

Using Picard Iteration and Cauchy Products
to Solve Initial Value Problem Ordinary
Differential Equations in Maple

James H. Money

James S. Sochacki

Outline of Talk

- A preliminary example
- The Main Theorem of Parker and Sochacki
- C++-Code Implementation of the Theorem
- Example - N Body Problem
- Example - Burger's Equation
- Using Cauchy Products
- Cauchy Product codes in C/Fortran
- Example - Springy Pendulum
- Future Directions - Pade, Error Analysis

Consider the IVP ODE

$$x'(t) = \sin x; \quad x(0) = \alpha \quad (1)$$

We let $x_2 = \sin x$ and $x_3 = \cos x$ and then get

$$x'_2(t) = \cos x x'$$

and

$$x'_3(t) = -\sin x x'$$

Hence we get the new system:

$$\begin{cases} x'(t) = x_2; & x(0) = \alpha \\ x_2'(t) = x_2 x_3; & x_2(0) = \sin \alpha \\ x_3'(t) = -x_2^2; & x_3(0) = \cos \alpha \end{cases}$$

We can use the fact that

$$x_2^2 + x_3^2 = 1$$

and we get that

$$x_3'(t) = x_3^2 - 1$$

and we find the solution is

$$x_3 = \frac{1 - e^{2t+2B}}{1 + e^{2t+2B}}$$

From this we obtain

$$x_2 = \frac{4e^{2t+2B}}{(1 + e^{2t+2B})^2}$$

and

$$x = 2 \arctan e^{t+B}$$

For continuous $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ the IVP ODE

$$y'(t) = F(t, y(t)) ; y(t_0) = y_0$$

is equivalent to the integral equation

$$y(t) = y(t_0) + \int_{t_0}^t F(s, y(s)) ds.$$

Theorem

Let $F = (F_1, \dots, F_n) : \mathbb{R}^n \rightarrow \mathbb{R}^n$ be a polynomial and

$$Y = (y_1, \dots, y_n) : \mathbb{R} \rightarrow \mathbb{R}^n$$

Consider the IVP ODE

$$y'_j = F_j(Y) ; y_j(0) = \alpha_j ; j = 1, \dots, n$$

and the Picard iterates

$$p_{j,1}(t) = \alpha_j ; j = 1, \dots, n$$

$$p_{j,k+1}(t) = \alpha_j + \int_0^t F_j(P_k(s)) ds ; k = 1, 2, \dots ; j = 1, \dots, n$$

then $p_{j,k+1}$ is the k th MacLaurin Polynomial for y_j plus a polynomial all of whose terms have degree greater than k .

(Here $P_k(s) = (p_{1,k}(s), \dots, p_{n,k}(s))$).

The above Examples and Theorem lead to the following definition

Definition

Let $x : \mathbb{R} \rightarrow \mathbb{R}$ then x is said to be projectively polynomial if there is an $n \in \mathbb{N}$ and $\alpha_2, \dots, \alpha_n \in \mathbb{R}$, $x_2, \dots, x_n : \mathbb{R} \rightarrow \mathbb{R}$ and a polynomial $Q : \mathbb{R}^n \rightarrow \mathbb{R}^n$ so that $y = (x, x_2, \dots, x_n)^T$ satisfies

$$y' = Q(y) ; y(0) = \begin{pmatrix} x(0) \\ \alpha_2 \\ \cdot \\ \cdot \\ \cdot \\ \alpha_n \end{pmatrix}$$

If the degree of $Q = k$ then we write $x \in P_{n,k}$. We let $P_k = \cup_n P_{n,k}$ and $P = \cup_k P_k$.

Properties of Projectively Polynomial Functions

- [1] $P_m \subset P_k$ for $k > m$.
- [2] If $f \in P$ then $f' \in P$
- [3] If $f, g \in P$ then $f + g, fg$ and $f \circ g \in P$.
- [4] If $f \in P$ and $f(0) \neq 0$ then $\frac{1}{f} \in P$
- [5] If $f \in P$ and $f'(0) \neq 0$ then $f^{-1} \in P$
- [6] $P_2 = P$.
- [7] $P \subset A$. (HARD! A is the set of analytic functions)
- [8] $P, A/P$ are dense in A .
- [9] $P_1 = \text{span} \{t^n e^{\alpha t} | n \in \{0, 1, 2, \dots\}, \alpha \in C\}$
($x' = Mx$; M an n by n matrix).

[10] $u \in P$ if and only if there is a polynomial $Q : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ so that $Q(u, u', \dots, u^n) = 0$. (Grobner Bases)

[11] $\tan t \in P_2$ ($y' = 1 + y^2$) $\tan t \notin P_1$.

[12] (Pade') $x(t) = \frac{\sum_{i=0}^N a_i t^i}{1 + \sum_{i=1}^J b_i t^i} = \sum_{i=0}^K M_i t^i$

[13] (*A-priori*) Taylor error bounds for approximations to solutions to initial value ordinary differential equations can be obtained independent of the derivative.

As a result of this theorem, a C++ code was developed to handle **arbitrary** systems of ODEs that are projectively polynomial. The code consists of several parts:

- Language parser for the input files
- Routines to handle polynomial manipulation for any dimension n . This is done via a hash table and manual computation of the multiplications.
- Output routines for exporting data to Maple.
- A polynomial integrator for handling projectively polynomial systems. This is performed term by term for each polynomial.

In addition, the code was parallelized for use on multiple CPUs:

- The code is parallelized by running each equation in the system on a separate CPU.
- Each node performs the integration, and then evaluates the polynomial at completion to update all the corresponding nodes that need the changes.
- The parallelization was originally written in PVM then later ported to MPI. This allows it to run on a single CPU machine if necessary.

Example - N Body Problem

$$x_i''(t) = \sum_{j \neq i} \frac{m_j(x_j - x_i)}{r_{i,j}^{\frac{3}{2}}}$$

$$y_i''(t) = \sum_{j \neq i} \frac{m_j(y_j - y_i)}{r_{i,j}^{\frac{3}{2}}}$$

$$z_i''(t) = \sum_{j \neq i} \frac{m_j(z_j - z_i)}{r_{i,j}^{\frac{3}{2}}}$$

where $r_{i,j} = [(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2]$, $i = 1, \dots, N$.

If we let

$$s_{i,j} = r_{i,j}^{-\frac{1}{2}}$$

this system for the N-body problem can be posed as

$$x'_i = u_i$$

$$y'_i = v_i$$

$$z'_i = w_i$$

$$u'_i = \sum_{j \neq i} m_j (x_j - x_i) s_{i,j}^3$$

$$v'_i = \sum_{j \neq i} m_j (y_j - y_i) s_{i,j}^3$$

$$w'_i = \sum_{j \neq i} m_j (z_j - z_i) s_{i,j}^3$$

$$s'_{i,j} = -\frac{1}{2} s_{i,j}^3 [2(x_i - x_j)(u_i - u_j) + 2(y_i - y_j)(v_i - v_j) + 2(z_i - z_j)(w_i - w_j)]$$

Example 2 : Burger's Equation

$$\frac{\partial u}{\partial t}(x, t) = -\frac{1}{2} \frac{\partial u^2}{\partial x} + \alpha \frac{\partial u}{\partial x} + \beta \frac{\partial^2 u}{\partial x^2}$$

with

$$\alpha = -5 \quad \beta = 10^{-5}$$

and the initial condition

$$u(x, 0) = -\sin x$$

which we expand as a MacLaurin series to the needed precision.

Cauchy Products

Now lets generate the MacLaurin polynomial for the solution of the initial value problem

$$y'(t) = A + By + Cy^2 = Q(y) \quad y(0) = \alpha.$$

We start the Picard process with

$$P_1(t) = \alpha = M_1(t)$$

and then define

$$P_{k+1}(t) = \alpha + \int_0^t Q(P_k(s))ds = \alpha + \int_0^t (A + BP_k(s) + CP_k(s)^2)ds.$$

This integral equation is equivalent to the IODE

$$P'_{k+1}(t) = A + BP_k(t) + CP_k(t)^2 = Q(P_k(t)) \quad y(0) = \alpha.$$

Since

$$P_k(t) = M_k(t) + \sum_{n=k}^{2^k-1} b_n t^n,$$

where

$$M_{k+1}(t) = \sum_{n=0}^k a_n t^n$$

is the degree k MacLaurin polynomial for the solution y to the IODE. One can use the integral equation or differential equation for $P_{k+1}(t)$ to solve for a_k and b_k . Since $a_0 = \alpha$ and $a_1 = Q(\alpha)$, this only needs to be done for $k \geq 2$.

We now derive the Cauchy Product form. Using the series expansion we have:

$$\begin{aligned} P'_{k+1} &= \sum_{n=1}^k n a_n t^{n-1} + \sum_{n=k+1}^{2^{k+1}-1} n b_n t^{n-1} \\ &= \sum_{n=0}^{k-1} (n+1) a_{n+1} t^n + \sum_{n=k+1}^{2^{k+1}-1} n b_n t^{n-1} \end{aligned}$$

While using the quadratic form of the ODE:

$$\begin{aligned}
 P'_{k+1} &= A + B \sum_{n=0}^{2^k-1} b_n t^n + C \sum_{n=0}^{2^k-1} b_n t^n \sum_{n=0}^{2^k-1} b_n t^n \\
 &= A + B \sum_{n=0}^{2^k-1} b_n t^n + C \sum_{j=0}^{2^k-1} \sum_{n=0}^{2^k-1} b_j b_n t^{n+j} \\
 &= A + B \sum_{n=0}^{2^k-1} b_n t^n + C \sum_{n=0}^{2(2^k-1)} d_n t^n,
 \end{aligned}$$

where $b_n = a_n$ for $n \leq k$ and $d_n = \sum_{j=0}^n b_j b_{n-j}$ for $0 \leq n \leq 2^k - 1$ and $d_n = \sum_{j=n-2^k+1}^{2^k-1} b_j b_{n-j}$ for $2^k \leq n \leq 2(2^k - 1)$.

By equating like powers we end up with

$$ka_k = Ba_{k-1} + C \sum_{j=0}^{k-1} a_j a_{k-1-j}.$$

That is,

$$M_{k+1}(t) = M_k(t) + \frac{(Ba_{k-1} + C \sum_{j=0}^{k-1} a_j a_{k-1-j})}{k} t^k.$$

That is, we can obtain the MacLaurin polynomial algebraically.

Newer routines were written to handle Quadratic Polynomial systems using the Cauchy Products that were discovered:

- Both the integrator and Cauchy product codes are implemented in Fortran 90/C.
- Both sets of code allows the user to input the values for the coefficients for the polynomial system and the corresponding initial conditions.
- Both programs calculate the values at each step and then outputs the data to a file for importing into Maple.
- The Cauchy product codes requires the system of differential equations to be in quadratic form while the Picard integrator code only requires the system of differential equations to be in polynomial form.

Example 3 : Springy-Pendulum

$$\frac{d^2}{dt^2}\theta = -\frac{2\left(\frac{d}{dt}r(t)\right)\frac{d}{dt}\theta(t) + g\sin(\theta)}{r}$$

$$\frac{d^2}{dt^2}r = r\left(\frac{d}{dt}\theta(t)\right)^2 + g\cos(\theta) - Kr + Kr_{eq}$$

Let

$$y_1 = \theta, y_2 = y_1', y_3 = r, y_4 = y_3', y_5 = \sin(\theta), y_6 = \cos(\theta), y_7 = r^{-1}$$

Future Directions

Several areas are currently being researched:

- Using Pade rational functions for singular and stiff systems - since we have the Taylor expansion for the function, it is straightforward to calculate the Pade approximates.
- Error Analysis - Paul Warne has completed an error analysis of the method including an error bound on the method. Using this information, we can adapt the timestep lengths to correct amount and not waste unnecessarily long expansions or shortened timesteps for computations.