Comparison of Numerical Solvers for Differential Equations for Holonomic Gradient Method in Statistics

N.Takayama, T.Yaguchi, Y.Zhang

November 23, 2021

1 Introduction

Definite integrals with parameters of holonomic functions satisfy holonomic systems of linear partial differential equations. When we restrict parameters to a one dimensional curve, then the system becomes a linear ordinary differential equation (ODE). For example, the definite integral with a parameter t

$$Z(t) = \int_0^{+\infty} \exp(ty - y^3) dy$$

satisfies the ODE $(3\partial_t^2 - t)Z(t) = 1$ where $\partial_t = \frac{d}{dt}$ (see [13, Chap 6] as an introductory exposition). We can evaluate the integral by solving the linear ODE numerically. The method of solving the linear ODE is not a high precision method, but it gives values of the integral at several evaluation points at once. This approach to evaluate numerically definite integrals is called *the holonomic gradient method* (HGM) and it is useful to evaluate several normalizing constants in statistics. See references in [12] or in ,e.g., [11], [26].

The HGM consists of 3 steps. The first step is to find a linear ODE for the definite integral $Z(t)$ with a parameter t. The 2nd step is to find value of $Z(t)$ and its derivatives at some points of t. The 3rd step is to solve the ODE. We can apply several results in computer algebra and in the theory of special functions of several variables for the first and the 2nd step. The HGM is successful for a broad class of distributions and the 3rd step is relatively easy in most of them. When the function $Z(t)$ is the dominant solution (the maximal growth rate solution) for the ODE when $t \to +\infty$ as in the case [11, Theorem 2], the 3rd step can be performed by a simple application of standard numerical solver for ODEs. However, there are some HGM problems for which we encounter difficulties in the 3rd step. The difficulty is that when the function $Z(t)$ is not dominant and we solve the linear ODE in a long interval, the numerical solution might become a false one. We have this difficulty in our analysis of the outage probability of MIMO WiFi systems (see the section 5 and [6]) and in solving an ODE numerically of the expectation of the Euler characteristic of a random manifold (see the section 6 and [26]). In principle, if we give an initial value vector exactly and the time step is sufficiently small, we do not get a false solution. However, these assumptions are not always satisfied in practice.

In this paper, we will discuss and compare methods to solve linear ODE's in the 3rd step of the HGM. We intend to provide references to the HGM users for choosing numerical solvers of ODE's for the HGM. We will present methods that work and methods that does not work. Of course, it depends on a given problem.

2 Methods to test

We will compare some methods mainly from a point of view of using for the 3rd step of the HGM[1]. In the HGM, the function $Z(t)$ to evaluate has an integral representation and satisfies an ODE

$$Lf = b, \quad L = \sum_{k=0}^{r} c_k(t)\partial_t^k, \ \partial_t = \frac{d}{dt} \tag{1}$$

where $b(t)$ is an inhomogeneous term and $f(t)$ is an unknown function. See, e.g., [6], [11], and papers in [12] as to examples. The differential equation (1) might have a solution u which dominates Z at $+\infty$. In other words, there might be a solution u such that $|u/Z| \to \infty$ when $t \to +\infty$. In such cases, it is not easy to obtain numerical values of $Z(t)$ globally by solving the ODE even when the almost accurate initial values of $Z(t), Z'(t), \ldots$ are known. See Examples 1 and 2. We call such case *the instable HGM problem*. We want to obtain correct values of Z as large interval of t as possible.

Here is a list of methods we try.

1. The Runge-Kutta method (see, e.g., [10, Chapter 2]).
2. The implicit Runge-Kutta method (see, e.g., [10, Chapters 2 and 4]).
3. Multi-step methods (see, e.g., [10, Chapters 3 and 5]).
4. The spectral method in the approximation theory (see, e.g., [2], [19], [27]).
5. Defusing method: restricting the initial value vector to a sub linear space (see, e.g., Section 3).
6. Sparse interpolations/extrapolations by ODE: solving a generalized boundary value problem by solving a linear equation or by solving an optimization problem (see, e.g., [3], [8] and the Section 4).

We compare these methods by different implementations on different languages: [2], [7], [9], [16], [17], [22], [23], [24], [25]. The robustness for input errors is an important point in our comparison, because initial values or boundary values might be evaluated by Monte-Carlo integrators.

The first 4 approaches are standard. In the HGM, we may be able to evaluate the target function $Z(t)$ at any point by numerical integration methods. Of course, the value obtained is not always accurate. For example, the accuracy of Monte Carlo integrators are not high. The last two methods of the defusing method and sparse interpolation/extrapolation method by ODE are expected to be used mainly under this situation of the HGM.

The program codes of this paper are obtainable from `http://www.math.kobe-u.ac.jp/OpenXM/Math/defusing/ref.html`.

3 Defusing method

In the HGM, we do not always have an exact initial value vector. We often have the case that the target function (the target integral) have a moderate growth, but the ODE has rapid growth solutions, too. Then, we need to correct the initial value vector so that the rapid growth solution causes a blow-up of the numerical solution by an error of the initial value vector. We will propose a heuristic method, which we will call *defusing method*, to avoid a blow-up of a solution under this situation. Any solution of the ODE can be expressed as a linear combination of basis functions of different growth order. The defusing method removes some basis functions of specified growth

[1]Programs of this paper can be obtainable from `http://www.math.kobe-u.ac.jp/OpenXM/Math/defusing/ref.html`. The folder/program names are given in blue letters. (Ignore the folder names if the folder does not exist.) The letter "-" in the file name may be "_" for some cases.

order to obtain the correct solution. The defusing method is an analogy of filtering some specified frequencies from wave data. This method may be well-known, but we do not find a relevant literature and we describe this method in this section.

In the HGM, we want to evaluate numerically a definite integral with a parameter t on some interval of t. We put this function $f(t)$. The HGM gives an algebraic method to find an ordinary differential equation (ODE) for $F(t) = (f(t), f'(t), f''(t), \ldots, f^{(r-1)})^T$ wher r is the rank of the ODE. We evaluate the integral by Monte-Carlo integration methods or by the saddle point approximation for large t's. Since $F(t)$ satisfies the differential equation, we can apply algorithms (see, e.g., [4], [14]) to find a basis of series solutions at $t = \infty$. Let F_{*j}, $j = 1, \ldots, r$ be the basis of series solutions around $t = \infty$ of the ODE. We denote by F_{ij} is the ith component of F_{*j} and F_i is the ith component of F. Any solution of the ODE is written as a linear combination of F_{*j}'s over $\mathbf{C}$. We assume

$$F_{11} > F_{12} > \cdots > F_{1m} > \cdots > F_{1r} > 0$$

with different growth orders when $t \to +\infty$. We suppose that

$$\left| \frac{\text{numerical approximation of } F_1(t)}{F_{1j}(t)} \right| \to 0, \quad t \to +\infty \tag{2}$$

for $1 \leq j < m$. See (25) as an example. Under these assumptions, we can conclude that $F(t)$ is approximated as a linear combination of $F_{*m}, \ldots, F_{*r}$. The coefficients of the linear combination can be estimated by numerical approximation values of $f(t)$ for large t's. Thus, we can interpolate and extrapolate numerical values of $F(t)$ around $t = +\infty$ by a set of fundamental solutions of the ODE. We call this method the defusing method by series solutions around $t = +\infty$. See the Example 7.

We want to find a numerical solution of the initial value problem of the ordinary differential equation (ODE)

$$\frac{dF}{dt} = P(t)F \tag{3}$$

$$F(t_0) = F_0^{\text{true}} \in \mathbf{R}^r \tag{4}$$

where $P(t)$ is an $r \times r$ matrix, $F(t)$ is a column vector function of size r, and F_0^{true} is the initial value of F at $t = t_0$. Let us explain a defusing method for the initial value problem. Solving this problem is the final step of the holonomic gradient method (HGM) [12]. We often encounter the following situation in the final step.

Situation 1

1. The initial value has at most 3 digits of accuracy. We denote this initial value F_0.
2. The property $|F| \to 0$ when $t \to +\infty$ is known, e.g., from a background of the statistics.
3. There exists a solution $\tilde{F}$ of (3) such that $|\tilde{F}| \to +\infty$ or non-zero finite value when $t \to +\infty$.

Under this situation, the HGM works only for a very short interval of t because the error of the initial value vector makes the fake solution $\tilde{F}$ dominant and it hides the true solution $F(t)$. We call this bad behavior of the HGM *the instability of the HGM*.

Example 1

$$\frac{d}{dt}F = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} F$$

The solution space is spanned by $F^1 = (\exp(-t), 0, 0)^T$, $F^2 = (0, \exp(-t), 0)^T$, $F^3 = (1, 1, 1)^T$. The initial value $(1, 0, 0)^T$ at $t = 0$ yields the solution F_1. Add some errors $(1, 10^{-30}, 10^{-30})^T$ to the initial value. Then, we have

t	value F_1 by RK	difference $F_1 - F_1^1$
50	1.92827e-22	9.99959e-31
60	8.75556e-27	1.00000e-30
70	1.39737e-30	$\boxed{\text{1.00000e-30}}$
80	1.00002e-30	$\boxed{\text{1.00000e-30}}$

We can see the instability.

Example 2

$$P(t) = \begin{pmatrix} 0 & 1 \\ t & 0 \end{pmatrix}.$$

This differential equation is obtained from the Airy differential equation

$$y''(t) - ty(t) = 0$$

by putting $F = (y(t), y'(t))^T$. It is well-known that the Airy function

$$\mathrm{Ai}(t) = \frac{1}{\pi} \lim_{b\to+\infty} \int_0^b \cos\left(\frac{s^3}{3} + ts\right) ds$$

is a solution of the Airy differential equation and

$$\begin{aligned}
\mathrm{Ai}(0) &= \frac{1}{3^{2/3}\Gamma(2/3)} = 0.355028053887817\cdots \\
\mathrm{Ai}'(0) &= \frac{1}{3^{1/3}\Gamma(1/3)} = -0.258819403792807\cdots \\
\lim_{t\to+\infty} \mathrm{Ai}(t) &= 0 \\
\lim_{t\to+\infty} \mathrm{Ai}'(t) &= 0
\end{aligned}$$

Figure 1 is a graph of Airy Ai function and Airy Bi function. The function $F(t) = (\mathrm{Ai}(t), \mathrm{Ai}'(t))^T$ satisfies the condition 2 of the Situation 1 of the instability problem.

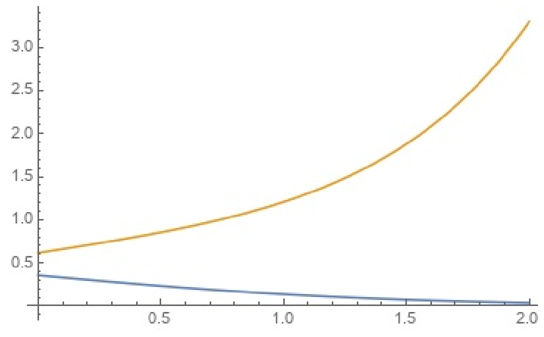

Figure 1: Airy Ai(x) and Bi(x) drawn by Mathematica

We can also see that the condition 3 of the Situation 1 holds by applying the theory of singularity of ordinary differential equations (see, e.g., the manual `DEtools/formal_sol` of Maple [15] and its references on the theory, which has a long history). In fact, the general solution of the Airy differential equation is expressed as

$$\begin{aligned}
& C_1 t^{-1/4} \exp\left(-\frac{2}{3}t^{3/2}\right)(1 + O(t^{-3/2})) \\
+ \quad & C_2 t^{-1/4} \exp\left(\frac{2}{3}t^{3/2}\right)(1 + O(t^{-3/2}))
\end{aligned}$$

when $t \to +\infty$ where C_i's are arbitrary constants.

We note that the high precision evaluation of the Airy function is studied by several methods (see, e.g., [5] and its references). Some mathematical software systems have evaluation functions of the Airy function. For example, `N[AiryAi[10]]` gives the value of $\mathrm{Ai}(10)$ on Mathematica. By utilizing these advanced evaluation methods, we use the Airy differential equation for our test case to check the validity of our heuristic algorithm.

We are going to propose some heuristic methods to avoid the instability problem of the HGM. Numerical schemes such as the Runge-Kutta method obtain a numerical solution by the recurrence

$$F_{k+1} = Q(k,h)F_k \tag{5}$$

from F_0 where $Q(k,h)$[2] is an $r \times r$ matrix determined by a numerical scheme and h is a small number The vector F_k is an approximate value of $F(t)$ at $t = t_k = t_0 + hk$.

Example 3 The Euler method assumes $dF/dt(t)$ is approximated by $(F(t+h) - F(t))/h$ and the scheme of this method is

$$F_{k+1} = (E + hP(t_k))F_k$$

where E is the $r \times r$ identity matrix.

In case that the initial value vector F_0 contains an error, the error may generate a blow-up solution $\tilde{F}$ under the Situation 1 and we cannot obtain the true solution.

Let N be a suitable natural number and put

$$Q = Q(N-1,h)Q(N-2,h)\cdots Q(1,h)Q(0,h) \tag{6}$$

We call Q the *matrix factorial* of $Q(k,h)$. The matrix Q approximates the fundamental solution matrix of the ODE. We assume the eigenvalues of Q are positive real and distinct to simplify our presentation. The following heuristic algorithm avoids to get the blow-up solution.

Algorithm 1

1. *Obtain eigenvalues $\lambda_1 > \lambda_2 > \cdots > \lambda_r > 0$ of Q and the corresponding eigenvectors $v_1, \ldots, v_r$.*
2. *Let λ_m be the eigenvalue which is almost equal to 0.*
3. *Express the initial value vector F_0 containing errors in terms of v_i's as*
$$F_0 = f_1 v_1 + \cdots + f_r v_r, \quad f_i \in \mathbf{R} \tag{7}$$
4. *Choose a constant c such that $F_0' := c(f_m v_m + \cdots + f_r v_r)$ approximates F_0.*
5. *Determine F_N by $F_N = QF_0'$ with the new initial value vector F_0'.*

We call this algorithm the *defusing method* for initial value problem. This is a heuristic algorithm and the vector F_0' gives a better approximation of the initial value vector than F_0 in several examples and we can avoid the blow-up of the numerical solution.

Example 4 We set $t_0 = 0$, $h = 10^{-3}$, $N = 10 \times 10^3$ and use the 4-th order Runge-Kutta scheme. We have $\lambda_1 = 9.708 \times 10^9$, $v_1 = (-5.097, -159.919)^T$ and $\lambda_2 = 3.247 \times 10^{-7}$, $v_2 = (-5.09798, 37.1648136496805760375399714182094650\,86)^T = (a,b)$. Then, $m = 2$. We assume the 3 digits accuracy of the value $\mathrm{Ai}(0)$ as 0.355 and set $F_0' = (0.355, 0.355b/a)$. Then, the obtained value F_{5000} at $t = 5$ is $(0.000108088745179140, -0.000246853220440734)$. We have the following accurate value by Mathematica

[2]It was denoted by $Q(t_0 + kh, h)$ in the previous section. We denote $Q(t_0 + kh, h)$ by $Q(k,h)$ as long as no confusion arises.

```
In[1]:= N[AiryAi[5]]
Out[1]= 0.000108344
In[2]:= N[D[AiryAi[x],{x}] /. {x->5}]
Out[2]= -0.000247414
```

Note that 3 digits accuracy has been kept for the value Ai(5). On the other hand, we appy the 4th order Runge-Kutta method with $h = 10^{-3}$ for $F_0 = (0.355, -0.259)^T$, which has the accuracy of 3 digits. It gives the value at $t = 5$ as $(-0.147395, -0.322215)$, which is a completely wrong value, and the value at $t = 10$ as $(-102173, -320491)$, which is a blow-up solution.

This heuristic algorithm avoids the blow-up of the numerical solution. Moreover, when the numerical scheme gives a good approximate solution for the exact initial value, we can give an error estimate of the solution by our algorithm. Let $|\cdot|$ be the Eucledian norm.

Lemma 1 *Let $F(t)$ be the solution. When $|QF_0^{\rm true} - F(Nh)| < \delta$ holds, we have*

$$|QF_0' - F(Nh)| < |QF_0'| + |F(Nh)| + 2\delta \tag{8}$$

for any $F_0' \in \mathbf{R}^n$.

Proof. It is a consequence of the triangular inequality. In fact, we have

$$\begin{aligned}
& |QF_0' - F(Nh)| \\
= \;& |QF_0' - QF_0^{\rm true} + QF_0^{\rm true} - F(Nh)| \\
\leq \;& |QF_0' - QF_0^{\rm true}| + |QF_0^{\rm true} - F(Nh)| \\
\leq \;& |QF_0'| + |QF_0^{\rm true}| + \delta \\
\leq \;& |QF_0'| + |F(Nh)| + 2\delta
\end{aligned}$$

//

Under the Situation 1, $|F(Nh)|$ is small enough. Then, it follows from the Lemma that $|QF_0'|$ should be small. In this context, we can give an error estimate of our algorithm. However, our numerical experiments present that the algorithm shows a better behavior than this theoretical error estimate. Then, we would like to classify our defusing method as a heuristic method.

4 Sparse interpolation/extrapolation method

We solve the ODE (1) of rank r when (approximate) values of $f(t) = Z(t)$ at $t = p_1, p_2, \ldots, p_r$ are known. We call the points $(p_i, q_i), q_i = f(p_i)$ *data points*. In other words, we want to interpolate or extrapolate values of f from these r values. This is a generalization of the boundary value problems. The number of data points may be more than the rank or less than the rank of the ODE in the methods B and C in this section. A standard numerical method to find values of f on an interval $[t_s, t_e]$ is as follows. Devide $[t_s, t_e]$ into N intervals. Let t_i be $t_s + hi$ where $h = (t_e - t_s)/N$. We assume that the set of all p_j's are a subset of the set $\{t_i\}$. We denote by f_i the value of $f(t_i)$. We introduce the backward shift operator $\nabla f_i = f_i - f_{i-1}$. Then $\frac{\nabla^k f_i}{h^k}$ is approximately equal to $f^{(k)}(t_i)$ or $f^{(k)}(t_{i-1})$ or ... or $f^{(k)}(t_{i-k})$. For example, when $k = 1$, we have $f(t_i) - f(t_{i-1}) = f'(t_i)h + O(h^2)$ if we make the Taylor expansion of $f(t_{i-1}) = f(t_i - h)$ at $t = t_i$ and $f(t_i) - f(t_{i-1}) = f'(t_{i-1})h + O(h^2)$ if we make the Taylor expansion of $f(t_i) = f(t_{i-1} + h)$ at $t = t_{i-1}$. We have

$$\sum_{k=0}^{r} c_k(t_i) \frac{\nabla^k f_{i+s_k}}{h^k} \sim b(t_i), \quad 0 \leq s_k \leq k. \tag{9}$$

Here s_i is an integer to choose an approximation of $f^{(k)}(t_i)$. By assuming the left hand side and the right hand side are equal and giving values f_j for $t = p_j, 1 \leq j \leq r$, we have a system of linear equations of the form

$$A \begin{pmatrix} f_0 \\ f_1 \\ \cdot \\ \cdot \\ \cdot \\ f_N \end{pmatrix} = B \tag{10}$$

where A is a $(N+1)\times(N+1)$ matrix and B is a column vector of length $N+1$. Solving this equation, we obtain approximate values of $f(t_i)$. We call this method *the sparse interpolation/extrapolation method A for HGM.*

Example 5 We solve $Lf = 0$, (intro/2021-06-11-sparse-interp.ipynb) $L = (\partial_t - 1)(\partial_t^2 - t) = \partial_t^3 - \partial_2^2 - t\partial_t + t - 1$ on $t \in [-9, 0]$ with $f(-9) = \mathrm{Ai}(-9) \sim -0.0221337215473414$, $f(-4) = \mathrm{Ai}(-4) \sim -0.0702655329492895$, $f(0) = \mathrm{Ai}(0) \sim 0.355028053887817$ and $N = 100$. Then, we obtain approximate values of the Airy function $\mathrm{Ai}(t)$. The numerical result shows that we do not have a false solution in spite of the factor $\partial_t - 1$.

An alternative method to solve (10) when $b = 0$ is to construct r linearly indedent solutions (fundamental solutions) of (9) and to find coefficients to express the solution of the generalized boundary conditions as the linear combination of the fundamental solutions.

Let us introduce *the sparse interpolation/extrapolation method B for HGM.* This is also a standard method and using it in the HGM is very useful. A numerical integration for a function g can be expressed as

$$I_N(g) = \sum_{j=0}^{N} T_j g(t_j) \tag{11}$$

where $t_0 = t_2 < t_1 < \cdots < t_{N-1} < t_N = t_e$ and $T_j \in \mathbf{R}_{\geq 0}$. We fix a numerical integration method. For example, the trapezoidal method can be expressed as $h = \frac{t_e - t_s}{N}$, $t_i = t_s + hi$, $T_j = h$ for $1 \leq j < N$ and $T_0 = T_N = h/2$. We approximately expand the solution f by a given basis functions $\{e_k(t)\}$, $k = 0, 1, \ldots, M$ as

$$f(t) = \sum_{k=0}^{M} f_k e_k(t), \quad f_k \in \mathbf{R}. \tag{12}$$

We put $M + 1 = m$ to be compatible with the variable name in our program. Put this expression into $Lf = b$. We minimize the loss function, which is the numerical integration value of $\int_{t_s}^{t_e} |Lf(t) - b(t)|^2 dt$

$$\begin{aligned} \ell(\{f_k\}) &= \sum_{j=0}^{N} |(Lf)(t_j) - b(t_j)|^2 T_j \\ &= \sum_{j=0}^{N} \left| \sqrt{T_j} \sum_{k=0}^{M} f_k (Le_k)(t_j) - \sqrt{T_j} b(t_j) \right|^2 \end{aligned} \tag{13}$$

under the constraints at data points

$$\sum_{k=0}^{M} f_k \cdot e_k(p_i) = q_i, \quad i = 1, 2, \ldots, r. \tag{14}$$

This is a least mean square problem under constraints (see, e.g., [18]). Since the loss function (13) is defined by the numerical integration formula (11), we have the following estimate.

Lemma 2 *The norm $\int_{t_s}^{t_e} |Lf(t)-b(t)|^2 dt$ is bounded by $\ell(\{f_k\})+e_N(|Lf-b|^2)$ where $e_N(|Lf-b|^2)$ is the error of the numerical integration method.*

Solvers of the least mean square problem output the value of the loss function, then we can estimate the L^2 norm of $Lf-b$ by this Lemma.

We can also consider a least mean square problem with no constraints by the loss function

$$\tilde{\ell}(\{f_k\}) = \alpha\ell(\{f_k\}) + \beta \sum_{i=1}^{r} \left(\sum_{k=0}^{M} f_k \cdot e_k(p_i) - q_i \right)^2 + \gamma \sum_{i=0}^{N} f_i^2. \tag{15}$$

Here α, β, γ are hyperparameters for the optimization. In particular, $\gamma > 0$ is used to avoid an overfitting.

Example 6 We solve the differential equation of Example 5. The method B does not work well on the interval $[-9, 0]$ because of the oscillation of the solution, but it works on a smaller interval $[-4, 0]$. We use Chebyshef polynomials on $[-4, 0]$ upto degree 9 as basis functions and $h = 0.01$. Data points are $[[-4, -0.0702655329492895], [-3, -0.37881429], [-2, 0.22740743]]$. (intro/sib-intro.py) (asir-tmp/sib-intro.rr, trybintro3())

This method works well for some problems which are hard by standard methods. See the Example 12 as to an application of this method to a 4th order ODE. An application of this method to a problem on random matrices is given in Example 14.

Let us introduce *the sparse interpolation/extrapolation method C for HGM.* We use the basis functions $e_j(t)$ as in the method C. We construct an $m \times m$ symmetric matrix S of which (i, j) element is

$$\int_{t_s}^{t_e} (Le_i(t))(Le_j(t))dt \tag{16}$$

Let F be the column vector of length $m = M + 1$: $F = (f_0, f_1, \ldots, f_M)^T$. Then, we have

$$\int_{t_s}^{t_e} \left(L \sum_{j=0}^{M} f_j e_j(t) \right)^2 dt = F^T S F. \tag{17}$$

It follows from this idenity that when $\sum_{j=0}^{M} f_j e_j(t)$ is an approximate solution of $Lf = 0$, we may expect the value of the quadratic form $F^T S F$ is small. Let (p_k, q_k)'s be given data points of $(t, f(t))$. The method C finds the vector F by solving the quadratic programming problem of minimizing

$$\begin{aligned} & F^T S F + \sum_{k=1}^{r} \left(\sum_{j=0}^{M} f_j e_j(p_k) - q_k \right)^2 \\ = \quad & F^T S F + \sum_{k=1}^{r} \left(F^T S_k^2 F - 2q_k S_k F + q_k^2 \right) & (18) \\ = \quad & F^T S F + F^T (P_e^T P_e) F - 2P_e^T Q + Q^T Q & (19) \end{aligned}$$

where $S_k = \mathrm{diag}(e_0(p_k), \ldots, e_M(p_k))$ and

$$P_e = \begin{pmatrix} e_0(p_1) & \cdots & e_M(p_1) \\ e_0(p_2) & \cdots & e_M(p_2) \\ \cdot & \cdots & \cdot \\ e_0(p_r) & \cdots & e_M(p_r) \end{pmatrix}, \quad Q = (q_1, \ldots, q_r)^T \tag{20}$$

Or, we may solve the quadratic programming problem of minimizing $F^T SF$ under the constraints

$$\sum_{j=0}^{M} f_j e_j(p_k) - q_k = 0, \quad k = 1, \ldots, r. \tag{21}$$

The method C can also solve the problem of Example 6. (intro/sic-intro.py) (asir-tmp/sic-intro.rr, trycintro3())

5 Tests of the methods to the function H_n^k

Let n and k be positive integers. We define the function $H_n^k(x,y)$ by

$$\begin{aligned} H_n^k(x,y) &= \int_0^x t^k \exp(-t)\,{}_0F_1(;n;yt)dt \qquad (22)\\ &= \frac{\Gamma(n)}{\sqrt{\pi}\Gamma(n-1/2)} \int_{D(x)} t^k (1-s^2)^{n-3/2} \exp(-t-2s\sqrt{yt})dtds \qquad (23)\\ &\quad \text{where } D(x) = \{(t,s) \in [0,x] \times [-1,1]\} \end{aligned}$$

This function appears in studies of the outage probability of MIMO WiFi systems (see, e.g., [6]). We will compare the methods in the section 2 for this function on several systems. The function satisfies the following system of linear partial differential equations.

Proposition 1 ([6])

1. *The function $u = H_n^k(x,y)$ satisfies*

$$\begin{aligned} \{\theta_y(\theta_y + n - 1) + y(\theta_x - \theta_y - k - 1)\} \bullet u &= 0, \\ (\theta_x - \theta_y - k - 1 + x)\, \theta_x \bullet u &= 0. \end{aligned}$$

   where $\theta_x = x\frac{\partial}{\partial x}$, $\theta_y = y\frac{\partial}{\partial y}$. The holonomic rank of this system is 4.

2. *The function u is a solution of the following ODE with respect to y.*

$$y^2\partial_y^4 + (-y+2n+2)y\partial_y^3 + (-yx+(-k-n-3)y+n(n+1))\partial_y^2 + ((y-n)x - n(k+2))\partial_y + (k+1)x \tag{24}$$

(Prog_paper/19-a19-n-pf.rr)

When $y \to +\infty$, solutions of the system has the following asymptotic behavior. It is shown by the `DEtools[formal_sol]` function of Maple [15]. (Hkn/a4-f2.ml)

$$\begin{aligned} h_1 &= (xy)^{-1/2(1/2+n)} \exp(-2(xy)^{1/2})(1 + O(1/y^{1/2})), \\ h_2 &= y^{-k-1}(1 + O(1/y)), \\ h_3 &= (xy)^{-1/2(1/2+n)} \exp(2(xy)^{1/2})(1 + O(1/y^{1/2})), \\ h_4 &= y^{1-n+k} \exp(y)(1 + O(1/y)), \end{aligned}$$

Which is the asymptotic behavior of the function $H_n^k(x,y)$ when x is fixed? We compare the value of h_4 and the value by a numerical integration in Mathematica[3]. The integration by Mathematica is done as follows.

```
--> hh[k_,n_,x_,y_]:=NIntegrate[t^k*Exp[-t]*HypergeometricPFQ[{},{n},t*y],{t,0,x}];
--> hh[10,1,1,1000]
```

[3]The quality of `HypergeometricPFQ` of mathematica is extremely high. However, the method to evaluate hypergeometric functions in Mathematica is still a black box. It is not easy to give a numerical evaluator of hypergeometric functions which matches to Mathematica in all ranges of parameters and independent variables.

y	Ratio
1000	7.36595030875893e-452
2000	2.64621603289928e-881
3000	2.67723893601667e-1311

where

$$\text{Ratio} = (H_1^{10}(1/2, y))/(y^{1-n+k}\exp(y)). \tag{25}$$

This computational experiments suggest that H_n^k is expressed by h_1, h_2, h_3 without the dominant component h_4 as explained in (2).

Example 7 We approximate $H_1^{10}(1, y)$ by h_3 of the asymptotic series terms truncated by $O(y^{-3})$. (Hkn/2021-06-06-asymp.rr) Let y_s be a sufficiently large number. We determine the constant C_s by $H_1^{10}(1, y_s)/h_3(y_s)$. Then we have the following data of relative errors $r_e(y) = \frac{h_3(y) - H_1^{10}(1,y)}{H_1^{10}(1,y)}$.

y_s	$r_e(y_s + 10)$	$r_e(y_s + 90)$
10^2	0.0763	0.259
10^4	5.7×10^{-10}	5.09×10^{-9}

The data tells that the defusing by series around $y = +\infty$ works well near $y = 10^4$, but we need to use other methods near $y = 10^2$. The sparse interpolation/extrapolation method B in the Example 12 is a more refined variation of this method.

Let us apply methods presented in the Section 2 to evaluate $H_1^{10}(1, y)$.

Example 8 We apply the implicit Runge-Kutta method to $H_1^{10}(1, y)$ which is a solution of the ODE (24). We translate the ODE into a system of ODE for U by $U = (u, u', u'', u''')^T$ and derive an ODE for $F = U\exp(-y)y^{-(1-n+k)}$ This is the Gauge transformation by the exponential part of h_4 so that the solutions keep bounded values. Then, the column vector function F satisfies $F' = PF$ where

$$P = \begin{pmatrix} \frac{-y-k+n-1}{y} & 1 & 0 & 0 \\ 0 & \frac{-y-k+n-1}{y} & 1 & 0 \\ 0 & 0 & \frac{-y-k+n-1}{y} & 1 \\ \frac{(-k-1)x}{y^2} & \frac{(-y+n)x+nk+2n}{y^2} & \frac{yx+(k+n+3)y-n^2-n}{y^2} & \frac{-k-n-3}{y} \end{pmatrix}. \tag{26}$$

The following data is obtained by the implicit Runge-Kutta method (Gauss method of order 10, [10, IV.5]) implemented in Mathematica [16]. The Figure 2 presents the value of $\log H_1^{10}(1, y)$ (the almost straight graph), the value by the implicit Runge-Kutta method, and the value by `NDSolve` with the `ExplicitRungeKutta` option (the graph of yellow green). (Hkn/2021-05-17-i-RK.nb) The initial value vector is evaluated at $y = 1$ by the numerical integration with the accuracy about 16 digits and the initial step size is 10^{-3}. The implicit Runge-Kutta method works upto about $y = 25$. This interval $[1, 25]$ is larger than the valid interval $[1, 4.5]$ of the `ExplicitRungeKutta` method of `NDSolve`. Why the implicit method works well? By virtue of the Gauge transformation, the maximal eigenvalue of the constant part of the coefficient matrix is close to 0. Then, it seems that we have relatively larger valid interval by the implicit Runge-Kutta method (see, e.g., [10, IV.3]).

We note that the implicit Runge-Kutta method is used in a large scale computation with the spectral diferred correction (SDC), see, e.g., [28], [20]. It is a method to accelerate the implicit Runge-Kutta method by making a preconditioning to iteration schemes for solving algebraic equations of the implicit method. We do not need this method for our relatively small rank ODE's in this paper. However, it will be useful for solving high rank ODE's in HGM. (num-ht3/pySDC/airy_playground.py, Airy_implicit.py)

When we can obtain exact initial values by the numerical integration (e.g., of (22)), it will be a good strategy to extrapolate the values in a short interval by solving the ODE by the implicit

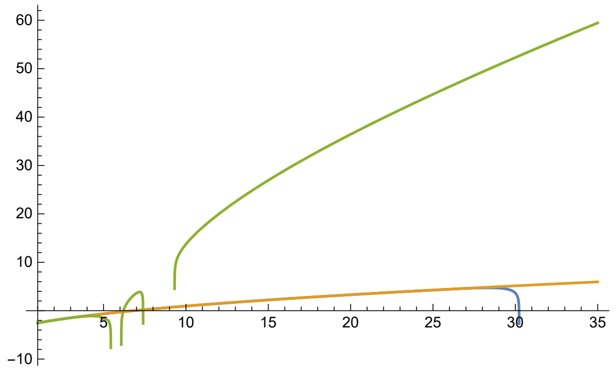

Figure 2: The Runge-Kutta(yellow green), the implicit Runge-Kutta method(blue), the exact value(orcher)

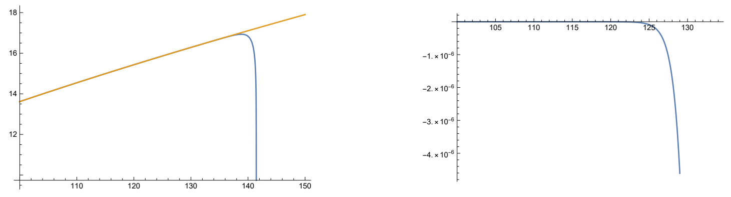

Figure 3: The implicit Runge-Kutta method vs the exact value. The right graph is the relative error $(H_r - H)/H$ where H is the exact value and H_r is the value by the implicit Runge-Kutta method.

Runge-Kutta method. The Figure 3 describes the value of $\log H_1^{10}(1,y)$ and the value by the implicit Runge-Kutta method on the interval $[100,150]$. The initial value vector is evaluated at $y=100$ by the numerical integration with the accuracy about 16 digits and the inital step size is 10^{-3}. It works upto about $y=130$.

Example 9 The spectral method in the approximation theory is a powerful method and we applied the ApproxFun package [2] implemented in Julia for solving the ODE (24). The outputs are remarkable. We solve the ODE on $[1,40]$ with the initial value at $y=1$ as $(u,u',u'',u''')(1) = (0.07810139136088563, 0.05096276584900834, 0.02050273784371611, 0.005887855153702640)$ where $u(y) = H_1^{10}(1,y)$. (Julia/hkn4i.jl) Then, the output value at $y=40$ is 815.0105773595695, which agrees with the value of numerical integration by Mathematica 815.0105773587113 upto 11 digits. It is also powerful to solve boundary value problems. We give the boundary values $(u(1),u(1.1),u(39.9),u(40)) = (0.0781014, 0.0833012, 803.121, 815.011)$, which is less accurate than the previous example. (Julia/hkn4b.jl) Then, the output value at $y=20$ is 27.021711397385513, which agrees with the value of numerical integration by Mathematica 27.021701160033859079 upto 6 digits. Note that we only give the boundary values in 6 digits accuracy. Although it works remarkably with a proper setting, we have had some troubles. For example, when we input the ODE $(24)\times y^2$, the program returns `NaN`. Another example is that it did not work for a first order system obtained from (24). (Julia/hkn3b.jl)

Example 10 We can try several solvers of ODE by `solve_ivp` on scipy/python [23]. (Hkn/2021-06-02-hkn.ipynb) We solve the initial value problem of (26) on $y \in [1,30]$ with the absolute tolerance 0 and relative tolerance 10^{-3} [4]. Note that the default values of the absolute and relative tolerances does not work well. Here is a table of y's for which the relative error becomes more than 0.3. Refer to [23] on details on the following methods. (Hkn/2021-06-08-hkn-rerr2.ipynb)

[4]`atol=0, rtol=1e-3`

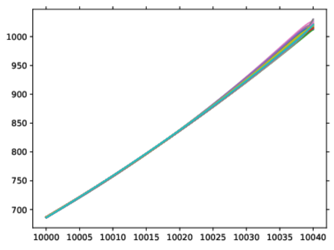
Figure 4: Solving by the sparse interpolation A

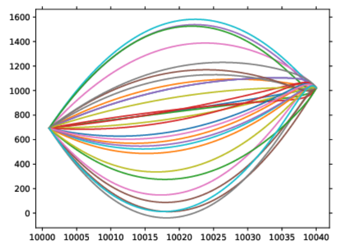
Figure 5: Solving the boundary value problem with random errors

Method	Failure y (relative error > 0.3)
RK45	21.5
Radau(Implicit Runge-Kutta)	26.1
BDF	16.8
LSODA(Adams/BDF on ODEPACK)	20.0

We can also try the deSolve package in R [7]. Some codes are common (e.g., FOTRAN package ODEPACK) with the python package and results are analogous with above. (Hkn/hkn.r)

Example 11 We apply the sparse interpolation/extrapolation method A. In other words, we solve a generalized boundary value problem of the ODE (24) on $y \in [10^4, 10^4 + 40]$. (Hkn/hkn-10-1-10000si-random.py) We give generalized boundary values of $u(y) = H_n^k(1, y) \times 10^{-80}$ at $p_1 = 10^4, p_2 = 10^4 + 1333h, p_3 = 10^4 + 2000h, p_4 = 10^4 + 4000h = 10^4 + 40$ where $h = 0.01$. We approximate derivatives as

$$\begin{aligned} u^{(1)}(y) &= \frac{1}{h}\left(u(y) - u(y-h)\right) \\ u^{(2)}(y) &= \frac{1}{h^2}\left(u(y+h) - 2u(y) + u(y-h)\right) \\ u^{(3)}(y) &= \frac{1}{h^3}\left(u(y+h) - 3u(y) + 3u(y-h) - u(y-2h)\right) \\ u^{(4)}(y) &= \frac{1}{h^4}\left(u(y+2h) - 4u(y+h) + 6u(y) - 4u(y-h) + u(y-2h)\right) \end{aligned} \tag{27}$$

and solve the linear equation (10). The linear equation is solved by the function `linsolv` in the scipy package [22]. The condition number of the matrix A becomes about 2.6×10^{13}. We give random errors by assuming the significant digits of the given boundary values are 3. The Figure 4 is a graph of the solutions of 30 tries of random errors. It works well. We note that the robustness with respect to random errors depends on a choice of p_i's. (Hkn/hkn-10-1-10000bv-random.py) For example, we change p_2 to $p_1 + h$ and p_3 to $p_4 - h$. This stands for solving the boundary value problem with the boundary values of f and f' at the boundary p_1 and p_4. We solve the linear equation with random errors. We can see that errors are magnified in middle from the Figure 5.

Although the boundary value problem with exact boundary values of $H_n^k(1, y)$ and its first derivatives can be solved nicely on $[1, 40]$, application of this method on $[1, 40]$ with internal reference points is not good. (Hkn/hkn-10-1-1-40-si.py, Hkn/2021-06-11-sparse-interp.ipynb) We take the four points as $1, 13.99675, 20.5, 40$ and $h = 0.00975$ and apply the method. It has an acceptable relative error 33.23 in the interval $[1, 6]$ and the relative error becomes acceptable smaller errors out of this interval.

(Hkn/2021-06-09-riccati-Hkn-rtol.ipynb) A clever method to solve boundary value problems is to use the Riccati transformation [8, p.1059] (see also [1, p.149]). We implemented this method with `solve_ivp` on scipy/python. It works for a small interval, e.g., $y \in [1, 2]$. But, it fails on $y \in [1, 10]$. The numerical solution of the associated Riccati equation increases from 0 to -1×10^{65} by the Runge-Kutta method (`method=RK4` with `rtol=1e-13, atol=1e-10`) and the backward equation cannot be solved because of the overflow. We tried on a smaller interval $[1, 3]$ with the Adams/BDF method (`method=LSODA`). Evaluation of the Riccati equation did not stop in 8 minutes on the google colaboratory.

Example 12 We apply the sparse interpolation/extrapolation method B explained in Section 4. By utilizing the local solution expansion at the infinity of the ODE (24), we use the set of the 4 functions

$$e_j(t) = t^{-3/4} \exp(2t^{1/2}) t^{-j/2}, \quad j = 0, 1, 2, 3, \; t = y \tag{28}$$

as the basis for (12). We will approximate the solution on $[t_s, t_e]$. The p_i's in the constraint (14) are

$$p_0 = t_s, p_1 = t_s + 5, p_2 = t_s + 10, \ldots, p_k = t_e - 1.$$

We do not use the constraint and add the difference of the approximate value and the true value to the the loss function as

$$\sum_{j=0}^{k} \left((\text{approximate value at } p_j) - H_1^{10}(1, p_j) \right)^2$$

We expect that this basis will give a good approximation when t_s and t_e are large. In fact, when $[t_s, t_e] = [1, 40]$, the maximum of the relative error is 5.47 by our implementation on `least_squares` in `scipy/python`. On the other hand, this basis gives a good approximation on $[20, 20 + 40]$ and $[10^4, 10^4 + 40]$. We give random errors to the value of $q_j = H_1^{10}(1, p_j)$ as $q_j \times (1 + O(10^{-3}))$. It is also robust to these random errors. (Hkn/hkn-10-1-10000sib-random.py Hkn/hkn-10-1-20-60sib-random.py)

$[t_s, t_e]$	$[20, 60]$	$[10^4, 10^4 + 40]$
max of relative errors with exact q_j	6.21×10^{-3}	2.67×10^{-12}
max of relative errors with 30 random errors	1.39×10^{-2}	4.07×10^{-3}

In summary, we should use the implicit Runge-Kutta method or the sparse interpolation/extrapolation method A when t_s and t_e are small and use the sparse interpolation/extrapolation method B when they become large for the problem H_n^k. An alternative choice when t_s and t_e are relatively small will be the following defusing method.

Example 13 We implement the defusing method in `tk_ode_by_mpfr.rr` [5] for the Risa/Asir [21]. It generates C codes utilizing the MPFR [17] for bigfloat and the GSL [9] for eigenvalues and eigenvectors. We apply the defusing method for initial value problem to $H_1^{10}(1, y)$ which is a solution of the ODE (24). We use the step size $h = 10^{-3}$ and the bigfloat of 30 digits of accuracy. (Hkn/defusing/tmp-test.c, tmp-proj.c) (Generated by asir-tmp/tk-ode-assert.rr, tk-ode-assert.hkn1(), tk-ode-assert.hkn2()) The Figure 6 shows that the adaptive Runge-Kutta method of GSL [9] fails before y becomes 30. (Hkn/defusing/a26-y.c) The Figure 7 presents the relative error of values by the defusing method and exact values. It shows that the defusing method works even when $y = 10^3$.

6 Tests of some methods to the function $E[\chi(M_t)]$

We consider the integral studied in [26]

$$\begin{aligned} \mathrm{E}[\chi(M_t)] &= F(s_1, s_2, m_{11}, m_{21}, m_{22}; t) \\ &= \frac{1}{2} \int_t^{\infty} d\sigma \int_{-\infty}^{\infty} db \int_0^{2\pi} d\theta \int_0^{2\pi} d\phi (\sigma^2 - b^2) \frac{s_1 s_2}{(2\pi)^2} \exp\left\{ -\frac{1}{2} R \right\}, \end{aligned} \tag{29}$$

[5] `http://www.math.kobe-u.ac.jp/OpenXM/Current/doc/asir-contrib/ja/tk_ode_by_mpfr-html/tk_ode_by_mpfr-ja.html`. Todo, English manual.

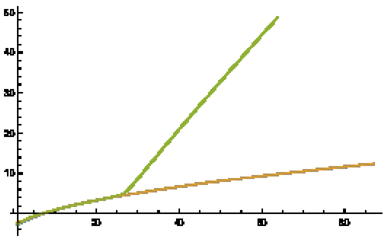

Figure 6: $\log H_1^{10}(1,y)$. Exact value (by numerical integration) and the value by our defusing method agree. The adaptive Runge-Kutta method with the initial relative error 10^{-20} (upper curve) does not agree with the exact value when y is larger than about 25.

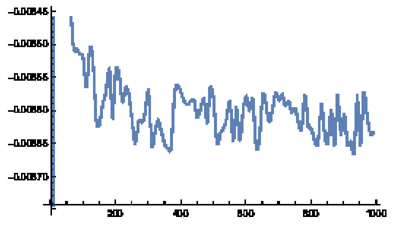

Figure 7: The relative error of $H_1^{10}(1,y)$ of our defusing method. The relative error is defined as $(H_d - H)/H$ where H_d is the value by the defusing method and H is the exact value.

where

$$\begin{aligned} R &= s_1 \left(b \sin\theta \sin\phi + \sigma\cos\theta\cos\phi - m_{11}\right)^2 + s_2 \left(\sigma\sin\theta\cos\phi - b\cos\theta\sin\phi - m_{21}\right)^2 \\ &\quad + s_1 \left(\sigma\cos\theta\sin\phi - b\sin\theta\cos\phi\right)^2 + s_2\left(b\cos\theta\cos\phi + \sigma\sin\theta\sin\phi - m_{22}\right)^2 . \end{aligned}$$

and

$$m_{11} = 1, m_{21} = 2, m_{22} = 3, s_1 = 10^3, s_2 = 10^2.$$

By virtue of the Euler characteristic method, the expectation of the Euler characteristic of a random manifold M_t $E[\chi(M_t)]$ approximates the probability that the maximal eigenvalue of 2×2 random matrices is less than t. See [26, Sec 3] for details.

The function $E[\chi(M_t)]$ is a solution of the rank 11 ODE $Lf = 0$ discussed in [26, Example 5]. The operator L is of the form

$$\left((-4.72\times 10^{-52} t^{29} + \cdots)\partial_t^{10} + \cdots + (-7.78\times 10^{-22} t^{35} + \cdots)\right)\partial_t \tag{30}$$

by multiplying a constant $10^{-31}/8.66$ [6] and ∂_t from the right to the operator given in `https://yzhang1616.github.io/ec1/ec1.html`. (or ec/tryb6/ann3.txt, variable ODE) Note that this ODE is of a form of the singular perturbation problem and it seems to be hard to solve initial value problems like the Runge-Kutta method. In [26], since the exact numerical integration of (29) is not easy, they use values of the Monte-Carlo simulation on $t \in [3.8, 3.81]$ to solve the ODE by power series. This series approximates the expectation upto $t = 3.8633$, but it does not when t is larger than it. See [26] for details. The sparse interpolation/extrapolation method B overcomes this difficulty.

Example 14 (ec/tryb6/2021-07-09-try6b-tmpb.py) (It is generated by asir-tmp/sib-yi5b.rr, tryb6()) The sparse extrapolation method B gives an approximate solution which has larger valid domain than the solution by the power series method. We use the basis functions

$$e_j(t) = (t - 3.8055)^j. \tag{31}$$

Let $p_i = 3.8 + i/1000$, $i = 0, 1, \ldots, 9$. The values q_i at p_i are

$$\begin{aligned} &[0.067160, 0.065485, 0.064732, 0.063315, 0.061814, \\ &\quad 0.060477, 0.059611, 0.058257, 0.057520, 0.055971] \end{aligned} \tag{32}$$

[6]This constant is chosen so that the maximal absolute value of the coefficients of f_k's in (13) of the sparse interpolation/extrapolation method B is 1 in Example 14.

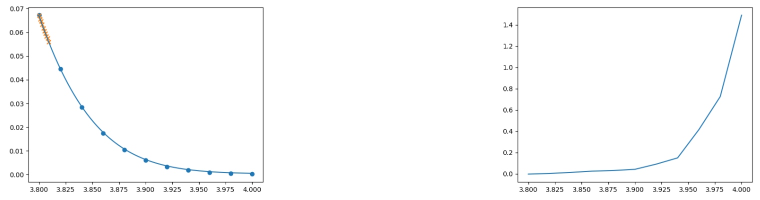

Figure 8: The graph of $F_{29}(t)$ and simulation values in the left and relative errors in the right. The data points are marked with 'x'.

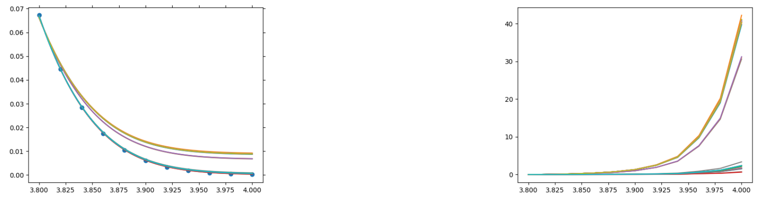

Figure 9: The graph of $F_{29}(t)$'s with random errors for data points and simulation values and relative errors in the right.

by a Monte-Carlo simulation. (ec/tryb6/yiex5b.r) (p_i, q_i)'s are data points for the sparse extrapolation method B. We divide the interval $[t_s, t_e] = [3.8, 4.0]$ into 200 segments and use the trapezoidal formula for $\ell(f)$. The optimal solution

$$\begin{aligned} & [f_0, f_1, \ldots, f_{29}] \\ = \; & [0.060145405402867516, -1.20074804549872, 9.438660716835336, \\ & -29.022737131194667, -46.606911486598264, 587.9594544508735, \ldots] \end{aligned}$$

can be found in about 18 seconds [7] by `least_squares` of the scipy package [24] on python. The value of the loss function $\tilde{\ell}$ is equal to 3.85×10^{-7}. The integral

$$\int_{t_s}^{t_e} (LF_{29})^2 \, dt \text{ where } F_{29}(t) = \sum_{j=0}^{29} f_j e_j(t) \qquad (33)$$

is equal to 2.58×10^{-4} and $|F_{29}(p_i) - q_i| < 3.18 \times 10^{-4}$ for all i. The left graph of Figure 8 is the graph of $F_{29}(t)$ and dots are approximate values of $E(\chi(M_t))$ by the Monte-Carlo simulation. The data points are marked with 'x'. The right graph of Figure 8 is the relative error of the values of F_{29} and the values by the simulation. Although the relative error is rather large, we disagree that the method is useless. For example, we have $F_{29}(3.935) = 0.00241$, then we can conclude the value of t satisfying $f(t) = 0.001$ may be found in the domain $t > 3.935$ assuming that the relative error is less than 1.4.

Example 15 (ec/tryb5/yi5b-random.py) We give relative errors less than 10^{-3} for q_k of the data points (p_k, q_k). We use the same scheme as Example 14. We execute 30 tries and obtain the results in Figure 9. Relative errors may be more than 40 at far points from the data points. We conclude that we should give accurate values of data points to extrapolate by the sparse extrapolation method B for this problem.

We expected that the sparse interpolation/extrapolation method C works for this problem, too. However, it does not seem to work well in real implementations on the double arithmetic.

[7]Debian 10.2 with Intel(R) Xeon(R) CPU E5-4650 2.70GHz

In fact, the cvxopt package [25] in python returns `ArithmeticError:5` in our implementation of the method C. (ec/tryb6/2021-07-14-tryc6-tmpb-modifed.py) We expect that an implementation of the quadratic programming with the bigfloat will solve this problem.

Acknowledgments. The first and the second authors are supported by the JST CREST Grant Number JP19209317. The first author is supported by the JSPS KAKENHI Grant Number 21K03270. The thrid author is supported by XJTLU Research Development Funding RDF-20-01-12, NSFC Young Scientist Fund No. 12101506, and Natural Science Foundation of the Jiangsu Higher Education Institutions of China Programme-General Programme No. 21KJB110032.

References

[1] F. S. Acton, Numerical Methods that Work, 1990, Mathematical Association of America.

[2] `ApproxFun.jl`, Julia package for function approximation,
`https://github.com/JuliaApproximation/ApproxFun.jl`,
`https://juliaapproximation.github.io/ApproxFun.jl/latest/`.

[3] U.M.Ascher , R.M.M.Mattheij, R.D.Russel, Numerical Solution of Boundary Value Problems for Ordinary Differential Equations, 1987, SIAM.

[4] M. Barkatou, An algorithm to Compute the Exponential Part of a Formal Fundamental Matrix Solution of a Linear Differential System, Applicable Algebra in Engineering, Communication, and Computing 8 (1997), 1–23.

[5] S. Chevillard, M. Mezzarobba, Multiple-precision evaluation of the Airy Ai function with reduced cancellation, arxiv:1212.4731.

[6] F.H.Danufane, K.Ohara, N.Takayama, C.Siriteanu, Holonomic Gradient Method-Based CDF Evaluation for the Largest Eigenvalue of a Complex Noncentral Wishart Matrix.
`https://arxiv.org/abs/1707.02564`

[7] DeSolve (R-package), Solvers for Initial Value Problems of Differential Equations,
`http://desolve.r-forge.r-project.org/`

[8] L. Dieci, M. Osborne, R. Russell, A Riccati Transformation Method for Solving Linear BFPs I, SIAM Journal on Numerical Analysis 25 (1988), 1055–1073.

[9] GSL, GNU scientific library,
`https://www.gnu.org/software/gsl/`

[10] E.Hailer, S.P.Norsett, G.Wanner, Solving Ordinary Differential Equations I, II. Springer, 1993.

[11] H.Hashiguchi, Y.Numata, N.Takayama, A.Takemura, Holonomic gradient method for the distribution function of the largest root of a Wishart matrix, Journal of Multivariate Analysis, 117, (2013) 296-312.

[12] References for HGM, `http://www.math.kobe-u.ac.jp/OpenXM/Math/hgm/ref-hgm.html`

[13] T. Hibi and et al, Gröbner Bases: Statistics and Software Systems, 2013, Springer

[14] M. van Hoeij, Formal Solutions and Factorization of Differential Operators with Power Series Coefficients, Journal of Symbolic Computation 24 (1997), 1–30.

[15] DEtools in Maple,
`https://www.maplesoft.com/support/help/maple/view.aspx?path=DEtools`

[16] `NDSolve` in Mathematica,
`https://reference.wolfram.com/language/tutorial/NDSolveImplicitRungeKutta.html`.

[17] The GNU MPFR library, `https://www.mpfr.org/`

[18] J.Nocedal, S.Wright, Numerical Optimization, 2006, Springer.

[19] S. Olver, A. Townsend, A fast and well-conditioned spectral method, SIAM Review 55 (2013), 462489.

[20] pySDC project, a Python implementation of the spectral diferred correction,
`https://parallel-in-time.org/pySDC/`

[21] Risa/Asir, a Computer algebra system, `http://www.openxm.org`

[22] `scipy.linalg.solve`
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve.html#scipy.linalg.solve`

[23] `scipy.integrate.solve_ivp`,
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html`

[24] `scipy.optimize.least_squares`,
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html`

[25] `cvxopt.solver.qp`,
`https://cvxopt.org/userguide/coneprog.html#quadratic-programming`

[26] N.Takayama, L.Jiu, S.Kuriki, Y.Zhang, Computations of the Expected Euler Characteristic for the Largest Eigenvalue of a Real Wishart Matrix, Journal of Multivariate Analysis 179 (2020), 104642.

[27] L. N. Trefethen, Approximation Theory and Approximation Practice, 2020, SIAM.

[28] M.Winkel, R.Speck, D.Ruprecht, A high-order Boris integrator, Journal of Computational Physics 295 (2015), 456–474.