A NUMERICAL ALGORITHM FOR SOLVING MANIPULATOR FORWARD DYNAMICS

Hajrudin Pasic
Robert L. Williams II
Chunwu Hui
Department of Mechanical Engineering
Ohio University
Athens, OH 45701

Mechanism and Machine Theory
Vol. 34, pp. 843-855
1999

Contact author information:
Robert L. Williams II
Assistant Professor
Department of Mechanical Engineering
257 Stocker Center
Ohio University
Athens, OH 45701-2979
phone: (740) 593 - 1096
fax: (740) 593 - 0476
email: bobw@bobcat.ent.ohiou.edu
URL: http://www.ent.ohiou.edu/~bobw

A NUMERICAL ALGORITHM FOR SOLVING MANIPULATOR FORWARD DYNAMICS

Hajrudin Pasic[1]
Robert L. Williams II[2]
Chunwu Hui[3]

Department of Mechanical Engineering
Ohio University
Athens, Ohio

ABSTRACT

A new algorithm is presented for iterative solution of systems of nonlinear ordinary differential equations (ODEs) with any order for multibody dynamics and control problems. The collocation technique (based on the explicit fixed-point iteration scheme) may be used for solving both initial value problems (IVPs) and boundary value problems (BVPs). The BVP is solved by first transforming it into the IVP. If the Lipschitz constant is large and the algorithm diverges in a single (‘long') domain, the domain is partitioned into a number of subdomains and the local solutions of the corresponding BVPs are matched either locally (in parallel) or globally. The technique is general and may be applied to general systems of ODEs in any field. As an illustration, the forward dynamics problem of a manipulator is solved as an IVP and then as a BVP.

---

[1] Professor.
[2] Assistant Professor, Corresponding Author.
[3] Graduate Research Assistant.

1. Introduction

When solving multibody dynamics and control problems one is to solve the nonlinear second-order system of ODEs, either as IVPs or BVPs. Many past works have presented efficient methods for solution of the computed torque (inverse dynamics) problem (e.g. [1]). Fewer methods are available for solution of manipulator forward dynamics and most focus on the IVP problem [2,3,4].

One of the most popular methods for solving the general BVP is the shooting method [5]. In the so-called simple shooting, one turns the BVP into the first-order IVP and tries to get the solution based on a set of unspecified initial conditions which are then corrected through an iterative procedure which uses the Newton method, satisfying the boundary conditions. This procedure requires the transformation into state-space form and also requires expensive evaluations of the Jacobian. The success of the procedure depends on a good initial guess.

In problems with large Lipschitz constant, the single shooting fails and one has to use the multiple shooting method [5]. It consists in partitioning the domain into a n small subdomains and matching the local solutions found by the Runge-Kutta method. The matching procedure is based on the Newton method and when applied to a system of m second-order equations results in matrices whose dimension is $2m(n+1)$ and is therefore expensive.

In this paper, we present an alternative procedure for finding both local and global solutions of the BVP recently developed by the first author [6,7], and show how it may be implemented in second-order ODE systems. The approach has much in common with the shooting method. The solutions of the local BVP is also found by transforming the BVP into the IVP. However, the boundary conditions are kept satisfied at all times while the resulting nonlinear system is solved by

fixed-point iteration. In this way, even very complicated boundary conditions may be easily handled and the expensive evaluation of the Jacobian is avoided. The procedure has a lower (linear) convergence rate and somewhat shorter convergence interval. The first problem may be overcome by using some of the well known acceleration techniques [5]. We have found the technique presented in this article very simple, easy to program and very robust. Also, the solution outcome does not depend on the initial guess.

If the system of ODEs to be solved has a large Lipschitz constant, i.e., when solutions in 'long' domains are sought, just as in multiple shooting, the domain is partitioned into a number of shorter ones. Then one solves the BVPs, not IVPs as in the multiple shooting, and does the matching only on the first derivatives. If n subdomains are used and a system of m second-order equations are solved, the resulting Jacobian has dimension $m(n+1)$, half that of multiple shooting.

In what follows we first discuss the forward dynamics problem of manipulators, then the new algorithm for solving BVPs is explained. The forward dynamics of a planar RR manipulator is solved first as an IVP and then as a BVP with this algorithm. Comparison is made with the shooting methods.

2. Manipulator Forward Dynamics Problem

The equations of motion form for a serial manipulator is [8]:

$$M(\Theta)\ddot{\Theta} + V(\Theta, \dot{\Theta}) + G(\Theta) = \tau(t) \tag{1}$$

where $\tau(t)$ is the $N \times 1$ vector of applied joint torques/forces, N is the number of joints, $M(\Theta)$ is the $N \times N$ manipulator inertia matrix, $V(\Theta, \dot{\Theta})$ is the $N \times 1$ vector of centrifugal and Coriolis terms, and $G(\Theta)$ is the $N \times 1$ vector of gravity terms. $\Theta, \dot{\Theta}, \ddot{\Theta}$ are the $N \times 1$ joint angle, rate, and acceleration vectors. The independent variable is time t. For implementing numerical algorithms, equation (1) is often written as:

$$M(\Theta)\ddot{\Theta} = f(t, \Theta, \dot{\Theta}) \tag{2}$$

where $f(t, \Theta, \dot{\Theta}) = \tau(t) - V(\Theta, \dot{\Theta}) - G(\Theta)$. In many existing numerical algorithms for IVP and BVP solutions [e.g. 9,10], equation (2) must be first rewritten as $\ddot{\Theta} = M(\Theta)^{-1} f(t, \Theta, \dot{\Theta})$ and further expressed as a system of first order ODEs. Even though the inertia matrix is symmetric and positive-definite (thus always guaranteeing the existence of the matrix inverse) the symbolic expression for the inverse inertia matrix is quite complex for even the simplest manipulator. The new algorithm presented can solve the forward dynamics problem in the form of equation (2), without the inverse inertia matrix. Also, the new method can handle any order ODEs.

The forward dynamics problem (FDP) solves for the motion $\Theta, \dot{\Theta}, \ddot{\Theta}$ given the forcing functions $\tau(t)$. Depending on the conditions given, the FDP may be solved as either initial value problem (IVP) or the boundary value problem (BVP). The former solves forward dynamics for any

arbitrary time given initial conditions $\Theta_0, \dot{\Theta}_0$, and the latter solves forward dynamics for a specific time interval given a mixture of boundary conditions involving initial and final positions and velocities, i.e. $\Theta_0, \dot{\Theta}_0, \Theta_f, \dot{\Theta}_f$.

Equation (2) is difficult to solve because it is nonlinear, coupled, and multiple solutions may exist to the BVP. Also it often exhibits stiff behavior. Previous work on the FDP of manipulators has focused mostly on the algorithms for IVPs [2,3,4]. The Euler method and Runge-Kutta method have been used to integrate equation (2). This paper studies the algorithms for BVPs of manipulators. The BVP is of practical importance in pick and place tasks and other manipulator applications.

3. New Algorithm for Solving BVPs as IVPs

Let the BVP be solving the system of *m* ODEs

$$A(y)y''(x) = f(x, y, y') \tag{3}$$

in the domain $0 \le x \le e$, with *2m* boundary conditions involving any linear combinations of the values of *y* and y' at *x*=[0, *e*]. If the boundary conditions are nonlinear, they may be linearized. In equations (1) and (2), $x \to t, y \to \Theta, A \to M$. The BVP algorithm presented here consists in converting the BVP into a sequence of IVPs and solving them with the collocation technique [6]. If the domain is long, this algorithm may fail. Similar to the case in which a single shooting must be replaced by a multiple shooting [5], a technique similar to the multiple shooting method is used. The domain is divided into *n* subdomains, and the BVP solution in each of them is found and then these solutions are matched.

3.1 Transformation from BVP to IVP

Define a transformation

$$y(x) = w(x) + \varphi(x) \tag{4}$$

where both $w(x)$ and $\varphi(x)$ are $m \times 1$ vectors, such that

$$\begin{aligned} y'(x) &= w'(x) + \varphi'(x) \\ y''(x) &= w''(x) \end{aligned} \tag{5}$$

i.e. $\varphi''(x)=0$ and therefore, $\varphi(x)=a_0+a_1x$. The *2m* unknown coefficients in the vectors a_0 and a_1 are to be found from the *2m* boundary conditions. A similar transformation may be used in an ODE of any order *k* [6], in which case $\varphi^{(k)}(x)=0$. If $w(x)$ is chosen such that $w(0)=w'(0)=w''(0)=0$, then if the boundary conditions are satisfied, the required 2*m* coefficients depend only on $w(e)$ and $w'(e)$. For example, for the system of the first boundary value problems (FBVPs) with the boundary conditions *y*(0)=*a* and *y*(e)=*b*, where *a* and *b* are *m* ×1 vectors, we find:

$$a_0=a \qquad\qquad a_1=(b-a-w(e))/e \tag{6}$$

Similar to equation (6) for the FBVPs, the transformation formulas for other types of BVPs may be derived [6]. For general linear boundary conditions, the coefficients in $\varphi(x)$ may involve both $w(e)$ and $w'(e)$. Thus, with the transformation defined by equations (4) and (5), the original problem, equation (3), becomes:

$$\begin{gathered}\bar{A}(w,w(e))w''(x)=\bar{f}(x,w,w',w(e),w'(e))\\ w(0)=w'(0)=0\end{gathered} \tag{7a}$$

where $\bar{A}(w,w(e))=A(y)$ and $\bar{f}(x,w,w',w(e),w'(e))=f(x,y,y')$. This transformed problem is an IVP if $w(e)$ and $w'(e)$ are known. Many numerical techniques are available to solve equation (7a). For instance, we may use a variation of the shooting method, in which Newton's method is employed to find $w(e)$ and $w'(e)$ iteratively, and the Runge-Kutta algorithm is used to solve the resulting IVP in each iteration. Additionally, the first author has recently introduced a new algorithm to solve equation (7a), in which the fixed-point iteration is combined with the collocation technique to find

$w(e)$ and $w'(e)$ and solve the resulting IVP [6]. With fixed-point iteration, equation (7a) is written as:

$$\begin{aligned} \overline{A}(w_i, w_i(e))w''_{i+1}(x) &= \overline{f}(x, w_i, w'_i, w_i(e), w'_i(e)) \\ w_{i+1}(0) &= w'_{i+1}(0) = 0 \end{aligned} \tag{7b}$$

Alternatively, equation (7b) may be rewritten into the following form which is more convenient for practical implementation:

$$\begin{aligned} A(y_i)w''_{i+1}(x) &= f(x, y_i, y'_i) \\ w_{i+1}(0) &= w'_{i+1}(0) = 0 \end{aligned} \tag{7c}$$

When using the collocation technique to solve equation (7c), we have the freedom of choosing proper collocation points to achieve higher accuracy. For example, if Gauss points are used, the algorithm will have a global error of O(h^{2s}) [10], where s is the number of collocation points and h is the step size.

In [6] and many standard software algorithms for IVP and BVP, the matrix $A(y)$ must be symbolically inverted to the right-hand-side of equation (3). However, this restriction (which generally results in complicated symbolic terms) is not necessary, Since y_i, y'_i are known from the previous iteration solution, both matrix A and the right-hand-side of equation (7c) are known. $w''_{i+1}(x)$ is then found from equation (7c) using Gaussian elimination.

An initial guess $y_0(x)$ is required which must satisfy all of the prescribed boundary conditions. For example, it may be the solution to the equation $y''_0(x) = 0$, i.e., $y_1(x) = c_0 + c_1 x$. Assume the boundary conditions are $y(0) = a$ and $y(e) = b$; then it can be easily found that $c_0 = a$,

$c_1 = (b - a)/e$. The iteration procedure is repeated until the following terminating criterion is satisfied:

$$\left| y_{i+1}(x_j) - y_i(x_j) \right| \le \varepsilon_1 \tag{8}$$

where i=1, 2, ..., is the iteration index, j =1, 2, ..., s is the collocation point index, and ε_1 is the prescribed tolerance.

3.2 Solving BVPs in Long Domains

If the Lipschitz constant is large, the convergence interval will be short [6]. In such cases, the domain is partitioned into subdomains as shown in Fig. 1.

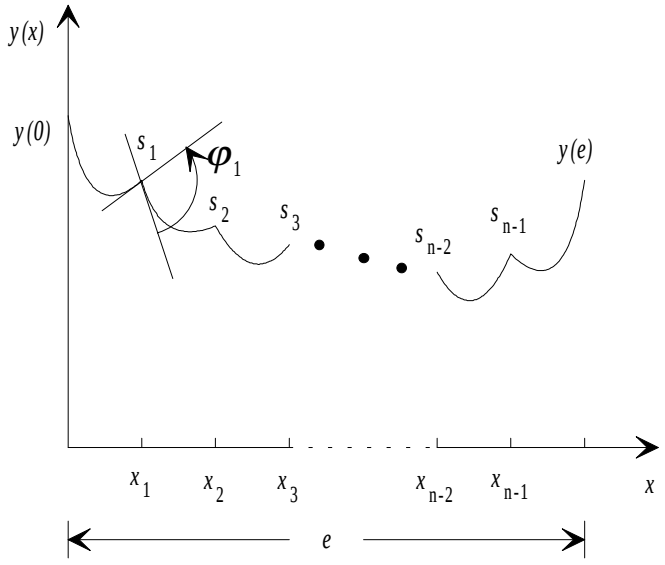

Figure 1. Domain Partitioning

We can try to solve BVP in each of these subdomains with any popular algorithm for BVPs (including the new method presented above) and match the solutions so a smooth curve is obtained

in the whole domain. The matching may be done on either local level or global level. The former, explained in detail in [7], is slower than the latter, but suitable for parallel processing, i.e., both local (subdomain) solutions and their matching may be performed in parallel. The procedure with global matching is explained below.

Assume that the interface values of the solution are s_1, s_2, ..., s_{n-1}, where n is the number of subdomains, then the following n BVPs are formed:

$$\begin{aligned} y'' &= f(x, y, y') && x_{i-1} \le x \le x_i \\ y(x_{i-1}) &= s_{i-1} \qquad y(x_i) = s_i && i = 1, 2, \ldots, n \end{aligned} \tag{9}$$

where $x_0 = 0$, $x_n = e$, $s_0 = a$, $s_n = b$. If s_i, $i = 1, 2, \ldots, n-1$, happen to be the exact solutions at the interfaces, then after solving the BVP in each subdomain, we would get the approximate solution in the whole domain, the slope of which at each interface would be continuous. Otherwise, the curve is non-smooth. In other words, the difference between the right and left derivatives at each interface is not sufficiently small (as shown in Fig. 1). Define the error functions φ_i to be the difference in right and left slopes:

$$\varphi_i = y'_r(x_i) - y'_l(x_i) \qquad i = 1, 2, \ldots, n-1, \tag{10}$$

where $y'_r(x_i)$ and $y'_l(x_i)$ are the right and left derivatives of two neighboring domain solutions at x_i. These errors are functions of the interface values $\mathbf{s} = \{s_1, s_2, \ldots, s_{k-1}\}^T$ only, and they may be written as vector φ:

$$\varphi = \varphi(\mathbf{s}) = \begin{Bmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_{n-1} \end{Bmatrix} = \begin{Bmatrix} \varphi_1(\mathbf{s}) \\ \varphi_2(\mathbf{s}) \\ \vdots \\ \varphi_{n-1}(\mathbf{s}) \end{Bmatrix} \tag{11}$$

Consequently, our problem becomes that of finding the solutions of the BVPs in *n* small subdomains by using any appropriate techniques (including that described in Sec. 3.1) and enforcing the interface values **s** such that $\varphi = 0$. This procedure yields a system of nonlinear equations that may be solved iteratively using Newton's method. The iteration formula is as follows:

$$\begin{aligned} J\Delta\mathbf{s}_i &= -\varphi(\mathbf{s}_i) \\ \mathbf{s}_{i+1} &= \mathbf{s}_i + \Delta\mathbf{s}_i \end{aligned} \tag{12}$$

where *i* is the current iteration number, ***J*** is the Jacobian, and $\mathbf{s}_i$ and $\mathbf{s}_{i+1}$ are the interface values vectors at the *i*-th and (*i*+1)-th iteration, respectively.

In the case of solving a single differential equation, **s** and φ are column vectors of length *n*-1, where *n* is the number of subdomains. ***J*** is a (*n*-1) by (*n*-1) matrix, and its elements are defined as:

$$J(i, j) = \frac{\partial \varphi_i(\mathbf{s})}{\partial s_j} \qquad i, j = 1, 2, ..., n\text{-}1 \tag{13}$$

In general, *J*(*i, j*) is approximated by a finite difference quotient:

$$J(i, j) \cong \frac{\Delta \varphi_i(\mathbf{s})}{\Delta s_j} \qquad i, j = 1, 2, ..., n\text{-}1 \tag{14}$$

where:

$$\Delta\varphi_i(\mathbf{s}) = \varphi_i(s_1,...,s_j+\Delta s_j,...,s_{n-1}) - \varphi_i(s_1,...,s_j,...,s_{n-1}) \tag{15}$$

Δs_j is a small quantity, and its choice has significant effects on the accuracy, convergence rate and stability of numerical methods. The reader is referred to [9,11] for detailed discussion on the rules for choosing Δs_j.

The calculation of $\Delta\varphi_i(\mathbf{s})$ requires $\varphi_i(s_1,...,s_j,...,s_{n-1})$ and $\varphi_i(s_1,...,s_j+\Delta s_j,...,s_{n-1})$ from solutions of the corresponding BVPs. When a specific component s_j of the interface condition is perturbed, only the solutions in the two neighboring subdomains are changed (see Fig. 1). The three corresponding error functions, i.e., φ_{j-1}, φ_j and φ_{j+1} will be perturbed. However, if j=1 (or j=n-1), then only two error functions are perturbed, i.e., φ_1 and φ_2 (or φ_{n-2} and φ_{n-1}). Therefore, the Jacobian is a tri-diagonal matrix. The computation of $\boldsymbol{J}$ may be simplified by means of some techniques [5] to increase efficiency.

In the case of systems of BVPs, the interface value vector **s** and the error vector φ are formed in the following way:

$$\mathbf{s} = (s_{11}, s_{21},..., s_{m1}, s_{12}, s_{22},..., s_{m2},..., s_{1(n-1)}, s_{2(n-1)},..., s_{m(n-1)})^T$$

$$\varphi = (\varphi_{11}, \varphi_{21},..., \varphi_{m1}, \varphi_{12}, \varphi_{22},..., \varphi_{m2},..., \varphi_{1(n-1)}, \varphi_{2(n-1)},..., \varphi_{m(n-1)})^T \tag{16}$$

where m is the number of equations. The first subscript denotes the number of the unknown functions, and the second the number of the interface. For example, s_{23} is the interface condition of

the second unknown function at the third interface, and φ_{23} represents the slope error of the second unknown function at the third interface. The resulting Jacobian is a tri-diagonal matrix with block elements:

$$J = \begin{bmatrix} J_{11} & J_{12} & 0 & \cdots & 0 \\ J_{21} & J_{22} & J_{23} & \cdots & 0 \\ 0 & J_{32} & J_{33} & \cdots & 0 \\ 0 & 0 & J_{43} & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & J_{(n-2)(n-1)} \\ 0 & 0 & 0 & \cdots & J_{(n-1)(n-1)} \end{bmatrix} \tag{17}$$

where:

$$J_{ij} = \begin{bmatrix} \dfrac{\partial \varphi_{1j}}{\partial s_{1j}} & \dfrac{\partial \varphi_{1j}}{\partial s_{2j}} & \cdots & \dfrac{\partial \varphi_{1j}}{\partial s_{mj}} \\ \dfrac{\partial \varphi_{2j}}{\partial s_{1j}} & \dfrac{\partial \varphi_{2j}}{\partial s_{2j}} & \cdots & \dfrac{\partial \varphi_{2j}}{\partial s_{mj}} \\ \vdots & \vdots & \cdots & \vdots \\ \dfrac{\partial \varphi_{mj}}{\partial s_{1j}} & \dfrac{\partial \varphi_{mj}}{\partial s_{2j}} & \cdots & \dfrac{\partial \varphi_{mj}}{\partial s_{mj}} \end{bmatrix} \qquad i, j = 1, 2, \ldots n\text{-}1 \tag{18}$$

The size of the Jacobian is $[m\times(n\text{-}1)] \times [m\times(n\text{-}1)]$. Since it has banded structure, the linear system equation (12) may be efficiently solved by standard algorithms [9]. The matching procedure is terminated when the norm of the error functions is less than a prescribed tolerance ε_2:

$$\|\varphi\| \le \varepsilon_2 \tag{19}$$

4. Solution of Manipulator Forward Dynamics

To illustrate solutions with the proposed method, the two-dof RR planar manipulator of Fig. 2 is used. The forward dynamics problem (FDP) is solved as an IVP by the scheme of [6]. The same problem is then solved as a BVP using the new algorithm and the shooting method.

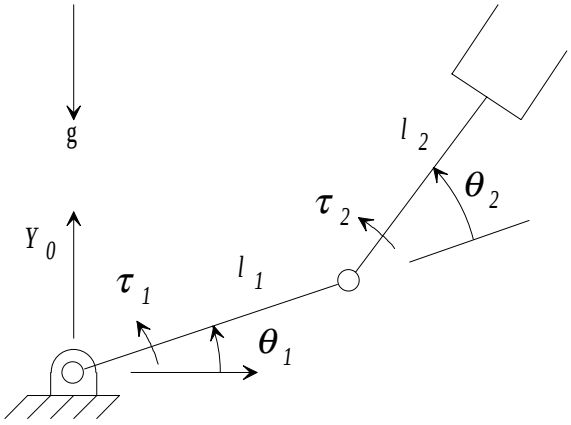

Figure 2. Planar RR Manipulator

4.1 Dynamics Equations of RR Planar Manipulator

Denavit-Hartenberg convention [8] is used. Each of the two links in Fig. 2 has length l_i, mass m_i, and mass moment of inertia about the center of mass I_{i33}, where i=1,2 is now the link index. The vectors which locate the center of mass for each link are l_{c_i}. The external end-effector forces/moments are $f_3 = \left\{ f_{3x} \quad f_{3y} \right\}^T$ and $n_3 = n_{3z}$. Using the Newton-Euler or Euler-Lagrange methods the dynamics equations of motion equation (2) are derived, where $\Theta = \left\{ \theta_1 \quad \theta_2 \right\}^T$ and:

$$M(\Theta) = \begin{bmatrix} m_{11} & m_{12} \\ m_{12} & m_{22} \end{bmatrix} \qquad f(t, \Theta, \dot{\Theta}) = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{20}$$

$$m_{11} = I_{133} + I_{233} + m_1 l_{c_1}{}^2 + m_2 l_1{}^2 + m_2 l_{c_2}{}^2 + 2m_2 l_1 l_{c_2} c_2$$

where: $$m_{12} = I_{233} + m_2 l_{c_2}{}^2 + m_2 l_1 l_{c_3} c_2$$

$$m_{22} = I_{233} + m_2 l_{c_2}{}^2$$

$$f_1 = \tau_1 + 2m_2 l_1 l_{c_2} s_2 \dot{\theta}_1 \dot{\theta}_2 + m_2 l_1 l_{c_2} s_2 \dot{\theta}_2{}^2 - m_2 g l_{c_2} c_{12} - m_1 g l_{c_1} c_1$$
$$- m_2 g l_1 c_1 - n_{3z} - l_1 f_{3x} s_2 - l_1 f_{3y} c_2$$

$$f_2 = \tau_2 - m_2 l_1 l_{c_2} s_2 \dot{\theta}_1{}^2 - m_2 g l_{c_2} c_{12} - n_{3z} - l_2 f_{3y}$$

The assumed parameters are: l_1 = 0.40 *m*, l_{c_1} = 0.20 *m*, l_2 = 0.20 *m*, l_{c_2} = 0.10 *m*, I_{133} = 0.2 *kg-m*^{2}, I_{233} = 0.04 *kg-m*^{2}; m_1 =12 *kg*, m_2 = 6 *kg*; f_3 = [0 0]^T, n_3 = 0; and g = 9.81 m/s^2. The prescribed joint torques τ_1 and τ_2 are shown in Fig. 3.

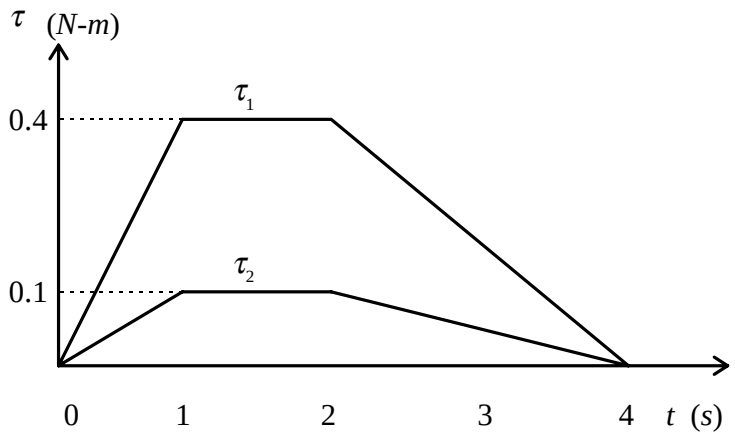

Figure 3. Given Joint Torques

The FDP (for both IVP and BVP) in the current example is stated: Given the input joint torques τ calculate the motion Θ.

4.2 Solving FDP as IVP

Taking the initial conditions to be $\Theta(0) = \{0 \quad 0\}^T$ and $\dot{\Theta}(0) = \{0 \quad 0\}^T$, uniform subdomain length 0.5, and the number of iterations i_{MAX}=5, equation (2) is solved in the time domain [0, 4]. The FDP solution is shown in Fig. 4. To avoid high acceleration which may result in a stiff problem and great difficulty in numerical solution, the joint torques were chosen to be relatively small. Thus the negative vertical gravity vector has a significant effect on the solution, and the motion of the links is similar to that of a double pendulum. The same IVP was also solved with the Runge-Kutta method, and the same solution of Fig. 4 was obtained.

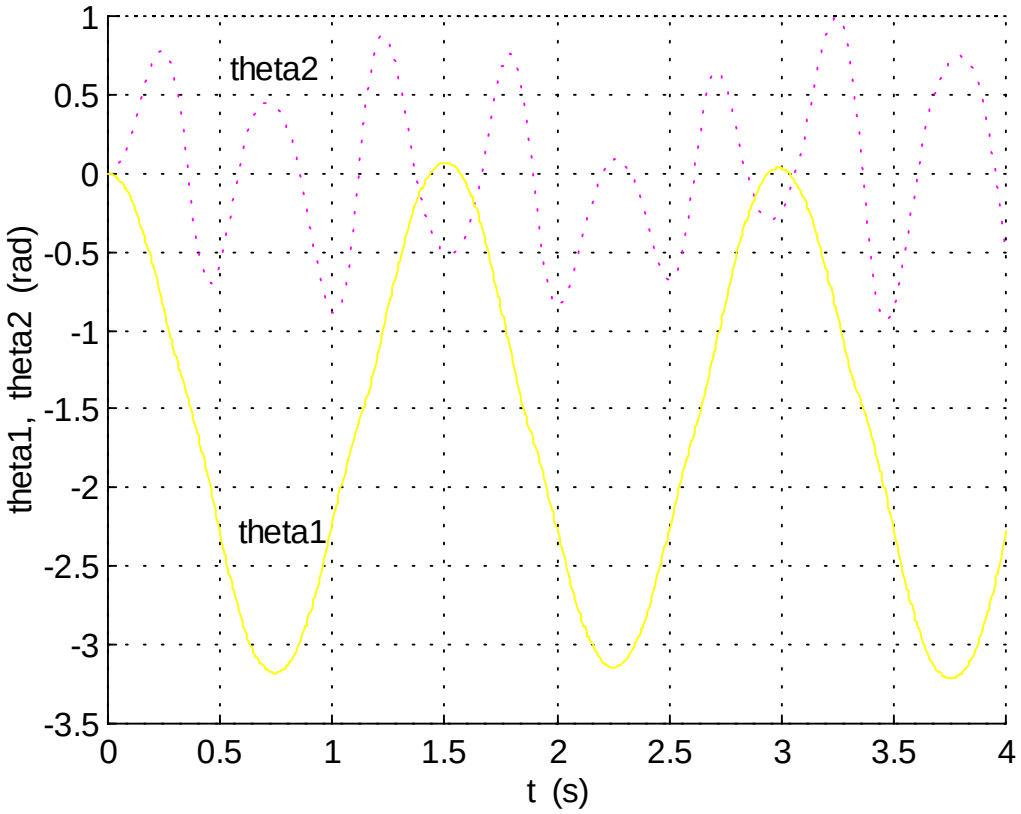

Figure 4. Solution to Equation (2) as IVP

4.3 Solving FDP as BVP

Using the new algorithm, the FDP is solved in the time domain [0, 0.3] with the boundary conditions $\Theta(0) = \{0 \quad 0\}^T$ and $\Theta(0.3) = \{-1.1525 \quad 0.5915\}^T$ (*rad*) where $\Theta(0.3)$ came from the IVP solution. The result is the same as that of the IVP in the same domain (a small portion of Fig. 4). If we solve the BVP in the domain [0, 0.35], a solution different from the IVP result is obtained as shown in Fig. 5. We can prove that this is also a solution to the BVP in the following way. After solving the BVP, we know $\dot{\Theta}(0)$. With this slope, we solve an IVP, and get the same solution in [0, 0.35] as that of the BVP. This example clearly illustrates the existence of multiple solutions, typical

of BVPs and nonlinear problems. For a longer domain, the solution diverges. This is because the convergence is guaranteed only when the domain length is within a certain value as required by the Lipschitz condition [6].

4.4 Solving FDP as BVP in Long Domains

Since the first approach did not converge in a longer domain, the domain partitioning and solution matching scheme have to be implemented. The problem is now solved in the domain [0, 1] with boundary conditions $\Theta(0) = \{0 \quad 0\}^T$, $\Theta(1) = \{-2.2332 \quad -0.8835\}^T$. where $\Theta(1)$ came from the IVP solution. The domain is divided into 20 subdomains of equal size and each of them has 3 Gauss collocation points. The solution of the BVP is the same as that of the IVP (the [0, 1] portion of Fig. 4). The problem has also been solved in the domain [0, 1.5]. However, in this case, the solution is different from that obtained from the IVP, as shown in Fig. 6. It is proved to be one of the multiple solutions in the manner described above. The method fails to solve the BVP in a longer domain. We notice that the derivatives of the solution of the IVP corresponds to very steep slope. This may imply that we have a stiff system, which needs very small subdomain size if the solution is possible with this method. Another reason is that Newton's method is employed to solve systems of nonlinear algebraic equations, which is convergent only if we have a good initial guess.

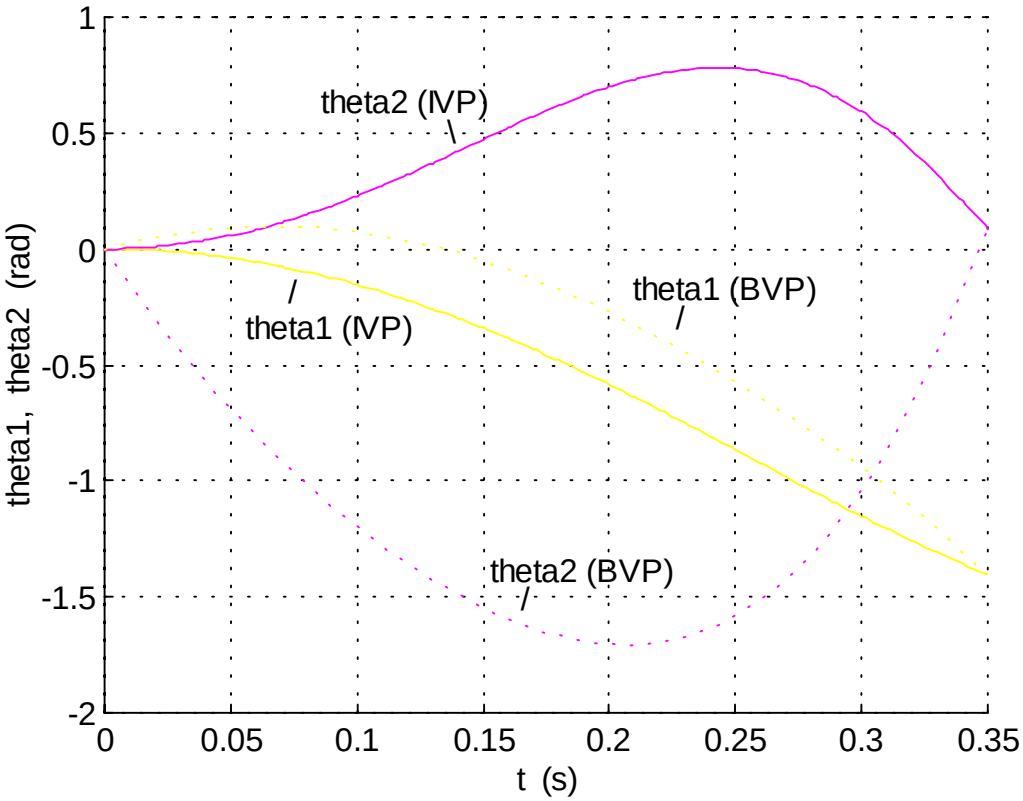

Figure 5. Multiple Solutions for BVP

The shooting method was used to solve the same BVP. The problem could be solved only in the time domain [0, 1] and the results are identical to the IVP results in the [0, 1] portion of Fig. 6 (same as [0, 1] portion of Fig. 4). Beyond this domain the solution always diverged using shooting.

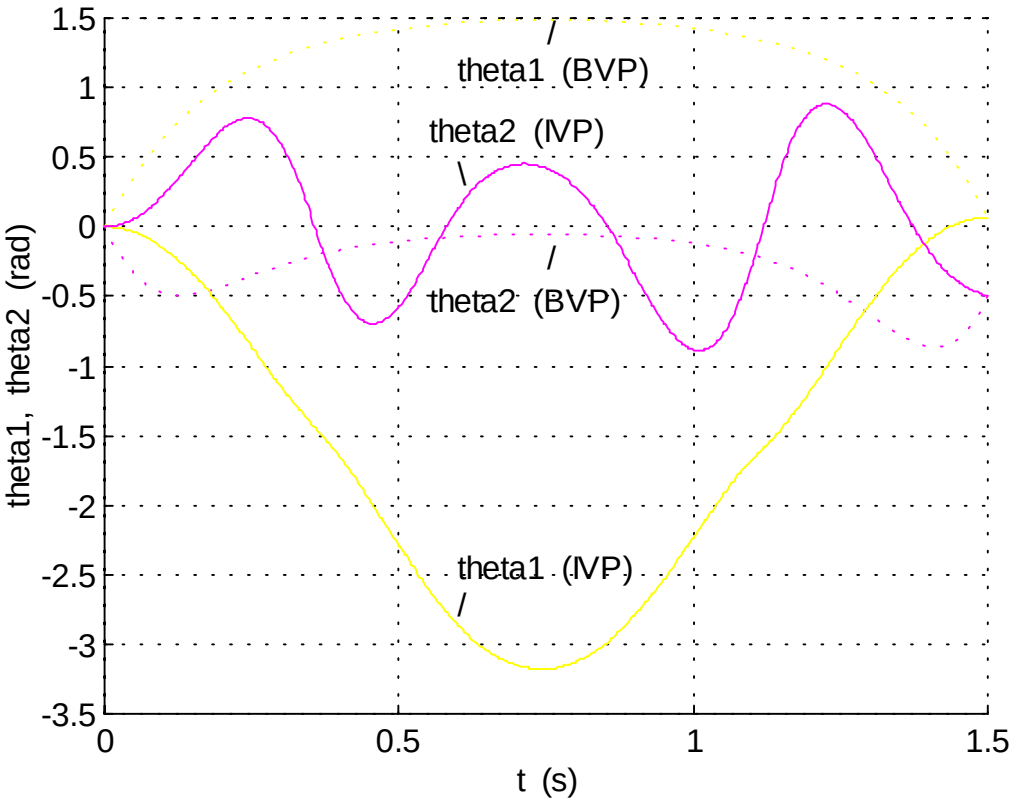

Figure 6. Solution of BVP with Domain Partitioning

5. Comparison with the Shooting Method

It is difficult to make quantitative comparison between different numerical methods. Therefore we discuss some aspects of these methods from the point of view of their practical implementation.

As explained above, the algorithm presented here solves the BVP by transforming it into IVPs. The number of function evaluations required for BVP is the same as that for IVP if the same number of iterations and steps are used. In the simple shooting method, however, more function evaluations are needed for calculating the Jacobian. Another good feature is that various boundary conditions can be conveniently incorporated. For different boundary conditions, only the corresponding transformations from BVP to IVP are different.

Some standard multiple shooting algorithms [e.g. 12] require that the analytical expression of the Jacobian be provided by the user, which is very difficult if not impossible for a complicated BVP involving the equations of motion of a manipulator. We use a multiple shooting program which evaluates the Jacobian numerically.

In both the new BVP algorithm and the multiple shooting method, the domain has to be partitioned into small subdomains, and the solution in each of them is found and matched. The major difference between them is that the new algorithm solves the original systems as BVPs in subdomains and matches these solutions according to the criterion of derivative continuity, while multiple shooting solves IVPs and matches the solutions with the requirement of function continuity. Consequently, for a system of second-order BVPs, the Jacobian for the multiple shooting has a dimension twice that in our algorithm. Also, this new algorithm may implement the matching either on the local or global level [7]. The former will be advantageous if parallel processing is used. Any

valid BVP algorithm (including the new algorithm presented) may be used to find the solution in each of the subdomains.

6. Conclusion

A new algorithm has been introduced for solving forward dynamics problems of manipulators. A planar RR manipulator is studied to validate the algorithm against the popular shooting method.

When solving a BVP without partitioning the domain, we find the new algorithm more efficient than the simple shooting because no Jacobian is required. This is especially true for systems with a large number of equations. Also, the algorithm handles various boundary conditions with ease. The weak point of this algorithm is its somewhat shorter domain of convergence.

When forward dynamics problems of manipulators are to be solved as BVPs over long domains, the domain has to be partitioned into subdomains, and the problems are solved either as multi-point IVPs (multiple shooting method) or as multi-point BVPs (the method presented). Both schemes use Newton's method to improve the solution iteratively and thus have to evaluate the Jacobian. However, the size of the Jacobian in the former is twice as large as that in the latter.

As illustrated, multiple solutions may arise in non-linear ODE BVPs. Typically this occurs when the BVP is solved in a domain whose length is close to certain critical value. Solution in a longer domain will fail, no matter how small the step size is and how many subdomains are used. This may be due to the stiff behavior of the manipulator system and/or divergence of Newton's method. These problems will be addressed in future development of this algorithm.

References

1. C.J. Li and T.S. Sankar, *Mechanism and Machine Theory*, 27 (6), 741 (1992).

2. B. Heimann and H. Loose, *Mechanism and Machine Theory*, 25 (6), 655 (1990).

3. K.W. Lilly, *Efficient Dynamic Simulation of Robotic Mechanisms*, Kluwer Academic Publishers (1993).

4. J. Angeles, *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms*, Springer, (1997).

5. J. Stoer and R. Burlisch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, (1993).

6. H. Pasic, "*Solving Boundary-Value Problems in ODEs by a Simple Transformation into the Initial-Value Problems*", to appear in Mathematical Modeling and Scientific Computing, 8 (1997).

7. H. Pasic and Y. Zhang, "*Parallel Solutions of BVPs in ODEs Based on Local Matching*", 8^{th} SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis (1997).

8. J.J. Craig, *Introduction to Robotics: Mechanics and Control*, Addison-Wesley Publishing Company, Inc. (1989).

9. K.E. Brenan, S.L. Campbell and L.R. Petzold, "*Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*", SIAM (1996).

10. E. Hairer and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problem*, Springer, (1996).

11. J.E. Dennis Jr. and R.B. Schnabel, "*Numerical Methods for Unconstrained Optimization and Nonlinear Equations*", SIAM (1996).

12. IMSL Inc., *IMSL User's Manual*, (1992).