```
#include <math.h>
#include <stdio.h>
#include "/usr/local/src/recipes_c−ansi/include/nr.h"
#include "/usr/local/src/recipes_c−ansi/include/nrutil.h"
//#include <stdlib.h>
#define NRANSI

// cc −o stiff stiff.c /usr/local/lib/librecipes_c.a

#define DT (0.0001) /* Time interval for saving data */
#define EPS 1e−6 /* Desired accuracy */
#define MINDT 0.0 /* Minimum allowed timestep */
#define TSTART 0.001 /*15 /* Start time */
#define TIME 10000.0 /* End time */
#define NDEG 2 /* Number of state variables (= number of coupled eq's) */

/* Global variables required by odeint() */
int kmax, kount;
float *xp, **yp, dxsav;

/* Globals required by rkdumb() */
float *xx, **y;

/* Global required by jacobn() */
float *qscale;


int main(void)
{
   float t, dt, dt1, qprime[3], total, *q, *q1; /* state variable */
   long seed;
   int i, j, nok, nbad;
   void equations(float t, float q[], float dt[]);
   FILE *datafile;

   q=vector(1,NDEG); /* state variable, accessed with Q macro */
   q1=vector(1,NDEG);

   // Set up length scale measures for jacobn()
   qscale=vector(1,NDEG);
   qscale[1]=50.0;
   qscale[2]=5.0;

   /* SET UP GLOBALS FOR odeint() I/O */
   kmax=TIME/DT;
   kount=0; /* redundant; odeint() does this too */
   dxsav=DT;
   xp=vector(1,kmax);
   yp=matrix(1,NDEG,1,kmax);

   /* Initial Conditions at time TSTART  */
   // Begin with 100 rabbits and 6 foxes.
   q[1]=q1[1] = 100.0;
   q[2]=q1[2] = 6.0;

   /* INTEGRATE EQUATIONS OF MOTION with EMBEDDED RUNGE−KUTTA*/
   odeint(q1, NDEG, TSTART, TIME, EPS, DT, MINDT, &nok, &nbad, equations, bsstep
);
   /* SAVE THE DATA */
   datafile=fopen("stiff_rkqs.txt","w");
   printf("odeint() returned; kount=%d; nok=%d; nbad=%d\n",kount,nok,nbad);
   for (i=1; i<=kount; i++) {
      fprintf(datafile, "%e", xp[i]); //write time to datafile
      for (j=1; j<=NDEG; j++) {
         /* Write state variable to datafile */
```

```
         fprintf(datafile," %e",yp[j][i]);
      }
      fprintf(datafile,"\n");
   }
   fclose(datafile);

#define NSTEPS 8000
   /* SET UP GLOBALS FOR rkdumb() I/O */
   xx=vector(1,NSTEPS+1);
   y=matrix(1,NDEG,1,NSTEPS+1);

   /* INTEGRATE EQUATIONS OF MOTION with fixed-step Runge-Kutta */
   rkdumb(q, NDEG, TSTART, TIME, NSTEPS, equations);
   /* SAVE THE DATA */
   datafile=fopen("stiff_rkdumb.txt","w");
   printf("rkdumb() returned after %i steps.\n",NSTEPS);
   // Fix broken initial y values returned from rkdumb():
   y[1][1]=q[1];
   y[2][1]=q[2];
   for (i=1; i<=NSTEPS; i++) {
      fprintf(datafile, "%e", xx[i]); //write time to datafile
      for (j=1; j<=NDEG; j++) {
         /* Write state variable to datafile */
         fprintf(datafile," %e",y[j][i]);
      }
      fprintf(datafile,"\n");
   }
   fclose(datafile);

   /* INTEGRATE EQUATIONS OF MOTION with stiff solver */
   odeint(q, NDEG, TSTART, TIME, EPS, DT, MINDT, &nok, &nbad, equations, stifbs)
;
   /* SAVE THE DATA */
   datafile=fopen("stiff_impl.txt","w");
   printf("odeint() returned; kount=%d; nok=%d; nbad=%d\n",kount,nok,nbad);
   for (i=1; i<=kount; i++) {
      fprintf(datafile, "%e", xp[i]); //write time to datafile
      for (j=1; j<=NDEG; j++) {
         /* Write state variable to datafile */
         fprintf(datafile," %e",yp[j][i]);
      }
      fprintf(datafile,"\n");
   }
   fclose(datafile);
}

/* Equations of motion are embodied in this... */
void equations(float t, float *q, float *d_t)
{
   // These are the equations for the ''ecology'' example...
   d_t[1] = q[1] - 0.01*q[1]*q[1] -7.0*q[2];
   d_t[2] = 0.1*q[2] - 2.0*q[2]*q[2]/q[1];
}

/* This is a generalized routine that calculates the Jacobian (matrix
  of partial derivatives of the equations of motion)
  by finite diffferencing. For lazy people like me, this saves the
  effort of analytically differentiating the equations of motion---
  which can be tedious for a large, nonlinear set of coupled equations. */
void jacobn(float t, float *q, float *dfdt, float **dfdq, int n)
{
   int i,j;
   float f0[NDEG+1], f1[NDEG+1], q0[NDEG+1], q1[NDEG+1];
   volatile float delt;
   extern float *qscale; //characteristic scales of all q[i]
```

```
   /* Calculate the Jacobian by finite differencing */
   for (i=1; i<=NDEG; i++) {
      q0[i]=q1[i]=q[i]; //initialize q0,q1.
      dfdt[i]=0.0; //no explicit time dependence in the ODE.
   }
   for (i=1; i<=NDEG; i++) {
      delt = 3.9e-3 * qscale[i];
         //NR eq. 5.7.8, using cube root of floating point machine precision
      q0[i] = q[i]-delt;
      q1[i] = q[i]+delt;
      delt = q1[i]-q0[i]; //this is now 2*delt... NR eq. 5.7.4
      //printf("qscale[%i] = %f, delt = %f\n",i,qscale[i],delt);
      equations(t,q0,f0); // ''equations'' function is expected to exist.
      equations(t,q1,f1);
      for (j=1; j<=NDEG; j++) {
         dfdq[j][i] = (f1[j]-f0[j])/delt; ///NR eq. 5.7.7
      }
      q0[i] = q1[i] = q[i]; //restore initial value
   }

}
```