中華民國八十六年全國計算機會議

Multibody 機械系統運動學模擬器
A Simulator for the Kinematic of Multibody Mechanical Systems

周建群
Chou Chien-Chun

淡江大學資訊工程研究所
Tamkang University, Taipei, Taiwan, ROC
chou@hawkeye.cs.tku.edu.tw

蔡妙玲
Thai Miao-Ling

淡江大學資訊工程研究所
Tamkang University, Taipei, Taiwan, ROC
crs@hawkeye.cs.tku.edu.tw

陳士農
Chen Shyh-Nong

淡江大學資訊工程研究所
Tamkang University, Taipei, Taiwan, ROC
csnong@hawkeye.cs.tku.edu.tw

摘要

由於許多機械系統可以用 multibody 機械系統的模型去模擬，因此我們希望建立一套完整的 multibody 機械系統運動學模擬器的環境。並且當系統處於 under constraint 的情況下，模擬器仍然可能求出一組最爲平滑之模型的運動情形。在本論文中將介紹以電腦模擬方法來分析 multibody 機械系統的運動方式。

關鍵字：機械系統,限制等式,自動微分,微分矩陣

Abstract

Robot systems, vehicles, and human bodies can all be modeled as multibody mechanical systems. The kinematics and dynamics' behavior of these mechanical systems can also be analyzed by using time-driven continuous simulation technique. In this thesis, we have designed and constructed a Multibody system Kinematics Simulator (MKSim) to assist the modeling process and analyzing process of a multibody mechanical system. Not only can MKSim analyze a fully constraint system, but also can find the smoothest kinematics movement (if there is one) for an under-constraint model.

Keywords: multibody, constraint, triad, joint, driver, automatic differential, jacobian, hessian

一、概論

在真實世界中有許多機械系統，例如：機械手臂、汽車機械主體、人體骨架、卡通動畫的設計過程....等都可以用 multibody 機械系統的模型去模擬。因此我們希望建立一套完整的 multibody 機械系統運動學模擬器的環境。在我們的 multibody 機械系統運動學模擬器（簡稱 MKSim：Multibody system Kinematic Simulation)中，使用者可以很容易的將 multibody 機械系統中四項之基本元件：rigid body、triad、joint 以及 driver，依據實際的需要組合成各種模型[1]。在 MKSim 的環境中，採用 lex 與 yacc 所產生的 parser 來剖析輸入檔案[2]，並將輸入檔案轉換成數學公式，即各種限制等式(constraint equations)來表示模型的架構，並用自動微分系統的記錄器，以 C++ operator overloading 的方式來記錄限制等式的運算過程。接著用數值分析的方法來分析模型的運動行爲。在做運動學分析時，則採用了自動微分系統的運算器，根據自動微分系統的記錄器所記錄結果來計算運動學分析時所需要的微分矩陣。最後用 C++ builder 所寫成的動畫顯示器來顯示模擬結果。除此之外，當系統處於 under constraint 的情形下，也就是說模型可能產生眾多種的運動行爲中，MKSim 仍然可能分析出一組最爲平滑的模型可能之運動行爲。此項功能是 MKSim 的重要突破之一。

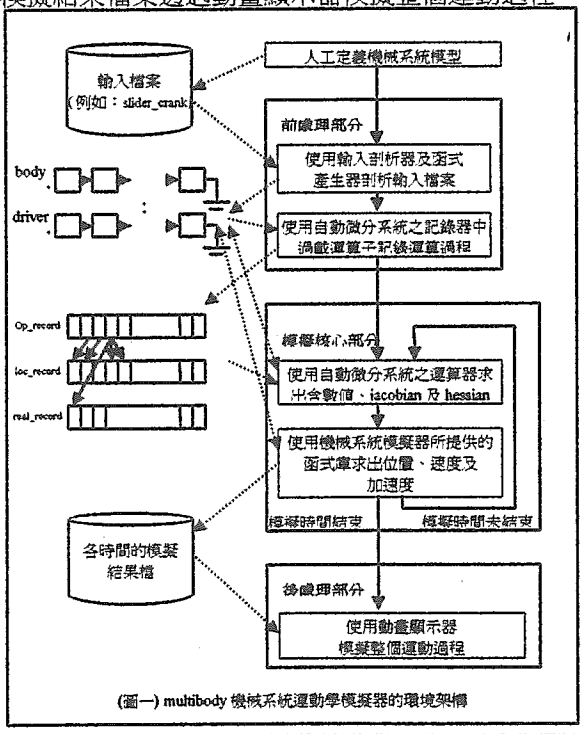
對於一個輸入模型而言，是以 rigid body(以下簡稱 body)來組成此模型的基本骨架，再用連接各 body 的各種型態之 joint 來限制 body 間的運動方式。接著用 triad(座標系統)來定義 body 與 joint 在空間中的位置和方向：代表 body 在空間中的位置和方向，是用附著在各 body 上的重心座標系統(cg triad) 來表示；代表 joint 在空間中的位置和方向，是用相對於各 body 的 cg triad 且分別附著在此 joint 所連接的兩個 body 上的 joint 座標系統(joint triad) 來表示。最後再加上與時間有關的 driver 來使此機械系統產生與時間相關的運動。

在 MKSim 環境中將各 body 的 normal constraint、地板所產生的 ground constraint、各 joint 所產生的 joint constraint 及與時間相關的 driving constraint 等，組合而得到模型之總限制等式。接著藉由運動學分析的函式庫和自動微分系統的運算，來從這些已知的限制等式中，求得各 body 在不同時間其重心參考座標所在的位置、速度及加速度。在 MKSim 環境系統中目前並不考慮 body 受力的影響，所以只能稱爲運動學(Kinematic)的模擬。如果再加上力的考慮，則可稱爲動力學(Dynamic)的模擬[1]；這是以後擴充的重要目標之一。

二、模擬器環境架構

MKSim 的整體環境架構如(圖一)所示，它包含了由輸入檔案剖析器(input parser)及函式產生器(function generator)及自動微分系統之記錄器所組成的前處理部分。加上由自動微分系統之運算器、機械系統函式庫及運動學分析迴圈所組成的模擬器核心部分。最後則是由動畫顯示器將模擬結果做動畫顯示的後處理部

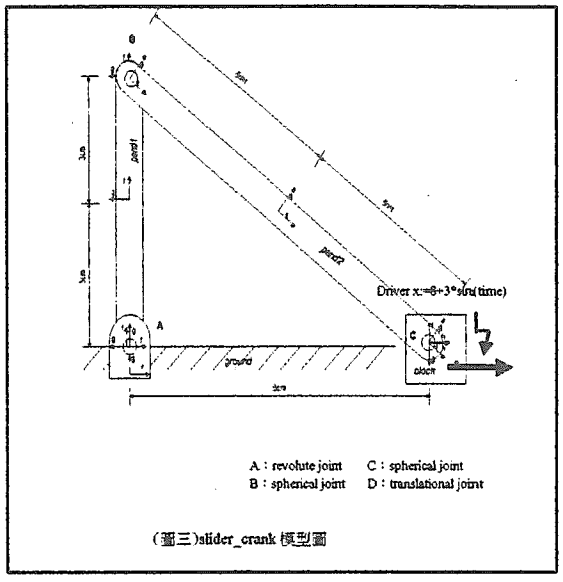
是由動畫顯示器將模擬結果做動畫顯示的後處理部分。在執行的流程方面：我們先用模型定義語言定義機械系統模型的輸入檔案，(圖二)爲 slider_crank 的模型，與其相對應的輸入檔案如(圖三)所示。此檔案經由前處理部分之輸入剖析器及函數產生器，將輸入檔案經過適當的轉換成機械系統模型內定的資料結構，並利用這些資料結構中的值，計算出限制等式的值，且透過自動微分系統之記錄器用 C++ operator overloading，將要記錄的變數設爲 adouble 的類別，並將相關於 adouble 類別的變數運算過程記錄下來。在模擬器的核心部分則利用前處理部分自動微分系統記錄器所記錄的結果，配合機械系統模擬器所提供函式庫(例如：linear solver、Newton iteration) 的運算及自動微分系統之運算器的運算求得各 body 所在的位置、速度及加速度。MKSim 會把每一間隔時間計算出來的模擬位置、速度及加速度記錄在模擬結果檔案中，最後把模擬結果檔案透過動畫顯示器模擬整個運動過程。



(圖一) multibody 機械系統運動學模擬器的環境架構

```
input file : slider_crank
MODEL slider_crank
SYSTEM
 ( KINEMATIC ANALYSIS, starting time = 0.0, ending time = 8.0,
   print interval = 0.1, lu tolerance = 0.00000000001,assembly tolerance = 0.001,);
   BODY G1 ( ground );
   BODY pend1 ( center of gravity = (0,3,0), pqr = [ (0,3,0),(0,3,1),(0,4,0) ] );
   BODY pend2 ( center of gravity = (4,3,0), pqr = [ (4,3,0),(4,3,1),(8,0,0) ] );
   BODY block ( center of gravity = (8,0,0), pqr = [ (8,0,0), (8,0,1),(9,0,0) ] );
   triad rev1ground
   ( associated body = g1, origin = (0,0,0), pqr = [(0,0,0),(0,0,1),(1,0,0)]);
   triad rev1pend1
   ( associated body = pend1, origin = (-3,0,0), pqr = [(-3,0,0),(-3,0,1),(-2,0,0)]);
   triad sph1pend1 ( associated body = pend1, origin = (3,0,0), pqr = [(3,0,0),(3,0,1),(4,0,0)] );
   triad sph1pend2 ( associated body = pend2, origin = (-5,0,0), pqr = [(-5,0,0),(-5,0,1),(-4,0,0)] );
   triad sph2pend2 ( associated body = pend2, origin = (5,0,0), pqr =[(5,0,0),(5,0,1),(6,0,0)] );
   triad sph2block ( associated body = block, origin = (0,0,0), pqr = [(0,0,0),(1,0,0),(0,1,0)] );
   triad tran1block ( associated body = block, origin = (0,0,0), pqr = [(0,0,0),(1,0,0),(0,1,0)] );
   triad tran1g1    ( associated body = g1, origin = (0,0,0), pqr = [(0,0,0),(1,0,0),(0,0,1)] );
   revolute joint rev1 ( triad = rev1ground, triad = rev1pend1 );
   spherical joint sph1 ( triad = sph1pend1, triad = sph1pend2 );
   spherical joint sph2 ( triad = sph2pend2, triad = sph2block );
   translation joint tran1 ( triad = tran1block , triad = tran1g1 );
   driver d1 ( blockcx = 8.00 + 3*sin( TIME) )
ENDMODEL
```

(圖二) slider_crank 模型的輸入檔案



(圖三)slider_crank 模型圖

三、MKSim 的核心與實作

在 MKSim 環境中，是使用一組變數來決定所有的 body 在空間上的位置及方向，這組變數稱爲 generalized coordinates[1]。當模型產生運動時，generalized coordinates 也會隨著時間做改變。我們通常用向量 $q=[q_1,q_2,\cdots,q_{nc}]^T$ 來描述出整個機械模型的狀態，其中 nc 代表在系統中 generalized coordinates 的總數。由於每個 body 用其重心的原點 x,y,z 三個變數來代表 body 在空間中的位置，以及用尤拉參數方向座標表示法中 e_0,e_1,e_2,e_3 等四個參數來代表 body 在空間中的方向，共七個變數來代表 body 在空間中的位置與方向，所以 nc 等於 7 乘上 body 的個數。如果想唯一的去決定所有 generalized coordinates 的值，就必須要有 nc 個限制等式。如果模型中有 ne 個 normal constraint，nj 個 joint constraint，ng 個 ground constraint 和 nd 個 driving constraint，若此時 ne+nj+ng+nd=nc 則可以解出唯一之 generalized coordinates 的值；如果(ne+nj+ng+nd)<nc 時，表示此模型有 nc-(ne+nj+ng+nd)的自由度，這種情況稱 under constraint，在 MKSim 的環境中能針對 under constraint 的情形求出一組最爲平滑的解(如果有解存在的情況)。此項功能是本論文的重要突破之一。

下面的各節中將深入介紹在 MKSim 環境中各種限制等式的數學原理以及如何利用這些限制等式，配合自動微分系統以及機械系統模擬器函式庫的運算(例如：利用 Newton iteration 的方法解非線性函式，以及在 under constraint 的情形下用 Gaussian 消去法解線性函式)，求得各 body 在不同時間所在的位置、速度以及加速度的方法。

3.1 限制等式(constraint equation)

由於 MKSim 採用尤拉參數方向座標表示法中 e_0,e_1,e_2,e_3 等四個參數來代表 body 在空間中的方向，因

此產生了 normal constraint，我們用 Φ^{nrml} 來代表，它四個參數的平方和等於 1

$$\Phi^{nrml} \equiv e_0^2 + e_1^2 + e_2^2 + e_3^2 - 1 = 0$$

另外一種關於 body 的限制是產生在地板(ground body) 上，稱作 ground constraint。它代表 x，y，z，e_1，e_2，e_3 六個參數皆不隨時間改變：

$$\Phi^1 = x - x^O = 0 \quad \Phi^2 = y - y^O = 0 \quad \Phi^3 = z - z^O = 0$$

$$\Phi^4 = e_1 - e_1^O = 0 \quad \Phi^5 = e_2 - e_2^O = 0 \quad \Phi^6 = x_3 - e_3^O = 0$$

其中 x，y，z，e_1，e_2，e_3 代表此 ground body 在任何時間 generalized coordinates 的值，而 x^O、y^O、z^O、e_1^O、e_2^O、e_3^O 則是代表此 body generalized coordinates 的初始值。另外再加上每個 body 皆有的 normal constraint，則 e_0 的值也是固定的。即 ground body 之 generalized coordinates 的七個參數皆爲固定值。

發生在 joint 的上的限制爲 joint constraint。以下我們用 P 點來表示 joint 在空間中的位置；用 f、g、h 三個互相垂直的單位向量來表示 joint 在空間中的方向。在 MKSim 中提供了五種基本 joint constraint[1]，它是根據不同型態的 joint 而產生相對應的 joint constraint。第一種基本 joint 是 spherical joint，它是指此 joint 所連接的兩個 joint triad P_i 與 P_j 的原點是共點，也就是說有三個旋轉的自由度，符合 spherical constraint：

$$\Phi^S(P_i, P_j) \equiv P_i - P_j = \begin{bmatrix} x_i - x_j \\ y_i - y_j \\ z_i - z_j \end{bmatrix} = 0$$

所以此 joint 之 constraint 的總數爲 3。第二種基本 joint 是 universal joint，它是指此 joint 所連接的兩個 joint triad P_i 與 P_j 的原點是共點且 h 軸垂直，符合 $\Phi^S(P_i, P_j) = 0$ 且 $\Phi^{d1}(h_i, h_j) = 0$，其中 $\Phi^{d1}(h_i, h_j) \equiv h_i^T h_j = 0$，所以此 joint 之 constraint 的總數爲 4。第三種基本 joint 是 cylindrical joint，它是指此 joint 所連接的兩個 joint triad 之間有共同的旋轉軸(h 軸)，且在這個軸上 body 之間可以做相對的移動與轉動，即有一個旋轉的自由度與一個位移的自由度。也就是符合：

$$\Phi^{p1}(h_i, h_j) \equiv \begin{bmatrix} \Phi^{d1}(f_i, h_j) \\ \Phi^{d1}(g_i, h_j) \end{bmatrix} = 0$$

$$\Phi^{p2}(h_i, d_{ij}) \equiv \begin{bmatrix} \Phi^{d2}(f_i, d_{ij}) \\ \Phi^{d2}(g_i, d_{ij}) \end{bmatrix} = 0$$

其中 d_{ij} 代表 body i 上的一點 P_i 到 body j 上的一點 P_j 的距離向量。所以此 joint 之 constraint 的總數爲 4。第四種基本 joint 是 revolute joint，它是指此 joint 所連接的兩個 joint triad 之間有共同的旋轉軸(h 軸)，並且只能在同一個平面(f、g 平面)上做相互轉動，因此只有一個旋轉的自由度。也就是說此 joint 所連接的兩個 joint triad P_i 與 P_j 的原點共點且 h 軸平行，即符合 $\Phi^S(P_i, P_j) = 0$ 且 $\Phi^{p1}(h_i, h_j) = 0$，所以此 joint 之 constraint 的總數爲 5。最後一種基本 joint 是 translational joint，它是指此 joint 所連接的兩個 joint triad 只能在同一個軌道(h 軸) 上做相對的移動，因此只有一個位移的自由度，即符合 $\Phi^{p1}(h_i, h_j) = 0$、$\Phi^{p2}(h_i, d_{ij}) = 0$ 以及 $\Phi^{d1}(f_i, f_j) = 0$。所以此 joint 之 constraint 的總數爲 5。然而在真實的模型中還有許多複雜的 joint，這將是以後擴充的重要範圍之一。

在 MKSim 中最後一種限制是與時間有關的 driving constraint，它是根據不同型態的 driver 而產生相對應的 driving constraint。下面分別介紹三種基本 driver 的特性以及相對應的 driving constraint，其中 $f(t)$ 表示一個與時間相關的函數。第一種基本 driver 是 absolute driver，它是指在 body 上的某一個軸(X、Y 或 Z)是一個時間的函數，與它相對應的 driving constraint 爲：$\Phi^{abs}(a,t) \equiv a - f(t) = 0$，其中 a 向量代表 X、Y 或 Z 中的任一個軸。第二種基本 driver 是 distance driver，它表示兩個 joint triad 之間的距離是一個時間的函數，與它相對應的 driving constraint 爲：$\Phi^{dist}(d_{ij}, t) \equiv d_{ij}^T \cdot d_{ij} - (f(t))^2 = 0$。最後一種基本 driver 是 angle driver，它是指兩個 triad 之間某一軸(f、g、h) 的夾角 θ 是一個時間的函數，與它相對應的 driving constraint 爲：$\Phi^{ang}(\theta, t) \equiv \theta - f(t) = 0$。

接下去我們以 slider_crank 模型的例子來介紹如何利用這些限制等式求得各時間 body 的位置、速度與加速度的方法。

3.2 各時間 body 位置、速度與加速度的方法

在 MKSim 環境中根據輸入的模型檔案可以得到各 body 初始的位置及方向，這些位置及方向經轉換成統一的 generalized coordinates，成爲自動微分系統中 independent 的值，我們用數學符號 q 向量 $q \equiv [q_1, q_2, \ldots, q_{nc}]$ 來表示所有的 independent 的值。其中 nc 代表所有 independent 的總數，由於每個 body 皆以七個參數(即 x，y，z，e_0，e_1，e_2，e_3) 代表其位置及方向，所以 nc=7*nb(其中 nb 代表 body 的總數)。我們用數學符號 Φ 向量來表示所有限制等式的值。當時間 t 等於某一個值時，我們想要求出 $\Phi(q,t) = 0$ 中所有 q 值的解。我們以 slider_crank 模型的例子來介紹如何利用這些限制等式求得各時間 body 的位置、速度與加速度的方法。在 slider_cranker 中有四個 body，它們別是 ground、pend1、pend2、以及 block。所以模型的 generalized coordinates 的總數爲 28。下面我們將 generalized coordinates 的總數(nc)、constraint 的總數(nh) 與自由度(DOF)的個數，整理成下列的表：

Model：slider_crank	
Bodies	nc=28
4 個 body	
Constraints:	
Revolute joint(ground ,pend1)	5
Spherical joint(pend1 , pend2)	3
Spherical joint(pend2 , block)	3
Translational joint(block , ground)	5
Ground constraint	6
Normal constraint	4
Absolute driving constraint(block:x=8+3sin(t))	1
自由度（DOF）=28-27=1	nh=27

在這個例子中有 28 個 generalized coordinates，也就是說有 28 個未知數；有 27 個限制等式也就是說有 27 個方程式。這種方程式個數少於未知數個數的情況稱爲 under constraint。在一般的機械系統模擬器中只能針對 full constraint 的情形求解，也就說方程式個數等於未知數個數。然而在 MKSim 的模擬器中不但能針對 full constraint 的情形求解，也能針對 under constraint 的情形求出一組最爲平滑的解(如果有解存在的情況)。由於 $\Phi(q,t)=0$ 是屬於非線性等式(nonlinear equations)，所以我們利用 Newton iteration 的方法來求解。所得到的解即是 body 在時間 t 的位置和方向。Newton iteration 求解的步驟[1]：

$$\Phi_1(q)=\Phi_1(q_1,q_2,.....,q_n)=0$$
$$\vdots$$
$$\Phi_m(q)=\Phi_n(q_1,q_2,.....,q_n)=0$$

1.首先我們估計初始值 $q^{(0)}$當作的方程式的解。

2.反覆以 i=0,1,2,....代入,用自動微分系統計算$\Phi(q^{(i)})$和$\Phi_q(q^{(i)})$的值。假如當下列兩個式子的值小於 equation error tolerance ε_e 和 solution error tolerance ε_s 時，就結束所有的步驟。

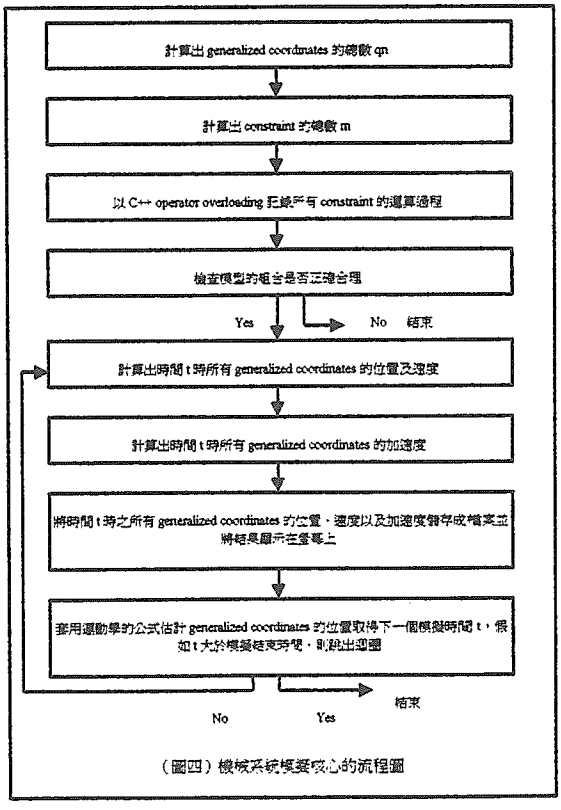
$$\left|\Phi(q^{(i)})\right|\le\varepsilon_e \quad \left|q^{(i)}-q^{(i-1)}\right|\le\varepsilon_s$$

3.代入 $\Phi_q(q^{(i)})\Delta q^{(i)}=-\Phi(q^{(i)})$ 的公式，解出$\Delta q^{(i)}$ 的值，並求出新的$q^{(i+1)}=q^{(i)}+\Delta q^{(i)}$回到步驟 2，並以 i+1 代替 i。

在 Newton iteration 求解的過程中，Jacobian 矩陣 $\Phi_q(q^{(i)})$ 和限制等式$\Phi(q^{(i)})$ 的值皆爲已知數，而$\Delta q^{(i)}$爲未知數。由於解$\Delta q^{(i)}$：$\Phi_q(q^{(i)})\Delta q^{(i)}=-\Phi(q^{(i)})$是屬於線性等式可以用 linear solver 的方式解出$\Delta q^{(i)}$。當$\left|\Phi(q^{(i)})\right|$趨近 0 且$q^{(i)}-q^{(i-1)}$也趨近 0 時，則找到 $q^{(i)}$爲$\Phi(q)=0$之近似解。求出$\Phi(q,t)=0$的解q^*，而q^*就是各 body 在時間 t 的位置。接著我們在等式$\Phi(q,t)=0$兩邊同時對時間 t 做微分得到速度等式如下：$\dot{\Phi}=\Phi_q\dot{q}+\Phi_t=0$ *or* $\Phi_q\dot{q}=-\Phi_t$。在上式中我們可以用自動微分系統算出Φ_q與Φ_t的值，所以未知數只有$\dot{q}$，而$\dot{q}$也就是各 body 在時間 t 的速度。由於速度等式是屬於線性等式所以我們也用 linear solver 求$\dot{q}$。接著，我們在速度等式的兩邊同時對時間 t 做微分，得到加速度等式如下：$\Phi_q\ddot{q}=-(\Phi_q\dot{q})_q\dot{q}-2\Phi_{qt}\dot{q}-\Phi_{tt}$。我們稱矩陣$(\Phi_q\dot{q})_q$爲 Hessian 矩陣。與速度等式相同的道理，在加速度等式中未知數只有$\ddot{q}$，我們也可以用 linear solver 的方式求出$\ddot{q}$，就是各 body 在時間 t 的加速度。

最後將模擬器核心的運作流程整理如（圖四）所示。在模擬器核心運作完成後，MKSim 會將所儲存的模擬結果檔案給動畫軟體作爲輸入資料，再利用動畫顯示軟體將模擬結果以動畫方式顯示在螢幕上。此外，在 MKSim 的環境中記錄了一些評估效能的參數。例如：記錄了使用 Newton iteration 的方法中每一次及總共花費多少迴圈才求出解、使用自動微分系統運算的次數有多少、使用 linear solver 求解的次數有多少等資料，以作爲評估模擬的效能之依據。

在求各 body 在時間 t 的位置、速度與加速度時，會用到 linear solver。接下去將介紹如何在 under constraint 的情形下解線性等式解的方法。

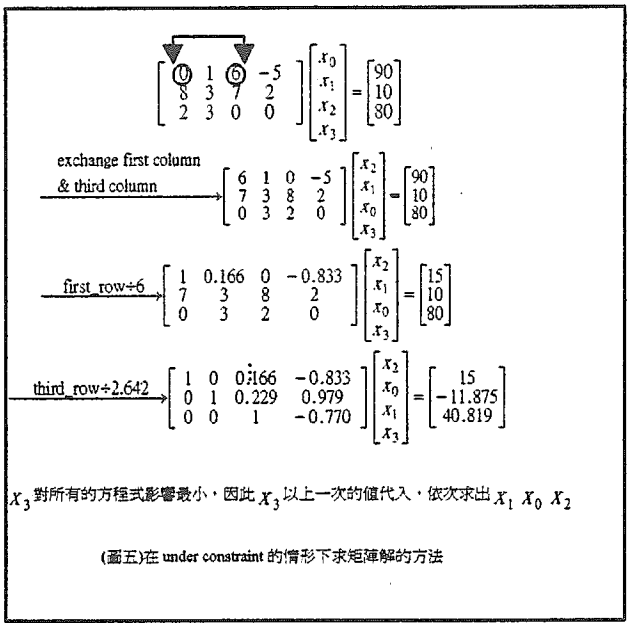


（圖四）機械系統模擬核心的流程圖

3.3under constraint 的情形下求線性解的方法

在 MKSim 中對於在 under constraint 的情形下求出線性等式 Ax=b 中的 x 的值，它所使用的計算原理與 Gaussian 消去法類似。我們以(圖五)的矩陣說明運算的

過程，在轉換成下三角形的項目爲零的步驟中，我們將每一個 row 中絕對值最大的 element 與對角線的 element 做交換，再將對角線的 element 化爲 1。接著再運用 Back Substitution 的方法從最後一個 row 反代回去求出未知數 x 的值。在 column 編號大於 row 編號之子矩陣中未知數的部分，在運算交換的過程中它們被移至矩陣的右邊，表示它們對所有的限制等式影響力最小(insignificant)的未知數，也就是說其變化量最少，我們將這些未知數以原值代回矩陣求解。由於在圖五的例子中有一個自由度，所以我們可以選擇固定任一個未知數的值，則可以求出其它未知數的解。然而我們選擇固定的未知數是變化量最少的未知數，因此所得到的解是最爲平滑的解。



(圖五)在 under constraint 的情形下求矩陣解的方法

四、自動微分系統

在 MKSim 中經常要使用大量的數學運算，其中包含求限制等式的函數值、求一次微分矩陣及二次微分矩陣等。由於在運動學分析的迴圈中，同樣的數學運算式反覆的被運算，所不同的是代入的值不斷的改變而已。因此，我們可以運用自動微分系統來加速運算的時間。舉例來說，計算函數$f: x \in R^n \longrightarrow y \in R^m$，我們想用$x$來計算$y$的微分值，我們稱$x$爲 independent 變數，稱$y$爲 dependent 變數。自動微分(Automatic Differentiation)它採取嵌入法則(chain rule) ：

$$\frac{\partial}{\partial t} f(g(t))\Big|_{t=t_0} = \left(\frac{\partial}{\partial s} f(s)\Big|_{s=g(t_0)}\right)\left(\frac{\partial}{\partial t} g(t)\Big|_{t=t_0}\right)$$

來處理複雜的運算組合。在 MKSim 中以自動微分方式配合 C++的 operator overloading 來自動產生所需要的微分運算。我們除了使用自動微分中的正向模式(Forward mode)來計算微分值之外，還使用反向模式(Reverse mode) [3][4][5][6]來計算比正向模式高一次的微分值，如此更加速了計算的速度。

在實作自動微分系統模擬器時，我們是以 Griewank[5]所提出的方法作爲基礎，加上一些修正以符合我們的需求。在 MKSim 環境中的自動微分系統是針對 multibody 模擬器所需要的微分部份以 C++的物件方式寫成。自動微分系統分成記錄器和運算器兩個部分。在記錄器中是用 C++ operator overloading 的方式來記錄限制等式的運算過程。在運算器的部分是將記錄器的記錄結果用正向模式 與反向模式的方式，計算限制等式的函式值、Jacobian 矩陣與 Hessian 矩陣值。因此在 MKSim 環境中除了使用 Griewank 所定義的 adouble 類別外還新定義了 trace 類別，並在 trace 類別中定義了 forward、reverse、jacobian 與 hessian 等有關於運算的成員函數(member function)，以及 trace_on 與 trace_off 等有關於記錄的成員函數。

在自動微分系統記錄器中，我們可以將所有的限制等式的方程式放在設定記錄區段中，並將所有的 generalized coordinate 的值設爲 independent 的變數，限制等式的值設爲 dependent 的變數。所謂的設定記錄區段是指包含在 trace 物件中 trace_on 與 trace_off 成員函數之間的所有程式碼。如（圖六 a）是一個簡單包含記錄區段程式碼的例子。我們要記錄方程式：y=f(x1,x2,x3)=-x1+(x2*x3*x2) 的運算過程。因此我們將 x1，x2，x3 與 y 四個變數設成 adouble 的類別，並且把 x1，x2，x3 設成 independent 的變數，把 y 設成 dependent 的變數;記錄區段程式碼中，使用過載運算子 "<<="和 ">>="來區別 independent 與 dependent 的變數。在記錄的過程中自動微分系統將所有複雜的運算式切割成單一或二元的運算式，並將運算的過程記錄成樹狀的結構，如(圖六 b)所示。通常除了 independent 和 dependent 的變數被設爲 adouble 的類別以外，相關 indepentent 在過程中，所產生的中間值也會自動被設爲 adouble 的類別，如圖六 b 中 t1，t2，t3 這三個中間值。凡是在設定記錄區段中，當變數被設爲 adouble 的類別時，此變數在做加減乘除等運算等運算時，這些 operator 會被 overloading，它會把相關於此變數的所有運算的過程，記錄在 trace 物件的資料結構中。

由於我們在記錄的過程中將所有複雜的運算式都切割成單一或二元的運算式，所以在自動微分系統運算器中，可以利用 chain rule 的法則(也就是正向模式)反覆代入運算式中求出函數的微分值，如(圖六 c)所示。另外我們可以根據正向模式所計算出來的結果，反向推導出更高一次的偏微分值，這就是反向模式。反向模式是依據 adjoint 的法則，如下所示：

if $s = f(t)$, then $tbar += sbar * (df / dt)$

if $s = f(t,u)$, then $tbar += sbar * (df / dt)$

$ubar += sbar * (df / du)$

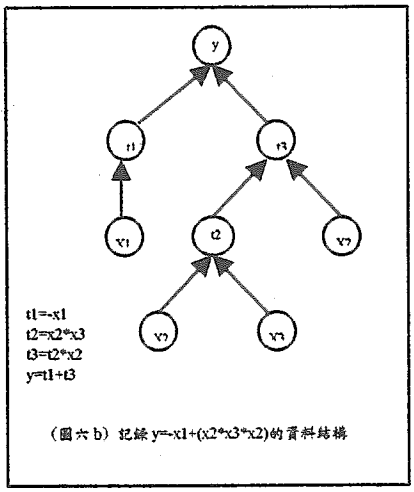
其中 tbar 與 sbar 分別爲 t 與 s 的 adjoint 值，其最後計算所得的值即爲 t 與 s 兩個 independent 變數的一次偏

微分值。(圖六 d) 即爲經過反向模式的運算結果。

在 MKSim 的環境中利用自動微分系統，求出所有限制等式的值，以判定模型的組合是否正確；計算出所有限制等式的 Jacobian 的矩陣，以求得模型的速度；最後計算出所有限制等式的 Hessian 的矩陣，以求得模型的加速度。

```
double *ax ,* ay;              // ax 是儲存 double 的陣列
trace    tr;          // tr 是 trace 的物件
ax[0]=10;
ax[1]=20;
tr.trace_on( );      //開始紀錄在 trace 的資料結構中
adouble x1,x2,x3,y;     //x1,x2,x3,y 是 adouble 的變數
x0<<=ax[0];             //value(x0)=ax[0]
x1<<=ax[1];             //value(x1)=ax[1]
y=-x1+(x2*x3*x2);
y>>=ay[0];              //ay[0]=value(y)
tr.trace_off( );        //結束紀錄
```

(圖六 a)一個簡單包含紀錄區段的程式碼



(圖六 b) 紀錄 y=-x1+(x2*x3*x2)的資料結構

$$\nabla t1 = -1 \cdot \nabla x1$$
$$\nabla t2 = x3 \cdot \nabla x2 + x2 \cdot \nabla x3$$
$$\nabla t3 = x2 \cdot \nabla t2 + t2 \cdot \nabla x2$$
$$\nabla y = \nabla t1 + \nabla t3$$

(圖六 c) y=-x1+(x2*x3*x2)的一次微分的 forward mode

//initialize adjoint quantites// $\overline{y} = 1 \quad \overline{x1} = \overline{x2} = \overline{x3} = \overline{t1} = \overline{t2} = \overline{t3} = 0$

//adjoints for y = t1 + t3 // $\overline{t1} += \overline{y} \qquad \overline{t3} += \overline{y}$

//adjoints for t3 = t2 * x2 // $\overline{t2} += \overline{t3} * x2 \quad \overline{x2} += \overline{t3} * t2$

//adjoints for t2 = x2 * x3 // $\overline{x2} += \overline{t2} * x3 \quad \overline{x3} += \overline{t2} * t3$

//adjoints for t1 = −x1 // $\overline{x1} += -\overline{t1}$

(圖六 d) y=-x1+(x2*x3*x2)的一次偏微分的 reverse mode

五、結論與未來研究

在本論文中有兩項功能是我們想要達到的。其一是建立一套完整的 multibody 運動學模擬器的環境。因此在 MKSim 環境中，在前處理部分將使用者輸入的模型檔案，經過剖析器自動轉成系統化的公式以及模擬器內部的資料，在核心部分有運動學分析的完整函式庫，其中並用自動微分系統處理有關運動學分析時所需要用到的微分矩陣，並在後處理部分用動畫顯示器顯示模擬的結果。其二是當系統處在 under constraint 的情況下，也就是說在模型產生眾多可能的運動行爲中，在 MKSim 環境中儘可能的分析出一組最爲平滑的運動行爲。

MKSim 設計的主要目的是讓使用者可以從模擬結果的數據中得到一些訊息，使他們能夠確定設計是否符合原來的要求。對於複雜的 multibody 機械系統模型來說，這種模擬方式要比用人工計算方便許多，而且透過動畫軟體的顯示，使用者可以預知整個模型設計的運動情形。然而因時間的關係，有一些功能尚未實現，在此提出以作爲未來的改進與研究重點：未來若能加上外力的考量，成爲較完整的動力學模擬器，並且模擬更多型式的 joint，使得模型的運動更驅多元化。

參考文獻

[1] Edward J. Haug, "Computer Aided Kinematics and Dynamics of Mechanical Systems Volumel I: Basic Methods", Allyn and Bacon Series in Engineering, 1989

[2] Brian W. Kernighan, Rob Pike, "The UNIX Programing Environment", Englewood Cliffs, NJ:Prentic-Hall Software Series,1984

[3] Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, "ADIFOR：Automatic Differentiation in a Source Translator Environment", Mathematics and Computer Science Division, Argonne National Laboratory , 1992

[4] Christian Bischof, Alan Carle, Peyvand Khademi, Andrew Mauer, "The ADIFOR 2.0 System for the Automatic Differentiation of Fortran 77 Programs", Mathematics and Computer Science Division, Argonne National Laboratory, 1994

[5] Andreas Griewank, and David Juedes, and Jay Srinivasan, and Charles Tyner, "ADOL—C：A Package for the Automatic Differentiation of Algorithms Written in C/C++", Mathematics and Computer Science Division, Argonne National Laboratory , 1990

[6] Andreas. Griewank and Georage Corliss, "Automatic Differentiation of Algorithms", SIAM, Philadelphia, 1991