Solving BVPs using two-point Taylor formula by a symbolic software

F. Costabile, A. Napoli*

Dipartimento di Matematica, Università della Calabria via P. Bucci 30A, 87036 Rende (Cs), Italy

Abstract

For the numerical solution of boundary value problems a global method, based on two-point Taylor formula is proposed. A Mathematica package to compute approximate solution of BVPs is presented.

Numerical examples provide favorable comparisons with other existing methods, especially with respect to accuracy.

© 2006 Elsevier B.V. All rights reserved.

MSC: 02.60.Lj; 02.70.Jn

Keywords: Boundary value problem; Two-point Taylor series

1. Introduction

In the present paper we will be concerned with the global (analytical polynomial) approximation of the solution of the following two-point boundary-value problem (BVP):

\[
\begin{align*}
y''(x) &= f(x, y(x), y'(x)), & x \in [a, b], & f \in C^n([a, b] \times \mathbb{R}^2, \mathbb{R}), \\
y(a) &= \alpha, \\
y(b) &= \beta.
\end{align*}
\]

Problems of this kind arise in engineering and other branches of physical sciences. We assume that all the hypothesis for the existence and the uniqueness of the solution \(y(x)\) of (1.1) are satisfied.

One may solve it numerically by first reducing it to a system of first-order equations. But it is well known that several advantages (substantial gains in efficiency, lower storage requirements, etc.) are realized when the equations are treated in their original second-order form.

BVPs form an active area of research and there exists a large number of methods to compute their solution [1,9]. The “shooting technique” is discussed in [11]; finite difference methods are treated in [8,7]. High-order finite difference methods can be found in [12,4,2,3].

For global approximation of (1.1) there is not an extensive literature. For the special nonlinear case \(y''(x) = f(x, y)\), \(y(a) = \alpha, y(b) = \beta\), in [6] a class of collocation methods is proposed, based on interpolation by Lagrange polynomial.

* Corresponding author.
E-mail addresses: costabil@unical.it (F. Costabile), a.napoli@unical.it (A. Napoli).
They are global methods which give an approximation of the solution of problem (1.1) in polynomial form. This type of problems are treated also via variational techniques in [5].

In [13] wavelets and Galerkin approximation were used for the numerical integration of a linear two-point BVPs in the case of Dirichlet, Neumann and mixed boundary conditions.

In the present paper we will consider the general case (1.1). Under the assumption that the solution of (1.1) exists, we will apply to \( y(x) \) the two-point Taylor formula or Hermite interpolation in order to find an approximate solution in polynomial form. To compute the coefficients of the polynomial we use the initial conditions, the idea of collocation and a symbolic software. The principal strength of the method is the ease to get the analytic expression of the approximate polynomial of a fixed degree.

In Section 2 we will illustrate the method and in Section 3 we will study its error. Another of our goals is to show that this method can, from a numerical point of view, be simply and efficiently applied with the use of automatic codes. To illustrate this, in Section 4 we will present numerical results for some test problems. Then, in Section 5, the method will be extended to general \( n \)-order systems of first-order equations. Finally, in Appendices A and B we will give Mathematica codes which implement the above algorithms.

### 2. The method

First we give some results about the two-point Taylor formula, which is the base of the proposed method for the global approximation of problem (1.1).

**Theorem 1.** If \( f(x) \in C^{2n}[0,1] \), then [10]

\[
f(x) = P_{2n-1}[f](x) + R_n[f](x),
\]

where

\[
P_{2n-1}[f](x) = \sum_{j=0}^{n-1} \left[ C_{n,j}(x) f^{(j)}(0) + (-1)^j C_{n,j}(1-x) f^{(j)}(1) \right],
\]

\[
C_{n,j}(x) = \frac{x^j}{j!} (1-x)^n \sum_{k=0}^{n-j-1} \binom{n+k-1}{k} x^k,
\]

\[
R_n[f](x) = \frac{f^{(2n)}(\xi)}{(2n)!} x^n (1-x)^n, \quad \xi \in (0,1).
\]

Moreover, if \( f(x) \in C^t[0,1] \) with \( t \geq 2n \) and \( x \in [0,1] \), then

\[
|R_n[k][f](x)| \leq \frac{[x(1-x)]^{n-k}}{k!(2n-2k)!} \| f^{(2n)} \|_{\infty}.
\]

If \( y(x) \in C^{2n}[0,1] \) is the solution of problem (1.1), then, from Theorem 1, it can be written in the form:

\[
y(x) = P_{2n-1}[y](x) + R_n[y](x),
\]

where

\[
P_{2n-1}[y](x) = \sum_{j=0}^{n-1} \left[ C_{n,j}(x) y^{(j)}(0) + (-1)^j C_{n,j}(1-x) y^{(j)}(1) \right],
\]

\[
R_n[y](x) = \frac{y^{(2n)}(\xi)}{(2n)!} x^n (1-x)^n, \quad \xi \in (0,1).
\]
In order to find a global approximation for \( y(x) \) we assume
\[
y(x) \approx y_n(x) = P_{2n-1}[y](x). \tag{2.9}\]
The computation of \( P_{2n-1}[y](x) \) requires the values of the derivatives \( y^{(k)}(0) \) and \( y^{(k)}(1) \), \( k = 0, \ldots, n-1 \). These values can be obtained from the known initial conditions and the differential equation if we know \( y'(0) \) and \( y'(1) \). In order to approximate \( y'(0) \) and \( y'(1) \) we can use a collocation type technique or the well-known Taylor formula. In the first case, if \( x_1 \) and \( x_2 \) are two arbitrarily fixed points in \((0, 1)\), we solve the nonlinear, in general, system of two equations:
\[
\begin{align*}
P''_{2n-1}[y](x_1) - f(x_1, P_{2n-1}[y](x_1), P'_{2n-1}[y](x_1)) &= 0, \\
P''_{2n-1}[y](x_2) - f(x_2, P_{2n-1}[y](x_2), P'_{2n-1}[y](x_2)) &= 0 \tag{2.10}
\end{align*}
\]
in the unknowns \( y'(0) \) and \( y'(1) \). In the second case we solve the system:
\[
\begin{align*}
T_m(0, 1) &= \beta, \\
T_m(1, 0) &= \alpha, \tag{2.11}
\end{align*}
\]
where \( T_m(0, x) \) and \( T_m(1, x) \) are the Taylor polynomials of a fixed degree \( m \) at points, respectively, 0 and 1.
If \( \tilde{y}'(0) \) and \( \tilde{y}'(1) \) are the solutions of (2.10) or (2.11), by the use of the initial data, the values \( \tilde{y}^{(j)}(0) \) and \( \tilde{y}^{(j)}(1) \), \( j = 2, \ldots, n-1 \), can be computed. Thus, the polynomial
\[
\tilde{P}_{2n-1}[\tilde{y}](x) = \sum_{j=0}^{n-1} \left[ C_{n,j}(x) \tilde{y}^{(j)}(0) + (-1)^j C_{n,j}(1-x) \tilde{y}^{(j)}(1) \right] \tag{2.12}
\]
can be considered as a global approximate solution of problem (1.1)
\[
y_n(x) \approx \tilde{y}_n(x) = \tilde{P}_{2n-1}[y](x). \tag{2.13}\]
The computation of (2.13) can be easily done by a symbolic software. In Appendix A a Mathematica code is given.
All the above considerations can be extended on a general interval \([a, b]\), by a linear transformation of variable.

3. Error and stability

The analysis of the error
\[
y(x) - \tilde{y}_n(x) \tag{3.1}\]
for \( n > 1 \) is not easy (if \( n = 1 \) the error is given directly by (2.8)) owing to the computation of \( \tilde{y}'(0) \) and \( \tilde{y}'(1) \). However, in the hypothesis
\[
|\tilde{y}^{(j)}(h) - y^{(j)}(h)| \leq \varepsilon_j < \varepsilon, \quad j = 0, \ldots, n-1, \quad h = 0, 1,
\]
an a priori estimate of the global error (3.1) and an estimate of the error in \( P_{2n-1}[\tilde{y}](x) \) can be obtained.

**Theorem 2.** With the previous notations we have
\[
|P_{2n-1}[y](x) - \tilde{P}_{2n-1}[\tilde{y}](x)| \leq 2\varepsilon M_n, \tag{3.2}\]
where
\[
M_n = \max_{0 \leq x \leq 1} \sum_{j=1}^{n-1} C_{n,j}(x). \tag{3.3}\]

**Proof.**
\[
|P_{2n-1}[y](x) - \tilde{P}_{2n-1}[\tilde{y}](x)| \leq \sum_{j=1}^{n-1} C_{n,j}(x)|\varepsilon_j| + \sum_{j=1}^{n-1} C_{n,j}(1-x)|\varepsilon_j|.
\]
Since $C_{n,j}(x)$ and $C_{n,j}(1-x)$ are symmetric with respect to $x = \frac{1}{2}$, we have that

$$M_n = \max_{0 \leq x \leq 1} \sum_{j=1}^{n-1} C_{n,j}(1-x).$$

Hence the result follows. □

From numerical results we can see that $M_n$ increases very slowly as $n$ goes up. Now we can prove the following theorem.

**Theorem 3 (A priori bound).** Let $y(x)$ be the solution of (1.1), $y(x) \in C^2[0, 1]$, $\varepsilon$ the upper bound of the error in the approximation of the derivatives, $\tilde{P}_{2n-1}(\tilde{y})(x)$ the polynomial (2.12) and $M_n$ defined in (3.3). Then

$$|y(x) - \tilde{P}_{2n-1}(\tilde{y})(x)| \leq \frac{\|y^{(2n)}\|_{\infty}}{(2n)!} [x(1-x)]^n + 2\varepsilon M_n$$

(3.4)

with $M_1 = 0$.

**Proof.** We just observe that

$$|y(x) - \tilde{P}_{2n-1}(\tilde{y})(x)| \leq |y(x) - P_{2n-1}(y)(x)| + |P_{2n-1}(y)(x) - \tilde{P}_{2n-1}(\tilde{y})(x)|$$

and then we apply (2.8) and (3.2). □

4. Numerical tests

The proposed method gives an approximate global solution of (1.1) if we are able to calculate the partial derivatives of $f(x, y, y')$ and the solution of the nonlinear system (2.10). We can solve these two problems using a symbolic software.

In this section we report the results obtained by applying the proposed package **TwoPoint** (Appendix A) to find numerical approximations of the solutions to some test problems. For most of the following problems we considered for $n = 4$ in (2.12); different values of $n$ are specified. As we know the exact solutions, we plotted the error functions.

Few software packages give the analytic solution of differential equations. For example, the Matlab ODE solver **bvp4c**, which is a finite difference code that implements the three-stage Lobatto IIIa formula, provides accurate numerical solutions but not their explicit expressions. The solver **ApproxSol** of the Mathematica package **LdeApprox** allows to find polynomial approximation for solutions of only differential equations with polynomial coefficients on a given interval and it does not work for systems of equations. If the given LDE has nonpolynomial coefficients one can use the package function **ToRatCoeffs** which gives a rational interpolation of the coefficients but, of course, the results are less accurate.

The following examples show that **TwoPoint** can be an efficient solver for linear BVPs even when the coefficients are not polynomials.

**Example 1.**

$$\begin{cases}
ky'' - xy' - y = -(1 + k \pi^2) \cos(\pi x) + \pi x \sin(\pi x), \\
y(0) = 1, \\
y(1) = -1
\end{cases}$$

(4.1)

with solution $y(x) = \cos(\pi x)$.

By **TwoPoint** we can plot the error functions, for several values of $k$ (Fig. 1). In this case, **ApproxSol** is not able to find polynomial approximation for solution of problem (4.1). If we first get rational coefficients of the LDE on the interval $[0, 1]$ the error functions are the ones in Fig. 2.
Example 2.

\begin{equation}
\begin{aligned}
ky'' &= y - (1 + k\pi^2) \cos(\pi x), \\
y(0) &= 1 + e^{-1/\sqrt{k}}, \\
y(1) &= -1 + e^{-2/\sqrt{k}}
\end{aligned}
\end{equation}

with solution \( y(x) = \cos(\pi x) + e^{-(x+1)/\sqrt{k}}. \)

For \( k = 0.0001 \) by TwoPointTM2 we get the approximate polynomial

\[
y(x) \approx 1 + 4.8705 \cdot 10^{-6}x - 4.9348x^2 + 0.008122x^3 + 3.9813x^4 \\
+ 0.2902x^5 - 1.8827x^6 + 0.5379x^7
\]

and the error function in Fig. 3.

Using ApproxSol, after rationalizing the coefficients of the LDE on interval \([0, 1]\), we obtain the error function in Fig. 4.

Example 3.

\begin{equation}
\begin{aligned}
\frac{d^2y}{dx^2} + (2 + \cos(\pi x))y' - y = -(1 + k\pi^2) \cos(\pi x) - (2 + \cos(\pi x))\pi \sin(\pi x), \\
y(0) &= 1, \\
y(1) &= -1
\end{aligned}
\end{equation}

with solution \( y(x) = \cos(\pi x). \)

We look for an approximating polynomial of degree 7.

If we approximate the solution of this problem by (2.12) the error function is the one plotted in Fig. 5.
If we use \( \text{ApproxSol} \), also in this case we first need rational coefficients of the LDE on interval \([0, 1]\). The error function is in Fig. 6.

In the case of nonlinear problems TwoPointTM2 can be modified by replacing

\[
\begin{align*}
  s &= \text{Solve}\left\{ R\left(\frac{b-a}{3}\right) = 0, R\left(2\frac{b-a}{3}\right) = 0 \right\}, \{ A, B \}; \\
  A &= s[[1,1,2]]; \quad B = s[[1,2,2]]; \\
  w &= \frac{(y_1-y_0)}{(b-a)}; \\
  s &= \text{FindRoot}\left\{ R\left(\frac{b-a}{3}\right) = 0, R\left(2\frac{b-a}{3}\right) = 0 \right\}, \{ A, w \}, \{ B, w \}; \\
  A &= s[[1,2]]; \quad B = s[[2,2]];
\end{align*}
\]

by

\[
\begin{align*}
  s &= \text{Solve}\left\{ R\left(\frac{b-a}{3}\right) = 0, R\left(2\frac{b-a}{3}\right) = 0 \right\}, \{ A, B \}; \\
  A &= s[[1,1,2]]; \quad B = s[[1,2,2]]; \\
  w &= \frac{(y_1-y_0)}{(b-a)}; \\
  s &= \text{FindRoot}\left\{ R\left(\frac{b-a}{3}\right) = 0, R\left(2\frac{b-a}{3}\right) = 0 \right\}, \{ A, w \}, \{ B, w \}; \\
  A &= s[[1,2]]; \quad B = s[[2,2]];
\end{align*}
\]
Example 4.

\[
\begin{aligned}
\frac{1}{k}y'' &= y + y^2 - e^{-2x\sqrt{k}}, \\
y(0) &= 1, \\
y(1) &= e^{-\sqrt{k}}
\end{aligned}
\]  

(4.4)

with solution \( y(x) = e^{-x\sqrt{k}} \).

For \( k = 0.5 \), by \texttt{TwoPointTM2} we get the approximate polynomial

\[
y(x) \approx 1 - 0.70711x + 0.25x^2 - 0.05892x^3 + 0.01041x^4 - 0.00147x^5 + 0.00017x^6 - 0.00001x^7
\]

and the error function in Fig. 7.
In this case, ApproxSol is not able to find a solution.

5. Systems of first-order equations

For a general \(n\)th order system of first-order equations with boundary conditions

\[
y'(x) = f(x, y(x)), \quad x \in [a, b],
\]

where \(y = (y_1, y_2, \ldots, y_n)\), \(f = (f_1, f_2, \ldots, f_n)\) are vectors and \(f_k = f_k(x, y_1, \ldots, y_n)\) are functions of \(n+1\) variables, the \(n\) boundary conditions may be, say,

\[
y_1(a) = \alpha_1, \quad y_2(a) = \alpha_2, \ldots, y_{m_1}(a) = \alpha_{m_1}, \quad y_{m_1+1}(b) = \beta_1, \quad y_{m_1+2}(b) = \beta_2, \ldots, y_n(b) = \beta_m,
\]

\(m_1, m_2 > 0, \quad m_1 + m_2 = n.\) (5.2)

We will apply the two-point Taylor formula (2.7) to \(y(x)\) in order to find a polynomial approximating the solution of problem (5.1)–(5.2). For the computation of its coefficients we use the differential equation (5.1). Since we need the values of all functions \(y_i, i = 1, \ldots, n\) at the two extremes of the interval, we require the residual \(P'_{2n-1}[y] - f\) to vanish at one point of \((a, b)\), for example the midpoint, to compute the unknown values.

Therefore, to accomplish the method, we have to solve the system

\[
P'_{2n-1}[y] \left( \frac{a + b}{2} \right) - f \left( \frac{a + b}{2}, P_{2n-1}[y] \left( \frac{a + b}{2} \right) \right) = 0
\]

in the unknowns \(y_j(b), j = 1, \ldots, m_1\) and \(y_k(a), k = m_1 + 1, \ldots, n\).

The following examples we will show the numerical results obtained by the code TwoPointTM in Appendix B for some test problems. In these cases, ApproxSol cannot work, thus we have compared our results with the ones obtained by the Matlab solver bvp4c.

Example 5.

\[
\begin{align*}
y_1' &= y_2, & y_1(0) &= 0, \\
y_2' &= y_3, & y_2(0) &= 0, \\
y_3' &= y_2 + 1, & y_3(1) &= e.
\end{align*}
\]

Real solution: \(y_1(x) = e^x - x - 1, \quad y_2(x) = e^x - 1, \quad y_3(x) = e^x.\)

The graph of the error functions is shown in Fig. 8.

Using the Matlab solver bvp4c with an initial mesh of 15 equally spaced points we get error of the same order (Fig. 9).
TwoPointTM 2 gives also the approximate polynomials

\[
p_1(x) = 0.5x^2 + 0.16667x^3 + 0.04162x^4 + 0.00849x^5 + 0.00116x^6 + 0.00033x^7,
\]
\[
p_2(x) = x + 0.5x^2 + 0.16667x^3 + 0.04162x^4 + 0.00849x^5 + 0.00116x^6 + 0.00033x^7,
\]
\[
p_3(x) = 1 + x + 0.5x^2 + 0.16667x^3 + 0.04162x^4 + 0.00849x^5 + 0.00116x^6 + 0.00033x^7.
\]

To solve many nonlinear problems of second-order we write them as systems of first-order equations and use TwoPointTM with just little obvious changes.

**Example 6.**

\[
\begin{align*}
  y'' &= \frac{1}{3} \left[ (2 - x)e^{2y} + \frac{1}{1 + x} \right], \\
  y(0) &= 0, \quad \text{with solution } y(x) = \ln \frac{1}{1 + x}, \\
  y(1) &= -\ln 2.
\end{align*}
\]
The error function using the approximation (2.12) with $n = 5$ and bvp4c are plotted, respectively, in Figs. 10 and 11.

Example 7.

\[
\begin{align*}
\begin{cases}
y'' &= e^y, \\
y(0) &= 0, \\
y(1) &= 0,
\end{cases}
\end{align*}
\]  
(5.5)

where $c \approx 1.33605569490611$ satisfies $c^2 = 1 + \cos(c/2)$.

In this case, the error function using TwoPointTM is drawn in Fig. 12.

The approximate polynomial is

\[
y(x) \approx -0.46363x + 0.5x^2 - 0.07727x^3 + 0.04998x^4 - 0.01361x^5 \\
+ 0.00454x^6 + 1.99105 \cdot 10^{-7}x^7.
\]

The error given by bvp4c is displayed in Fig. 13.
Appendix A.

The function TwoPointTM2 implements the algorithm of two-point-Taylor series method for the numerical solution of BVPs of second order. It generates the two-point Taylor polynomial of a given degree $n$ approximating the solution of the problem in the interval $[a, b]$.

TwoPointTM2 takes as input

- $f$: the function $f(x, y)$,
- $\{x, a, b\}$: $x$ is the independent time variable, $a$ and $b$ the extremes of the integration interval,
- $\{y, y0, y1\}$: $y$ is the dependent variable of $f$, $y0$ the initial value of $y$ at $x = a$, $y1$ the initial value of $y$ at $x = b$,
- $m$: the order of the method.

The output is a polynomial $y = y(t)$ of degree $2m - 1$.

This is the function TwoPointsTM2:

\[
\text{TwoPointsTM2}[f,\{x, a, b\},\{y, y0, y1\},n] := \\
\text{Module}[\{DD, dy, p, fp, R, s, A, B, t = x\}, \\
\]

$y''(x) = f; \ y[a] = y_0; \ y'[a] = A; \ y'[b] = B$

$$DD[j_, t_, z1_, z2_] := \frac{(t-z1)^n}{(z2-z1)^n!} \sum_{k=0}^{n-j-1} \binom{n+k-1}{k} \frac{(t-z2)^{k+j}}{(z1-z2)^k};$$

$$dy[i_, z_] := \text{D}[y[x], \{x, i\}] /. x \to z;$$

$$p[t_] := n-1 \sum_{j=0}^{n-1} (DD[j, t, b, a] \ dy[j, a] + DD[j, t, a, b] \ dy[j, b]);$$

$$fp[t_] := f/.y[x] \to p[t]/. y'[x] \to p'[t] /. x \to t;$$

$$R[t_] := p''[t] - fp[t];$$

$$s = \text{Solve}\{R[(b-a)/3] = 0, R[2(b-a)/3] = 0\}, \{A, B\};$$

$$A = s[[1, 1, 2]]; \ B = s[[1, 2, 2]];$$

$$y \to \text{Collect}[p[t], t] // \text{N}$$

The formal syntax is:

TwoPointsTM2[f, {t, a, b}, {y, y0, y1}, n]

To plot the approximate polynomial $y$: Plot[y /% , {t, a, b}]

Appendix B.

The function TwoPointTM implements the algorithm of two-point Taylor series method for the numerical solution of a system of first-order differential equations with boundary conditions (5.1). It generates the two-point Taylor polynomial of a given degree $n$ approximating the solutions $y_1, \ldots, y_n$ of the problem in the interval $[a, b]$.

TwoPointTM takes as input

- $f$: the function $f(x, y)$,
- $\{x, x0, y0\}$: $x$ is the independent time variable, $x0 = \{a, b\}$ with $a = \{a_1, \ldots, a_m\} \in \mathbb{R}^m$ and $b = \{b_1, \ldots, b_m\} \in \mathbb{R}^m$,
- $yy = \{y_1, \ldots, y_n\}$,
- $n$: the order of the method.

The output is a polynomial $y = y(t)$ of degree $2n - 1$.

TwoPointsTM[F_, {x_, x0_, y0_}, yy_, n_] :=
Module[{YX, YT, YP, m0, M0, dy, dd, DD, h, sol, nf = Length[F], t = x, i},

$$DD[j_, t_, z1_, z2_] := \frac{(t-z1)^n}{(z2-z1)^n!} \sum_{k=0}^{n-j-1} \binom{n+k-1}{k} \frac{(t-z2)^{k+j}}{(z1-z2)^k};$$

$$m0 = \text{Min}[x0]; \ M0 = \text{Max}[x0];$$

If[(nf = = 2 && yy[[1]] = = yy[[2]])],
YX = {ToExpression[StringJoin[ToString[yy[[1]]]],
"[",ToExpression[x, "]"]",F[[1]]]},
YX = Table[ToExpression[StringJoin[ToString[yy[[i]]]],"[",ToExpression[x, "]"]",{i,nf}]],
YT = Table[ToExpression[StringJoin[ToString[yy[[i]]]],"[",ToExpression[x0[[i]]]],""]",{i,nf}]],

dd = D[YX, x];
dy = NestList[(D[#1, t]/.Thread[dd \[Rule] F])&,F,n-1];
h[t_] = (DD[0, t, M0, m0] (YX/.x \[Rule] m0) + DD[0, t, m0, M0] (YX/.x \[Rule] M0) +
\sum_{j=1}^{n-1} (DD[j, t, b, a] (dy[[j]])/.x \[Rule] m0) + DD[j, t, a, b] (dy[[j]])/.x \[Rule] m0))/. MapThread[Rule[#1, #2]&,\{yt, y0\}];

If[(nf = = 2 && yy[[1]] = = yy[[2]])],
YP = Table[ToExpression[StringJoin[ToString[F[[i]]]],
If[x0[[i]] = = m0, x \[Rule] M0, x \[Rule] m0]]],{i,nf}];
YP = Table[ToExpression[StringJoin[ToString[yy[[i]]]], "[", If[x0[[i]] == m0, ToString[M0], ToString[m0]], "]"]], {i, nf}];
sol = FindRoot[((D[h[t], t] - F /. Thread[YX → h[t]]) /. t → (m0 + M0)/2) == 0, Thread[{{YP, y0}}]];
y → Collect[h[t]/.sol , t]//N

For the problem of Example 5
x0 = {0, 0, 1}; Y0 = {0, 0, Exp[1]}; F = {y2[x], y3 [x], y2 [t] + 1};
TM[F, {x, x0, Y0}, {y1, y2, y3}, 4]

For the problem of Example 6
x0 = {0, 1}; Y0 = {0, -Log[2]}; F = {y2[x], 1/3 ((2-x)*Exp[2 y1[x]]+1/(1+x))};
TM[F, {x, x0, Y0}, {y1, y1}, 5]

References