Modified Chebyshev Picard Iteration for Efficient Numerical Integration of Ordinary Differential Equations

Brent Macomber, Robyn M. Woollands, Austin Probe, Ahmad Bani Younes, John L. Junkins

Texas A & M University, Aerospace Engineering Dept, H.R. Bright, 3141 TAMU, College Station, TX, 77843-3141

Xiaoli Bai

Optimal Synthesis, Inc, 95 1st St. Suite 240, Los Altos, CA 94022

ABSTRACT

Modified Chebyshev Picard Iteration (MCPI) is an iterative numerical method for approximating solutions of linear or non-linear Ordinary Differential Equations (ODEs) to obtain time histories of system state trajectories. Unlike other step-by-step differential equation solvers, like the Runge-Kutta family of numerical integrators, MCPI approximates long arcs of the state trajectory with an iterative path approximation approach, and is ideally suited to parallel computation. Orthogonal Chebyshev Polynomials are used as basis functions during each path iteration, and the integrations of the Picard iteration are then done analytically. The orthogonality of the Chebyshev basis functions mean that the least square approximations can be computed without a matrix inversion; the coefficients are conveniently computed robustly from discrete inner products. As a consequence of discrete sampling and weighting adopted for the inner product definition, the Runge phenomena errors that usually occur near the ends of the approximation intervals are significantly minimized. The MCPI algorithm utilizes a vector-matrix framework for computational efficiency. Additionally, all Chebyshev coefficients and integrand function evaluations are independent, meaning they can be simultaneously computed in parallel for further decreased computational cost. Over an order of magnitude speedup from traditional methods is achieved in serial processing, and an additional order of magnitude is achievable in parallel architectures.

This paper presents a new MCPI library, a modular toolset designed to allow MCPI to be easily applied to a wide variety of ODE systems. Library users will not have to concern themselves with the underlying mathematics behind the MCPI method. Inputs are the boundary conditions of the dynamical system, the integrand function governing system behavior, and the desired time interval of integration, and the output is a time history of the system states over the interval of interest.

Examples from the field of astrodynamics are presented to compare the output from the MCPI library to current state-of-practice numerical integration methods. It is shown that MCPI is capable of out-performing the state-of-practice in terms of computational cost and accuracy.

1. INTRODUCTION

Modified Chebyshev Picard Iteration (MCPI) is an iterative numerical method for solving linear or non-linear ordinary differential equations. It combines the discoveries of two great mathematicians: Émile Picard (Picard Iteration) and Rafnuty Chebyshev (Chebyshev Polynomials). The decision to make use of these techniques in a simultaneous manner was first proposed by Clenshaw and Norton in 1963 [1].

Picard stated that any first order differential equation

$$\dot{x}(t) = f(t, x(t)), \quad x(t_0), \qquad (1.1)$$

with an initial condition $x(t_0) = x_0$, may be rearranged without approximation to obtain the integral equation shown in Eq. (1.2).

$$x(t) = x(t_0) + \int_{t_0}^{t} f(\tau, x(\tau))d\tau. \quad (1.2)$$

A sequence of approximate solutions, $x^i(t)$, (i = 1, 2, 3, …, ∞), to this differential equation may be obtained through Picard iteration using the following formula:

$$x^i(t) = x(t_0) + \int_{t_0}^{t} f(\tau, x^{i-1}(\tau))d\tau, \quad i = 1,2,... \quad (1.3)$$

In the MCPI method, orthogonal Chebyshev polynomials are used as basis functions to approximate the integrand in the Picard integral. Chebyshev polynomials reside in the domain $\tau = [-1,1]$, and can be defined recursively as:

$$T_0(\tau) = 1, \quad (1.4)$$

$$T_1(\tau) = \tau, \quad (1.5)$$

$$T_{k+1}(\tau) = 2\tau T_k(\tau) - T_{k-1}(\tau). \quad (1.6)$$

Unlike traditional step-by-step integrators, for example the Runge-Kutta methods, MCPI is unique in that long state trajectory arcs are approximated during the Picard iteration. The system dynamics are normalized such that the timespan of integration is projected onto the domain of the Chebyshev polynomials, thus the system states can be approximated using the Chebyshev polynomial basis functions. The orthogonal nature of the basis functions means that the coefficients that linearly scale the basis functions can be computed independently as simple ratios of inner products with no matrix inversion.

As a consequence of the independence of the basis functions, the coefficients multiplying the Chebyshev basis functions may be computed in parallel by separate processor threads. This is the first of two available layers of parallelization in the MCPI method. The second layer of parallelization is enabled by the fact that the entire state trajectory over the time interval of interest is estimated at once. Thus the calculation of the integrand function (which is a function of the system states) can be performed all at once on parallel processor threads. Using MCPI, over an order of magnitude speedup from traditional methods is achieved in serial processing, and an additional order of magnitude is achieved in parallel architectures.

A key feature of MCPI is a non-uniform cosine density sampling of the domain of the Chebyshev basis functions called Chebyshev-Gauss-Lobatto (CGL) nodes, defined in Eq. (1.7).

$$\tau_j = \cos(j\pi / N), j = 0,1,2...,N \quad (1.7)$$

This sampling scheme has much higher density towards the edges, which enables a higher accuracy solution near the boundaries of the state trajectory. This scheme eliminates the Runge phenomena, a common issue in function approximation whereby noisy estimates are returned near the edges due to lack of knowledge of the states on the other sides of the boundaries. The coefficients multiplying the Chebyshev basis functions are approximated by the method of least squares, which generally requires a matrix inversion. A wonderful side effect of the cosine sampling scheme is that the matrix required to be inverted in the Normal Equations of least squares is diagonal, thus the inverse is trivial.

In 2010, Bai's dissertation [2] laid the groundwork of MCPI and proved the capability of the method to outperform the state of the practice for numerical integration of ODEs. Bai and Junkins applied MCPI to non-linear IVPs and orbit propagation in [3], and showed that MCPI can outperform other higher order integrators such as Runge-Kutta-

Nystrom 12(10). In [4] Bai and Junkins applied MCPI to efficiently solving Lambert's transfer problem, and to solving an optimal control trajectory design problem more accurately and efficiently than the Chebyshev pseudospectral method. In [5] Bai and Junkins use MCPI in a complex three-body station-keeping control problem formulated as a BVP. Subsequent publications by Junkins et al. [6], [7], and [8] further clarify the concept and derivation of MPCI and orthogonal approximation in general, and apply the method to problems in the field of astrodynamics.

A full derivation of MCPI is beyond the scope of this short paper. Instead we present a flow chart in Fig. 1 briefly summarizing the mathematics underlying the MCPI method for solution of an Initial Value Problem (IVP). Fig. 2 is the same mathematics represented in the more elegant vector/matrix formulation, which is computationally the most efficient way to implement the method. Any of the above references provide more detailed derivations, as well as examples and results that demonstrate the power of the MCPI algorithm with regard to speed and accuracy. Additionally, those references contain comparisons to other well-known integrators including high-order Runge-Kutta methods and the Gauss-Jackson method.

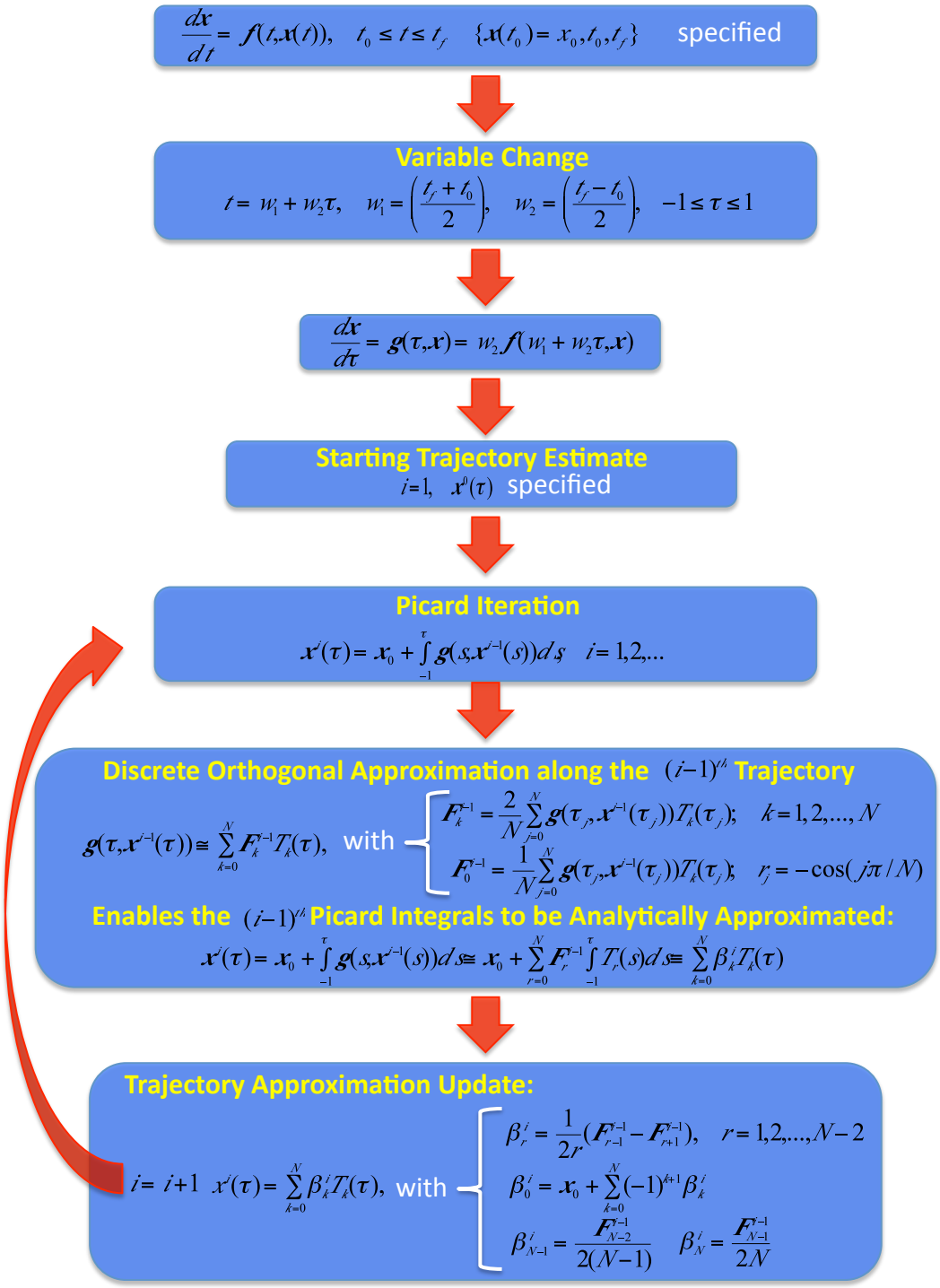

Fig. 1. Flow diagram of MCPI Initial Value Problem implementation.

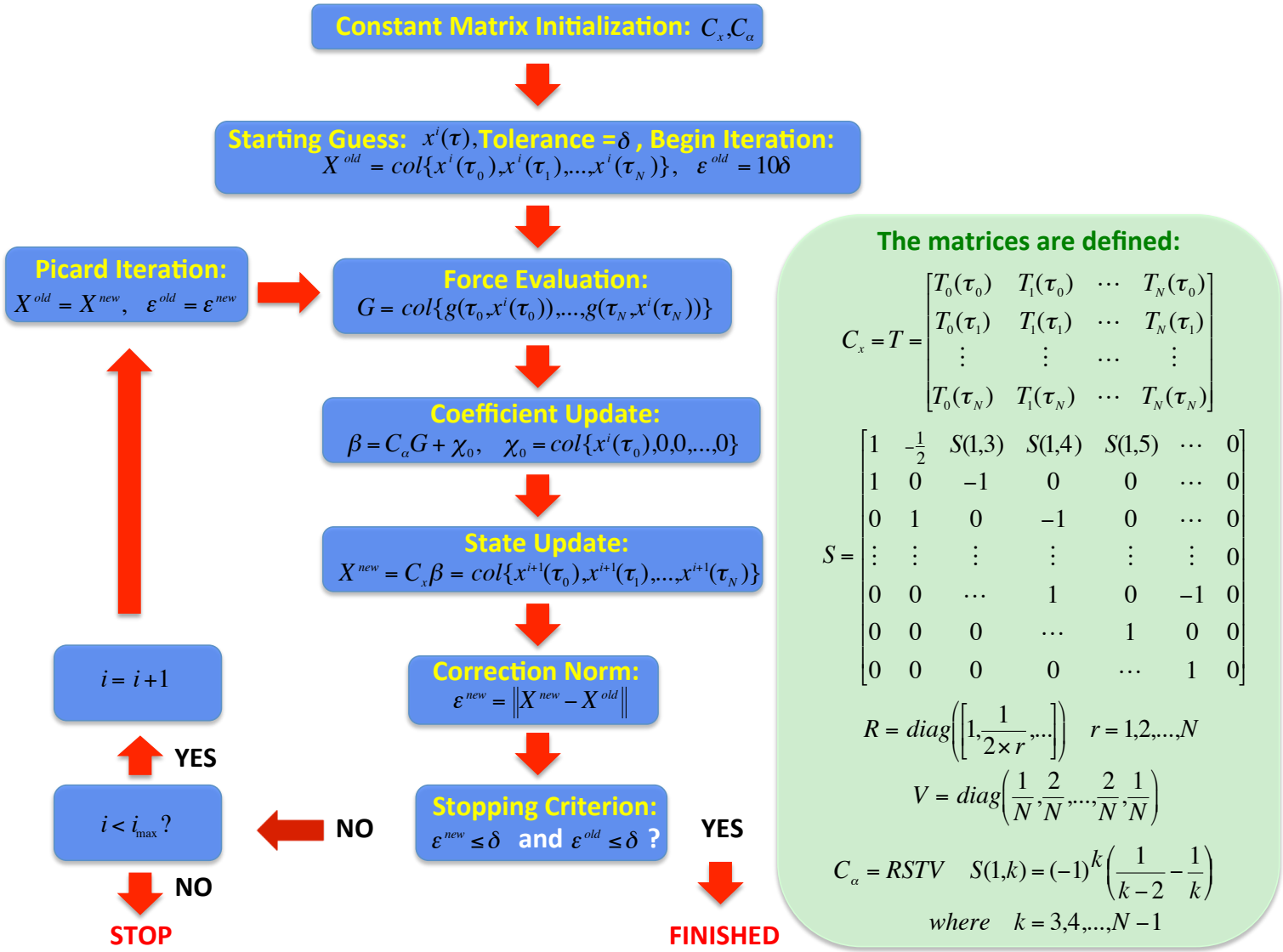

Fig. 2. Flow diagram of MCPI algorithm in vector-matrix form.

2. TAMU MCPI LIBRARY

This paper introduces the Texas A&M University MCPI libraries (TAMU MCPI), which have been created to encourage widespread use of the MCPI method for solution of Ordinary Differential Equations. The goal of the project is to create an easy to use toolset that effectively eliminates the learning curve of using MCPI methods, but at the same time is versatile and powerful enough for application to a variety of projects. The user is not required to have a thorough understanding of the inner-workings of MCPI in order to implement it in their own projects. TAMU MCPI is a set of efficient and lightweight classes for solution of Initial Value Problems (IVPs) and Boundary Value Problems (BVPs). Solvable ODEs can be linear or non-linear, autonomous or non-autonomous, and first-order or second-order. Higher order systems are solvable by decomposition to a first-order or second-order system by the inclusion of additional states that are the time derivatives of lower order states.

Fig. 3 shows a high-level overview of the TAMU MCPI structure from an implementation point of view. The user provides a handle to an integrand function for the problem at hand, that is, the update function that describes how the time derivatives of the system states behave. Additionally, the user provides the relevant boundary conditions for the system states, defined at the initial time, the final time, or both, depending upon the problem to be solved. If the system has time-varying parameters, or other numerical data is required in the integrand function, these may be inputted as well. Given these inputs, TAMU MCPI will iteratively attempt to numerically solve the state-space trajectories of the system over the desired time interval. If a solution is found, the time history of the system states over the interval of interest is returned.

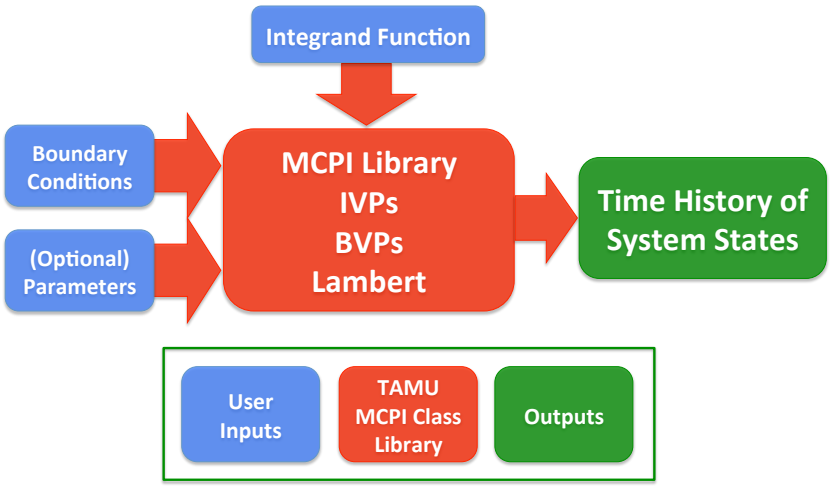

Fig. 3. High-level overview of TAMU MCPI library.

The TAMU MCPI library is available in Matlab, C++, and as Matlab wrapper functions to the CUDA parallel computation environment. CUDA stands for Compute Unified Device Architecture, and is a parallel computing language developed by NVIDIA for use upon their Graphics Processing Units (GPUs); effectively it allows lightweight parallel computation at a desktop workstation. TAMU MCPI is fully cross-platform, and has been tested on Windows, Linux, and Apple computers. The structure of the libraries is hierarchical, with an abstract parent class and derived child classes tailored to the solution of various problem types. This modular approach is to allow for future expansion, or application-specific customization and optimization. Control parameters can be set from a configuration file or interactively by the user.

The C++ libraries can be distributed as source code with minimal external dependencies (the only dependencies are headers from the Boost cross-platform library[1]), or as pre-compiled binaries and header files for many widely used operating systems. Compiling the libraries from source is possible with any reasonable C++ compiler, and include

[1] Boost is a set of cross-platform C++ tools to accomplish common tasks. TAMU MCPI uses header-only Boost libraries to avoid inclusion of large binary files. See http://www.boost.org/ for more information.

files and linking are managed with CMake[2]. The CUDA libraries utilize the Matlab Parallel Computation Toolbox, and require Matlab 2010 or newer (2011 or newer recommended), and an NVIDIA GPU with compute capability of 1.3 or greater.

3. EXAMPLE: ORBITAL PROPAGATION OF DEBRIS CLOUD

In this example, we forward propagate the orbital motion of a cloud of 1000 simulated debris objects in Low Earth Orbit. Initially the cloud is a three-dimensional Gaussian distribution with mean initial position and velocity and distribution parameters as shown in Table 1. The mean particle orbital eccentricity is e = 0.0099, and the mean orbital period is P = 5.3905 x 10^3 seconds. The motion of each object is propagated forward by one (mean) orbital period using a simple inverse square gravity model. The initial and final distributions are shown in Fig. 4 (note that the Earth is shown solely to provide scale, the coordinate system is arbitrary).

This numerical integration is performed using the TAMU MCPI Initial Value Problem library running in Matlab 2013, and benchmarked against the native Matlab Runge-Kutta 4(5) variable step size numerical solver ODE45. The comparison is carried out on a laptop computer with an Intel Core i7 2.3GHz processor, and16GB of RAM. The accuracy of the numerical solution is verified against the analytic F and G solution, and both algorithms are tuned to have similar accuracy as shown in Fig. 5, in which the motion of a single particle is propagated forward by several orbits. In this arrangement, the Matlab implementation of TAMU MCPI forward propagates the particle cloud motion five times faster than ODE45, and with comparable accuracy.

Table 1: Parameters required for the IVP solution.

Orbit Parameters	
Propagation Time (s)	5.3905 x 10^3
Mean Particle Initial Position Vector (km)	[-464.856, 6667.880, 574.231]
Mean Particle Initial Velocity Vector (km/s)	[-2.8381,-0.7872,7.0830]
Standard Deviation Particle Position (km)	0.1
Standard Deviation Particle Velocity (km/s)	0.1

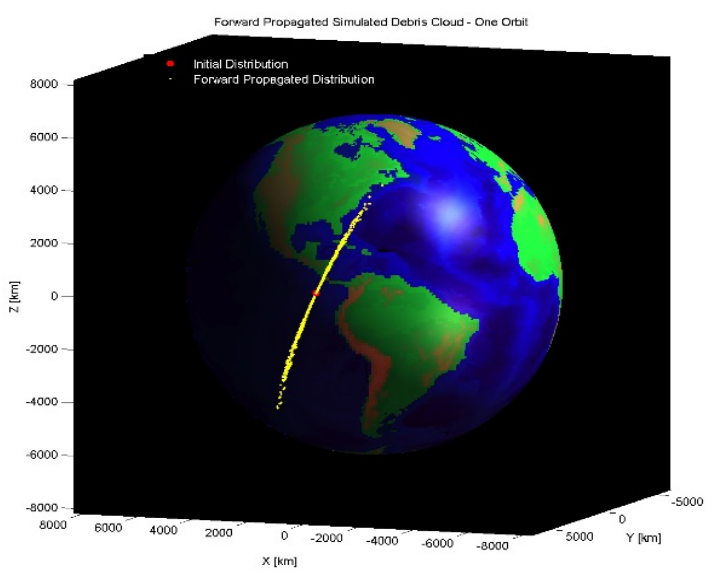

Fig. 4: Simulated debris cloud, initial Gaussian distribution and final distribution after forward propagation by one mean orbital period.

[2] Cmake is a cross-platform build tool that creates projects such that the native compiler can build applications from source code. See http://www.cmake.org/ for details.

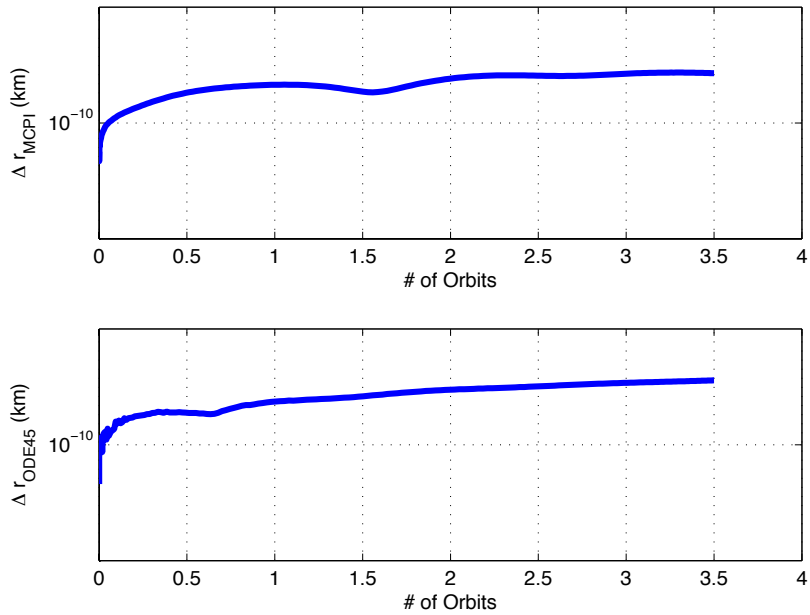

Fig. 5. Position errors of the MCPI algorithm (top panel) and the ODE45 (bottom panel) compared with the analytic F and G solution.

4. EXAMPLE: LAMBERT'S TRANSFER PROBLEM

We solve the orbital motion for a section of a Low Earth Orbit given boundary conditions on the initial and terminal position as well as the time taken for the motion, a formulation called Lambert's Problem. These input parameters are shown in Table 2. The period of the chosen orbit is P = 5.3905 x 10^3 seconds, and the eccentricity is e = 0.0099.

This problem is solved with the TAMU MCPI Second Order Boundary Value (Lambert-Style) library running in Matlab 2013, and benchmarked against the Shooting Method using fsolve and ODE45. The comparison is carried out on a laptop computer with an Intel Core i7 2.3GHz processor, and16GB of RAM. The output from the two solvers are verified against the analytic F and G solution, and the parameters of both algorithms are tuned until the accuracy is comparable, as shown in Fig. 7. Depending upon the desired arc-length of solution, the Matlab implementation of TAMU MCPI is able to solve the Lambert Problem 20-60 times faster than the shooting method with fsolve and ODE45, and with comparable accuracy.

For this given orbit, the MCPI BVP algorithm maximum arc length over which convergences occurs is 38% of an orbital period. We are currently investigated promising new methods to increase this arc length, and these will appear in subsequent publications.

Table 2: Parameters required for the BVP solution.

Orbit Parameters	
Propagation Time (s)	0.38 * 5.3905 x 10^3
Initial Position Vector (km)	[-464.856, 6667.880, 574.231]
Final Position Vector (km)	[-1386.506,-5174.986,3873.216]

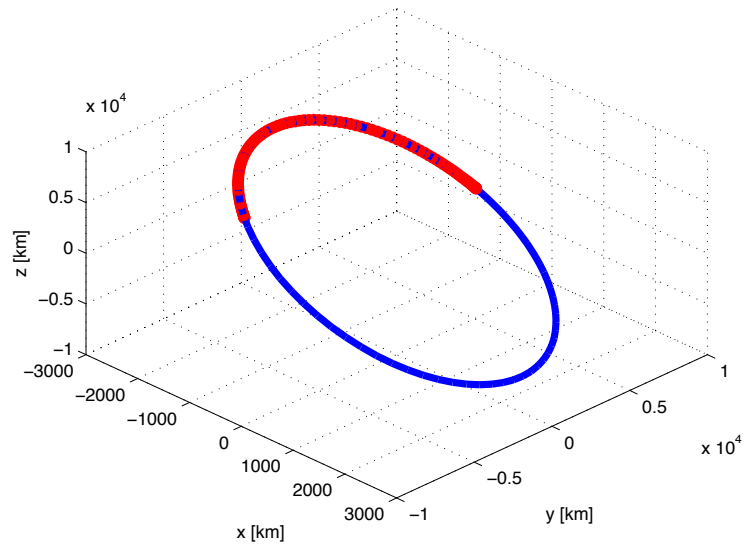

Fig. 6: The reference orbit generated from an F and G solution (blue), and the 38% time period arc (red) propagated with the BVP algorithm.

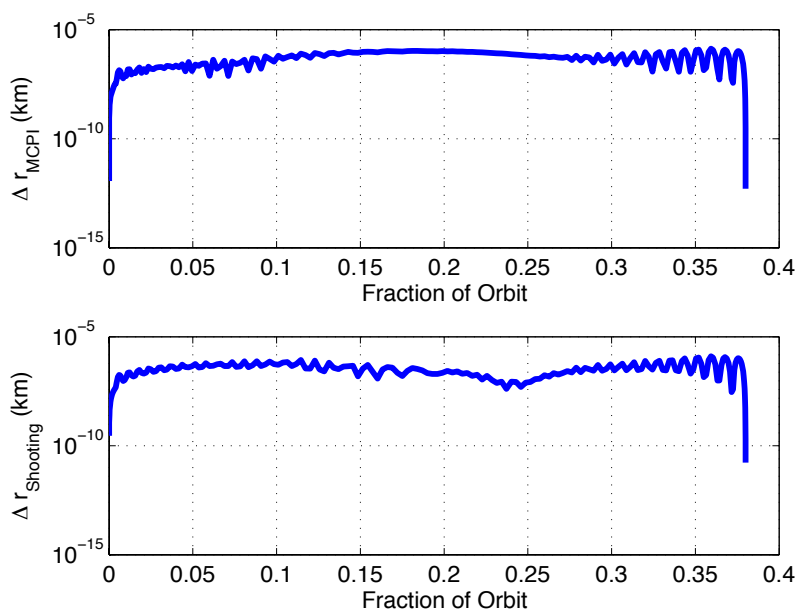

Fig. 7. Position errors of the MCPI algorithm (top panel) and the shooting method (bottom panel) compared with the analytic F and G solution.

5. CONCLUSION

Modified Chebyshev Picard Iteration (MCPI) is an iterative numerical method for approximating solutions of linear or non-linear Ordinary Differential Equations (ODEs). Unlike other step-by-step differential equation solvers, like the Runge-Kutta family, MCPI approximates long arcs of the state trajectory with an iterative path approximation approach, and is ideally suited to parallel computation. Orthogonal Chebyshev Polynomials are used as basis functions during each path iteration, and the integrations of the Picard iteration are then carried out analytically. The orthogonality of the Chebyshev basis functions allows the least square approximations to be computed without

matrix inversion. Instead the coefficients are computed robustly from discrete inner products. The discrete sampling and weighting that is adopted to satisfy the inner product definition creates that added benefit that the approximation errors are minimized near the ends of the interval.

The MCPI algorithm utilizes a vector-matrix framework for computational efficiency. All Chebyshev coefficients and integrand function evaluations are independent, meaning they can be simultaneously computed in parallel for further decreased computational cost. Over an order of magnitude speedup from traditional methods is achieved in serial processing, and an additional order of magnitude is achievable in parallel architectures.

In this paper we have presented the new TAMU MCPI library that allows the user to easily apply the MCPI method to their own ODE systems. The TAMU MCPI library is available in Matlab, C++, and as Matlab wrappers for CUDA parallel computation. It is fully cross-platform for Windows, Linux, and Apple, and can be compiled from source by the user, or distributed as a binary library for many common operating systems. The idea is that the user does not need to concern themselves with the underlying mathematics behind the MCPI algorithm, but simply inputs the boundary conditions of the dynamical system, the integrand function governing system behavior, and the desired time interval of integration. The algorithm outputs the time history of the system states over the interval of interest.

Two astrodynamic examples are presented to demonstrate the capability of the algorithm for the initial value and boundary value problems respectively. For the first example (IVP) we forward propagate a simulated cloud of debris particles in a low earth orbit. Compared to a native Matlab ODE45 integrator, we are able to forward propagate the motion five times faster with the same accuracy. For the second example (BVP) we consider Lambert's problem and present a convergence arc length of 38% of the orbit. Depending upon the arc-length of the orbit in the Lambert's problem, MCPI is able to obtain a solution 20-60 times faster than the shooting method. We have demonstrated the power of our MCPI algorithm in numerous publications, and we are excited at the prospect of sharing this new library to afford other researchers the opportunity to benefit from these tools.

6. REFERENCES

1. C. W. Clenshaw and H. J. Norton, *The solution of Nonlinear Ordinary Differential Equations in Chebyshev Series,* The Computer Journal, 6(1):88-92,1963.

2. X. Bai, *Modified Chebyshev-Picard Iteration Methods for Solution of Initial Value and Boundary Value Problems*, Ph.D. dissertation, Texas A&M University, College Station, Tex, USA, 2010.

3. X. Bai and J. L. Junkins, *Modified Chebyshev-Picard Iteration Methods for Solution of Initial Value Problems*, Advances in the Astronautical Sciences, vol. 139, pp. 345–362, 2011.

4. X. Bai and J. L. Junkins, *Modified Chebyshev-Picard Iteration Methods for Solution of Boundary Value Problems*, Advances in the Astronautical Sciences, vol. 140, pp. 381–400, 2011.

5. X. Bai and J. L. Junkins, *Modified Chebyshev Picard Iteration Methods for Station-Keeping of Translunar Halo Orbits*, Mathematical Problems in Engineering, vol. 2012, Article ID 926158, 2012.

6. John L. Junkins, Ahmad Bani Younes, Robyn M. Woollands and Xiaoli Bai, *Orthogonal Approximation in Higher Dimensions: Applications in Astrodynamics*, ASS 12-634, Jer-nan Juang Astrodynamics Symposium, College Station, TX, June 14-26, 2012.

7. John L. Junkins, Ahmad Bani Younes, Robyn M. Woollands and Xiaoli Bai, *Orthogonal Approximation in Higher Dimensions: Applications in Astrodynamics,* submitted to The Journal of the Astronautical Sciences, June, 2013.

8. John L. Junkins, Ahmad Bani Younes, Robyn M. Woollands and Xiaoli Bai, *Picard Iteration, Chebyshev Polynomial and Chebyshev Picard Methods: Application in Astrodynamics*, accepted in The Journal of the Astronautical Sciences, July, 2013.