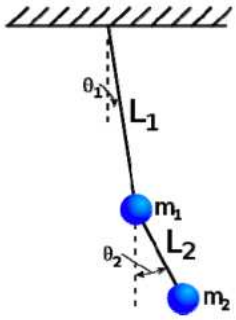
"Sviluppo di un oggetto in ambiente Pure Data che simuli il moto di un doppio pendolo"

di Paolo Gatti

Sistema fisico

Il doppio pendolo è un sistema meccanico provvisto di due pendoli collegati. Il moto che ne deriva in particolari condizioni energetiche è ascrivibile ad un moto caotico.



Dato un sistema di riferimento cartesiano che abbia centro nel gancio, asse x parallelo al soffitto e asse y verticale con direzione verso il basso, le coordinate delle due masse porgeranno:

$$P_1 = (l_1 \sin\theta(t), l_1 \cos\theta(t))$$
$$P_2 = (l_1 \sin\theta(t) + l_2 \sin\phi(t), l_1 \cos\theta(t) + l_2 \cos\phi(t))$$

Definendo opportunamente le condizioni iniziali, è possibile studiare il sistema in termini di *meccanica Lagrangiana*. La *Lagrangiana*, definita come la differenza fra l' energia cinetica e l' energia potenziale del sistema, ci consente di derivare le quattro equazioni fondamentali del moto:

$$\dot{\theta}_1 = 6\frac{2p_{\theta_1} - 3cos\,(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9cos^2(\theta_1 - \theta_2)} \quad (1)$$

$$\dot{\theta}_2 = 6\frac{8p_{\theta_2} - 3cos\,(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9cos^2(\theta_1 - \theta_2)} \quad (2)$$

$$\dot{p}_{\theta_1} = -\tfrac{1}{2}[\dot{\theta}_1\dot{\theta}_2 \sin(\theta_1 - \theta_2) + 3gsin\theta_1] \quad (3)$$

$$\dot{p}_{\theta_2} = -\tfrac{1}{2}[-\dot{\theta}_1\dot{\theta}_2 \sin(\theta_1 - \theta_2) + gsin\theta_2] \quad (4)$$

In definitiva, le equazioni sopra descritte, esplicitano le derivate degli angoli e le derivate dei momenti cinetici (ovvero le derivate dei momenti della quantità di moto rispetto agli assi di rotazione x, y). E' possibile ottenere l'evoluzione temporale del sistema attraverso le coordinate delle masse in funzione del tempo tramite il metodo di integrazione numerica di *Runge - Kutta*. L' algoritmo di Runge – Kutta, è un metodo di risoluzione efficace per le equazioni differenziali ordinarie. Tale risoluzione matematica, offre una soluzione al generico *problema di Cauchy* nella forma:

$$y' = f(t, y)$$
$$y(t_0) = y_0$$

Dato un intervallo [t_0 , t_f] e suddiviso quest' ultimo in un intervallo uniforme, è possibile ricavare una

approssimazione dei valori di y(t_j), potendo ricostruire l' andamento della funzione (l'esattezza della ricostruzione aumenta per un numero maggiore di discretizzazioni dell' intervallo).
Nel caso del doppio pendolo, la relazione che consente l' integrazione delle equazioni di cui sopra è la seguente:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{i+1} = t_i + \Delta t$$

y_{i+1} è l' approssimazione di Runge – Kutta del quarto ordine della y(t_j) .

$$k_1 = \Delta t[f(t_i, y_i)]$$
$$k_2 = \Delta t[f(t_i + \frac{\Delta t}{2}, y_i + \frac{k_1}{2})]$$
$$k_3 = \Delta t[f(t_i + \frac{\Delta t}{2}, y_i + \frac{k_2}{2})]$$
$$k_4 = \Delta t[f(t_i + \Delta t, y_i + k_3)]$$

Delta t è l'ampiezza dell' intervallo. I valori successivi della funzione sono ottenuti dal presente valore di y più il prodotto della ampiezza intervallare per un fattore di pendenza medio. Il fattore di pendenza medio è un valore pesato delle pendenze *k1*, *k2*, *k3*, *k4*. Più precisamente, *k1* è la pendenza all' inizio dell' intervallo, *k2* è la pendenza al punto di mezzo in funzione di *k1*, *k3* è la pendenza al punto di mezzo in funzione di k2 e *k4* è la pendenza alla fine dell' intervallo in funzione di *k3*.

Implementazione in C

L' oggetto Pure Data **dpendulum**, invia in uscita i valori delle posizioni (coppia di coordinate x, y) nel tempo delle due masse *m1* e *m2*. Dalle relazioni (1) e (2), si ottengono le derivate rispetto al tempo degli angoli *theta1* e *theta2* attraverso la porzione di codice:

```
double f1 ( double t, double u )
{
  double dudt;

  dudt = u * 6*(2 * x3 - 3 * cos (x1 - x2) * x4) / ((16 - 9 * cos (x1 - x2)) * (16 - 9 * cos (x1 - x2)));
  return dudt;
}

double f2 ( double t, double u )
{
    double dudt;
    dudt = u * 6*(8 * x4 - 3 * cos (x1 - x2) * x3) / ((16 - 9 * cos (x1 - x2)) * (16 - 9 * cos (x1 - x2)));
    return dudt;
}
```

Successivamente, tali valori vengono presentati in ingresso all'algoritmo di *Runge – Kutta* mediante la chiamata alla funzione rk4:

```
u1_1 = rk4 ( t0, x->u0_1, x->dt, f1 );
u1_2 = rk4 ( t0, x->u0_2, x->dt, f2 );
```

L' algoritmo di Runge – Kutta, è eseguito all' interno del sorgente rk4.inc

```
/*
Input, double T0, the current time.
Input, double U0, the solution estimate at the current time.
Input, double DT, the time step.
Input, double F ( double T, double U ), a function which evaluates  the derivative, or right hand side of the problem.
Output, double RK4, the fourth-order Runge-Kutta solution estimate at time T0+DT.
*/
{
 double f0;
 double f1;
```

```
 double f2;
 double f3;
 double t1;
 double t2;
 double t3;
 double u;
 double u1;
 double u2;
 double u3;
/*
 Get four sample values of the derivative.
*/
 f0 = f ( t0, u0 );

 t1 = t0 + dt / 2.0;
 u1 = u0 + dt * f0 / 2.0;
 f1 = f ( t1, u1 );

 t2 = t0 + dt / 2.0;
 u2 = u0 + dt * f1 / 2.0;
 f2 = f ( t2, u2 );

 t3 = t0 + dt;
 u3 = u0 + dt * f2;
 f3 = f ( t3, u3 );
/*
Combine them to estimate the solution.
*/
 u = u0 + dt * ( f0 + 2.0 * f1 + 2.0 * f2 + f3 ) / 6.0;
return u;
}
```

Attraverso la stima delle pendenze f_0, f_1, f_2, f_3, vengono approssimate le du / dt (nel nostro caso le derivate prime di *theta1* e *theta2*), ricavandone l' andamento rispetto al tempo (ovvero l' integrazione numerica in funzione del parametro tempo). A questo punto, i valori di uscita *u1_1*, *u1_2* (i valori di *theta1(t)* e *theta2(t)*) vanno ad aggiornare le variabili locali della struttura dati dell' oggetto Pd:

```
x->u0_1 = u1_1;
x->u0_2 = u1_2;
x->u0_3 = u1_3;
x->u0_4 = u1_4;
```

Conoscendo la lunghezza delle asticelle l_1 e l_2, è possibile ricavare le posizioni delle masse nel tempo mediante la trasformazione da coordinate polari a coordinate cartesiane:

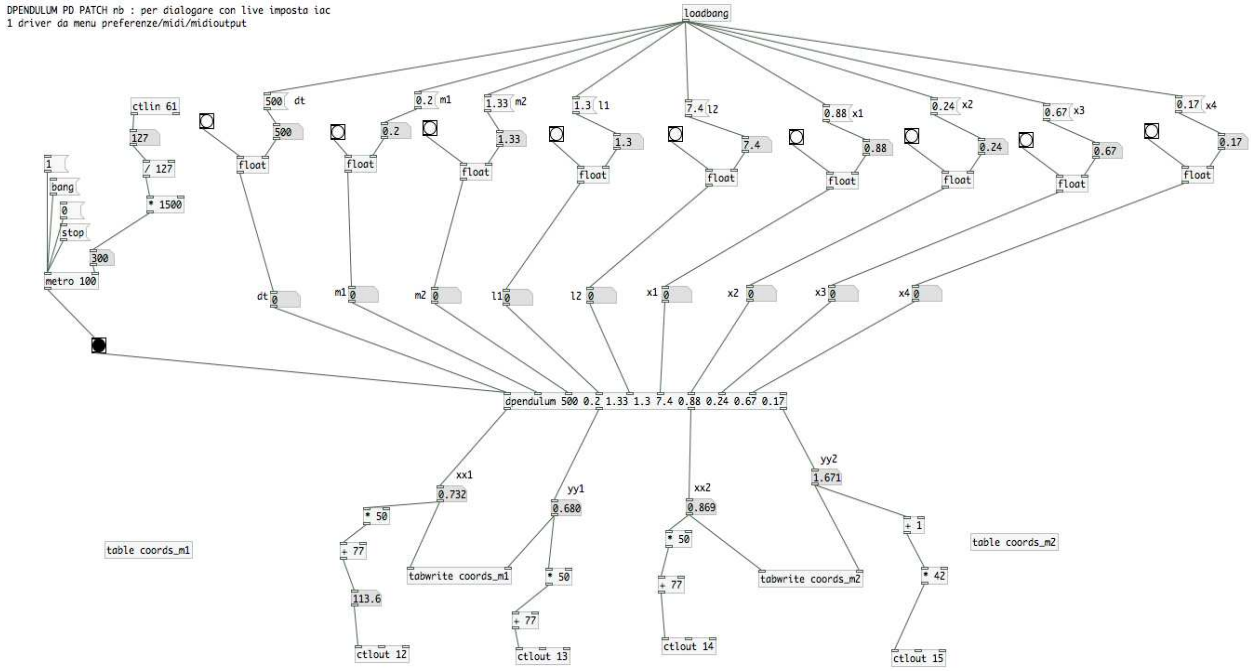
```
x->xx1 = x->l1 * sin(x->u0_1);
x->yy1 = x->l1 * cos(x->u0_1);
x->xx2 = x->l1 * sin(x->u0_1) + x->l2 * sin(x->u0_2);
x->yy2 = x->l1 * cos(x->u0_1) + x->l2 * cos(x->u0_2);
```

Istanziando quattro output per l' oggetto Pd e associando ad un metodo “bang” dell'oggetto l' invio in output dei suddetti valori, si ottiene il profilo del moto pendolare mediante le due coppie di coordinate in funzione delle condizioni iniziali (masse, lunghezze, posizioni iniziali). Questa è la routine che associa il comando bang alla chiamata della funzione rungekutta( ) :

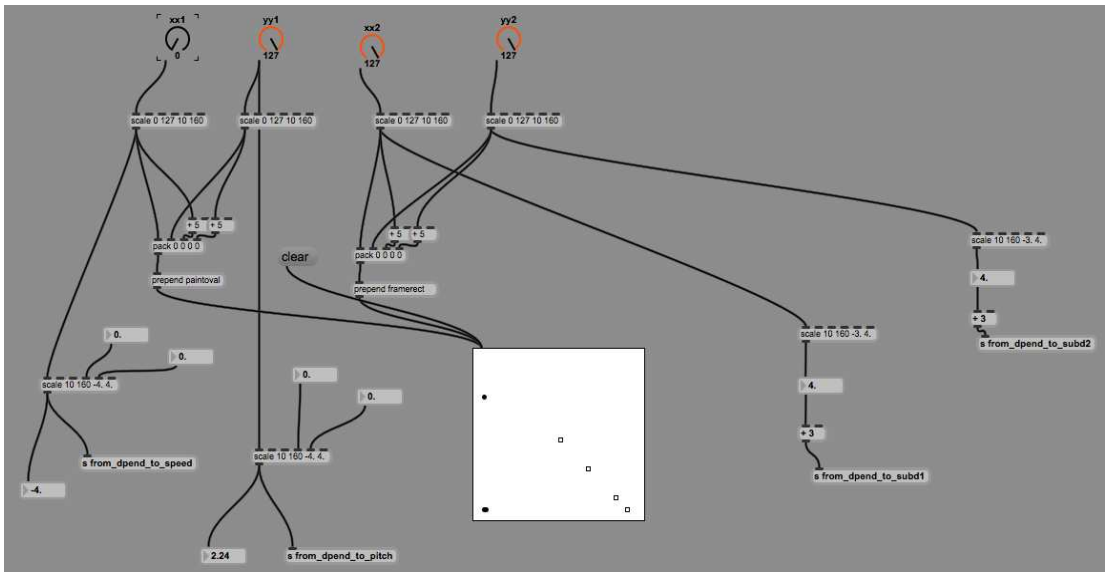
```
static void dpendulum_bang(t_dpendulum *x)
{
rungekutta(x);

outlet_float(x->x_outletxx1, x->xx1);
outlet_float(x->x_outletyy1, x->yy1);
outlet_float(x->x_outletxx2, x->xx2);
outlet_float(x->x_outletyy2, x->yy2);
}
```

Di seguito, uno screen- shot dell 'oggetto in funzione:



e una visualizzazione delle coordinate in funzione del tempo tramite l' oggetto table di Max4Live:



Applicazioni Musicali

Lo scopo del presente lavoro, è stato quello di implementare un sistema fisico caratterizzato da valori caotici nel tempo per poterlo utilizzare come strumento di controllo nella composizione "Poltergeist" (che pone le proprie basi espressive sull' impiego di sistemi caotici). La coppia di coordinate nel tempo può essere impiegata per pilotare l' andamento di taluni parametri in un sistema di sintesi sonora, filtraggio, seguendo l' approccio creativo scelto dal compositore. Nella composizione argomento del presente lavoro, i segnali in uscita dall' oggetto, opportunamente riscalati, sono stati inviati ad un patch Max4Live adibito alla gestione di tali dati come variabili in input ad un banco di oggetti elasticX~ (oggetti Max similari a Groove~, con controllo indipendente di pitch e velocità di lettura del campione i-esimo).

Sviluppi futuri

Il sistema doppio pendolo, può essere studiato qualitativamente anche in termini energetici; ragionando in termini di energia cinetica complessiva del sistema, si potrebbero inserire i parametri velocità come variabili di controllo. L'andamento quadratico della velocità aggiungerebbe ulteriori caratteristiche di non linearità al sistema; guardando il problema dal punto di vista hamiltoniano inoltre, si studierebbero gli andamenti oscillatori del sistema attorno a configurazioni di equilibrio stabile. Imponendo da input il raggiungimento di questi equilbri in un certo istante, si avrebbero in uscita i valori delle pulsazioni ed il tempo impiegato per discostarsi dalla posizione di equilibrio. Parametri questi di sicuro interesse dal punto di vista della applicazione musicale. Altro obiettivo futuro, sarà graficare opportunamente l' andamento del moto, potendo avere una correlazione in tempo reale fra l' aspetto visivo ed il risultato acustico correlato alle coordinate del moto in questione.

Listati

```
/*  dpendulum.c  */

#include "m_pd.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "rk4.h"
#include "rk4.inc"

//static char *version = "dpendulum v0.1, written by Paolo Gatti <www.paologatti.org>";

double f1 ( double t, double u );
double f2 ( double t, double u );
double f3 ( double t, double u );
double f4 ( double t, double u );

double g = 9.81,x1,x2,x3,x4;

typedef struct dpendulum
{
  t_object x_ob;
  t_inlet  *x_inleft;
  t_outlet *x_outletxx1;
  t_outlet *x_outletyy1;
  t_outlet *x_outletxx2;
  t_outlet *x_outletyy2;

  t_float x1;
  t_float x2;
  t_float x3;
  t_float x4;
  t_float m1;
  t_float m2;
  t_float l1;
  t_float l2;
  t_float dt;

  t_float xx1;
  t_float xx2;
  t_float yy1;
  t_float yy2;

  t_float u0_1;
  t_float u0_2;
  t_float u0_3;
  t_float u0_4;

} t_dpendulum;

void rungekutta ( t_dpendulum *x  );

static void dpendulum_bang(t_dpendulum *x)
{
    rungekutta(x);
```

```
    outlet_float(x->x_outletxx1, x->xx1);
    outlet_float(x->x_outletyy1, x->yy1);
    outlet_float(x->x_outletxx2, x->xx2);
    outlet_float(x->x_outletyy2, x->yy2);
}

static void dpendulum_setx1(t_dpendulum *x, t_floatarg f)
{
        x->x1 = f;
    post("set x1 to: %f", f);
        dpendulum_bang(x);
}

static void dpendulum_setx2(t_dpendulum *x, t_floatarg f)
{
    x->x2 = f;
    post("set x2 to: %f", f);
    dpendulum_bang(x);
}

static void dpendulum_setx3(t_dpendulum *x, t_floatarg f)
{
    x->x3 = f;
    post("set x3 to: %f", f);
    dpendulum_bang(x);
}

static void dpendulum_setx4(t_dpendulum *x, t_floatarg f)
{
    x->x4 = f;
    post("set x4 to: %f", f);
    dpendulum_bang(x);
}

static void dpendulum_setl1(t_dpendulum *x, t_floatarg f)
{
    x->l1 = f;
    post("set pendulum length 1 to: %f", f);
    dpendulum_bang(x);
}

static void dpendulum_setl2(t_dpendulum *x, t_floatarg f)
{
    x->l2 = f;
    post("set pendulum length 2 to: %f", f);
    dpendulum_bang(x);
}

static void dpendulum_setm1(t_dpendulum *x, t_floatarg f)
{
    x->m1 = f;
    post("set pendulum mass 1 to: %f", f);
    dpendulum_bang(x);
}

static void dpendulum_setm2(t_dpendulum *x, t_floatarg f)
{
    x->m2 = f;
    post("set pendulum mass 2 to: %f", f);
    dpendulum_bang(x);
}

static void dpendulum_setdt(t_dpendulum *x, t_floatarg f)
{
    x->dt = f;
    post("set delta time to: %f", f);
    dpendulum_bang(x);
}

static t_class *dpendulum_class;

static void *dpendulum_new(t_symbol *s, t_int argc, t_atom* argv)
{
    t_dpendulum *x = (t_dpendulum *)pd_new(dpendulum_class);

    x->x1=0.1;
```

```
    x->x2=0.44;
    x->x3=0.76;
    x->x4=0.4;
    x->m1=1.0;
    x->l1=1.0;
    x->m2=1.0;
    x->l2=1.0;
    x->dt=0.1;

    x->u0_1=0.5;
    x->u0_2=0.2;
    x->u0_3=0.3;
    x->u0_4=0.8;

    switch(argc)
    {
        default:
        case 1:
            x->dt = atom_getfloat(argv);
        case 2:
            x->m1 = atom_getfloat(argv+1);
        case 3:
            x->m2 = atom_getfloat(argv+2);
        case 4:
            x->l1 = atom_getfloat(argv+3);
        case 5:
            x->l2 = atom_getfloat(argv+4);
        case 6:
            x->x1 = atom_getfloat(argv+5);
        case 7:
            x->x2 = atom_getfloat(argv+6);
        case 8:
            x->x3 = atom_getfloat(argv+7);
        case 9:
            x->x4 = atom_getfloat(argv+8);
            break;
    }


    post("dpendulum initialization: dt %f, m1 %f, m2 %f, l1 %f, l2 %f, x1 %f, x2 %f, x3 %f, x4 %f ",x->dt,x->m1,x->m2, x->l1, x->l2, x->x1, x->x2, x->x3, x-
>x4);

    floatinlet_new(&x->x_ob, &x->x4);
    floatinlet_new(&x->x_ob, &x->x3);
    floatinlet_new(&x->x_ob, &x->x2);
    floatinlet_new(&x->x_ob, &x->x1);
    floatinlet_new(&x->x_ob, &x->l2);
    floatinlet_new(&x->x_ob, &x->l1);
    floatinlet_new(&x->x_ob, &x->m2);
    floatinlet_new(&x->x_ob, &x->m1);
    floatinlet_new(&x->x_ob, &x->dt);

            x->x_outletxx1 = outlet_new(&x->x_ob, gensym("float"));
            x->x_outletyy1 = outlet_new(&x->x_ob, gensym("float"));
            x->x_outletxx2 = outlet_new(&x->x_ob, gensym("float"));
            x->x_outletyy2 = outlet_new(&x->x_ob, gensym("float"));

    return (void *)x;
}

void dpendulum_setup(void)
{
    dpendulum_class = class_new(gensym("dpendulum"), (t_newmethod)dpendulum_new,
            0, sizeof(t_dpendulum), 0, A_GIMME, 0);

    class_addfloat(dpendulum_class, dpendulum_setx1);
    class_addfloat(dpendulum_class, dpendulum_setx2);
    class_addfloat(dpendulum_class, dpendulum_setx3);
    class_addfloat(dpendulum_class, dpendulum_setx4);
    class_addfloat(dpendulum_class, dpendulum_setl1);
    class_addfloat(dpendulum_class, dpendulum_setl2);
    class_addfloat(dpendulum_class, dpendulum_setm1);
    class_addfloat(dpendulum_class, dpendulum_setm2);
    class_addfloat(dpendulum_class, dpendulum_setdt);
            class_addbang(dpendulum_class, (t_method)dpendulum_bang);

    /*
```

```
    class_addmethod(dpendulum_class,
                (t_method)dpendulum_setlength, gensym("setlength"),
                A_DEFFLOAT, 0);
    class_addmethod(dpendulum_class,
                (t_method)dpendulum_setmass, gensym("setmass"),
                A_DEFFLOAT, 0);
    */

   //post(NULL, 4, version);

   class_sethelpsymbol( dpendulum_class, gensym("dpendulum-help") );
}

/*****************************************************************************/

void rungekutta ( t_dpendulum *x )
{
  //double dt = 0.1;
  double pi = 3.14159265;
  double t0 = 0.0;
  double t1;
  double tmax = 12.0 * pi;
  //double u0_1 = 0.5, u0_2 = 0.2, u0_3 = 0.3, u0_4 = 0.8;
  double u1_1, u1_2, u1_3, u1_4;
  //double xx1, xx2, yy1, yy2;

   //while ( 1 )
   //{
            x->xx1 = x->l1 * sin(x->u0_1);
            x->yy1 = x->l1 * cos(x->u0_1);
            x->xx2 = x->l1 * sin(x->u0_1) + x->l2 * sin(x->u0_2);
            x->yy2 = x->l1 * cos(x->u0_1) + x->l2 * cos(x->u0_2);

   //if ( tmax <= t0 )
   //{
     //break;
   //}

            x1 = x->x1;
            x2 = x->x2;
            x3 = x->x3;
            x4 = x->x4;

   t1 = t0 + x->dt;

   u1_1 = rk4 ( t0, x->u0_1, x->dt, f1 );
   u1_2 = rk4 ( t0, x->u0_2, x->dt, f2 );
   u1_3 = rk4 ( t0, x->u0_3, x->dt, f3 );
   u1_4 = rk4 ( t0, x->u0_4, x->dt, f4 );

   t0 = t1;

   x->u0_1 = u1_1;
   x->u0_2 = u1_2;
   x->u0_3 = u1_3;
   x->u0_4 = u1_4;
   //}

  return;
}

/*****************************************************************************/

double f1 ( double t, double u )
{
   double dudt;
   dudt = u * 6*(2 * x3 - 3 * cos (x1 - x2) * x4) / ((16 - 9 * cos (x1 - x2)) * (16 - 9 * cos (x1 - x2)));
   return dudt;
}

double f2 ( double t, double u )
{
   double dudt;
   dudt = u * 6*(8 * x4 - 3 * cos (x1 - x2) * x3) / ((16 - 9 * cos (x1 - x2)) * (16 - 9 * cos (x1 - x2)));
   return dudt;
}
```

```
double f3 ( double t, double u )
{
   double dudt, xprime1, xprime2;
   xprime1 = u * 6*(2 * x3 - 3 * cos (x1 - x2) * x4) / ((16 - 9 * cos (x1 - x2)) * (16 - 9 * cos (x1 - x2)));
   xprime2 = u * 6*(8 * x4 - 3 * cos (x1 - x2) * x3) / ((16 - 9 * cos (x1 - x2)) * (16 - 9 * cos (x1 - x2)));
   dudt = u * -(xprime1 * xprime2 * sin (x1-x2) + 3 * g * sin (x1)) / 2;
   return dudt;
}

double f4 ( double t, double u )
{
   double dudt, xprime1, xprime2;
   xprime1 = u * 6*(2 * x3 - 3 * cos (x1 - x2) * x4) / ((16 - 9 * cos (x1 - x2)) * (16 - 9 * cos (x1 - x2)));
   xprime2 = u * 6*(8 * x4 - 3 * cos (x1 - x2) * x3) / ((16 - 9 * cos (x1 - x2)) * (16 - 9 * cos (x1 - x2)));
   dudt = u * -(-xprime1 * xprime2 * sin (x1-x2) + g * sin (x2)) / 2;
   return dudt;
}

/******************************************************************************/
/* rk4.inc */

# include <stdlib.h>
# include <stdio.h>
# include <time.h>

# include "rk4.h"

/******************************************************************************/

double rk4 ( double t0, double u0, double dt, double f ( double t, double u ) )

/******************************************************************************/
/*
  Purpose:

    RK4 takes one Runge-Kutta step for a scalar ODE.

  Discussion:

    It is assumed that an initial value problem, of the form

      du/dt = f ( t, u )
      u(t0) = u0

    is being solved.

    If the user can supply current values of t, u, a stepsize dt, and a
    function to evaluate the derivative, this function can compute the
    fourth-order Runge Kutta estimate to the solution at time t+dt.

  Parameters:

    Input, double T0, the current time.

    Input, double U0, the solution estimate at the current time.

    Input, double DT, the time step.

    Input, double F ( double T, double U ), a function which evaluates
    the derivative, or right hand side of the problem.

    Output, double RK4, the fourth-order Runge-Kutta solution estimate
    at time T0+DT.
*/
{
  double f0;
  double f1;
  double f2;
  double f3;
  double t1;
  double t2;
  double t3;
  double u;
  double u1;
  double u2;
  double u3;
/*
  Get four sample values of the derivative.
*/
  f0 = f ( t0, u0 );

  t1 = t0 + dt / 2.0;
  u1 = u0 + dt * f0 / 2.0;
  f1 = f ( t1, u1 );

  t2 = t0 + dt / 2.0;
  u2 = u0 + dt * f1 / 2.0;
  f2 = f ( t2, u2 );

  t3 = t0 + dt;
  u3 = u0 + dt * f2;
  f3 = f ( t3, u3 );
/*
  Combine them to estimate the solution.
*/
  u = u0 + dt * ( f0 + 2.0 * f1 + 2.0 * f2 + f3 ) / 6.0;
```

```
  return u;
}
/******************************************************************************/

double *rk4vec ( double t0, int m, double u0[], double dt,
  double *f ( double t, int m, double u[] ) )

/******************************************************************************/
/*
  Purpose:

    RK4VEC takes one Runge-Kutta step for a vector ODE.

  Discussion:

    It is assumed that an initial value problem, of the form

      du/dt = f ( t, u )
      u(t0) = u0

    is being solved.

    If the user can supply current values of t, u, a stepsize dt, and a
    function to evaluate the derivative, this function can compute the
    fourth-order Runge Kutta estimate to the solution at time t+dt.


  Parameters:

    Input, double T0, the current time.

    Input, int M, the spatial dimension.

    Input, double U0[M], the solution estimate at the current time.

    Input, double DT, the time step.

    Input, double *F ( double T, int M, double U[] ), a function which evaluates
    the derivative, or right hand side of the problem.

    Output, double RK4VEC[M], the fourth-order Runge-Kutta solution estimate
    at time T0+DT.
*/
{
  double *f0;
  double *f1;
  double *f2;
  double *f3;
  int i;
  double t1;
  double t2;
  double t3;
  double *u;
  double *u1;
  double *u2;
  double *u3;
/*
  Get four sample values of the derivative.
*/
  f0 = f ( t0, m, u0 );

  t1 = t0 + dt / 2.0;
  u1 = ( double * ) malloc ( m * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
  {
    u1[i] = u0[i] + dt * f0[i] / 2.0;
  }
  f1 = f ( t1, m, u1 );

  t2 = t0 + dt / 2.0;
  u2 = ( double * ) malloc ( m * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
  {
    u2[i] = u0[i] + dt * f1[i] / 2.0;
  }
  f2 = f ( t2, m, u2 );

  t3 = t0 + dt;
  u3 = ( double * ) malloc ( m * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
  {
     u3[i] = u0[i] + dt * f2[i];
  }
  f3 = f ( t3, m, u3 );
/*
  Combine them to estimate the solution.
*/
  u = ( double * ) malloc ( m * sizeof ( double ) );
  for ( i = 0; i < m; i++ )
  {
     u[i] = u0[i] + dt * ( f0[i] + 2.0 * f1[i] + 2.0 * f2[i] + f3[i] ) / 6.0;
  }
/*
  Free memory.
*/
  free ( f0 );
  free ( f1 );
  free ( f2 );
  free ( f3 );
  free ( u1 );
  free ( u2 );
  free ( u3 );

  return u;
```

```
}
/******************************************************************************/

void timestamp ( void )

/******************************************************************************/
/*
  Purpose:

    TIMESTAMP prints the current YMDHMS date as a time stamp.

  Example:

    31 May 2001 09:45:54 AM

  Parameters:

    None
*/
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;

  now = time ( NULL );
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );

  fprintf ( stdout, "%s\n", time_buffer );

  return;
# undef TIME_SIZE
}

/******************************************************************************/
```

```
/* rk4.h */

double rk4 ( double t0, double u0, double dt, double f ( double t, double u ) );
double *rk4vec ( double t0, int n, double u0[], double dt,
  double *f ( double t, int n, double u[] ) );
void timestamp ( void );

/******************************************************************************/
```