**Les Cahiers du GERAD** ISSN: 0711–2440

# Tuning Runge-Kutta parameters on a family of ordinary differential equations

C. Audet

G–2017–18

March 2017



**GERAD** HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

**Tél. : 514 340-6053**
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

# Tuning Runge-Kutta parameters on a family of ordinary differential equations

**Charles Audet** [a]

[a] *GERAD & Département de mathématiques et génie industriel, Polytechnique Montréal (Québec) Canada, H3C 3A7*

charles.audet@gerad.ca

**March 2017**

**Les Cahiers du GERAD**

**G–2017–18**

**Abstract:** The Runge-Kutta class of iterative methods is designed to approximate solutions of a system of ordinary differential equations (ODE). The second-order class of Runge-Kutta methods is determined by a system of 3 nonlinear equations and 4 unknowns, and includes the modified-Euler and mid-point methods. The fourth-order class is determined by a system of 8 nonlinear equations and 10 unknowns. This work formulates the question of identifying good values of these 8 parameters for a given family of ODE as a blackbox optimization problem. The objective is to determine the parameter values that minimize the overall error produced by a Runge-Kutta method on a training set of ODE. Numerical experiments are conducted using the NOMAD direct-search optimization solver.

**Keywords:** Runge-Kutta, parameter tuning, blackbox optimization, direct-search

**Acknowledgments:** This Work was supported by Natural Sciences and Engineering Research Council of Canada (NSERC), Discovery grant #2015-05311.

# 1 Introduction

Many algorithms rely on a set of parameters. The question of identifying values that improve performance is known as *parameter tuning* [9]. Various metrics may be used to quantify the performance of an algorithm, including accuracy of the solutions and computational time. There are frameworks for tuning parameters on a collection of test problems [2, 4, 7].

The objective of the present paper is to illustrate the high-level simplicity of parameter tuning on the well-known class of fourth-order Runge-Kutta methods for solving systems of ordinary differential equations (ODE).

The question of tuning the parameter on a family of ODE is formulated as a blackbox optimization problem that takes as input values of the Runge-Kutta parameters, then solves a series of ODE systems and returns as output a measure of the norm of the errors produced by the Runge-Kutta method. This blackbox optimization problem is then solved by a direct-search solver. For cross-validation, the optimized parameters are then tested and compared on other members of the family of ODE.

The document is structured as follows. Section 2 presents the fourth-order Runge-Kutta class of methods that depend on two real parameters. Section 3 models the question of tuning these parameters as a blackbox optimization problem. Finally, Section 4 analyzes the results produced by applying the optimization framework on two specific families of ODE.

# 2 Fourth-order Runge-Kutta methods

Runge-Kutta methods solve a system of $\ell$ ordinary differential equations (ODE) of the type $y'(t) = f(t, y(t))$ where $y : \mathbb{R} \mapsto \mathbb{R}^\ell$ and $f : \mathbb{R} \times \mathbb{R}^\ell \mapsto \mathbb{R}^\ell$, subject to the initial condition $y(t^0) = y^0 \in \mathbb{R}^\ell$ with $t^0 \in \mathbb{R}$. The second-order class of Runge-Kutta methods is determined by a system of 3 nonlinear equations and 4 unknows, and the class includes the modified-Euler and mid-point methods.

Fourth-order Runge-Kutta methods use a constant step size $h \in \mathbb{R}_+$ and estimate $y(t^n)$ by the value $y^n$ through the following iterative process initiated at $n = 0$:

$$\begin{aligned}
t^n &= t^0 + nh \\
k_1 &= hf(t^n, y^n) \\
k_2 &= hf(t^n + \beta_1 h, y^n + \beta_1 k_1) \\
k_3 &= hf(t^n + \beta_2 h, y^n + \beta_3 k_2 + (\beta_2 - \beta_3) k_1) \\
k_4 &= hf(t^n + \beta_4 h, y^n + \beta_5 k_2 + \beta_6 k_3 + (\beta_4 - \beta_5 - \beta_6) k_1) \\
y^{n+1} &= y^n + \alpha_1 k_1 + \alpha_2 k_2 + \alpha_3 k_3 + \alpha_4 k_4 \\
n &\leftarrow n + 1
\end{aligned}$$

in which the parameters $(\alpha, \beta) \in \mathbb{R}^4 \times \mathbb{R}^6$ are chosen in such a way that $y(t^{n+1})$ is approximated by a fourth-degree polynomial around $y(t^n)$ with error term $\mathcal{O}(h^5)$. The cumulative errors on $y^n, y^{n-1}, \ldots, y^1$ add up to yield the approximation $y(t^n) = y^n + \mathcal{O}(h^4)$. Gill [6] shows that this is achieved when the ten parameters $\alpha \in \mathbb{R}^4$ and $\beta \in \mathbb{R}^6$ satisfy the eight nonlinear equations

$$\begin{aligned}
\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 &= 1, & \alpha_3\,\beta_1\,\beta_3 + \alpha_4\,\beta_1\,\beta_5 + \alpha_4\,\beta_2\,\beta_6 &= \tfrac{1}{6}, \\
\alpha_2\,\beta_1 + \alpha_3\,\beta_2 + \alpha_4\,\beta_4 &= \tfrac{1}{2}, & \alpha_3\,\beta_1\,\beta_2\,\beta_3 + \alpha_4\,\beta_1\,\beta_4\,\beta_5 + \alpha_4\,\beta_2\,\beta_4\,\beta_6 &= \tfrac{1}{8}, \\
\alpha_2\,\beta_1^2 + \alpha_3\,\beta_2^2 + \alpha_4\,\beta_4^2 &= \tfrac{1}{3}, & \alpha_3\,{\beta_1}^2\beta_3 + \alpha_4\,{\beta_1}^2\beta_5 + \alpha_4\,\beta_2^2\beta_6 &= \tfrac{1}{12}, \\
\alpha_2\,{\beta_1}^3 + \alpha_3\,\beta_2^3 + \alpha_4\,\beta_4^3 &= \tfrac{1}{4}, & \alpha_4\,\beta_1\,\beta_3\,\beta_6 &= \tfrac{1}{24}.
\end{aligned}$$

Any values that satisfy these equations define a valid Runge-Kutta method, and the classical ones are

$$\alpha = \left(\frac{1}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{6}\right), \quad \beta = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 1, 0, 1\right).$$

These equations possess two degrees of freedom, and given values of $\beta_1$ and $\beta_5$, the other parameters can be identified according to the following equations:

$$\begin{aligned}
\beta_2 &= \begin{cases} \dfrac{1}{2} & \text{if } \beta_1 = \dfrac{1}{2} \\ \dfrac{12\beta_1^3\beta_5 - 6\beta_1\beta_5 - \sqrt{\gamma} - 5\beta_1 + 5}{8(3\beta_1^2\beta_5 - 2\beta_1\beta_5 - \beta_1 + 1)} & \text{otherwise} \end{cases} \\
\text{where} \quad \gamma &= 144\beta_1^6\beta_5^2 - 384\beta_1^5\beta_5^2 + 400\beta_1^4\beta_5^2 - 40\beta_1^4\beta_5 - 192\beta_1^3\beta_5^2 + 72\beta_1^3\beta_5 \\
&\quad +36\beta_1^2\beta_5^2 + 16\beta_1^3 - 36\beta_1^2\beta_5 - 39\beta_1^2 + 4\beta_1\beta_5 + 30\beta_1 - 7; \\
\beta_3 &= \begin{cases} \dfrac{1}{2(1-\beta_5)} & \text{if } \beta_1 = \dfrac{1}{2} \\ \dfrac{\beta_2(\beta_1 - \beta_2)}{2\beta_1(2\beta_1 - 1)} & \text{otherwise;} \end{cases} \\
\beta_4 &= 1; \\
\beta_6 &= \frac{\beta_1(-\beta_1\beta_2\beta_5 + \beta_2^2\beta_5 + \beta_1\beta_3 - \beta_3)}{12\beta_1^3\beta_3^2 - 4\beta_1^2\beta_2\beta_3 - 8\beta_1^2\beta_3^2 + 4\beta_1\beta_2^2\beta_3 + \beta_1\beta_2^2 - \beta_2^3}; \\
\alpha_2 &= \frac{12\beta_1^2\beta_2^2\beta_3\beta_5 - 8\beta_1^2\beta_2\beta_3\beta_5 - 4\beta_1^2\beta_3^2 - 4\beta_1\beta_2^2\beta_3 + 4\beta_1\beta_2\beta_3 + \beta_2^3 - \beta_2^2}{24\beta_1^3\beta_3(-\beta_1\beta_2\beta_5 + \beta_2^2\beta_5 + \beta_1\beta_3 - \beta_3)}; \\
\alpha_3 &= \frac{12\beta_1^3\beta_3\beta_5 - 8\beta_1^2\beta_3\beta_5 - 4\beta_1^2\beta_3 + \beta_1\beta_2 + 4\beta_1\beta_3 - \beta_2}{24\beta_1^2\beta_3(\beta_1\beta_2\beta_5 - \beta_2^2\beta_5 - \beta_1\beta_3 + \beta_3)}; \\
\alpha_4 &= \frac{12\beta_1^3\beta_3^2 - 4\beta_1^2\beta_2\beta_3 - 8\beta_1^2\beta_3^2 + 4\beta_1\beta_2^2\beta_3 + \beta_1\beta_2^2 - \beta_2^3}{24\beta_1^2\beta_3(-\beta_1\beta_2\beta_5 + \beta_2^2\beta_5 + \beta_1\beta_3 - \beta_3)}; \\
\alpha_1 &= 1 - \alpha_2 - \alpha_3 - \alpha_4.
\end{aligned} \tag{1}$$

Other values than the classical ones appear in the literature. Gill [6] proposes the values

$$\begin{aligned}
\alpha &= \left(\frac{1}{6}, \frac{1}{3}(1 - \frac{1}{\sqrt{2}}), \frac{1}{3}(1 + \frac{1}{\sqrt{2}}), \frac{1}{6}\right), \\
\beta &= \left(\frac{1}{2}, \frac{1}{2}, 1 - \frac{1}{\sqrt{2}}, 1 - \frac{1}{\sqrt{2}}, 1, 1 + \frac{1}{\sqrt{2}}\right)
\end{aligned}$$

which minimize storage requirements by reusing function evaluations. Ralston [10] suggests the values

$$\begin{aligned}
\alpha &= \left(\frac{263}{1812} + \frac{2\sqrt{5}}{151}, -\frac{250\sqrt{5}}{957} + \frac{125}{3828}, \frac{3426304}{5924787} + \frac{553984\sqrt{5}}{1974929}, \frac{10}{41} - \frac{4\sqrt{5}}{123}\right), \\
\beta &= \left(\frac{2}{5}, \frac{7}{8} - \frac{3\sqrt{5}}{16}, \frac{3785}{1024} - \frac{405\sqrt{5}}{256}, 1, -\frac{1523\sqrt{5}}{1276} - \frac{975}{2552}, \frac{93408}{48169} + \frac{203968\sqrt{5}}{240845}\right)
\end{aligned}$$

that minimize the theoretical truncation error, i.e., the coefficient multiplying the $h^4$ error term. Some of these parameters are negative.

The present work suggests to determine values of $\alpha$ and $\beta$ by tailoring them to a specific family of systems of ODE. More precisely, the question is formulated in Section 3 as a blackbox optimization problem, in which the blackbox takes as input values of $\beta_1$ and $\beta_5$, uses them to solve a subset of the elements of the family of ODE, and returns a measure of the overall numerical error. A blackbox optimization solver then identifies the values of $\beta_1$ and $\beta_5$ that minimize the error. Computational experiments are conducted on two families of systems of ODE, and comparisons with the classical, Gill's and Ralston's parameters are presented in Section 4.

## 3 Blackbox formulation

Two families of systems of ODE, denoted $\mathcal{A}$ and $\mathcal{B}$, are considered in this work. Each of them is parameterized by an integer $\ell$ that defines the number of equations. The systems, initial conditions, and analytical solutions are

$$\mathcal{A}: \quad \begin{cases} y_i'(t) = \left(\frac{y_i}{t}\right)^2 + \frac{i(i+1)}{2y_{i+1}} - \frac{i^2}{t}, & i = 1, 2, \ldots, \ell - 1 \\ y_\ell'(t) = \left(\frac{y_\ell}{t}\right)^2 + \frac{\ell}{2y_1} - \frac{\ell^2}{t}, & \\ \text{with} \quad y_i(1) = i, & i = 1, 2, \ldots, \ell, \\ \text{solution:} \quad y_i(t) = i\sqrt{t} & i = 1, 2, \ldots, \ell. \end{cases}$$

$$\mathcal{B}: \quad \begin{cases} y_i'(t) = \frac{y_i^2}{t(1+i)} - (1+i)\sqrt{\frac{(i+1)(2+i+y_{i+1})}{2+i-y_{i+1}}}, & i = 1, 2, \ldots, \ell - 1 \\ y_\ell'(t) = \frac{y_\ell^2}{t(1+\ell)} - (1+\ell)\sqrt{\frac{2+y_1}{2-y_1}}, & \\ \text{with} \quad y_i(1) = 1 - i, & i = 1, 2, \ldots, \ell, \\ \text{solution:} \quad y_i(t) = \frac{(1+i)(1-i\,t^2)}{1+i\,t^2} & i = 1, 2, \ldots, \ell. \end{cases}$$

For both families, the Runge-Kutta will be used to estimate the value $y(4)$ from the initial solution $y(1)$ for a collection of values of $\ell$ and $n$. For a given value of the number of steps $n$, the step size used in the Runge-Kutta method will be $h = \frac{3}{n}$. In the region of interest, both ODE systems and their solutions are well defined and in $\mathcal{C}^\infty$.

In order to assess the quality of the Runge-Kutta algorithmic parameters, the extended-valued function $\psi : \mathbb{R}^2 \mapsto \mathbb{R} \cup \{\infty\}$ returning a measure of the error produced by the application of Runge-Kutta is introduced. The function $\psi$ consists of a simulation that takes as input the real values $\beta_1$ and $\beta_5$, then applies Equation (1) to determine the parameters $\alpha \in \mathbb{R}^4$ and $\beta \in \mathbb{R}^6$, and then uses these values to launch the Runge-Kutta method on a collection of ODE. Formally, the function $\psi$ returns:

$$\psi(\beta_1, \beta_5) \;=\; \begin{cases} \left( \sum_{\ell=4}^{7} \sum_{n=145}^{150} e_{\ell,n}^2 \right)^{\frac{1}{2}} & \text{if Runge-Kutta succeeds with } (\alpha, \beta) \\ \infty & \text{otherwise} \end{cases}$$

where $e_{\ell,n} = \|y^n - y(4)\|_2$ is the two-norm of Runge-Kutta approximation error. There are situations where Runge-Kutta fails to solve to system, for example

i- when $\beta_1 = \frac{1}{2}$ and $\beta_5 = 1$, the denominator used to compute $\beta_3$ in Equation (1) is null;

ii- the intermediate value $\gamma$ required to compute $\beta_2$ Equation (1) is negative;

iii- applying Runge-Kutta numerically fails as the vector $y^n$ grows unbounded (it is the case with the classical parameters on system $\mathcal{A}$ with $\ell = 9$ equations and $n = 150$ steps).

The error measure $\psi$ takes into account a total of 24 systems of ODE. The number of equations $\ell$ ranges from 4 to 7 and the number of steps $n$ ranges from 145 to 150. Of course, these values could have been chosen differently, thereby leading to a different analysis.

The question of tuning the Runge-Kutta algorithmic parameters reduces to an unconstrained optimization problem in $\mathbb{R}^2$ compactly written as

$$\min_{\beta_1, \beta_5} \quad \psi(\beta_1, \beta_5). \tag{2}$$

This problem falls into the realm of blackbox optimization [1] in which a simulation needs to be launched in order to compute the objective function value. The iterative nature of the Runge-Kutta method makes it improbable that the objective function is differentiable. Figure 1 plots the least value between $\psi$ and 2 for various values of $\beta_1$ and $\beta_5$ on the system $\mathcal{A}$. The holes in the plot surface correspond to values of the parameters for which the system (1) has no real solutions, i.e., when $\psi = \infty$. The error function $\psi$ to be minimized is clearly nonsmooth.

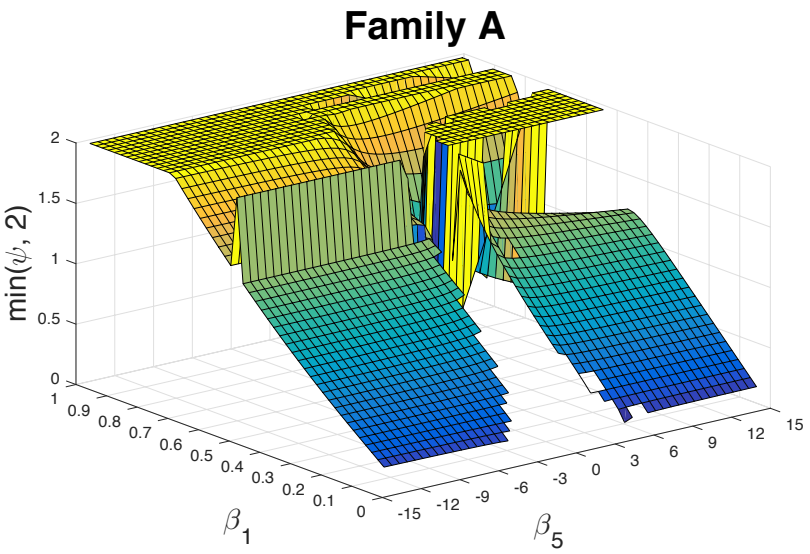


**Figure 1: Graph of the nonsmooth error function for family $\mathcal{A}$**

## 4 Numerical experiments

Coding of the Runge-Kutta methods and of the objective function $\psi$ was done in Matlab. The NOMAD [8] implementation[1] of the mesh adaptive direct search (MADS) algorithm [3] is used to solve Problem (2). MADS is designed for constrained or unconstrained blackbox optimization in which the functions defining the objective and constraints are typically computed through a time-consuming computer simulation. Many blackbox optimization applications are listed in the survey [1]. MADS is supported by a rigorous convergence analysis [3] based on the Clarke nonsmooth calculus [5].

NOMAD is launched within Matlab using all the default options, from the starting point $(\beta_1, \beta_5) = (0.5, 0)$ which corresponds to the classical parameters, and it terminates after a total of 100 blackbox evaluations. The actual Matlab syntax is

```
nomad(@fun,[0.5,0],[],[], nomadset('max_bb_eval',100));
```

in which the first parameter is the function that returns $\psi$, the second is the initial solution ($[\beta_1, \beta_5]$ from the classical parameters). The third and fourth are the bounds (none) and the last one provides the termination criteria for NOMAD.

Each blackbox evaluation takes $\beta_1$ and $\beta_5$ as input, extends these parameters to create $\alpha \in \mathbb{R}^4$ and $\beta \in \mathbb{R}^6$ and then returns the value $\psi$ after solving all 24 systems of ODE. Figure 2 gives a high-level illustration of the blackbox optimization framework.

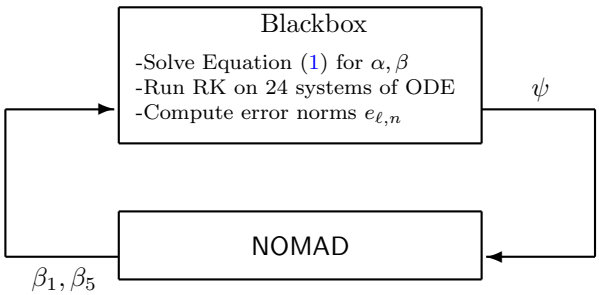


**Figure 2: The NOMAD solver pilots the blackbox optimization process**

[1] Software available at https://www.gerad.ca/nomad

The optimal values of $\alpha$ and $\beta$ produced by NOMAD for each family $\mathcal{A}$ and $\mathcal{B}$ are reported in Table 1.

**Table 1: Optimal parameters produced by NOMAD**

| Family | |
|---|---|
| $\mathcal{A}$ | $\alpha$ = ( 0.1176, 0.0580, 0.6594, 0.1650 ),<br>$\beta$ = ( 0.0328, 0.5052, 3.8995, 1.0000, -15.2460, 1.9777 ) |
| $\mathcal{B}$ | $\alpha$ = ( 0.1803, -11.6094, 12.2241, 0.2051 ),<br>$\beta$ = ( 0.6305, 0.6229, 0.0143, 1.0000, -21.7739, 22.4784 ) |

Table 2 compares the three parameter values from the literature to the two produced by the optimization process on the training set of ODE with $\ell \in \{4, 5, 6, 7\}$ and $n \in \{145, 146, \ldots, 150\}$. For both families, the table provides the corresponding objective function value $\psi$. Without any surprise, the smallest value for family $\mathcal{A}$ corresponds to the one generated by NOMAD on that family. A similar conclusion holds for family $\mathcal{B}$. With family $\mathcal{B}$, the gain produced by the optimization is by a factor of 5 over the parameters from the literature. With family $\mathcal{A}$, the gain is by a factor of more than 300.

**Table 2: Objective function value for both families of ODE with different parameters**

| | Family $\mathcal{A}$ | | | Family $\mathcal{B}$ | | |
|---|---|---|---|---|---|---|
| | training | cross-validation | | training | cross-validation | |
| Parameters | $\psi$ | average | worst | $\psi$ | average | worst |
| Classic | 1.364 | 1.000 | 1.000 | 2.565e-06 | 1.000 | 1.000 |
| Ralston | 1.055 | 0.757 | 0.774 | 2.254e-06 | 0.899 | 0.972 |
| Gill | 1.363 | 0.999 | 1.000 | 2.170e-06 | 0.866 | 0.909 |
| NOMAD$_{\mathcal{A}}$ | 0.0037 | 0.025 | 0.071 | 1.865e-06 | 0.728 | 0.968 |
| NOMAD$_{\mathcal{B}}$ | 1.834 | 1.397 | 1.737 | 4.758e-07 | 0.248 | 0.547 |

The table also summarizes the relative gain with respect to the classical parameters on other systems of ODE within the same families. For each family $\mathcal{A}$ and $\mathcal{B}$, a cross-validation group of 24 systems of ODE is solved: The number of equations $\ell$ ranges from 3 to 8 and the number of steps $n$ takes the values 120, 140, 160 and 180. None of these cross-validation systems overlaps with the training ones used to construct $\psi$. For each cross-validation family, the average and worst error (over the 24 systems) are reported for each of the five parameters in Table 2. These values are normalized so that the classical values equal 1. Once again, the optimized parameters outperform the three sets of parameters from the literature. On average, the gain over the classical parameters is by a factor of 40 for family $\mathcal{A}$, and the gain is by a factor of 4 for $\mathcal{B}$. The worst case associated to the optimized parameters is also systematically preferable to the classical parameters.

Figure 3 illustrates additional cross-validation tests. The figure plots the base 10 logarithm of the error term $e_{\ell,n}$ using a fixed number of steps $n = 150$, and for some values of $\ell$ outside of the training set $\{4, 5, 6, 7\}$. The plot on the left shows that the NOMAD$_{\mathcal{A}}$ parameters outperforms the four other sets of parameters for all values of $\ell$ ranging from 3 to 10 on family $\mathcal{A}$. Only the NOMAD$_{\mathcal{A}}$ parameters allow to solve the systems with 9 and 10 equations. The four other methods generate unbounded values. The NOMAD$_{\mathcal{A}}$ parameters fail to solve systems with 11 or more equations. The right part of the figure illustrates the same comparison but for family $\mathcal{B}$ and for larger systems: the number of equations $\ell$ varies form 3 to 25. Here, the NOMAD$_{\mathcal{B}}$ parameters systematically dominates the four others. The same behavior is observed for larger values of $\ell$.

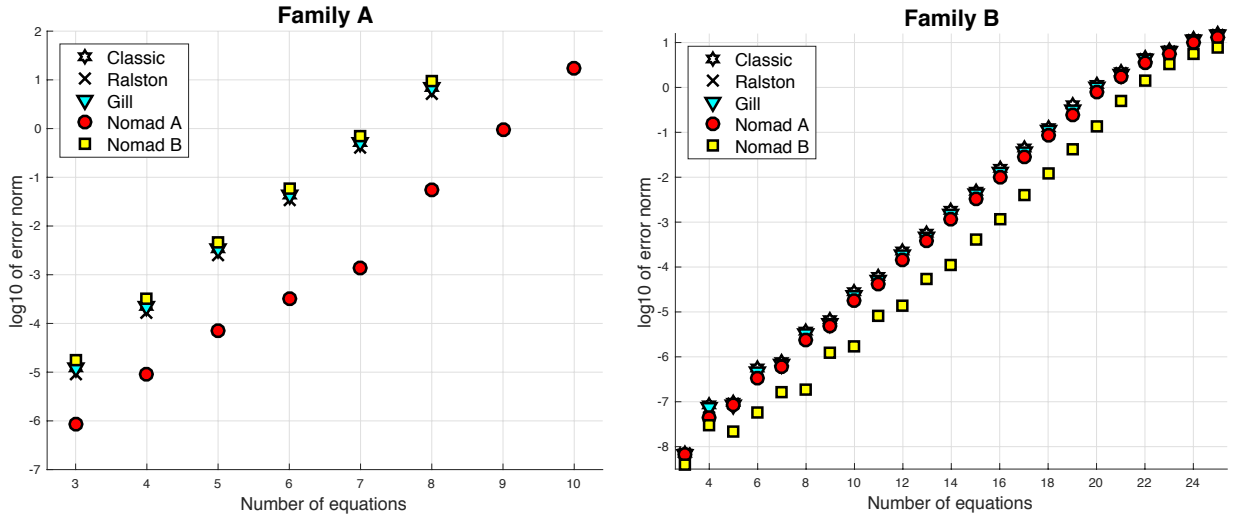


**Figure 3: Comparison of solutions on larger systems of ODE**

For comparison, the Matlab adaptive routine ode45 was launched on both families. In both cases, the errors are important. Figure 4 shows the solutions produced by ode45 on the problems of family $\mathcal{A}$ with $\ell \in \{4, 5, 6\}$. A difficulty arises when $\ell = 5$, as one of the function is clearly not convex. The top curve on the central plot fails to be a good approximation of the solution $y_5(t) = 5\sqrt{t}$ since at $t = 4$, the Runge-Kutta approximation $y^n > 12$ is far from the exact solution $y_5(4) = 10$. The plot on the right with $\ell = 6$ reports values of the order of $10^{14}$. Much smaller errors are observed on family $\mathcal{B}$.

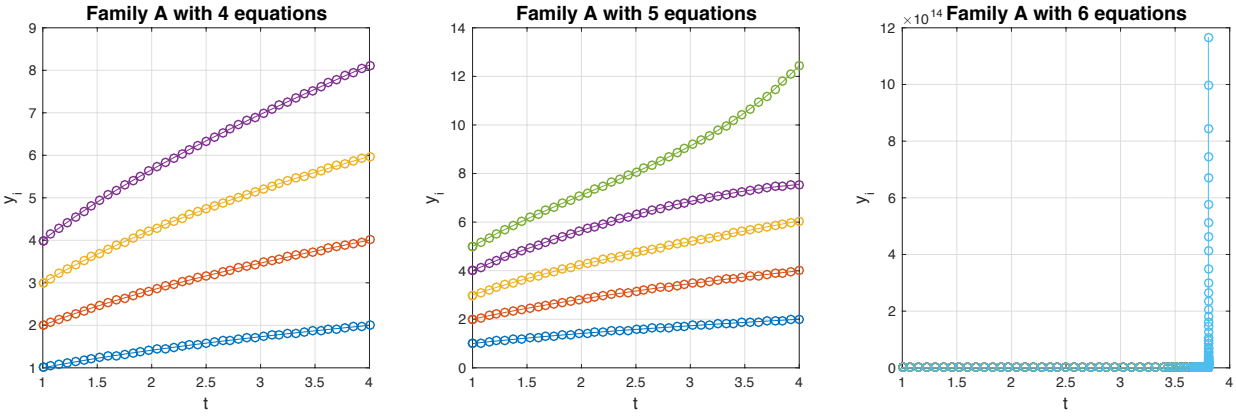


**Figure 4: Solutions produced by the Matlab** ode45 **command on family $\mathcal{A}$**

# 5 Discussion

Designers of algorithms often provide default values for their algorithmic parameters. The objective of this paper is to show that these parameters can be customized to improve the quality of the results on some families of problems. This is achieved by formulating a blackbox optimization problem, and using an appropriate optimization solver.

The methodology is applied to tune the parameters of the fourth-order Runge-Kutta methods on two families of ODE. The optimized parameters are specifically tuned for each family and may not be pertinent for use with other families.

# References


[1] C. Audet. A survey on direct search methods for blackbox optimization and their applications. In P.M. Pardalos and T.M. Rassias, editors, Mathematics without boundaries: Surveys in interdisciplinary research, chapter 2, pages 31–56. Springer, 2014.

[2] C. Audet, C.-K. Dang, and D. Orban. Optimization of algorithms with OPAL. Mathematical Programming Computation, 6(3):233–254, 2014.

[3] C. Audet and J.E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. SIAM Journal on Optimization, 17(1):188–217, 2006.

[4] P. Balaprakash, S.M. Wild, and B. Norris. Spapt: Search problems in automatic performance tuning. Procedia Computer Science, 9:1959–1968, 2012. Proceedings of the International Conference on Computational Science, ICCS 2012.

[5] F.H. Clarke. Optimization and Nonsmooth Analysis. John Wiley & Sons, New York, 1983. Reissued in 1990 by SIAM Publications, Philadelphia, as Vol. 5 in the series Classics in Applied Mathematics.

[6] S. Gill. A process for the step-by-step integration of differential equations in an automatic digital computing machine. Proceedings of the Cambridge Philosophical Society, 47:95–108, 1951.

[7] H.H. Hoos. Automated algorithm configuration and parameter tuning. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, Autonomous Search, pages 37–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[8] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear Optimization with the MADS algorithm. ACM Transactions on Mathematical Software, 37(4):44:1–44:15, 2011.

[9] K. Naono, K. Teranishi, J. Cavazos, and R. Suda. Software Automatic Tuning: From Concepts to State-of-the-Art Results. Springer, 2010.

[10] A. Ralston. Runge-Kutta methods with minimum error bounds. Mathematics of Computation, 16:431–437, 1962.