beta_2)}f, ..., D^{(\beta_n)}f), \tag{22}$$

where $\bar{p} \leq p$, $n = \bar{p}$, and $\sum_{i=1}^{n} \#\beta_i = p$. Then the partial derivative of $D^{(\beta)}f$ with respect to y is

$$\frac{\partial}{\partial y} D^{(\beta)} f = f^{(\bar{p}+1)}(D^{(\beta_1)}f, D^{(\beta_2)}f, ..., D^{(\beta_n)}f) + \sum_{i=1}^{n} f^{(\bar{p})}(..., \frac{\partial}{\partial y} D^{(\beta_i)}f, ...).$$

Notice that $f^{(\bar{p}+1)}(D^{(\beta_1)}f, D^{(\beta_2)}f, ..., D^{(\beta_n)}f)$ corresponds to the tree of order $p+2$ where a leaf is added to the root of β. The summation then adds up trees where a leaf is added to the root of each β_i, and it proceeds recursively. Thus, the partial derivative of $D^{(\beta)}f$ results in a summation of trees of order $p+2$ where each is obtained by adding a terminal leaf to a vertex of β.

$f'\varphi_{p+1}$ is by definition

$$\sum_{\#\beta=p+1} \alpha_\beta \left(b^{\mathrm{T}} A^{(\beta)} - \frac{1}{\beta!} \right) f'(D^{(\beta)}f).$$

Using (22) we can then calculate $f'(D^{(\beta)}f)$ to be

$$f'(D^{(\beta)}f) = f^{(\bar{p}+1)}(D^{(\beta_1)}f, D^{(\beta_2)}f, ..., D^{(\beta_n)}f).$$

Thus, $f'(D^{(\beta)}f)$ corresponds to the tree of order $p+2$ that is obtained by attaching the root of β to the terminal vertex of the second-order tree. A more visual description would be "putting β on a stem."

Hence, we define $g(\beta, \tau)$ to return the number of times a tree τ of order $p+2$ is produced by adding a terminal leaf to a vertex of β or by "putting β on a stem," completing the proof. □

2.3. **Optimizing Formulas.** As mentioned in Section 1.3, an ODE solver uses an estimate of the local error to adjust its step size. The algorithm that adjusts the step is based on the assumption that terms of order $p+1$ dominate in the local error.

Therefore, a program based on a fourth-order Runge-Kutta formula requires fifth-order error coefficients that are substantial enough to drown out error of the sixth order or higher. Minimizing the sixth-order terms of the p-order formula would also be beneficial to improve accuracy of both the formula and the local error estimates.

To optimize a fourth-order Runge-Kutta formula with fifth-order error terms $\hat{T}_5$ and sixth-order error terms $\hat{T}_6$, and to optimize a double-step formula with sixth-order error terms T_6, we want the formula to

obey the following conditions [23]:

$$\begin{aligned} &\|T_6\| \text{ minimized subject to} \\ &\hat{T}_5 \geq \text{lower bound} \\ \text{and } &\frac{\|T_6 - \hat{T}_6\|}{\|\hat{T}_5\|} \leq \text{upper bound} \end{aligned} \tag{23}$$

By using a Gröbner basis for the eight order conditions (9), we obtained normal forms for the fifth- and sixth-order error coefficients that simplified the calculations in the search for optimum formulas.

2.3.1. *Problems with lack of adequate parametric equations.* However, one difficulty in analyzing the fifth- and sixth-order error coefficients was due to artificial poles created by the parametric form of the equations. Although these special cases could be analyzed separately, it still prevented us from gathering accurate information at points on the contour maps close to these artificial poles.

The three cases where solutions to the error terms existed, but had to be analyzed separately, were $c_2 = 1/2$, $c_2 = c_3$, and $c_2 = 1/2(4c_3 - 3)/(3c_3 - 2)$. By adjusting the initial order conditions to reflect each of these cases, a special Gröbner basis could be calculated for that single scenario. Solving this Gröbner basis caused all of the parameters and error terms to be functions of just one free parameter.

Furthermore, the values for c_2 and c_3 were constricted to just a few cases. $c_2 = c_3$ implied that both terms were $1/2$. $c_2 = 1/2$ implied that $c_3 = 0$ or $1/2$. And the final condition only allowed for $c_2 = 1$ and $c_3 = 1/2$.

These three scenarios were each studied individually; however, the problem of distortion close to these points on the contour maps was unable to be resolved. Attempts at calculating Gröbner bases using different term orderings or with different variables in the term ordering proved to be unsuccessful.

2.4. **Testing of Runge-Kutta Formulas Via MATLAB.** We developed a program in MATLAB to test four-stage fourth-order Runge-Kutta formulas using step doubling (see [23]). The initial step size was determined using simple estimates of $\|\frac{\partial f}{\partial x}(x_0, y_0)\|$ and $\|\frac{\partial f}{\partial y}(x_0, y_0)\|$ based on ideas by Watts [29]. The algorithm used step doubling to estimate the local truncation error and achieve an extra order of accuracy. A proof of this can be found in Shampine [24]. Note that the program adjusts step sizes according to error per step, as opposed to error per

unit step. Hermite quintic interpolation [5] was used to evaluate the function at the final x-value.

A selection of three periodic orbit problems was used to test the error of each Runge-Kutta formula. The simplest orbit was a Keplerian orbit of eccentricity 0.9, used in the DETEST battery of ODE problems [12]. The second orbit was a plane-restricted, circular three-body problem based on the `orbitode` demonstration program in MATLAB. The most difficult orbit was the three-body Arenstorf orbit [9].

The error of a Runge-Kutta formula was calculated by taking the norm of the difference between the initial and final position and velocity vectors after one full period. Error was then plotted against the number of derivative evaluations for each Runge-Kutta formula, with absolute and relative error tolerances ranging from 10^{-4} to 10^{-10} to generate a graph of each formula's relative efficiency. A selection of graphs can be found in Appendix B.

2.4.1. *Results.* As expected, none of the formulas tested could consistently compete with `ode45`, although some formulas were able to achieve better efficiency on specific orbits and certain tolerances.

One of the most successful Runge-Kutta formulas was associated with $c_2 = 2/3$ and $c_3 = 1/2$. It was selected as a possible candidate due to the fact that it eliminated eight of the sixth-order error terms associated with the double-step formula. Interestingly, the b_2 parameter of this formula is 0, making this fourth-order formula similar to Simpson's rule, but it has an improved derivative evaluation at $h/2$. This formula was significantly more efficient than most of the previously-optimized Runge-Kutta formulas covered in Section 2. The only exception was the three-body Arenstorf orbit on tight error tolerances, where the classic formula and Ralston's optimized formula exhibited higher efficiency. This observation was surprising since the classic formula was not expected to perform well, especially on difficult problems.

The reason for the classic formula's success was revealed once a Gröbner basis for the case where $c_2 = c_3 = 1/2$ was calculated, showing that the classic formula eliminated ten of the sixth-order error terms. Further testing showed that a slight variation of the classic formula (changing a_{32} from 1/2 to 1/4) led to an even better performance on the three-body Arenstorf orbit, to the point where at tight tolerances the formula outperformed `ode45`. However, all variations of the classic formula performed relatively poorly on the Keplerian orbit and the orbit from `orbitode`. This evidence may lead to the conclusion that formulas with $c_2 = c_3 = 1/2$ are most effective on difficult problems

with very strict error tolerances, where sixth-order error may become more significant. In fact, a formula that minimized the sixth-order error ($c_2 = 0.5130$ and $c_3 = 0.4974$) was also most effective on the Arenstorf orbit, providing further proof that minimizing the sixth-order error is important for demanding ODEs, but it does not necessarily improve performance on easier problems.

Meanwhile, it appears as though previous Runge-Kutta formulas (besides the classic and Ralston) were primarily focused on reducing the fifth-order truncation error. The Kutta and Kuntzmann optimized formulas performed very similarly to a formula that minimized the fifth-order truncation error ($c_2 = 0.3578$, $c_3 = 0.5915$). Prior to the discovery of the $c_2 = 2/3$ and $c_3 = 1/2$ formula, these fifth-order minimizers were the most efficient on the Kepler orbit and even the orbit of `orbitode`. The higher-order error on these orbits was likely to be less significant. The fifth-order minimizers could often take larger steps without a significant loss in accuracy, which agrees with the groupings of formulas with comparable derivative evaluations for a given accuracy.

Why the formula with $c_2 = 2/3$ and $c_3 = 1/2$ performs so well on even the simple Kepler orbit is curious, as its fifth-order error coefficients are not minimized. A possible explanation is that it achieves a very good balance between maintaining a robust fifth-order error and minimizing the sixth-order error. Another possibility is that it eliminates the exact sixth-order elementary differentials that are especially prevalent in orbit problems.

Another odd result was that the formula with $c_2 = 2/3$ and $c_3 = 0.51$ showed even better results than with $c_3 = 1/2$, competing with `ode45` at high tolerances on the simpler orbits. No explanation has yet been found as to why this slight change to c_3 would improve results.

Testing was also done on the special cases where vales of c_2 and c_3 required the order conditions to be adjusted to find the error coefficients. From their initial appearance, these values of c_2 and c_3 appeared to not be conducive to producing effective formulas, as they either had $c_3 = 0$ or $c_2 = 1$. However, they produced reasonable results, proving that these special cases should not be simply disregarded when investigating Runge-Kutta formulas. The most successful formula of these tested was $c_2 = 1/2$ and $c_3 = 0$ with free parameter a_{43} set to $5/4$, which had the best relative efficiency on the simpler orbits at tight tolerances.

3. Runge-Kutta (4,5) Formula Pairs

Since Fehlberg embedding methods have become popular among software including MATLAB's `ode45`, we then chose to examine Runge-Kutta (4,5) formulas in more detail. Optimizing a formula of this type involved some significantly different strategies than with a fourth-order formula, though the overall process was similar.

3.1. **Solving Order Conditions.**

The first step toward finding an optimal (4,5) pair was the solving of a particular set of order conditions using a Gröbner basis. We made some simplifying assumptions about these order conditions to reduce the complexity of the optimization problem.

3.1.1. *Theorem of alternate conditions up to 5th order.* The paper by Papakostas and Papageorgiou [19] shows 20 conditions that are equivalent to the $17+8$ order conditions that $\begin{array}{c|c} c & A \\ \hline & b^{\mathrm{T}} \end{array}$ has order 5 and $\begin{array}{c|c} c & A \\ \hline & \hat{b}^{\mathrm{T}} \end{array}$ has order 4, if the conditions $Ae = c$ and $e_i^{\mathrm{T}}\left(AC - \frac{c^2}{2}\right)e = 0, \quad i = 3\ldots 6$ are assumed true. We developed a theorem using an added condition for stage order 3,

$$e_i^{\mathrm{T}}\left(AC^2 - \frac{C^3}{3}\right)e = 0, \qquad i = 3\ldots 6.$$

This condition constrained the solution set a little more, but it reduced the number of equivalent order conditions. When used with $Ae = c$ and $e_i^{\mathrm{T}}\left(AC - \frac{C^2}{2}\right)e, \quad i = 3\ldots 6$, it determines 16 conditions equivalent to the $17 + 8$ order conditions as previously stated.

Theorem 2. *Assume an s-stage Butcher tableau satisfies*

$$s = 7, \tag{24}$$

$$c_i \neq c_j \text{ when } i \neq j, \tag{25}$$

$$c_7 = 1, \text{ and} \tag{26}$$

$$b_i = a_{7i} \quad (i = 1\ldots 7). \tag{27}$$

Then if the conditions

$$Ae = c \tag{28}$$

$$e_i^T \left(AC - \frac{C^2}{2} \right) e = 0, \qquad i = 3 \dots 6 \tag{29}$$

$$e_i^T \left(AC^2 - \frac{C^3}{3} \right) e = 0, \qquad i = 3 \dots 6 \tag{30}$$

are satisfied, the following 16 order conditions are necessary and sufficient for the $17 + 8$ *order conditions that* $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$ *has order 5 and* $\begin{array}{c|c} c & A \\ \hline & \hat{b}^T \end{array}$ *has order 4:*

Orthogonality conditions:

$$b^T e_2 = b^T A e_2 = b^T C A e_2 = b^T A^2 e_2 = 0 \tag{31}$$

$$\hat{b}^T e_2 = \hat{b}^T A e_2 = 0 \tag{32}$$

Quadrature conditions:

$$b^T C^{k-1} e - \frac{1}{k} = 0, \qquad k = 1 \dots 5 \tag{33}$$

$$\hat{b}^T C^{k-1} e - \frac{1}{k} = 0, \qquad k = 1 \dots 4 \tag{34}$$

Tree condition:

$$b^T A C^3 e - \frac{1}{20} = 0 \tag{35}$$

Proof. For the sufficiency, assume that conditions (31) to (35) are satisfied. First we note that conditions (33), (34), and (35) are identical to 10 of the 25 conditions, leaving 15 others. Consider any of the remaining 15, such as $b^{\mathrm{T}} A^3 C e - \frac{1}{120} = 0$. By (29) and assumptions (24)-(27),

there exist nonzero scalars δ_2 and δ_3 such that

$$ACe = \frac{1}{2}C^2e + \delta_2 e_2 \tag{36}$$

$$AC^2e = \frac{1}{3}C^3e + \delta_3 e_3. \tag{37}$$

With this in mind,

$$\begin{aligned}
b^{\mathrm{T}}A^3Ce - \frac{1}{120} &= b^{\mathrm{T}}A^2(ACe) - \frac{1}{120} \\
&= b^{\mathrm{T}}A^2\left(\frac{1}{2}C^2e + \delta_2 e_2\right) - \frac{1}{120} && \text{by (36)} \\
&= \frac{1}{2}b^{\mathrm{T}}A^2C^2e + \delta_2 b^{\mathrm{T}}A^2e_2 - \frac{1}{120} \\
&= \frac{1}{2}b^{\mathrm{T}}A(AC^2e) + 0 - \frac{1}{120} && \text{by (31)} \\
&= \frac{1}{2}b^{\mathrm{T}}A\left(\frac{1}{3}C^3e + \delta_3 e_2\right) - \frac{1}{120} && \text{by (37)} \\
&= \frac{1}{6}b^{\mathrm{T}}AC^3e + \frac{1}{2}\delta_3 b^{\mathrm{T}}Ae_2 - \frac{1}{120} \\
&= \frac{1}{6}b^{\mathrm{T}}AC^3e + 0 - \frac{1}{120} && \text{by (31)} \\
&= \frac{1}{6}\left(\frac{1}{20}\right) - \frac{1}{120} && \text{by (35)} \\
&= 0
\end{aligned}$$

The 14 other conditions can be shown in a similar way to prove the sufficiency argument.

For the converse, assume that the $17+8$ "standard" order conditions are satisfied. We prove the 16 order conditions.

Once again, 10 of the conditions are identical, so it suffices to prove the orthogonality conditions (31) and (32). To show $b^{\mathrm{T}}CAe_2 = 0$, for example, notice that

$$\begin{aligned}
b^{\mathrm{T}}CA\left(AC - \frac{C^2}{2}\right)e &= b^{\mathrm{T}}CA^2Ce - \frac{1}{2}b^{\mathrm{T}}CAC^2e \\
&= \frac{1}{30} - \frac{1}{2}\left(\frac{1}{15}\right) \\
&= 0.
\end{aligned}$$

On the other hand,

$$b^{\mathrm{T}}CA\left(ACe - \frac{C^2}{2}e\right) = b^{\mathrm{T}}CA(\delta_2 e_2) \text{ by } 36$$
$$= \delta_2 b^{\mathrm{T}}CAe_2.$$

Since this means $\delta_2 b^{\mathrm{T}}CAe_2 = 0$ and $\delta_2 \neq 0, b^{\mathrm{T}}CAe_2 = 0$ follows. The necessary and sufficient directions have both been shown, so the proof is complete. □

3.1.2. *Choice of initial basis polynomials.* To reduce the number of variables in the eventual Gröbner basis, we used the first expressions in Equations (31) and (32) to make $b_2 = \hat{b}_2 = 0$. We also used Equations (29) and (30) with $i = 3$ to find $c_2 = \frac{2}{3}c_3$ and $a_{32} = \frac{3}{4}c_3$ by hand calculation.

The six order conditions involving $\hat{b}$ terms, found in Equations (31), (32), and (34), meant that a solution could be found for the $\hat{b}$ terms as a linear combination of a freely-chosen $\hat{b}$ term (we chose to make $\hat{b}_7$ free). So we planned to first find a Gröbner basis for the other 18 order conditions (reduced to 15 by the simplifications above).

Only one of these conditions, $b^{\mathrm{T}}A^2e_2 = 0$, was nonlinear, so by using an elimination order with one of c_4 or c_6 chosen with the lowest priority and the other 14 variables (all a's and b's, with no a_{i1} needed due to the fact that $a_{i1} = c_i - \sum_{j=2}^{i-1} a_{ij}$) having higher priority, a Gröbner basis was computed successfully.

3.1.3. *Two cases to consider.* When c_4 was included in the elimination order, then $c_4 - \frac{c_3}{2(5c_3^2-4c_3+1)}$ was part of the Gröbner basis. [19] showed that $c_4 = \frac{c_3}{2(5c_3^2-4c_3+1)}$ is equivalent to $\hat{b}_7 = 0$. This choice is undesirable because having the seventh $\hat{b}$ term equal 0 causes the flexibility of incorporating the first stage of a subsequent step (from the seventh stage of the current step) into the fourth-order error estimator to be lost. We expected a better error estimator if $\hat{b}_7 \neq 0$, which [19] also showed was equivalent to $b^{\mathrm{T}}(A + C - I) = 0$. As Section 2.2.1 showed with the fourth-order formula, this equation resulted in pairing among sixth-order error terms.

So we included c_6 in the elimination order; $c_6 - 1$ was part of the Gröbner basis and $\hat{b}_7$ could be chosen as a free parameter to determine the other $\hat{b}$ terms.

3.2. **Choosing Optimal Coefficients.** With a Gröbner basis generated, we then aimed to optimize a set of coefficients relative to the conditions in Equation (23). First, we normalized the sixth-order truncation error terms with respect to the basis, and then we looked to optimize the resulting expressions. It appeared promising when the 20 reduced errors could be placed in four groups, with each expression a multiple of the others in the same group. These expressions depended only on c_3, c_4, and c_5; there were 9 expressions in the first group, 6 in the second group, 3 in the third group, and 2 in the last group.

3.2.1. *Minimization of sixth-order truncation error.* It was possible to choose values for c_3, c_4, and c_5 so 14 or more of the 20 sixth-order error conditions were equal to 0. One solution even permitted all 20 errors to be 0. Unfortunately, none of these cases was feasible for our desired formula. When all 20 error conditions were set to 0, $c_4 = c_5 = c_6 = c_7 = 1$, which if implemented would have treated the formula like it had four stages instead of seven (or actually six stages since $c_6 = c_7 = 1$ already). When 18 error conditions were set to 0, the denominators of a_{52}, a_{53}, and a_{54} were forced to be 0 due to the structure of the terms in one error group. And when 14 error conditions were set to 0, one of the fifth-order error terms in $\hat{T}_5$ was forced to be 0 relative to $\begin{array}{c|c} c & A \\ \hline & \\ & \hat{b}^T \end{array}$, which would be undesirable if its corresponding elementary differential was a significant part of the magnitude of the estimated error.

Our most recent efforts focused on solving for two equations from two groups to cause 12 error terms to be 0. Doing this led to expressions for $c_3 = \frac{2(3c_5-2)}{3(5c_5-3)}$ and $c_4 = \frac{15c_5^2-19c_5+6}{15c_5^2-20c_5+7}$, so the remaining indeterminates were c_5 and $\hat{b}_7$.

3.2.2. *Choice of coefficients based on plots and tests.* When c_3 and c_4 were plotted as a function of c_5, it revealed several intervals of values that should not be assigned to c_5 due to the necessity of $0 \leq c_i \leq 1$ and assumptions (24)-(27). The interval $0.2 \leq c_5 \leq 0.55$ looked desirable due to the spread between c_3, c_4, and c_5. A plot of the sum of square of T_6 terms had a relative minimum when $c_5 \approx 0.51$ and took even smaller values when $0.7 \leq c_5 \leq 1$, so we examined the formula with values of c_5 in the neighborhood of 0.5 and 0.8.

Interestingly, it seems that varying $\hat{b}_7$ has little effect on a MATLAB graph of error versus number of functional evaluations. This is counterintuitive against the fact that when $\|\hat{b}_7\|$ increases, $\|\hat{b}_6\|$ increases

(but with opposite sign) about as quickly, but other $\hat{b}$ values are not affected as much.

3.3. **Testing.** Thus far, we have tested a limited number of formulas in the family of solutions determined by setting 12 of the T_6 errors equal to 0. The three periodic orbits used with the four-stage fourth-order formula were also tested here.

3.3.1. *RK44Auto Modified.* We made several changes to our `RK44Auto.m` function (and renamed it `RK45Auto.m`) so it would use a given fifth-order tableau $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$ of stage order 3 with fourth-order tableau $\begin{array}{c|c} c & A \\ \hline & \hat{b}^T \end{array}$.

3.3.2. *Comparison using test problems.* The Kepler, `orbitode`, and Arenstorff differential equation problems were used to assess the quality of chosen formula implementations compared to MATLAB's `ode45` function. Though this testing was limited in scope and depth due to the deadline of this project, some notable results have already been determined.

Appendix C contains three graphs that each plot error vs. effort for `ode45` and four Runge-Kutta (4,5) formulas distinguished by the value of c_5 ($\hat{b}_7 = 1$ in all cases). For the Kepler orbit, one formula that consistently beat `ode45` used $c_5 = 0.465$. This formula (and the ones in the other two graphs here) was found by comparing plots with different values of c_5 and zooming in where the error seemed to get lower relative to the others. With the Kepler orbit, for instance, we plotted error vs. effort starting with $c_5 \in \{0.4, 0.45, 0.5, 0.55\}$ and saw that $c_5 = 0.45$ had the smallest error for the same number of derivative evaluations. The next iteration involved $c_5 \in \{0.42, 0.44, 0.46, 0.48\}$, and so forth.

With the three-body orbit of MATLAB's `orbitode`, we were unfortunately unable to find a formula the surpassed `ode45` in an error range prior to the limits of the numerical software. But the formula with $c_5 = 0.8334$ gave us hope for our optimization technique when it performed better than `ode45` on the Arenstorf orbit when the error was between 10^{-5} and 10^{-8}. Further research is suggested so the behavior of the family of Runge-Kutta (4,5) formulas that we optimized for can be understood better.

4. Conclusion

We found that Gröbner bases are an effective and relatively simple way of simplifying the Butcher order conditions and reducing the higher-order error coefficients in explicit Runge-Kutta formulas.

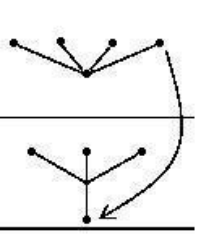
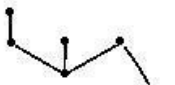
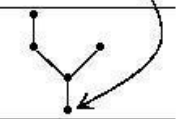
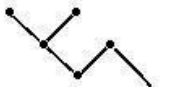
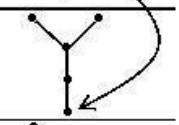
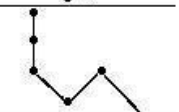
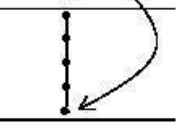
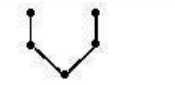
Also, by treating the double-step process for an s-stage formula of order p as a $(3s-1)$-stage formula with order $p+1$, we could then optimize a formula in terms of the order $p+2$ truncation error. This process led to the discovery of more efficient Runge-Kutta methods that generally increased in effectiveness on demanding problems.

Furthermore, we developed a new family of Runge-Kutta (4,5) formula pairs that are easy to derive. It is suggested by our most recent results that some optimal formula pairs can be competitive with `ode45`.

5. Acknowledgements

We wish to thank Professor Roger Alexander of Iowa State University for his instruction and guidance. Chris Kurth also provided assistance. We are grateful to Iowa State University where the research was performed. This project was sponsored by NSF REU grant #0353880.

Appendix A. Fifth-Order Conditions Reduced by Gröbner Basis for a Fourth-Order Runge-Kutta Formula

$b^T CCCCe - \frac{1}{5}$	$= \frac{1}{24}\left(\frac{1}{6}c_2c_3 - \frac{1}{12}c_2 - \frac{1}{12}c_3 + \frac{1}{20}\right)$	
$b^T ACCCe - \frac{1}{20}$	$= -\frac{1}{6}\left(\frac{1}{6}c_2c_3 - \frac{1}{12}c_2 - \frac{1}{12}c_3 + \frac{1}{20}\right)$	
$b^T CCACe - \frac{1}{10}$	$= -\frac{1}{2}\left(\frac{1}{24}c_3 - \frac{1}{40}\right)$	
$b^T ACACe - \frac{1}{40}$	$= 1\left(\frac{1}{24}c_3 - \frac{1}{40}\right)$	
$b^T CACCe - \frac{1}{15}$	$= -\frac{1}{2}\left(\frac{1}{24}c_2 - \frac{1}{60}\right)$	
$b^T AACCe - \frac{1}{60}$	$= \frac{1}{2}\left(\frac{1}{24}c_2 - \frac{1}{60}\right)$	
$b^T CAACe - \frac{1}{30}$	$= \frac{1}{120}$	
$b^T AAACe - \frac{1}{120}$	$= -\frac{1}{120}$	
$*b^T (ACe)^2 - \frac{1}{20}$	$= \frac{1}{2}\left(\frac{14c_2^2c_3 - 6c_2^2 - 30c_2c_3^2 + 42c_2c_3 - 15c_2 + 20c_3^2 - 27c_3 + 9}{240\left(-8c_2^2 + 12c_2^2c_3 - 14c_2c_3 + 10c_2 - 4c_3 - 3\right)}\right)$	

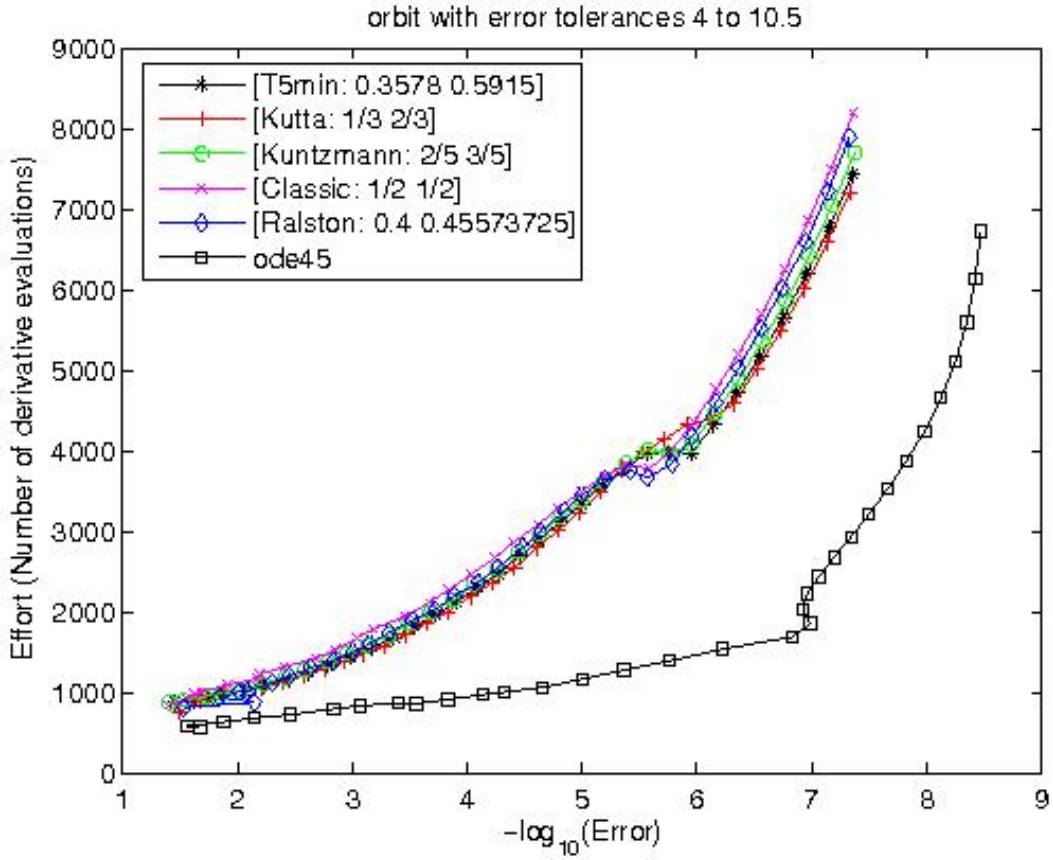
*Denotes coordinate-wise squaring

Appendix B. Error vs. Effort graphs for Fourth-Order Runge-Kutta Formulas and ode45

B.1. **Previously-discovered formulas.**

Previously-discovered formulas compared with a formula minimizing the fifth-order truncation error.

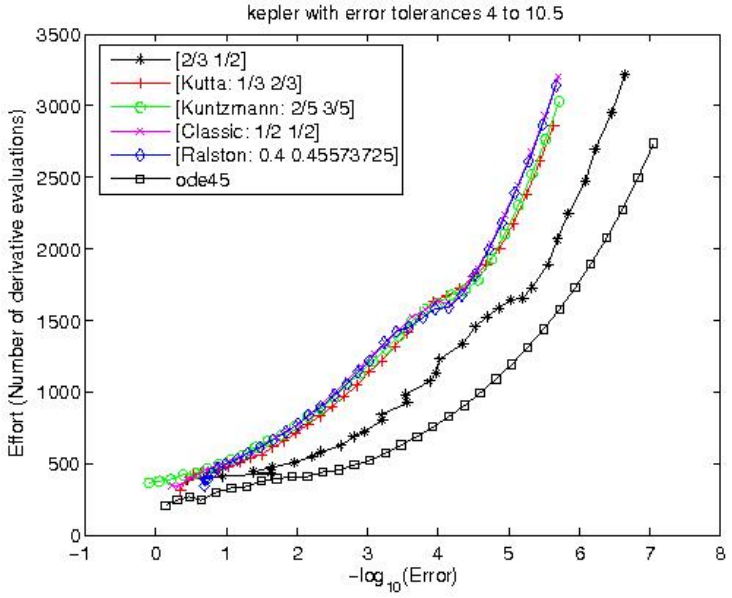
Three-body orbit of MATLAB's `orbitode`



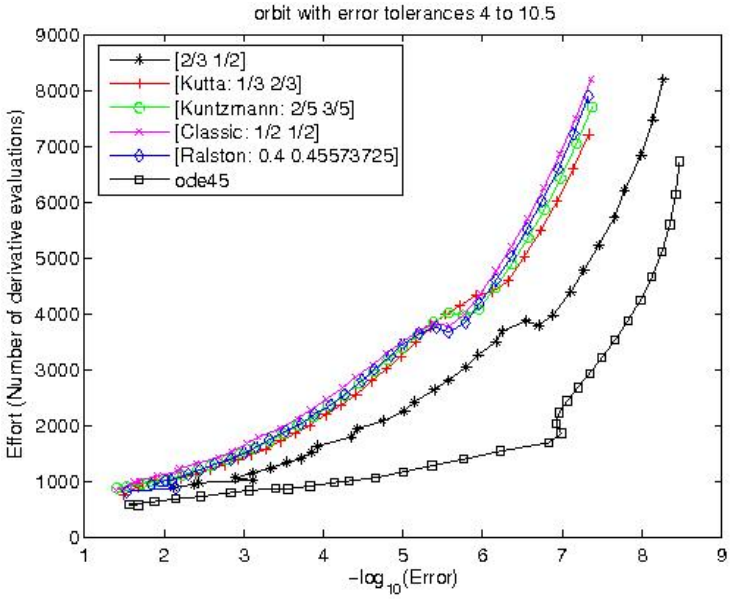
B.2. **Formula with $c_2 = 2/3$ and $c_3 = 1/2$.**

The formula with $c_2 = 2/3$ and $c_3 = 1/2$ compared to previously-optimized fourth-order formulas.

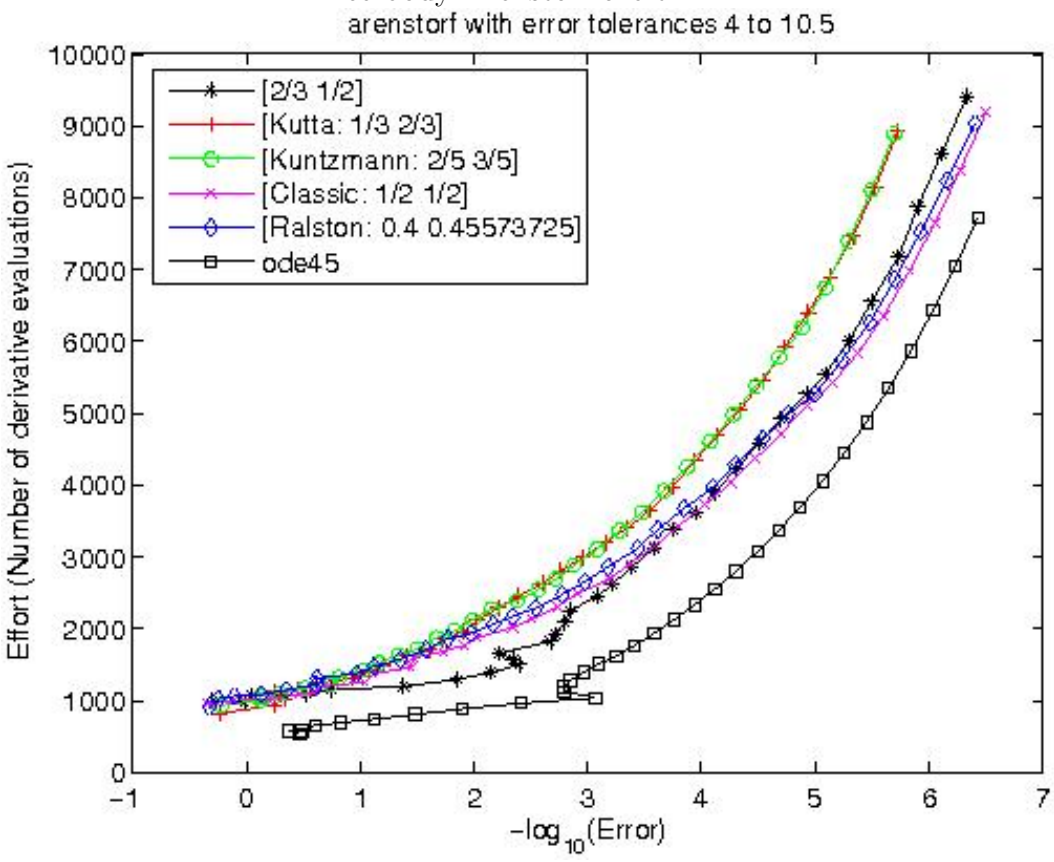
Kepler orbit with eccentricity 0.9



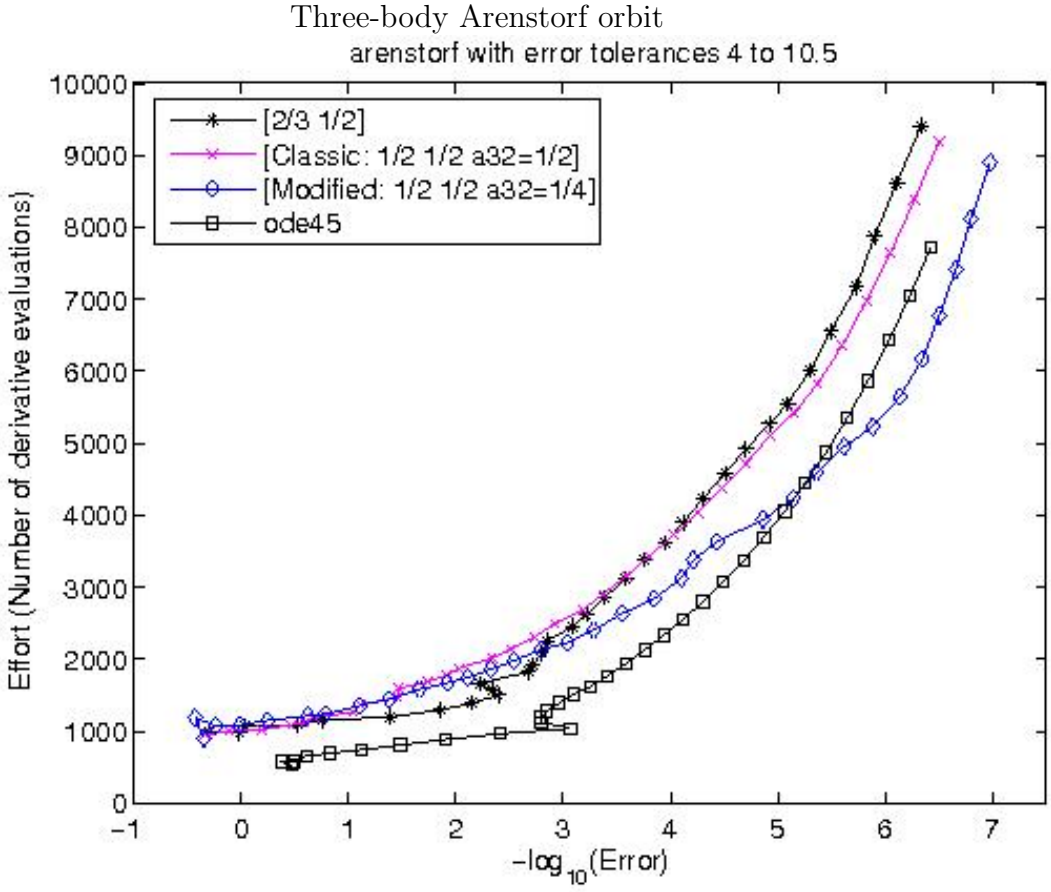
Three-body orbit of MATLAB's `orbitode`



Three-body Arenstorf orbit

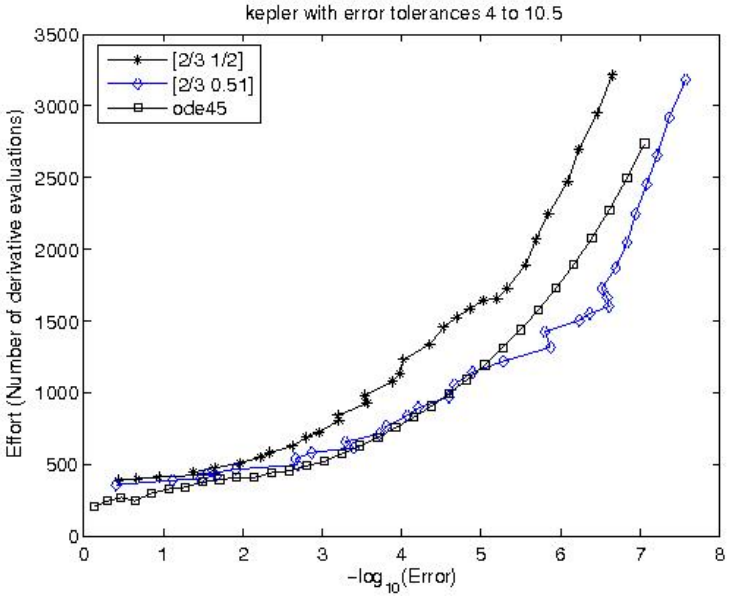


B.3. **Modifying a_{32} value of classic formula.**
The formula with $c_2 = 1/2$, $c_3 = 1/2$, and $a_{32} = 1/4$ compared to the classic formula ($a_{32} = 1/2$)

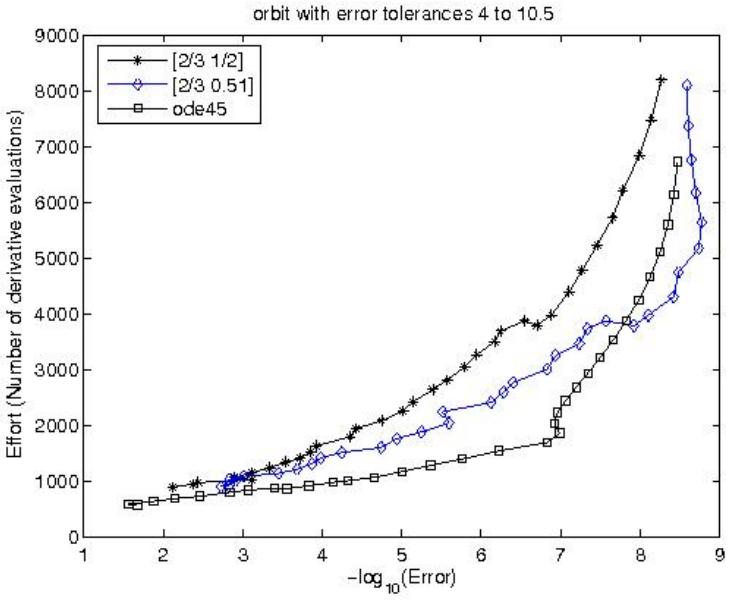


B.4. **Formula with $c_2 = 2/3$ and $c_3 = 0.51$.**
The formula with $c_2 = 2/3$ and $c_3 = 0.51$.

Kepler orbit with eccentricity 0.9



Three-body orbit of MATLAB's `orbitode`

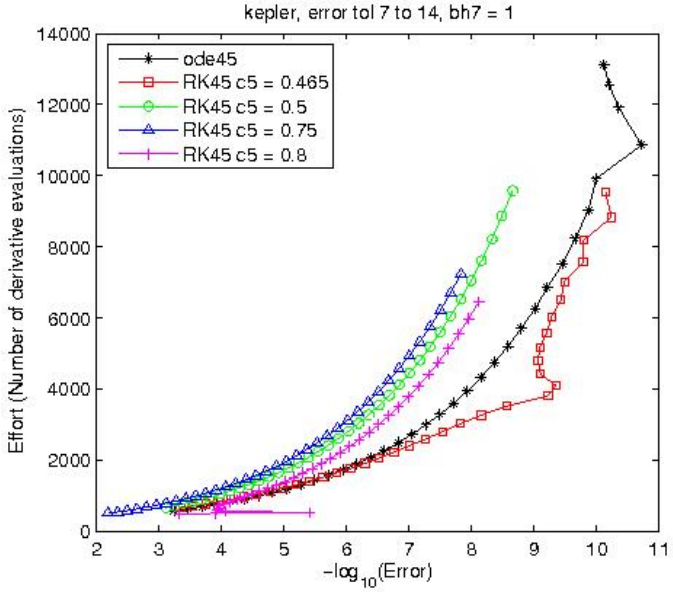


Appendix C. Error vs. Effort graphs for (4,5) Runge-Kutta Formulas and ode45

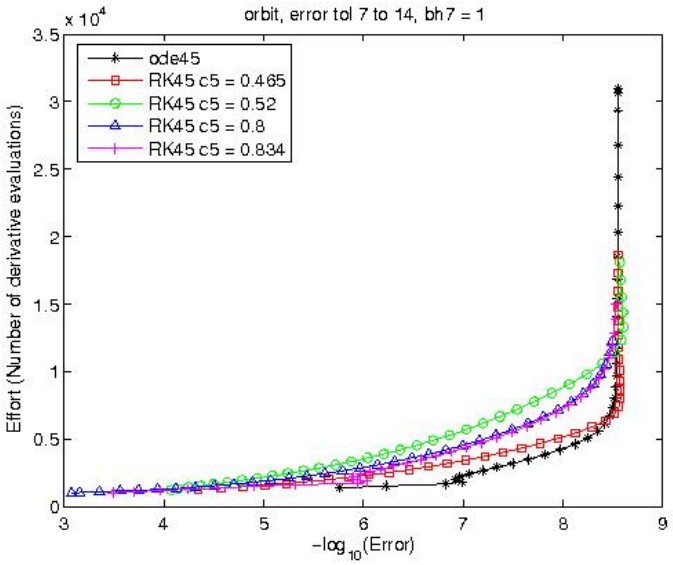
C.1. Formula with $\hat{b}_7 = 1$ and selected values of c_5.

RK (4,5) formulas (dependent on c_5) compared to `ode45`.

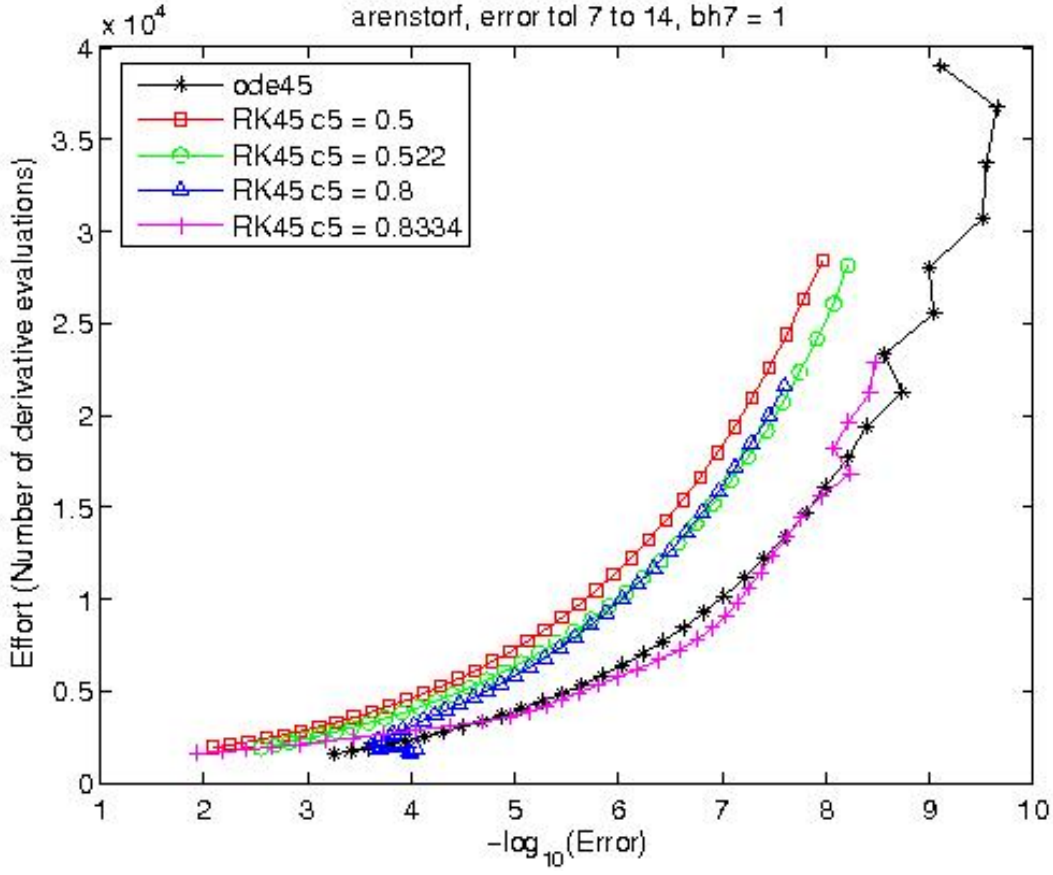
Kepler orbit with eccentricity 0.9



Three-body orbit of MATLAB's `orbitode`



Three-body Arenstorf orbit



References

[1] W. Adams, P. Loustaunau, *An Introduction to Gröbner Bases*, American Mathematics Society, Providence, RI, 1994.

[2] W. Boege, R. Gebauer, and H. Kredel. *Some examples for solving systems of algebraic equations by calculating Groebner bases*, J. Symb. Comp. 1 (1986), pp. 83-98.

[3] Folkmar Bornemann, *Runge-Kutta Methods, Trees, and Maple*, Selçuk Journal of Applied Mathematics, 2 (2001), pp. 3-15.

[4] J.C. Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*, John Wiley & Sons, Chichester, 1987.

[5] Samuel D. Conte, Carl de Boor, *Elementary Numerical Analysis*, McGraw-Hill, New York, 1980.

[6] J.R. Dormand, P.J. Prince, *A family of embedded Runge-Kutta formulas*, Journal of Computational and Applied Mathematics, 6 (1980), pp. 19-26.

[7] W.H. Enright, T.E. Hull, *Test Results on Initial Value Methods for Non-Stiff Ordinary Differential Equations*, SIAM Journal of Numerical Analysis, 13 (1976), pp. 944-961.

[8] Erwin Fehlberg, *Classical fifth, sixth, seventh, and eigth order Runge-Kutta formulas with step-size control*, NASA, Springfield, VA, 1968.

[9] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II* Springer-Verlag, New York, 1991.

[10] R. L. Honeycutt, *Stochastic Runge-Kutta algorithms. II. Colored Noise*, Phys. Rev. A., 45 (1992), pp. 604-610.

[11] F.Q. Hu, J.L. Matheny, M.Y. Hussaini, *Low-dissipation and low-dispersion Runge-Kutta schemes for computational acoustics*, Journal of Computational Physics, 124 (1996), pp. 177-191.

[12] T.E. Hull, W.H. Enright, B.M. Fellen, A.E. Sedgwick *Comparing Numerical Methods for Ordinary Differential Equations* SIAM Journal on Numerical Analysis, 9 (1972), pp. 603-637.

[13] T.E. Hull and R.L. Johnston, *Optimum Runge-Kutta methods*, Math. Comp., 18 (1964), pp. 306-310.

[14] Peter Kaps, Peter Renthrop, *Generalized Runge-Kutta methods of order four with stepsize control for stiff ordinary differential equations*, Numerishce Mathematik, 33 (1979), pp. 55-68.

[15] C.A. Kennedy, M.H. Carpenter, R.M. Lewis, *Low-storage, explicit Runge-Kutta schemes for the compressible Navier-Stokes equations*, Applied Numerical Mathematics, 35 (2000), pp. 177-219.

[16] J. Kuntzmann, *Deux Formules Optimales du type de Runge-Kutta*, Chiffres, 2 (1959), pp. 21-26.

[17] J.D. Lambert, *Numerical Methods for Ordinary Differential Systems*, John Wiley & Sons, New York, 1991.

[18] L. Lapidus and J. Seinfeld, *Numerical Solution of Ordinary Differential Equations*, Academic Press, New York, 1971.

[19] S. N. Papakostas and G. Papageorgiou, *A Family of Fifth Order Runge-Kutta Pairs*, Mathematics of Computation, 65 (1996), pp. 1165-1181.

[20] A. Ralston, P. Rabinowitz, *A First Course in Numerical Analysis*, McGraw-Hill, New York, 1978.

[21] S. Reich, *Multi-sympletic Runge-Kutta Collocation Methods for Hamiltonian wave equations*, Journal of Computational Physics, 157 (2000), pp. 473-499.
[22] Joseph J. Rotman, *A First Course in Abstract Algebra*, Prentice-Hall, New Jersey, 2000.
[23] Lawrence F. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman and Hall Mathematics, 1994.
[24] Lawrence F. Shampine, *Local error estimation by doubling*, Springer Wien, 34 (1985), pp. 179-190.
[25] L.F. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations*, W.H. Freeman & Co., 1975.
[26] L.F. Shampine and M.W. Reichelt, *The MATLAB ODE Suite*, SIAM Journal on Scientific Computing, 18-1, 1997.
[27] P.W. Sharp and J.H. Verner, *Explicit Runge-Kutta 4-5 Pairs wit Interpolants*, Mathematical Preprint No. 1995-03, Queen's University, (1995).
[28] J.G. Verwer, *Explicit Runge-Kutta methods for parabolic partial differential equations*, Applied Numerical Mathematics, 22 (1996), pp. 359-379.
[29] H.A. Watts, *Starting step size for an ODE solver*, J. Comput. Appl. Math., 9 (1983), pp. 177-191.

30401 Ashton Lane, Bay Village, OH 44140
E-mail address: dupalsm@rose-hulman.edu

2688 Marsh Drive, San Ramon, CA 94583
E-mail address: michael.yoshizawa@pomona.edu