```
//QUESTION 1ST

/* solves a PAIR of simultaneous 2nd-order ODEs
via 4th-order runge-kutta method */

#include <iostream>
#include <cmath>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>

using namespace std;

// modify f[0], f[1], f[2], f[3] in function func()
// GM is a constant number

int
     func (double t, const double y[], double f[],
           void *params)
     {
       double GM = *(double *)params;
       f[0] = y[1];
       f[1] = -GM*y[0]/(pow(y[0]*y[0] + y[2]*y[2],1.5));
       f[2] = y[3];
       f[3] = -GM*y[2]/(pow(y[0]*y[0] + y[2]*y[2],1.5));
       return GSL_SUCCESS;
     }

     int
     jac (double t, const double y[], double *dfdy,
          double dfdt[], void *params)
     {
       double GM = *(double *)params;
       gsl_matrix_view dfdy_mat
         = gsl_matrix_view_array (dfdy, 4, 4);
       gsl_matrix * m = &dfdy_mat.matrix;
       gsl_matrix_set (m, 0, 0, 0.0);
       gsl_matrix_set (m, 0, 1, 1.0);
       gsl_matrix_set (m, 0, 2, 0.0);
       gsl_matrix_set (m, 0, 3, 0.0);


       gsl_matrix_set (m, 1, 0, -GM*(-2.0*pow(y[0],2) + pow(y[2],2))/
                       (pow(y[0]*y[0] + y[2]*y[2],2.5)));
       gsl_matrix_set (m, 1, 1, 0.0);
       gsl_matrix_set (m, 1, 2, GM*(3.0*y[0]*y[2])/
                       (pow(y[0]*y[0] + y[2]*y[2],2.5)));
       gsl_matrix_set (m, 1, 3, 0.0);


       gsl_matrix_set (m, 2, 0, 0.0);
       gsl_matrix_set (m, 2, 1, 0.0);
       gsl_matrix_set (m, 2, 2, 0.0);
       gsl_matrix_set (m, 2, 3, 1.0);


       gsl_matrix_set (m, 3, 0, GM*(3.0*y[0]*y[2])/
                       (pow(y[0]*y[0] + y[2]*y[2],2.5)));
       gsl_matrix_set (m, 3, 1, 0.0);
       gsl_matrix_set (m, 3, 2, -GM*(-2.0*pow(y[2],2) + pow(y[0],2))/
                       (pow(y[0]*y[0] + y[2]*y[2],2.5)));
       gsl_matrix_set (m, 3, 3, 0.0);
```

```
       // since df/dt = 0 for all choices of f, these are zero:

       dfdt[0] = 0.0;
       dfdt[1] = 0.0;
       dfdt[2] = 0.0;
       dfdt[3] = 0.0;
       return GSL_SUCCESS;
     }


int main ()
     {
       const gsl_odeiv_step_type * T
         = gsl_odeiv_step_rk4;

       gsl_odeiv_step * s
         = gsl_odeiv_step_alloc (T, 4);

       double GM = 1.00;               //CHANGE ME. value of G*M_sun
       gsl_odeiv_system sys = {func, jac, 4, &GM}; // GM is ODE dependent

       double t = 0.0, t1 = 100.0;   // CHANGE ME. bounds.
       double h = 0.02;              // CHANGE ME. step size

       // initial conditions: y[0], y[2] = position, y[1],y[3] = velocity
       double y[4] = { 0.5, 0.0, 0.0, 1.25 }, y_err[4];
       double dydt_in[4], dydt_out[4];

        //initial tau, alpha, and signal
        double alpha_x=0; double alpha_y=0; double tau=0;
        int sign = 0;

       /* initialise dydt_in from system parameters */
       GSL_ODEIV_FN_EVAL(&sys, t, y, dydt_in);

       while (t < t1)
         {
           int status = gsl_odeiv_step_apply (s, t, h,
                                              y, y_err,
                                              dydt_in,
                                              dydt_out,
                                              &sys);

           if (status != GSL_SUCCESS)
               break;

     // caculate for tao and alpha

                if ( alpha_x < y[0]){
                        alpha_x = y[0];
                }
                if ( alpha_y < y[2]){
                        alpha_y = y[2];
                }

                if ( t > 0.5 && sign == 0 && abs((y[0]-0.5)/0.5) < 0.01){
                        tau = t;
                        //to get the period for the first round
                        sign = 1;
                }


           dydt_in[0] = dydt_out[0];
```

```
           dydt_in[1] = dydt_out[1];
           dydt_in[2] = dydt_out[2];
           dydt_in[3] = dydt_out[3];

           t += h;
           cout << t << " " << y[0] << " " << y[1] <<
             " " << y[2] << " " << y[3] << endl;

           //           printf ("%.5e %.5e %.5e\n", t, y[0], y[1]);
         }

     //print alpha and tau

         cout << " alpha x is : " << alpha_x << "\t alpha x ^ 3  is " <<  pow(alpha_x,3
) << "\t alpha y is " <<  alpha_y << "\t alpha y ^ 3  is " <<  pow(alpha_y,3) << "\t  t
au is: " << tau << "\t tau^ 3  is: " <<  pow(tau,3) << endl;

       gsl_odeiv_step_free (s);
       return 0;
     }



// QUESTION 2ND-----------------------------------------------------------------------
---------------------------------------


/* solves a PAIR of simultaneous 2nd-order ODEs
via 4th-order runge-kutta method */

#include <iostream>
#include <cmath>
#include <fstream>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>

using namespace std;

// modify f[0], f[1], f[2], f[3] in function func()
// GM is a constant number

int
     func (double t, const double y[], double f[],
           void *params)
     {
       double GM = *(double *)params;

        //1.5 - > 2 here
       f[0] = y[1];
       f[1] = -GM*y[0]/(pow(y[0]*y[0] + y[2]*y[2],2));
       f[2] = y[3];
       f[3] = -GM*y[2]/(pow(y[0]*y[0] + y[2]*y[2],2));
       return GSL_SUCCESS;
     }

     int
     jac (double t, const double y[], double *dfdy,
          double dfdt[], void *params)
     {
       double GM = *(double *)params;
       gsl_matrix_view dfdy_mat
         = gsl_matrix_view_array (dfdy, 4, 4);
```

```
       gsl_matrix * m = &dfdy_mat.matrix;
       gsl_matrix_set (m, 0, 0, 0.0);
       gsl_matrix_set (m, 0, 1, 1.0);
       gsl_matrix_set (m, 0, 2, 0.0);
       gsl_matrix_set (m, 0, 3, 0.0);

        //change codes here too
       gsl_matrix_set (m, 1, 0, -GM*(-3.0*pow(y[0],2) + pow(y[2],2))/
                        (pow(y[0]*y[0] + y[2]*y[2],3)));
       gsl_matrix_set (m, 1, 1, 0.0);
       gsl_matrix_set (m, 1, 2, GM*(4.0*y[0]*y[2])/
                        (pow(y[0]*y[0] + y[2]*y[2],3)));
       gsl_matrix_set (m, 1, 3, 0.0);


       gsl_matrix_set (m, 2, 0, 0.0);
       gsl_matrix_set (m, 2, 1, 0.0);
       gsl_matrix_set (m, 2, 2, 0.0);
       gsl_matrix_set (m, 2, 3, 1.0);


       gsl_matrix_set (m, 3, 0, GM*(4.0*y[0]*y[2])/
                        (pow(y[0]*y[0] + y[2]*y[2],3)));
       gsl_matrix_set (m, 3, 1, 0.0);
       gsl_matrix_set (m, 3, 2, -GM*(-3.0*pow(y[2],2) + pow(y[0],2))/
                        (pow(y[0]*y[0] + y[2]*y[2],3)));
       gsl_matrix_set (m, 3, 3, 0.0);

       // since df/dt = 0 for all choices of f, these are zero:

       dfdt[0] = 0.0;
       dfdt[1] = 0.0;
       dfdt[2] = 0.0;
       dfdt[3] = 0.0;
       return GSL_SUCCESS;
     }


int main ()
     {
       const gsl_odeiv_step_type * T
         = gsl_odeiv_step_rk4;

       gsl_odeiv_step * s
         = gsl_odeiv_step_alloc (T, 4);

       double GM = 2.00;               //CHANGE ME. value of G*M_sun
       gsl_odeiv_system sys = {func, jac, 4, &GM}; // GM is ODE dependent

       double t = 0.0, t1 =500.0;   // CHANGE ME. bounds.
       double h = 0.5;              // CHANGE ME. step size

       // initial conditions: y[0], y[2] = position, y[1],y[3] = velocity
       double y[4] = { 5, 0.0, 0.0, 12.5 }, y_err[4];
       double dydt_in[4], dydt_out[4];



       /* initialise dydt_in from system parameters */
       GSL_ODEIV_FN_EVAL(&sys, t, y, dydt_in);

        /* simple code to write to an output file. */
```

```
        ofstream out_file;

        // open output files:

        out_file.open("hw6_2.dat");

       while (t < t1)
         {
           int status = gsl_odeiv_step_apply (s, t, h,
                                              y, y_err,
                                              dydt_in,
                                              dydt_out,
                                              &sys);

           if (status != GSL_SUCCESS)
               break;




           dydt_in[0] = dydt_out[0];
           dydt_in[1] = dydt_out[1];
           dydt_in[2] = dydt_out[2];
           dydt_in[3] = dydt_out[3];

           t += h;
           cout << t << " " << y[0] << " " << y[1] <<
             " " << y[2] << " " << y[3] << endl;

        //output to "hw6_2.dat
        out_file << t << " " << y[0] <<
             " " << y[2] << endl;

           //           printf ("%.5e %.5e %.5e\n", t, y[0], y[1]);
         }


       gsl_odeiv_step_free (s);
       return 0;
     }

//then use gnuplot to plot the hw6_2.dat


// QUESTION 3RD-------------------------------------------------------------------------
--------------------------------------

#include <iostream>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <gsl/gsl_sf_trig.h>
 using namespace std;

     int
     func (double t, const double y[], double f[],
           void *params)
     {
       double g = *(double *)params;
       f[0] = y[1];
       f[1] = -g*gsl_sf_sin(y[0])/1 ;
       return GSL_SUCCESS;
     }
```

```
int
jac (double t, const double y[], double *dfdy,
     double dfdt[], void *params)
{
  double g = *(double *)params;
  gsl_matrix_view dfdy_mat
    = gsl_matrix_view_array (dfdy, 2, 2);
  gsl_matrix * m = &dfdy_mat.matrix;
  gsl_matrix_set (m, 0, 0, 0.0);
  gsl_matrix_set (m, 0, 1, 1.0);
  gsl_matrix_set (m, 1, 0, -g*gsl_sf_cos(y[0])/1);
  gsl_matrix_set (m, 1, 1, 0);
  dfdt[0] = 0.0;
  dfdt[1] = 0.0;
  return GSL_SUCCESS;
}

int
main (void)
{
  const gsl_odeiv_step_type * T
    = gsl_odeiv_step_rk8pd;

  gsl_odeiv_step * s
    = gsl_odeiv_step_alloc (T, 2);
  gsl_odeiv_control * c
    = gsl_odeiv_control_y_new (0.01, 0.0);
  gsl_odeiv_evolve * e
    = gsl_odeiv_evolve_alloc (2);

  double g = 9.8;
  gsl_odeiv_system sys = {func, jac, 2, &g};

  double t = 0.0, t1 = 20.0;
  double h = 0.01;
   double pi = 3.1415926;
   double initPos = 0;

   //get initial angle.
   cout << " input the initial angle: " << endl;
   cin >> initPos;



  double y[2] = { pi*initPos/180, 0.0 }, y_err[2];
   double dydt_in[2], dydt_out[2];

   //cout << " gsl sin is " << gsl_sf_sin(pi*initPos/180) << endl;
   //initial tau and signal
   double tau=0;
   int sign = 0;

  while (t < t1)
    {
      int status = gsl_odeiv_step_apply (s, t, h,
                                         y, y_err,
                                         dydt_in,
                                         dydt_out,
                                         &sys);

      if (status != GSL_SUCCESS)
          break;
```

```
                if ( t > 1 && sign == 0 && abs(int((y[0]-pi*initPos/180)/(pi*initPos/18
0))) < 0.001){
                        tau = t;
                        //to get the period for the first round
                        sign = 1;
                }

           dydt_in[0] = dydt_out[0];
           dydt_in[1] = dydt_out[1];
           t += h;

           cout << t << " " << y[0] << " " << y[1] << endl;

           //printf ("%.5e %.5e %.5e\n", t, y[0], y[1]);
         }
     cout << " the period for " << initPos << " is : " << tau << ", and k is : " << tau
/1.5 << endl;

       gsl_odeiv_evolve_free (e);
       gsl_odeiv_control_free (c);
       gsl_odeiv_step_free (s);
       return 0;
     }

/*

degree  k
90      1.18
60      1.07
45      1.04
30      1.02
10      1


// QUESTION 4th-----------------------------------------------------------------------
---------------------------------------


#include <iostream>
#include <fstream>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <gsl/gsl_sf_trig.h>
 using namespace std;

   struct parameters
{
        double a;
        double b;
        double c;
        double d;
};

     int
     func (double t, const double y[], double f[],
           void *params)
     {

        parameters p = * (parameters *) params;
       f[0] = y[0]*(p.a-p.b*y[1]);
```

```
  f[1] = -y[1]*(p.c-p.d*y[0]) ;
  return GSL_SUCCESS;
}

int
jac (double t, const double y[], double *dfdy,
     double dfdt[], void *params)
{
   parameters p = * (parameters *) params;
  gsl_matrix_view dfdy_mat
    = gsl_matrix_view_array (dfdy, 2, 2);
  gsl_matrix * m = &dfdy_mat.matrix;
  gsl_matrix_set (m, 0, 0, p.a-p.b*y[1]);
  gsl_matrix_set (m, 0, 1, -p.b);
  gsl_matrix_set (m, 1, 0, p.d);
  gsl_matrix_set (m, 1, 1, -p.c+p.d*y[0]);
  dfdt[0] = 0.0;
  dfdt[1] = 0.0;
  return GSL_SUCCESS;
}

int
main (void)
{
  const gsl_odeiv_step_type * T
    = gsl_odeiv_step_rk8pd;

  gsl_odeiv_step * s
    = gsl_odeiv_step_alloc (T, 2);
  gsl_odeiv_control * c
    = gsl_odeiv_control_y_new (0.01, 0.0);
  gsl_odeiv_evolve * e
    = gsl_odeiv_evolve_alloc (2);

   parameters p;
  p.a = 1, p.b = 0.5, p.c = 0.95, p.d = 0.25;
  gsl_odeiv_system sys = {func, jac, 2, &p};

  double t = 0.0, t1 = 100.0;
  double h = 0.1;

   //get initial population.
   double rabbit, fox;
   cout << " input the initial population for rabbits and fox: " << endl;
   cin >> rabbit >> fox;



  double y[2] = { rabbit, fox }, y_err[2];
   double dydt_in[2], dydt_out[2];

   /* simple code to write to an output file. */

   ofstream out_file;

   // open output files:

   out_file.open("hw6_4.dat");

  while (t < t1)
    {
      int status = gsl_odeiv_step_apply (s, t, h,
                                         y, y_err,
```

```
                                                dydt_in,
                                                dydt_out,
                                                &sys);

           if (status != GSL_SUCCESS)
               break;



           dydt_in[0] = dydt_out[0];
           dydt_in[1] = dydt_out[1];
           t += h;

           cout << t << " " << y[0] << " " << y[1] << endl;
           out_file << t << " " << y[0] << " " << y[1] << endl;
           //printf ("%.5e %.5e %.5e\n", t, y[0], y[1]);
         }


       gsl_odeiv_evolve_free (e);
       gsl_odeiv_control_free (c);
       gsl_odeiv_step_free (s);
       return 0;
     }
//then use gnuplot, then splot "hw6_4.dat" ;


// QUESTION 5th-----------------------------------------------------------------------------
--------------------------------------


#include <iostream>
#include <fstream>
#include <cmath>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <gsl/gsl_sf_trig.h>
 using namespace std;


     int
     func (double t, const double y[], double f[],
           void *params)
     {

        double k = * (double *) params;
       f[0] = y[1];
       f[1] = k*pow((1+y[1]*y[1]),0.5)/t;
       return GSL_SUCCESS;
     }

     int
     jac (double t, const double y[], double *dfdy,
          double dfdt[], void *params)
     {
        double k = * (double *) params;

       gsl_matrix_view dfdy_mat
         = gsl_matrix_view_array (dfdy, 2, 2);
       gsl_matrix * m = &dfdy_mat.matrix;
       gsl_matrix_set (m, 0, 0, 0);
```

```
  gsl_matrix_set (m, 0, 1, 1);
  gsl_matrix_set (m, 1, 0, 0);
  gsl_matrix_set (m, 1, 1, 2*k/pow((1+y[1]*y[1]),0.5)*y[1]/t);
  dfdt[0] = 0.0;
  dfdt[1] = -k*pow((1+y[1]*y[1]),0.5)/(t*t);
  return GSL_SUCCESS;
}

int
main (void)
{
  const gsl_odeiv_step_type * T
    = gsl_odeiv_step_rk8pd;

  gsl_odeiv_step * s
    = gsl_odeiv_step_alloc (T, 2);
  gsl_odeiv_control * c
    = gsl_odeiv_control_y_new (0.1, 0.0);
  gsl_odeiv_evolve * e
    = gsl_odeiv_evolve_alloc (2);

   double k;

  gsl_odeiv_system sys = {func, jac, 2, &k};

  double t = -50, t1 = 0;
  double h = 0.1;

   //get k.
   double a,b;
   cout << " input the initial speed a and b " << endl;
   cin >> a >> b;
   k=a/b;


  double y[2] = { 0, a }, y_err[2];
   double dydt_in[2], dydt_out[2];

   /* simple code to write to an output file. */

   ofstream out_file;

   // open output files:

   out_file.open("hw6_5.dat");

  while (t < t1)
    {
      int status = gsl_odeiv_step_apply (s, t, h,
                                         y, y_err,
                                         dydt_in,
                                         dydt_out,
                                         &sys);

      if (status != GSL_SUCCESS)
          break;



      dydt_in[0] = dydt_out[0];
      dydt_in[1] = dydt_out[1];
      t += h;
```

```
          cout << t << " " << y[0] << " " << y[1] << endl;
          out_file << t << " " << y[0] << endl;
          //printf ("%.5e %.5e %.5e\n", t, y[0], y[1]);
        }


      gsl_odeiv_evolve_free (e);
      gsl_odeiv_control_free (c);
      gsl_odeiv_step_free (s);
      return 0;
    }
//then use gnuplot, then plot "hw6_5.dat" ;
```