`Tau Toolbox` Technical Report **TR-1**
available at `https://cmup.fc.up.pt/tautoolbox/`

Tau Toolbox

for the solution of
integro-differential problems

Tau method in `Tau Toolbox` : the basics

`Tau Toolbox` 0.91.0:
J. A. O. Matos, J. M. A. Matos, P. B. Vasconcelos

`Tau Toolbox` 0.10:
J. M. A. Matos, M. Trindade, P. B. Vasconcelos

Last update: May 2024 (`Tau Toolbox` 0.91.0)
Previous updates: September 2017 (`Tau Toolbox` 0.10)

Tau spectral method: an introduction

J. A. O. Matos
CMUP and University of Porto - Faculty of Economics
J. M. A. Matos
CMUP and Polytechnic of Porto - School of Engineering
P. B. Vasconcelos
CMUP and University of Porto - Faculty of Economics

Abstract

This report introduces the Lanczos' Tau method to compute a polynomial approximate solution of differential problems. Also it explains the first steps on how to use the `Tau Toolbox` to solve the above mentioned problem.

1 Introduction

The Tau method, introduced by Cornelius Lanczos [2], is a spectral method originally developed to compute a polynomial that approximates the solution of a linear ordinary differential problem with polynomial coefficients. The method has been used since then and extended to problems with non-polynomial coefficients, nonlinear differential equations, partial differential equations, among others. This widespread was only possible from the pioneering work of Ortiz [10] with the introduction of an algebraic formulation of the method.

The `Tau Toolbox` is a project to aggregate all these contributions, to enhance the use of the method by developing more stable algorithms and to offer efficient implementations of its algebraic formulation. It is able to solve various integro-differential problems, linear and nonlinear, with initial and/or boundary or others conditions, working with the most common polynomial orthogonal bases.

Non-expert can now profit from this spectral method and its solutions properties. On the other hand, experts can easily analyze new problems by exploring the large set of building block functions provided.

2 Overview of the tau method

In what follows, we assume that the solution of the differential problem to be solved can be expressed by an orthogonal polynomial series development. Furthermore, the tau method is introduced considering linear differential equations with polynomial coefficients.

In the Tau method sense, we get an $(n-1)^{th}$ degree polynomial approximation y_n to the differential problem's solution y by imposing that y_n solves exactly the differential problem with a polynomial perturbation term τ_n in the differential equation (or system of differential equations). To achieve good minimization properties for the error, τ_n is projected onto an orthogonal polynomial basis.

Several studies applying the Tau method have been performed to approximate the solution of differential linear and nonlinear equations [8, 1, 3], partial differential equations [7, 9, 5] and integro-differential equations [11], among others. Nevertheless, in all these works the tau method is tuned for the approximation of specific problems.

The `Tau Toolbox` is based on the so called operational version of the tau method, introduced in [10]. The differential equation together with initial and/or boundary conditions, expressed in an orthogonal polynomial basis, are cast onto an infinite algebraic system of equations, relating the series coefficients. The approximate solution y_n, can be obtained solving a truncated algebraic linear system of order n. This truncation gives rise to an approximate solution in the Tau sense, that is, the residual polynomial τ_n in the differential equation has the maximum approximation order for an $(n-1)^{th}$ degree polynomial.

If the differential equation is nonlinear, an usual procedure is to build a succession of linear equations in a linearization process (we refer the reader to [6]). Other generalizations are tackled on other `Tau Toolbox` Technical Reports.

If we are interested in solving an ordinary (or partial) differential equation to high accuracy on a simple domain and if the data problem are smooth, then spectral methods are usually the best tool. With respect to the more usual finite differences or finite elements, spectral methods allow to achieve high order accuracy [12]. Their important spectral properties is a motivation to improve the Tau method with new techniques, and new software applications.

2.1 Preliminaries

Given $\mathcal{P} = [P_0, P_1, P_2, \ldots]$, an orthogonal polynomial basis, the Tau method's operational formulation is based on two operational matrices $\mathsf{M}_\mathcal{P}$ and $\mathsf{N}_\mathcal{P}$ translating differential operations on $\mathcal{P}$ into algebraic operations. Representing by $x\mathcal{P} = [xP_0, xP_1, xP_2, \ldots]$ and $\frac{d}{dx}\mathcal{P} = [\frac{d}{dx}P_0, \frac{d}{dx}P_1, \frac{d}{dx}P_2, \ldots]$, then $\mathsf{M}_\mathcal{P}$ and $\mathsf{N}_\mathcal{P}$ are defined by

$$x\mathcal{P} = \mathcal{P}\mathsf{M}_\mathcal{P}, \quad \frac{d}{dx}\mathcal{P} = \mathcal{P}\mathsf{N}_\mathcal{P}.$$

By linearity, we can extend those operations to Fourier series in orthogonal polynomials. Let $y = \sum_{k\geq 0} a_k P_k$ be a formal Fourier series in $\mathcal{P}$ and let $\mathsf{a}_\mathcal{P} = [a_0, a_1, \ldots]^T$ be it's coefficient's vector so that we can write $y = \mathcal{P}\mathsf{a}_\mathcal{P}$ and then formally $xy = \mathcal{P}\mathsf{M}_\mathcal{P}\mathsf{a}_\mathcal{P}$ and $\frac{d}{dx}y = \mathcal{P}\mathsf{N}_\mathcal{P}\mathsf{a}_\mathcal{P}$.

Tau method's classical references reports to differential operations acting on powers basis and Taylor series, translated by matrices

$$\mathsf{M}_\mathcal{X} = \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ & 1 & 0 & & \\ & & 1 & 0 & \\ & & & \ddots & \ddots \end{bmatrix} \quad \text{and} \quad \mathsf{N}_\mathcal{X} = \begin{bmatrix} 0 & 1 & & & & \\ & 0 & 2 & & & \\ & & 0 & 3 & & \\ & & & 0 & 4 & \\ & & & & \ddots & \ddots \end{bmatrix}.$$

Proposition 1 shows the basic procedure on how to change from the power basis $\mathcal{X}$ to an orthogonal polynomial basis $\mathcal{P}$.

Proposition 1. *Let $\mathcal{P} = [P_0, P_1, P_2, ...]$ be an orthogonal basis, $\mathsf{V}_\mathcal{P}$ the triangular matrix such that $\mathcal{P} = \mathcal{X}\mathsf{V}_\mathcal{P}$ and $\mathsf{a}_\mathcal{P} = \mathsf{V}_\mathcal{P}^{-1}\mathsf{a}_\mathcal{X}$. Then $y = \mathcal{P}\mathsf{a}_\mathcal{P}$, $xy = \mathcal{P}\mathsf{M}_\mathcal{P}\mathsf{a}_\mathcal{P}$ and $\frac{d}{dx}y = \mathcal{P}\mathsf{N}_\mathcal{P}\mathsf{a}_\mathcal{P}$, where*

$$\mathsf{M}_\mathcal{P} = \mathsf{V}_\mathcal{P}^{-1}\mathsf{M}_\mathcal{X}\mathsf{V}_\mathcal{P} \quad \text{and} \quad \mathsf{N}_\mathcal{P} = \mathsf{V}_\mathcal{P}^{-1}\mathsf{N}_\mathcal{X}\mathsf{V}_\mathcal{P}. \tag{1}$$

Proof. See [10] □

For theoretical purposes the similarity transformations (1) are established with no restrictions, since, for any orthogonal polynomial basis $\mathcal{P}$, each polynomial P_n has exact degree n, and so $\mathsf{V}_\mathcal{P}$ is regular since it is triangular with non null entries in the main diagonal. For implementation purposes, however, this matrix can be highly ill-conditioned and the computation of (1) must be avoided. Proposition 2 presents an equivalent procedure but numerically more stable to compute $\mathsf{M}_\mathcal{P}$ and $\mathsf{N}_\mathcal{P}$, based on recurrence relations.

For improved clarity, the subscripts denoting orthogonal polynomial basis $\mathcal{P}$ are omitted whenever it causes no ambiguity.

Proposition 2. *Let $\mathcal{P} = [P_0, P_1, P_2, \ldots]$ be an orthogonal basis satisfying the recurrence relation $xP_j = \alpha_j P_{j+1} + \beta_j P_j + \gamma_j P_{j-1}$, $j \geq 0$, $P_0 = 1$, $P_{-1} = 0$. The coefficients of $\mathsf{M} = \mathsf{M}_\mathcal{P} = [\mu_{ij}]$ can be obtained by*

$$\begin{cases} \mu_{j+1,j} = \alpha_{j-1}, \quad \mu_{j,j} = \beta_{j-1}, \quad \mu_{j,j+1} = \gamma_{j-1} \\ \mu_{i,j} = 0, \quad |i-j| > 1 \end{cases}, \ j = 1, 2, \ldots \tag{2}$$

and those of $\mathsf{N} = \mathsf{N}_{\mathcal{P}} = [\eta_{ij}]$ *by*

$$\begin{cases} \eta_{i,j+1} = \frac{1}{\alpha_j}\left[\alpha_{i-1}\eta_{i-1,j} + (\beta_i - \beta_j)\eta_{i,j} + \gamma_{i+1}\eta_{i-1,j} - \gamma_j\eta_{i,j-1}\right],\ i = 0,\dots,j-1 \\ \eta_{j,j+1} = \frac{1}{\alpha_j}(\alpha_{j-1}\eta_{j,j-1} + 1) \\ \eta_{i,0} = 0,\ \eta_{0,1} = \frac{1}{\alpha_0} \end{cases} ,\ j = 1,2,\dots. \quad (3)$$

Proof. See [4] □

Moreover, numerical implementations for the computation of powers of M and N matrices are also provided.

2.2 The Tau method for linear differential problems

Let $\mathcal{D} = \sum_{k=0}^{\nu} p_k \frac{d}{dx^k}$ represent an order ν linear differential operator acting on $\mathcal{P}$, where $p_k = \sum_{i=0}^{n_k} p_{ki}x^i$ are polynomial coefficients, $n_k \in \mathbb{N}_0$, $p_{k,i} \in \mathbb{R}$ and let $f \in \mathcal{P}$ with finite degree λ. The problem

$$\begin{cases} \mathcal{D}y = f \\ c_i(y) = s_i,\ i = 1,\dots,\nu \end{cases} , \quad (4)$$

has a matrix representation given by

$$\mathsf{T}\mathsf{a} = \mathsf{b}, \text{ with } \mathsf{T} = \begin{bmatrix} \mathsf{C} \\ \mathsf{D} \end{bmatrix} \text{ and } \mathsf{b} = \begin{bmatrix} \mathsf{s} \\ \mathsf{f} \end{bmatrix}. \quad (5)$$

where

$$\mathsf{C} = [c_{ij}]_{\nu\times\infty} = c_i(P_{j-1}),\ i = 1,\dots,\nu,\ j = 1,\dots,$$

$$\mathsf{D} = \sum_{k=0}^{\nu} p_k(\mathsf{M})\mathsf{N}^k, \quad p_k(\mathsf{M}) = \sum_{i=0}^{n_k} p_{ki}\mathsf{M}^i,$$

$\mathsf{a} = [a_0, a_1, \dots]^T$ contains the coefficients of the series representation of y in $\mathcal{P}$, $\mathsf{s} = [s_1,\dots,s_\nu]^T$ the right-hand-side vector of the boundary conditions and $\mathsf{f} = [f_1,\dots,f_{n-\nu},0,0,\dots]^T$ the coefficients of the right-hand-side differential equations on the basis $\mathcal{P}$. Matrix T has an upper trapezoidal structure, with $h = \max\{n_k - k,\ k = 0,\dots,\nu\}$ nonnull subdiagonals. This can be derived from

$$\begin{aligned} h &= \sup\{\deg(\mathcal{D}P_n) - n,\ n = 0,1,\dots\} \\ &= \sup\left\{\deg\left(\sum_{k=0}^{\nu} p_k \frac{d}{dx^k} P_n\right) - n,\ n = 0,1,\dots\right\} \\ &= \sup\{\max\{n_k + n - k,\ k = 0,\dots,\nu\} - n,\ n = 0,1,\dots\} \\ &= \sup\{\max\{n_k - k,\ k = 0,\dots,\nu\} + n - n,\ n = 0,1,\dots\} \\ &= \max\{n_k - k,\ k = 0,\dots,\nu\} \end{aligned}$$

Choosing an integer n, an $(n-1)^{th}$ degree polynomial approximate solution $\mathsf{y}_n = \mathcal{P}_n\mathsf{a}_n$, expressed by the coefficients $\mathsf{a}_n = [a_{n,0}, a_{n,1},\dots,a_{n,n-1}]$ on $\mathcal{P}_n = [P_0, P_1,\dots,P_{n-1}]^T$, is obtained by truncating system (5) to its first n columns. The resulting system has $n+\nu+h$ equations. Restricting this system to its first n equations, a linear system

$$\mathsf{T}_n\mathsf{a}_n = \mathsf{b}_n \quad (6)$$

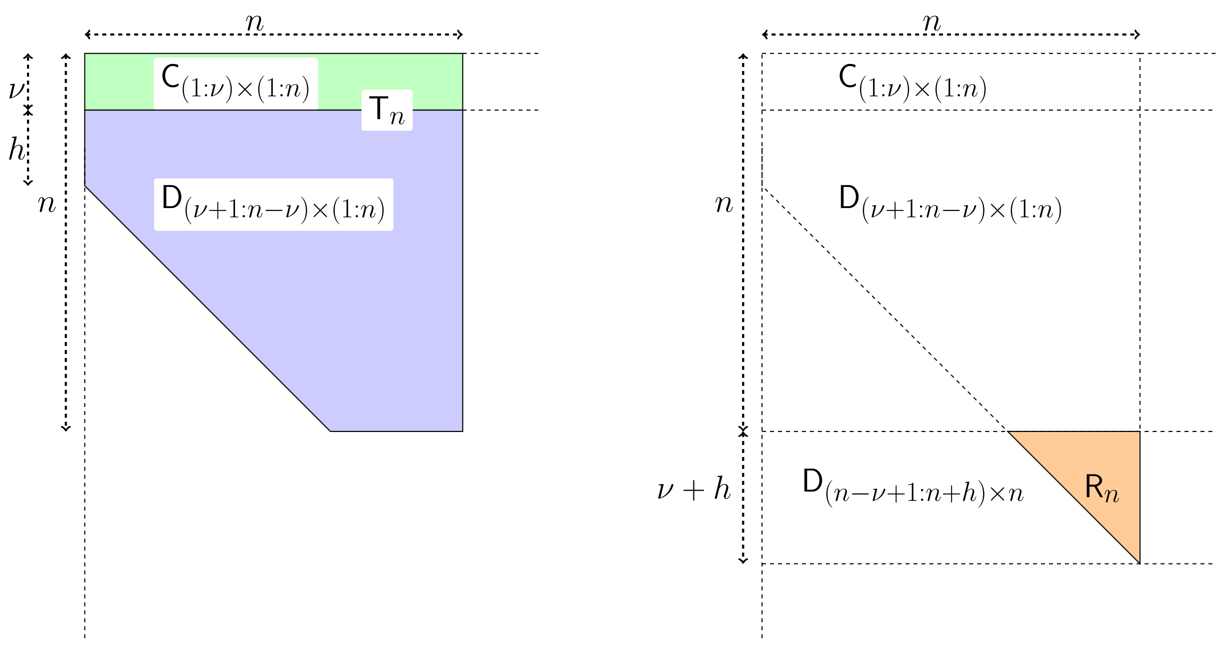
is obtained with T_n of dimension $n \times n$, which is equivalent to introduce a polynomial residual

$$\tau_n = \mathcal{D}y - \mathcal{D}y_n = \sum_{i=n+1}^{n+\nu+h} r_{n,i}\mathcal{P}_{i-\nu} \quad (7)$$

in the right-hand-side. Defining $\hat{\mathcal{P}}_n = [P_{n-\nu+1}, P_{n-\nu+2}, \dots, P_{n+h}]$ and $\mathsf{r}_n = [r_{n,n+1}, \dots, r_{n,n+\nu+h}]$, then equation (7) can be written as

$$\tau_n = \hat{\mathcal{P}}_n \mathsf{r}_n,\ \mathsf{r}_n = \mathsf{R}_n \mathsf{a}_n(n-\nu-h+1:n), \tag{8}$$

where R_n is the triangular block of matrix T_n, of order $\nu+h$, restricted to columns and rows $n+1, \dots, n+\nu+h$ (see Figure 1).



n
ν
h
n
$\mathsf{C}_{(1:\nu)\times(1:n)}$
T_n
$\mathsf{D}_{(\nu+1:n-\nu)\times(1:n)}$
n
n
$\mathsf{C}_{(1:\nu)\times(1:n)}$
$\mathsf{D}_{(\nu+1:n-\nu)\times(1:n)}$
$\nu+h$
$\mathsf{D}_{(n-\nu+1:n+h)\times n}$
R_n

(a) Matrix T_n on the truncated system (b) Matrix R_n on the truncated system

Figure 1: Matrix representations for the truncated system (6) and tau residual (8)

The approximate solution y_n obtained satisfies the perturbed system

$$\begin{cases} \mathcal{D}y_n = f + \tau_n \\ c_i(y_n) = s_i,\ i = 1, \dots, \nu \end{cases}. \tag{9}$$

3 Tau method on `Tau Toolbox`

In this section we will briefly explain how to work with `Tau Toolbox` to compute approximate solutions of differential systems of equations using the Tau method. In order to understand how the toolbox works internally we will now present some details of how the calculations are done although the toolbox can be used without the knowledge of the inner lower levels.

The `Tau Toolbox` is a MATLAB toolbox that uses object-oriented techniques in the programming design. It works with OCTAVE and a PYTHON version is also in development.

To begin using the toolbox, just set the path for the toolbox by running

```
>> tauPath
```

In order to simplify the naming of functions and to avoid name collisions with user's code all the functions and classes of the toolbox have a `tau.` prefix.

There are three classes that are the corner stones of the operation in Tau Toolbox are:

- `tau.polynomial` the data type the represents an orthogonal polynomial;
- `tau.settings` that hold all the settings used in the different methods;

- `tau.problem` the glue that holds together the different elements of the problems to be solved.

Let us take as an example the call `x = tau.polynomial` that creates the object `x`, a tau polynomial with support on a polynomial orthogonal basis; by default the Chebyshev of first kind:

```
>> x = tau.polynomial()

x =

  tau.polynomial object with properties:
          basis:     ChebyshevT
         degree:     1
         domain:     [-1, 1]
          coeff:     [2x1 double]
```

By default, the system creates a degree 1 polynomial in the chebyshev (first kind) orthogonal basis in [-1,1].

the `tau.settings` class helds together the different option elements that are used when working with the `Tau Toolbox` . The `basis` (e.g. Chebyshev of the first and second kind, Legendre, Laguerre among others), `domain` and `degree` can be set through `tau.settings` where the arguments form pairs with the property name first and its value later:

```
>> options = tau.settings('basis', 'LegendreP', 'domain', [0 1]);
>> x = tau.polynomial(options)

x =

  tau.polynomial object with properties:
          basis:     LegendreP
         degree:     1
         domain:     [0, 1]
          coeff:     [2x1 double]
```

Now if we write $x^2 - 4 * x^5$, since x is a `tau.polynomial`, the result is again a `tau.polynomial` object. The `coeff` property holds the coefficients of $x^2 - 4 * x^5$ in the selected basis.

```
>> x^2-4*x^5

ans =

  tau.polynomial object with properties:
          basis:     LegendreP
         degree:     5
         domain:     [0, 1]
          coeff:     [6x1 double]

>> ans.coeff

ans =

   -0.3333
   -0.9286
   -1.0238
   -0.5556
   -0.1429
   -0.0159
```

With these building blocks, that illustrate the inner working of `Tau Toolbox` , the scenario is set to tackle the solution of differential problems by the Tau method with ease. The following examples illustrate how to use the `Tau Toolbox` in the solution of linear differential problems.

Finally the class `tau.problem` allows to build integro-differential problem and prepare it to be solved via the `tau.solve` driver.

Example 1: a linear second order initial value problem

Using `Tau Toolbox` to approximate the solution of the following initial value problem

$$\begin{cases} y''(x) + y(x) = 0, & x \in]0,\ 5[\\ y(0) = 0, & y'(0) = 1 \end{cases}, \tag{10}$$

can be done using only two function calls:

`Tau Toolbox` code 1: basic level

```
% differential problem
equation = 'diff(y,2)+y = 0';
domain = [0 5];
conditions = {'y(0)=0'; 'y''(0)=1'};
problem = tau.problem(equation, domain, conditions);

% solution via tau method
yn = tau.solve(problem);
figure; plot(yn);

% compare approximation with the exact solution
u = linspace(yn);
figure; semilogy(u, abs(yn(u)-sin(u)), 'color', 'red');
```

- equation is the equation;
- domain is the integration domain;

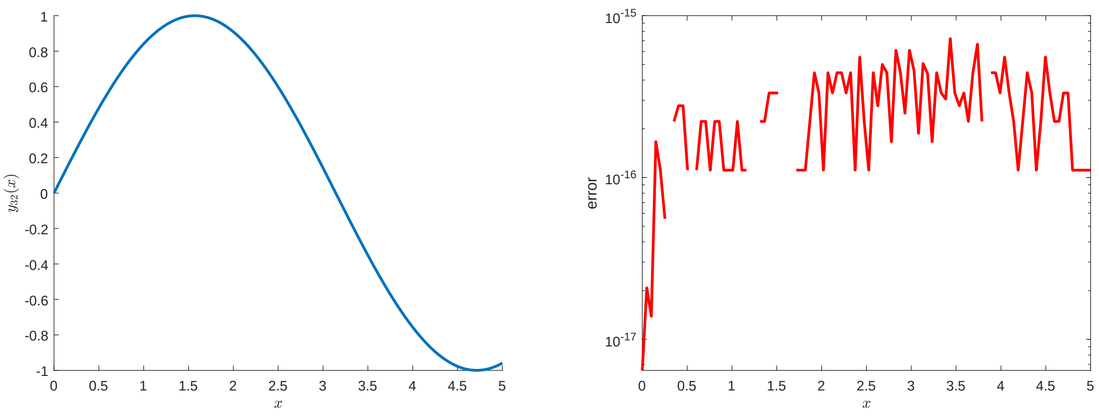
- conditions conditions and/or constraints that the variable satisfies;
- options a tau.settings object (that as the name says it is optional).

The `tau.problem` prepares all data to be processed by the solver and `tau.solve` computes the sought approximation: a 31th degree polynomial projected on the ChebyshevT basis (the default). The approximate solution `yn` is a `tau.polynomial` object with coefficients in `yn.coeff`:

```
yn =

  tau.polynomial object with properties:
          basis:      ChebyshevT
         degree:      31
         domain:      [0, 5]
          coeff:      [32x1 double]
```

Plotting the approximate solution is easy: either `tau.solve` is called without output, i.e. to call just `tau.solve(problem)`, or use the `plot` function that can handle the tau polynomial y_n. Figure 3 shows approximate solution $y_n(x) = \sum_{i=0}^{n-1} a_{n,i}T_i(x) = \mathcal{T}_n \mathsf{a}_n$, $a_n =$`yn.coeff`, and the true error $|y_n(x) - y(x)|$, $n = 32$.



(a) $y_{32}(x)$ (b) $|y_{32}(x) - y(x)|$

Figure 2: Approximate solution and error using a Chebyshev basis of dimension $n = 32$ with support on $[0, 5]$

Remark 1. Functions `plot`, `linspace` are `Tau Toolbox` functions that overload the ones from MATLAB in the case of one argument to be a `tau.polynomial` object. Also when doing `yn(u)` we are overloading `polyval` function.

Remark 2. The first derivative of y is written as `'y''(0)=1'` since the initial conditions is introduced as a string. An equivalent syntax is `'diff(y,0)=1'`.

Remark 3. Note that the output coefficients in vector `yn.coeff` are ordered from the lowest to the highest degree term.

Instead of using char arrays/strings to pass the arguments the same arguments can be provided using anonymous functions. At an intermediate level, the user can parameterize the `tau.solve` function by specifying input parameters and exploring the output ones. The following example illustrate these two features:

`Tau Toolbox` code 2: intermediate level

```
% differential problem
equation = @(x,y) diff(y,2)+y;
domain = [0 5];
conditions = @(y) {y(0); y(0)'-1};
options = tau.settings('basis', 'LegendreP', 'degree', 32);
problem = tau.problem(equation, domain, conditions, options);

% solution via tau method
[yn, info, residual, cauchy_error, tau_residual] = tau.solve(problem);
plot(tau_residual, 'color', 'magenta')
```

The output reveals `yn` and `tau_residual` as `tau.polynomial` objects, `info` provides information on the solution process and `Cauchy_error` the Cauchy error vector.

```
info =

  struct with fields:

      algorithm: 'gauss elimination'
           cond: 2.3813e+04
       residual: 6.9202e-16
    tauresidual: 9.2647e-31
    cauchyerror: 2.3529e-30
```

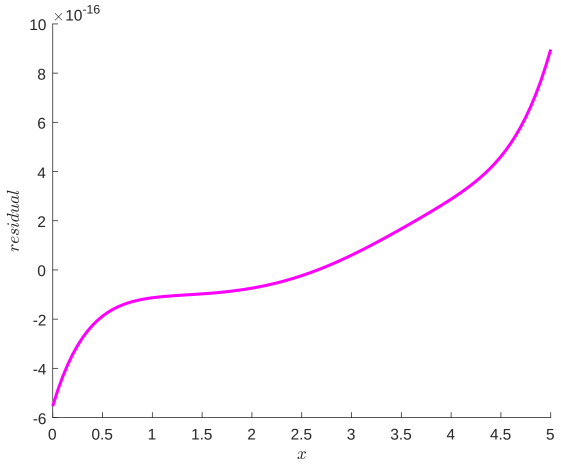


Figure 3: Tau residual $(\mathsf{D}(n-\nu+1:n+h,1:n)*\mathsf{a}_n)$ using a Legendre basis of dimension $n=32$ with support on $[0,5]$

More advanced `Tau Toolbox` users can detail their codes making use of more elaborate functions and explore new approaches. The matrix T_n and the vector b_n can be easily accessed using `Tau Toolbox` function `tau.getMatrix`, and the linear system $\mathsf{T}_n\mathsf{a}_n = \mathsf{b}_n$ can be solved by a specific solver. In the next illustration the Gaussian elimination provided by MATLAB is used.

Example 2: a linear third order boundary problem

Let us now solve the boundary value problem

$$\begin{cases}(x^2+1)y'''(x)-(x^2+3x)y''(x)+5xy'(x)-5y=60x^2-10 \quad x\in]-1,\ 1[\\ y(-1)=4, \quad y'(1)=2, \quad y''(0)=0\end{cases} \tag{11}$$

The problem can be solved with the usual syntax:

`Tau Toolbox` code 3: basic level

```
% differential problem
equation = '(x^2+1)*diff(y,3)-(x^2+3*x)*diff(y,2)+5*x*diff(y)-5*y = 60*x...
    ^2-10';
domain = [-1 1];
conditions = {'y(-1)=4'; 'y''(1)=2'; 'y''''(0)=0'};
options = tau.settings('degree', 5);
problem = tau.problem(equation, domain, conditions, options);

% solution via tau method
yn = tau.solve(problem);
```

The vector solution obtained `yn.coeff=` $[2,\ -2.375,\ 0,\ 0.3125,\ 0,\ 0.0625]$ corresponds to the Chebyshev form $y_6(x) = 2T_0(x) - \frac{19}{8}T_1(x) + \frac{5}{16}T_3(x) + \frac{1}{16}T_5(x) = 2 - 3x + x^5 = y(x)$ the exact solution (to see use `format rat`); that is with n large enough the exact solution is recovered whenever it is a polynomial.

To illustrate additional input choices available, the following code solves the same problem but using preconditioned preconditioned `gmres` to solve the involved linear system:

`Tau Toolbox` code 4: intermediate level

```
options = tau.settings('degree', 5, 'solver', 'gmres', 'milu', 'row');
problem = tau.problem(equation, domain, conditions, options);

% solution via tau method
yn = tau.solve(problem);
```

Example 3: a system of differential equations

In order to illustrate an application to a system of differential equations, the previous example is translated into a system of first order differential equations, as

$$\begin{cases} y_2(x) - y_1'(x) = 0 \\ y_3(x) - y_2'(x) = 0 \\ (x^2+1)y_3'(x) - (x^2+3x)y_2'(x) + 5xy_2(x) - 5y_1 = 60x^2 - 10 \\ y_1(-1) = 4, \quad y_2(1) = 2, \quad y_3(0) = 0 \end{cases} . \tag{12}$$

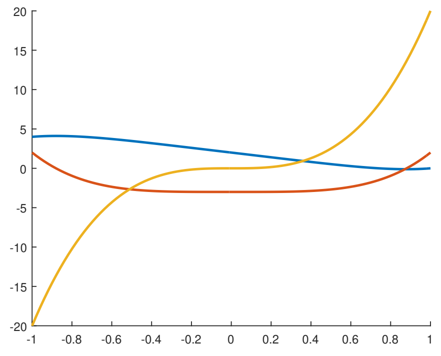
From the user perspective, `Tau Toolbox` `tau.solver` tackles systems of ode's similarly as single ode's:

`Tau Toolbox` code 5: basic level

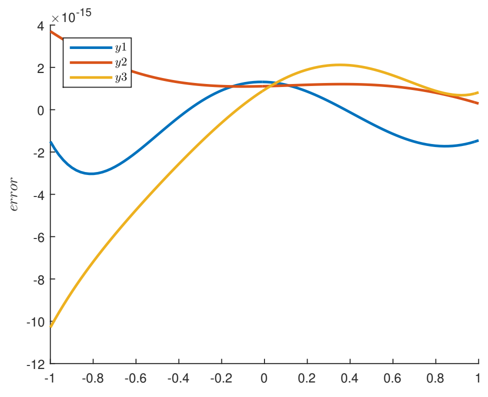
```
% differential problem
system = {'y2-diff(y1) = 0'; ...
          'y3-diff(y2) = 0'; ...
          '(x^2+1)*diff(y3)-(x^2+3*x)*diff(y2)+5*x*y2-5*y1 = 60*x^2-10'};
domain = [-1, 1];
conditions = {'y1(-1)=4';'y2(1)=2';'y3(0)=0'};
options = tau.settings('degree', 5);
problem = tau.problem(system, domain, conditions, options);

% solution via tau method and error
yn = tau.solve(problem);
ye = tau.polynomial({@(x) x.^5-3*x+2; @(x) 5*x.^4-3; @(x) 20*x.^3}, ...
    options);
figure; plot(yn), figure; plot(yn-ye);
```

and the solution and true error are drawn in the following



(a) Approximate solution



(b) True error

Figure 4: Approximate solution and true error

The differential problem is translated into an algebraic linear system $\begin{bmatrix} \mathsf{C} \\ \mathsf{D} \end{bmatrix} \mathsf{a} = \mathsf{b}$, where

$$\mathsf{D} = \begin{bmatrix} -\mathsf{N} & \mathsf{I} & \mathbf{0} \\ \mathbf{0} & -\mathsf{N} & \mathsf{I} \\ -5\mathsf{I} & 5\mathsf{MI} - (\mathsf{M}^2 + 3\mathsf{M})\mathsf{N} & (\mathsf{M}^2 + \mathsf{I})\mathsf{N} \end{bmatrix}$$

translates the differential operator and, since $\mathcal{T} = [T_0, T_1, \dots, T_{n-1}]$ then

$$\mathsf{C} = \begin{cases} c_{1,i} = T_i(-1) \\ c_{2,n+i} = T_i(1) \\ c_{3,2n+i} = T_i(0) \end{cases}, \quad i = 0, 1, \dots, n-1$$

translates the boundary conditions. The right-hand-side is also translated into the T basis: since $60x^2 - 10 = 30T_2(x) + 20T_0(x)$, then

$$\mathsf{b} = [\underbrace{4,\ 2,\ 0,}_{\nu=3}\ \underbrace{0,\ \dots,\ 0,}_{2(n-1)}\ 20,\ 0,\ 30,\ \underbrace{0,\ \dots,\ 0}_{n-4}]^T$$

.

`Tau Toolbox` functions can be used to construct all these parts. Namely, `tau.matrix` allows to get any of the matrices involved in the computations. A special `spy` function, called `tau.spy`, is useful to understand the structure of matrix T. A more detailed implementation is given in the following code excerpt

`Tau Toolbox` code 6: advanced level

```
% get matrix problem
T = tau.matrix('T', problem); b = tau.matrix('b', problem);
spy(T);

% solution
n = problem.options.n; nu = problem.nconditions; opts = problem.options;
yn = tau.polynomial(reshape(T\b, [n, nu]), opts);
```

4 Highlights

In this Technical Report:

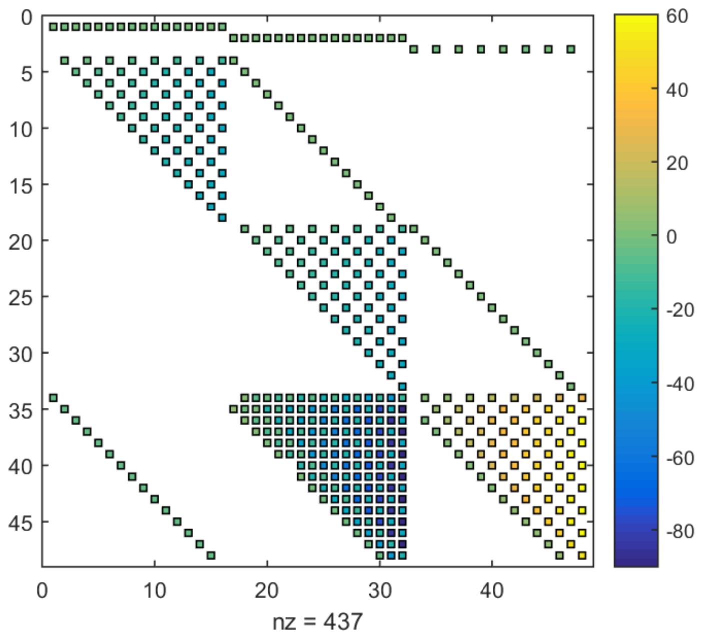


Figure 5: Spy of the T matrix for $n = 16$

- the Tau method is introduced;
- the first insights on the `Tau Toolbox` are provided
- the three level functions of `Tau Toolbox` are presented: basic, intermediate and advanced.

References

[1] M. R. Crisci and E. Russo. An extension of Ortiz recursive formulation of the Tau method to certain linear systems of ordinary differential equations. *Mathematics of computation*, pages 27–42, 1983.

[2] C. Lanczos. Trigonometric interpolation of empirical and analytical functions. *Journal of Mathematics and Physics*, 17(1):123–199, 1938.

[3] K. Liu and C. Pan. The automatic solution to systems of ordinary differential equations by the Tau method. *Computers & Mathematics with Applications*, 38(9):197–210, 1999.

[4] J. Matos, M. J. Rodrigues, J. C. de Matos, and M. Cruz. Avoiding similarity transformations in the operational Tau method. *arXiv:1703.00743*, pages 1–17, 2017.

[5] J. Matos, M. J. Rodrigues, and P. B. Vasconcelos. New implementation of the Tau method for pdes. *Journal of computational and applied mathematics*, 164:555–567, 2004.

[6] J. A. O. Matos, J. A. M. Matos, and P. B. Vasconcelos. Tau Toolbox for nonlinear ordinary differential systems. Technical Report TR5, CMUP and University of Porto, 2020.

[7] S. Namasivayam and E. L. Ortiz. Best approximation and the numerical solution of partial differential equations with the Tau method. *Portugaliae mathematica*, 40(1):97–119, 1981.

[8] E. Ortiz. On the numerical solution of nonlinear and functional differential equations with the Tau method. In *Numerical treatment of differential equations in applications*, pages 127–139. Springer, 1978.

[9] E. Ortiz and A. P. N. Dinh. Linear recursive schemes associated with some nonlinear partial differential equations in one dimension and the Tau method. *SIAM Journal on Mathematical Analysis*, 18(2):452–464, 1987.

[10] E. L. Ortiz and H. Samara. An operational approach to the Tau method for the numerical solution of non-linear differential equations. *Computing*, 27(1):15–25, 1981.

[11] J. Pour-Mahmoud, M. Rahimi-Ardabili, and S. Shahmorad. Numerical solution of the system of fredholm integro-differential equations by the Tau method. *Applied Mathematics and Computation*, 168(1):465–478, 2005.

[12] L. N. Trefethen. *Spectral methods in MATLAB*, volume 10. SIAM, 2000.