ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΕΠΙΤΡΟΠΗ ΕΡΕΥΝΩΝ

ΕΝΙΣΧΥΣΗ ΕΡΕΥΝΗΤΙΚΗΣ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ ΣΤΟ ΑΠΘ
ΔΡΑΣΗ Γ. Ενίσχυση Ερευνητικής Δραστηριότητας Βασικής Έρευνας

Κωδικός Έργου: 87840
ΔΥΝΑΜΙΚΟΙ ΜΗΧΑΝΙΣΜΟΙ ΕΚΤΟΞΕΥΣΗΣ ΠΛΑΝΗΤΩΝ ΚΑΙ ΟΡΦΑΝΟΙ ΠΛΑΝΗΤΕΣ.

ΑΡΙΘΜΗΤΙΚΗ ΕΠΙΛΥΣΗ ΤΟΥ ΕΠΙΠΕΔΟΥ ΓΕΝΙΚΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΤΩΝ 3 ΣΩΜΑΤΩΝ ΜΕ ΤΗ ΧΡΗΣΗ ΕΞΟΜΑΛΥΝΣΗΣ (regularization)

ΤΕΧΝΙΚΟ ΕΓΧΕΙΡΙΔΙΟ – ΠΕΡΙΓΡΑΦΗ ΚΩΔΙΚΑ C++

Γ. Βουγιατζής,
Επίκουρος Καθηγητής, Τμ. Φυσικής ΑΠΘ.

Θεσσαλονίκη, Δεκέμβριος 2012

Α. ΑΡΙΘΜΗΤΙΚΗ ΟΛΟΚΛΗΡΩΣΗ

Υλοποιήθηκε η μέθοδος αριθμητική μέθοδος επίλυση Συνήθων διαφορικών εξισώσεων Bulirsch-Stoer (BS), όπως περιγράφεται στο *Press et al, Numerical Recipes in C++, Cambridge University Press, 2nd edition, 2002.*

Ο κώδικας της ολοκλήρωσης BS υλοποιείται με την κλάση ODES

Αρχικοποίηση ODES :

i) Αριθμός εξισώσεων (N)
ii) Δείκτης στη συνάρτηση που περιλαμβάνει τις διαφορικές εξισώσεις (**DEQs**)
iii) Ελάχιστο και μέγιστο επιτρεπόμενο χρονικό βήμα
iv) Ακρίβεια αριθμητικής ολοκλήρωσης
v) Αριθμός εξισώσεων για τον έλεγχο ακρίβειας (obsolete – set 0)

Κοινόχρηστες Συναρτήσεις

i)
```
int bsstep(double y0[], double dydx[], double dt0, double yout[], double
*dtnext);
```

y0[] : πίνακας αρχικών συνθηκών (y0[0]=χρόνος, y0[i], i=1,N, μεταβλητές-συναρτήσεις των διαφορικών εξισώσεων
dydx[]: πίνακας παραγώγων
dt0: προτεινόμενο βήμα ολοκλήρωσης
yout[] : τιμές των άγνωστων συναρτήσεων στο χρόνο t=t0+dt (=yout[0])
dtnext: Προτεινόμενο χρονικό βήμα για την επόμενη επανάληψη της ολοκλήρωσης
Return =error (0:no error)

ii)
```
int bsDTstep(double yin[], double DT, double *dt, double yout[]);
```

```
Ολοκλήρωση για χρονικό διάστημα DT. Η συνάρτηση πραγματοποιεί επανάλειπτική
εκτέλεση της bsstep.

yin[] : αρχικές συνθήκες
yout[] : τελικές συνθήκες μετά από χρόνο DT
dt: προτεινόμενο βήμα για έναρξη ολοκλήρωσης και για το επόμενο βήμα
ολοκλήρωσης.
```

```
Ορισμός κλάσης: κώδικας ODES.h
Υλοποίηση συναρτήσεων : ODES.cpp
```

Β. ΟΛΟΚΛΗΡΩΣΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΤΩΝ ΤΡΙΩΝ ΣΩΜΑΤΩΝ – *ΧΩΡΙΣ* ΕΞΟΜΑΛΥΝΣΗ

DEQs: Εξισώσεις προβλήματος τριών σωμάτων σε αδρανειακό σύστημα με κέντρο το κέντρο μάζας. Η θέση-ταχύτητα του τρίτου σώματος καθορίζεται από το κέντρο μάζας στο μηδέν. Συμπεριλαμβάνονται "Stokes' dissipation terms"

Βασικός πίνακας μεταβλητών του συστήματος των Διαφορικών εξισώσεων (**DEQs**)

Αριθμός εξισώσεων N,
Πίνακας X μεταβλητών : μέγεθος N+1

X[0]=t (time)
X[1]= x1 (x-συνιστώσα θέσης πλανήτη P1)
X[2]= x2 (x-συνιστώσα θέσης πλανήτη P2)
X[3]= y1 (y-συνιστώσα θέσης πλανήτη P1)
X[4]= y2 (y-συνιστώσα θέσης πλανήτη P2)
X[5]= dot x1 (x-συνιστώσα ταχύτητας πλανήτη P1)
X[6]= dot x2 (x-συνιστώσα ταχύτητας πλανήτη P2)
X[7]= dot y1 (y-συνιστώσα ταχύτητας πλανήτη P1)
X[8]= dot y2 (y-συνιστώσα ταχύτητας πλανήτη P2)

Υλοποίηση στον κώδικα : **dsys3BPvmdis2glb.cpp**

Συνάρτηση (**DEQs**):

```
void dSystem3BPdsptv(double t, double χ[],double dd[])
```

t: τρέχον χρόνος,
χ[] : αρχικές συνθήκες
dd[] : οι χρονικές παράγωγες (τιμές των αριστερών μελών των εξισώσεων του συστήματος

Η χρήση της `dSystem3BPdsptv προαπαιτεί την αρχικοποίηση του συστήματος με την συνάρτηση`

```
void InitializeSystem(double mass1, double mass2, double mass3, double
dcoef1, double dcoef2, double vmcoef1, double vmcoef2)
```

mass1, mass2, mass3: μάζες σωμάτων (το τρίτο σώμα θεωρείτε το αστέρι του συστήματος)
dcoef1, dcoef2: Συντελεστές για Stoke's dissipation (βλ. Hadjidemetriou J.D. and Voyatzis G., "On the dynamics of extrasolar planetary systems under dissipation: Migration of Planets", Cel.Mech.Dyn.Astr. 107, 3-19 (2010))
vmcoef1, vmcoef2: Συντελεστές «α» και «β», αντίστοιχα για την απώλεια μάζας του αστέρα (βλ Voyatzis et al, " Multi-Planet Destabilisation and Escape in Post-Main Sequence Systems", 2013, submitted in MNRAS)

Σημείωση: `Στην συνάρτηση dSystem3BPdsptv η γραμμή κώδικα «if(VARMASS) varmassXXX(t);» καλεί μια από τις συναρτήσεις μεταβολής μάζας που ορίζονται στον κώδικα dsys3BPvmdis2glb.cpp` (Eddington-Jean's law)

Γ. ΟΛΟΚΛΗΡΩΣΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΤΩΝ ΤΡΙΩΝ ΣΩΜΑΤΩΝ *ΜΕ* ΕΞΟΜΑΛΥΝΣΗ

DEQs: Εξισώσεις προβλήματος τριών σωμάτων σε μεταβλητές εξομάλυνσης Levi-Civita. Ορίζονται ως σώματα 1,2 τα σώματα που βρίσκονται σε κοντινή διάβαση (close encounter) και 3 είναι το απομακρυσμένο σώμα.

Για την περιγραφή των εξισώσεων και των μεταβλητών βλ. Voyatzis et al, "Ejection of planets via stellar mass loss and chaotic scattering", preprint, 2013

Σημείωση : Δεν μπορεί ο κώδικας να υλοποιήσει ολοκλήρωση σε «τριπλή» κοντινή διάβαση» (δηλαδή και τα τρία σώματα να βρεθούν ταυτόχρονα πολύ κοντά το ένα στο άλλο)

Βασικός πίνακας μεταβλητών του συστήματος των Διαφορικών εξισώσεων (**DEQs**)

Αριθμός εξισώσεων N,
Πίνακας Y μεταβλητών : μέγεθος N+1

Y[0]=t (regularized dynamical time τ)
Y[1]= u1 (x-συνιστώσα LeviCivita)
Y[2]= u2 (y-συνιστώσα LeviCivita)
Y[3]= x3 (x-συνιστώσα θέσης σώματος 3)
Y[4]= y3 (y-συνιστώσα θέσης σώματος 3)
Y[5]= dot u1 (x-συνιστώσα ταχύτητας LeviCivita)
Y[6]= dot u2 (x-συνιστώσα ταχύτητας LeviCivita)
Y[7]= dot x3 (y-συνιστώσα ταχύτητας σώματος 3)
Y[8]= dot y3 (y-συνιστώσα ταχύτητας σώματος 3)
Y[9]=h (energy of the 1-2 body binary)
Y[10]=time (real time)

Υλοποίηση στον κώδικα : **dsys3BPvmdis2glb.cpp**

Συνάρτηση (**DEQs**):

```
void dSystem3BPreg(double t, double y[],double dd[])
```

t: τρέχον χρόνος (εξομαλυσμένος),
y[] : αρχικές συνθήκες
dd[] : οι χρονικές παράγωγες (τιμές των αριστερών μελών των εξισώσεων του συστήματος

Η χρήση της `dSystem3BPdsptv προαπαιτεί την αρχικοποίηση του συστήματος με την συνάρτηση`

```
void InitializeSystem(double mass1, double mass2, double mass3, double
dcoef1, double dcoef2, double vmcoef1, double vmcoef2)
```

(βλ. παραπάνω, παράγραφος Β)

Δ. ΜΕΤΑΤΡΟΠΕΣ ΜΕΤΑΒΛΗΤΩΝ

Οι μετατροπές μεταβλητών από το ένα σύστημα στο άλλο

ΑΔΡΑΝΕΙΑΚΕΣ X ←→ ΕΞΟΜΑΛΥΣΜΕΝΕΣ Y

γίνονται μέσω των συναρτήσεων

i) Από τις εξομαλυσμένες Y στις αδρανειακές X

```
void RegularizedToInertial(int collision, double Y[], double X[])
```

collision: ορίζει ποια σώματα βρίσκονται σε κοντινή διάβαση και περιγράφονται σε μεταβλητές Levi-Civita. Οι τιμές της μεταβλητής είναι οι

1: κοντινή διάβαση πλανήτη 1 με το Αστέρι (σώμα 3)
2: κοντινή διάβαση πλανήτη 2 με το Αστέρι (σώμα 3)
3: κοντινή διάβαση των πλανητών 1 και 2.

ii) Από τις αδρανειακές X στις εξομαλυσμένες Y

```
void InertialToRegularized(int collision, double X[], double Y[])
```

μεταβλητή collision : όπως παραπάνω

Οι παραπάνω συναρτήσεις προαπαιτούν αρχικοποίηση με την `InitializeSystem` (βλ. παράγραφο Β)

Υλοποίηση στον κώδικα : **dsys3BPvmdis2glb.cpp**

Η εύρεση της μεταβλητής collision επιστρέφεται από τη συνάρτηση

```
int CloseEncounter(double X[])
{
double Z[NEQ];
GetStarPos(X, Z);
double r12=sqrt((X[1]-X[2])*(X[1]-X[2])+(X[3]-X[4])*(X[3]-X[4]));
double r13=sqrt((X[1]-Z[1])*(X[1]-Z[1])+(X[3]-Z[2])*(X[3]-Z[2]));
double r23=sqrt((X[2]-Z[1])*(X[2]-Z[1])+(X[4]-Z[2])*(X[4]-Z[2]));
      if(r13<r23&&r13<r12) return 1; else
            if(r23<r13&&r23<r12) return 2; else return 3;
}
```

όπου η GetStarPos βρίσκει τη θέση και ταχύτητα του Αστέρα (μέσω του ολοκληρώματος του κέντρου μάζας)

Δ. ΣΥΝΘΕΤΗ ΟΛΟΚΛΗΡΩΣΗ

Γενικά επιλύεται το σύστημα των εξισώσεων του αδρανειακού συστήματος (παράγραφος Β). Η εξομάλυνση χρησιμοποιείτε όταν δύο σώματα έρχονται «σχετικά» κοντά το ένα στο άλλο. Στο κώδικα που υλοποιήσαμε χρησιμοποιούμε ως κριτήριο χρήσης των εξομαλυσμένων εξισώσεων την «μη δυνατότητα επίλυσης των αδρανειακών εξισώσεων με την ζητούμενη ακρίβεια».

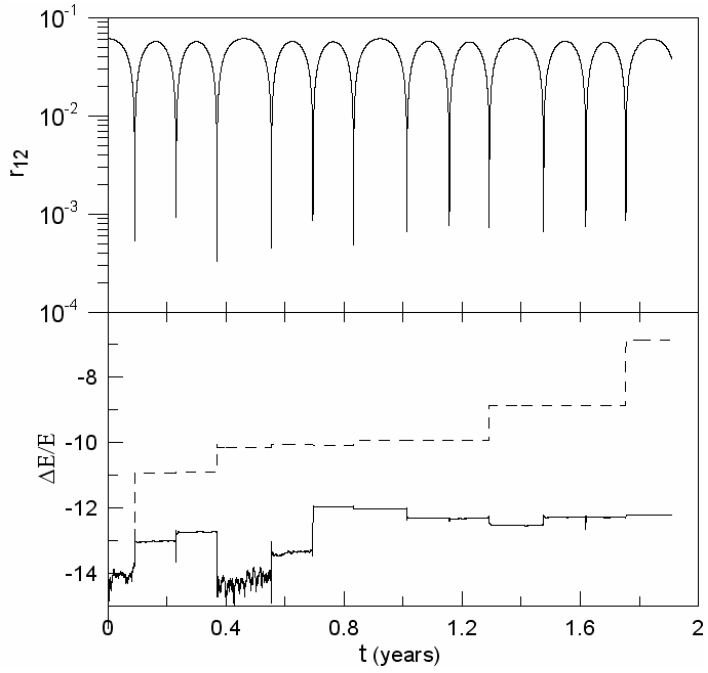
Συγκεκριμένα για το κάθε βήμα ολοκλήρωσης για πραγματικό χρόνο DT χρησιμοποιούμε την διαδικασία

Use as DEQs→ `dSystem3BPdsptv`

```
Inegrate with ODES. bsDTstep
If ERROR{
      InertialToRegularized X->Y
       Use as DEQs→ dSystem3BPreg
      Integrate with ODES.bsstep until to cover DT time span
      If ERROR exit program
      RegularizedToInertial Y->X
}
```

Το σφάλμα στην αριθμητική ολοκλήρωση το εκτιμούμε με βάση την διατήρηση της ενέργειας του συστήματος[1] , συγκεκριμμένα χρησιμοποιούμε τον δείκτη $\Delta=(E-E_0)/E_0$ όπου E_0 η αρχική ενέργεια του συστήματος.

Στο παρακάτω σχήμα δίνουμε ένα παράδειγμα της διατήρησης του δείκτη Δ. Στο πάνω πάνελ παρουσιάζεται σε λογαριθμική κλίμακα η απόσταση των δύο πλανητών. Στο κάτω πάνελ παρουσιάζεται σε λογαριθμική κλίμακα το σφάλμα Δ. Η διακεκομμένη γραμμή αντιστοιχεί σε ολοκλήρωση χωρίς τη χρήση των εξισώσεων εξομάλυνσης ενώ η συνεχής γραμμή αντιστοιχεί στη σύνθετη ολοκλήρωση που περιγράψαμε παραπάνω.



[1] Αυτό δεν είναι εφικτό όταν χρησιμοποιείται dissipation

Ε. ΒΑΣΙΚΟΙ ΚΩΔΙΚΕΣ ΥΛΟΠΟΙΗΣΗΣ

E1. ODES.H
E2. ODES.CPP

E3. dsys3BPvmdis.h
E4. dsys3BPvmdis2glb.cpp

E1. ODES.H

```
//////////////////////////////////////////
//   Class ODES                         //
//                                      //
//   G. Voyatzis, 2012                  //
//   voyatzis@auth.gr
//////////////////////////////////////////
// This class integrates a system of Ordinary Differential Equations (ODES)
using Bulirsch-Stoer Method
//

#ifndef  _GV_ODES_March_2012_
#define  _GV_ODES_March_2012_

#include <iostream>
#include <cassert>
#include <cmath>

#ifndef _BULSTOERSPARAMETERS_
#define _BULSTOERSPARAMETERS_
#define IMAX 11
#define SHRINK 0.95
#define GROW 1.2
#define NUSE 7
#endif

class ODES{
private:
    int     neq;                                                //number of
ODES
    void    (*derivs)(double t, double y[], double dydt[]);    //pointer to
function of ODES

   double BSAcc; //accuracy of calculations
      double BSMinStep; //minimum time step
      double BSMaxStep; //maximum time step
      int ner; //number of variables (starting from the 1st) to take into
acount for error calculation
      int BSINITIALIZED; //bool true if BS is already initialized
      double *Bx; //used
      double **Bd;//in extrapolation (preserve values through loops)
      //pointers for internal use
      double *y, *ysav, *dysav, *yseq, *yerr, *ym, *ynn;

//Polynomial Extrapolation
    void rzextr(int iest, double xest, double yest[], double yz[], double
dy[]);
        //  xest
        //  yest
        //  yz
        //  dy

//Modified Midpoint Method (private,for use in Bulirsch-Stoer)
    void mmid(double y[], double dydx[], double t0, double dt0, int nstep,
double yout[]);
        //  y      Current value of dependent variable (initial input data)
        //  dydx   derivative data produced by calling function "deriv"
        //  t0     Independent variable (usually time)
```

```
        //  dt0    total step to be made
        //  nstep  max number of substeps to be used

public:
//Constructor
    ODES();
        //  numODES                                         number of ODES
        //  DEs_system(double, nVector&, nVector&)      function of ODES

//Constructor For BS only!!
    ODES(int numODEs, void ODEs_system(double , double [], double []),
double minstep, double maxstep, double accuracy, int n_error_check);
        //  numODEs                                         number of ODEs
        //  DEs_system(double, double[], double[])      function of ODEs
        //  min/max step                                    minimum/maximum
step of integration
        //  accuracy                                        accuracy of
calculations
            //  n_error_check : number of (first) deqs used for accuary
control


    ~ODES(void);
//Destructor


//Basic step Integration
    int bsstep(double y0[], double dydx[], double dt0, double yout[],
double *dtnext);
          //  y0      Current value of dependent variable (initial input
data)
        //  dydx    derivative data produced by calling function "deriv"
          //  dt0     time step for integration (dt0 may shrinks internaly
- Correct step done = yout[0]-y0[0]
          //  yout    output data
          //  dtnext  suggested next step, (always smaller than MAXSTEP
defined)

//Bulirsch-Stoer Interval Integration
    int bsDTstep(double yin[], double DT, double *dt, double yout[]);
//Bulirsch-Stoer Integrator for time T  (from t0 to t0+T)
        //  yin     array of Initial conditions (yin[0]=t0)
        //  DT       Time for integration
        //  dt      internal time step (dt<=T)
        //  yout    array of Final conditions (yout[0]=t0+T)
        //  return  1: in case of complete integration
        //      or  0: in case of error integration
      void open(int numODES, void ODES_system(double , double y[], double
dydt[]), double minstep, double maxstep, double acc, int n_error_check);
      int test(void);
};

//End of Class ODES

#endif
```

E2. ODES.CPP

```
#include "ODES.h"

//============================ Private Methods ============
int ODES::bsstep(double y0[], double dydx[], double dt0, double yout[],
double *dtnext){
       int SMALLSTEP=0;
      static int nseq[IMAX+1]={0,2,4,6,8,12,16,24,32,48,64,96};

      double t=y0[0];
      for(int i=1;i<neq;i++) y[i]=y0[i];
      double h=dt0;
      double xsav=t;
      for (int i=1;i<neq;i++) {ysav[i]=y[i];dysav[i]=dydx[i];}
      int COMPLETE=0;
      do
      {
            for (int i=1;i<=IMAX;i++)
            {
                  mmid(ysav,dysav,xsav,h,nseq[i],yseq);
                  double xest=h*h/(nseq[i]*nseq[i]);
                  rzextr(i,xest,yseq,y,yerr);
                  double errmax=0.0;
                  for (int j=1;j<=ner;j++)
                  {
                        double tmp=fabs(yerr[j]);
                        if (errmax<tmp) errmax=tmp;
                  }
                  //old if (errmax < BSAcc) : working failure for the
"rotation" project
                  if (errmax < BSAcc && errmax>=0)    //correction in
21/4/2008
                  {
                        t+= h;
                        *dtnext = i==NUSE? h*SHRINK : i==NUSE-1? h*GROW :
(h*nseq[NUSE-1])/nseq[i];
                        COMPLETE=1; break; //return;
                  }
            }
            if(COMPLETE) break;
            h*= 0.25;
            for (int i=1;i<=(IMAX-NUSE)/2;i++) h *= 0.5;
            SMALLSTEP=(h<BSMinStep);
      }while(!SMALLSTEP);
      for(int i=1;i<neq;i++) yout[i]=y[i]; yout[0]=t;
      if(*dtnext>BSMaxStep) *dtnext=BSMaxStep;
      return (!SMALLSTEP);
}


void ODES::rzextr(int iest, double xest, double yest[], double yz[], double
dy[]){
      double fx[NUSE+1];
      Bx[iest]=xest;
      if (iest == 1)    for (int j=1;j<neq;j++) {yz[j]=yest[j];
      Bd[j][1]=yest[j];dy[j]=yest[j];}
      else
      {
            int m1=(iest < NUSE ? iest : NUSE);
```

```
            for (int k=1;k<m1;k++) fx[k+1]=Bx[iest-k]/xest;
            for (int j=1;j<neq;j++)
            {
                  double ddy;
                  double yy=yest[j];
                  double v=Bd[j][1];
                  double c=yy;
                  Bd[j][1]=yy;
                  for (int k=2;k<=m1;k++)
                  {
                        double b1=fx[k]*v;
                        double b=b1-c;
                        if(fabs(b)>BSAcc) { b=(c-v)/b;      ddy=c*b;
c=b1*b;}
                        else ddy=v;
                        if (k != m1) v=Bd[j][k];
                        Bd[j][k]=ddy;
                        yy += ddy;
                  }
                  dy[j]=ddy;
                  yz[j]=yy;
            }
      }
}

void ODES::mmid(double y[],double dydt[],double t0, double dt0, int nstep,
double yout[]){

      double h=dt0/nstep;
      for (int i=1;i<neq;i++)
      {
            ym[i]=y[i];
            ynn[i]=y[i]+h*dydt[i];
      }
      double t=t0+h;    derivs(t,ynn,yout);
      double h2=2.0*h;
      for (int n=2;n<=nstep;n++)
      {
            for (int i=1;i<neq;i++)
            {
                  double swap=ym[i]+h2*yout[i];
                  ym[i]=ynn[i]; ynn[i]=swap;
            }
            t += h;
            derivs(t,ynn,yout);
      }
      for (int i=1;i<neq;i++) yout[i]=0.5*(ym[i]+ynn[i]+h*yout[i]);
}


//============================ Public Methods  ======================//

//------------------ constructor - destructor ----------------------------
ODES::ODES(){

}

//only for yse in BS
ODES::ODES(int numODES, void ODES_system(double , double y[], double
dydt[]), double minstep, double maxstep, double acc, int n_error_check){
```

```
	if(numODES<2) {BSINITIALIZED=0; return;}
	derivs = ODES_system; if(derivs==NULL) {BSINITIALIZED=0; return;}
	neq=numODES;
	ner=n_error_check; if(ner<=0) ner=neq-1; //all vars
	acc=fabs(acc);
	if(acc>0.1||acc<1.0e-20) acc=0.001; BSAcc=acc;
	BSMinStep=minstep;
	if(maxstep<minstep) maxstep=minstep*100; BSMaxStep=maxstep;

	if(BSINITIALIZED==1) return;

	Bx=new double[IMAX+1];
	int NN; if(neq>NUSE) NN=neq+1; else NN=NUSE+1;
	Bd=new double* [NN];
	for(int i=0;i<NN;i++) Bd[i]=new double[NN];

	y=new double [neq];
	ysav=new double [neq];
	dysav=new double [neq];
	yseq=new double [neq];
	yerr=new double [neq];
	ym=new double[neq];
	ynn=new double[neq];
	BSINITIALIZED=1;
}

ODES::~ODES(void){
	if(BSINITIALIZED==1){
	int NN; if(neq>NUSE) NN=neq+1; else NN=NUSE+1;
	delete [] Bx;
	for(int i=0;i<NN;i++) delete [] Bd[i];

	delete [] y;
	delete [] ysav;
	delete [] dysav;
	delete [] yseq;
	delete [] yerr;
	delete [] ym;
	delete [] ynn;
	}
	BSINITIALIZED=0;
}

void ODES::open(int numODES, void ODES_system(double , double y[], double
dydt[]), double minstep, double maxstep, double acc, int n_error_check){

	if(numODES<2) {BSINITIALIZED=0; return;}
	derivs = ODES_system; if(derivs==NULL) {BSINITIALIZED=0; return;}
	neq=numODES;
	ner=n_error_check; if(ner<=0) ner=neq-1; //all vars
	acc=fabs(acc);
	if(acc>0.1||acc<1.0e-20) acc=0.001; BSAcc=acc;
	BSMinStep=minstep;
	if(maxstep<minstep) maxstep=minstep*100; BSMaxStep=maxstep;

	if(BSINITIALIZED==1) return;

	Bx=new double[IMAX+1];
	int NN; if(neq>NUSE) NN=neq+1; else NN=NUSE+1;
	Bd=new double* [NN];
	for(int i=0;i<NN;i++) Bd[i]=new double[NN];
```

```
      y=new double [neq];
      ysav=new double [neq];
      dysav=new double [neq];
      yseq=new double [neq];
      yerr=new double [neq];
      ym=new double[neq];
      ynn=new double[neq];
      BSINITIALIZED=1;
}

int ODES::test(){ return 0; }


//------------------------  Bulirsch-Stoer integrator ---------------------

int ODES::bsDTstep(double yin[], double DT, double *dt, double yout[]){
    double *y=new double[neq];
      double *dydx=new double[neq];
      for(int i=0;i<neq;i++) y[i]=yin[i];
      double restT=DT,dT,dtn;
      if(DT<*dt) dT=DT; else dT=*dt;
      do
      {
            derivs(y[0],y,dydx);
            if(!bsstep(y,dydx,dT,yout,&dtn)) {delete [] y;  delete [] dydx;
return 0;}
            restT-=(yout[0]-y[0]);
            for(int i=0;i<neq;i++) y[i]=yout[i];
            if(restT>=dtn) {dT=dtn; *dt=dT;}
            else break;
      }while(1);
      dT=restT;
      derivs(y[0],y,dydx);
      if(!bsstep(y,dydx,dT,yout,&dtn)) {delete [] y;  delete [] dydx;
return 0;}
      delete [] y;      delete [] dydx;
      return 1;
}
```

E3. dsys3BPvmdis.h

```
/DSYSTEM :  3-BODYPROBLEM PLANAR - DISSIPATION + VARMASS
//25/3/06 //G.Voyatzis
//rev 5/4/12 >> INCLUDES VARMASS functions - new version



#ifndef _DSYSG3BPR_H
#define _DSYSG3BPR_H

#include <math.h>
#define NEQ 9   //number of 1st order ODEs PLUS 1
#define RNEQ 11   //number of 1st order ODEs PLUS 1 for REGULARIZED
EQUATIONS
#define TINY_DT 1.0e-14
//========Global system variables ==================
extern double m1,m2,m3; //masses
extern double Ma1,Ma2; //anigmenes masses of system P1P3 and (P1P3)P2
extern double mu; //ratio m1/(m1+m3)
extern double Ener0,Lang0; //energy and angular momentum
extern double zacc;

void InitializeSystem(double mass1, double mass2, double mass3, double
dcoef1, double dcoef2, double vmcoef1, double vmcoef2);

void InertialToRegularized(int collision, double y[], double yr[]);
void RegularizedToInertial(int collision, double yr[], double y[]);
//--derivs---
void dSystem3BPdsptv(double t, double y[],double dd[]);
void dSystem3BPreg(double t, double y[],double dd[]);

//-----------------------------------------------------------------
void GetMasses(double *mass1, double *mass2, double *mass3);
void GetSysCoefs(double *d1,  double *d2, double *vm1,  double *vm2);

double Energy(double y[]);
double AngularMomentum(double y[]);

void GetStarPos(double y[], double z[]);
void SetSystemPos(double X1[], double X2[], double Y[]);

double GetPPdistance(double X[]);
double GetPPdistance(double X1[], double X2[]);
double GetSP1distance(double X[]);
double GetSP2distance(double X[]);


#endif
```

E3. dsys3BPvmdis.cpp

```
//DSYSTEM :  3-BODYPROBLEM PLANAR + DISSIPATION + VARMASS
//31/3/06  //G.Voyatzis
//         Mar-2012 : Regularization
//         Apr-2012 : Varmass Project : new version
//revised  May-2012 : choose varmass function

//Inertial Initial condition arrays
//    BODY1 BODY2 BODY3 (used as the Star)
//X   1     2     from CM
//Y   3     4
//VX  5     6
//VY  7     8


#include "dsys3BPvmdis.h"

void dSystem3BPdsptv(double t, double y[],double dd[]);  //dissipative deqs

//========Global system variables ==================
double m1,m2,m3,m30,m20,m10; //masses
double Rm1,Rm2,Rm3; //masses for regularized setup
int VARMASS=0;  //flag for variable mass
int DISSIPATION=0;  //flag for Dissipation terms
double da=1;//coefficient1, SET ZERO for DISSIPATION=FALSE
double db=1.0e-5; //coefficient2
double vm1=0;  //use depends on the model, SET ZERO for VARMASS=FALSE
double vm2=10;
double massend;
double Ma1,Ma2; //anigmenes masses of system P1P3 and (P1P3)P2
double mu; //ratio m1/(m1+m3)
double Ener0,Lang0; //energy and angular momentum
//double zacc;

void InitializeSystem(double mass1, double mass2, double mass3, double
dcoef1, double dcoef2, double vmcoef1, double vmcoef2)
{
      m1=mass1; m2=mass2; m3=mass3;
      vm1=vmcoef1;
      vm2=vmcoef2;
      db=dcoef1;
      da=dcoef2;
      mu=m1/(m1+m3);
      Ma1=m1*m3/(m1+m3);
      Ma2=(m1+m3)*m2/(m1+m2+m3);
      m10=m1; m20=m2; m30=m3;
      massend=m30*(1-vm1);
      if(vmcoef1==0||vmcoef2==0) VARMASS=0; else VARMASS=1;
      if(dcoef1==0) DISSIPATION=0; else DISSIPATION=1;
}

//Function for mass variation - Eddington-Jean's laws  \dot{m} = -a m^n

inline void varmassLin(double t) //n=0
//n=0, linear decay
//vm1: decay factor
//vm2: ratio of mass loss
{
```

```
	m3=m30-vm2*t;
	//double massend=m30*(1-vm1);
	if(m3<massend) m3=massend;
	mu=m1/(m1+m3);
	Ma1=m1*m3/(m1+m3);
	Ma2=(m1+m3)*m2/(m1+m2+m3);
}


inline void varmassSqrt(double t) //n=1/2
//n=1/2, dot m =-a m^n
//vm1: decay factor
//vm2: ratio of mass loss
{
	m3=m30-vm2*sqrt(m30)*t+0.25*vm2*vm2*t*t;
	double massend=m30*(1-vm1);
	if(m3<massend) m3=massend;
	mu=m1/(m1+m3);
	Ma1=m1*m3/(m1+m3);
	Ma2=(m1+m3)*m2/(m1+m2+m3);
}



inline void varmassExp(double t) //n=1
//n=1,  Exp decay to zero
//vm1: decay factor
//vm2: ratio of mass loss
{
	m3=m30*exp(-vm2*t);
	double massend=m30*(1-vm1);
	if(m3<massend) m3=massend;
	mu=m1/(m1+m3);
	Ma1=m1*m3/(m1+m3);
	Ma2=(m1+m3)*m2/(m1+m2+m3);
}


inline void varmassSqr(double t) //n=2
//n=2, dot m =-a m^n
//vm1: decay factor
//vm2: ratio of mass loss
{
	m3=m30/(1+vm2*m30*t);
	double massend=m30*(1-vm1);
	if(m3<massend) m3=massend;
	mu=m1/(m1+m3);
	Ma1=m1*m3/(m1+m3);
	Ma2=(m1+m3)*m2/(m1+m2+m3);
}


inline void varmassCub(double t) //n=3
//n=3, dot m =-a m^n
//vm1: decay factor
//vm2: ratio of mass loss
{
	m3=m30/sqrt(1+2*vm2*m30*m30*t);
	double massend=m30*(1-vm1);
	if(m3<massend) m3=massend;
	mu=m1/(m1+m3);
```

```
	Ma1=m1*m3/(m1+m3);
	Ma2=(m1+m3)*m2/(m1+m2+m3);
}



//======================DERIV FUNCTION==================
//var y[i] : y[1]=x1,y[2]=x2,y[3]=y1, y[4]=y2,y[5]=vx1,....,y[8]=vy2
void dSystem3BPdsptv(double t, double y[],double dd[])
{
	if(VARMASS) varmassLin(t);
//3rd body
	double yx3=-(m1*y[1]+m2*y[2])/m3;
	double yy3=-(m1*y[3]+m2*y[4])/m3;
	//squares
	double r12=(y[1]-y[2])*(y[1]-y[2])+(y[3]-y[4])*(y[3]-y[4]);
	double r13=(y[1]- yx3)*(y[1]- yx3)+(y[3]- yy3)*(y[3]- yy3);
	double r23=(y[2]- yx3)*(y[2]- yx3)+(y[4]- yy3)*(y[4]- yy3);
	//squareroots
	double sr12=sqrt(r12);double sr13=sqrt(r13); double sr23=sqrt(r23);
//squareroots
	//quartics
	double r123=r12*sr12; double r133=r13*sr13; double r233=r23*sr23;
	double r125=r123*r12; double r135=r133*r13; double r235=r233*r23;
	double sr133=sqrt(r133); double sr233=sqrt(r233);
	//Differential equations of 1st ORDER
	dd[1]=y[5];
	dd[2]=y[6];
	dd[3]=y[7];
	dd[4]=y[8];
	dd[5]=-m3*(y[1]-yx3)/r133-m2*(y[1]-y[2])/r123;
	dd[6]=-m3*(y[2]-yx3)/r233-m1*(y[2]-y[1])/r123;
	dd[7]=-m3*(y[3]-yy3)/r133-m2*(y[3]-y[4])/r123;
	dd[8]=-m3*(y[4]-yy3)/r233-m1*(y[4]-y[3])/r123;
	if(DISSIPATION){
		dd[5]-=db*(y[5]+da*y[3]/sr133);
		dd[6]-=db*(y[6]+da*y[4]/sr233);
		dd[7]-=db*(y[7]-da*y[1]/sr133);
		dd[8]-=db*(y[8]-da*y[2]/sr233);
	}
}


//======================DERIV FUNCTION==================
//var y[i] : y[1]=u1,y[2]=u2,y[3]=x3, y[4]=y3, y[5]=pu1, y[6]=pu2,
y[7]=px3, y[8]=py3, y[9]=h, y[10]=t
void dSystem3BPreg(double t, double y[],double dd[])
{
	double irm12=1.0/(Rm1+Rm2);
	double xx=y[1]*y[1]-y[2]*y[2];
	double yy=2*y[1]*y[2];
	double x1=(Rm2*xx-Rm3*y[3])*irm12; double y1=(Rm2*yy-Rm3*y[4])*irm12;
	double x2=-(Rm1*xx+Rm3*y[3])*irm12; double y2=-
(Rm1*yy+Rm3*y[4])*irm12;
	double x13=x1-y[3]; double x23=x2-y[3]; double y13=y1-y[4]; double
y23=y2-y[4];
	double r=x13*x13+y13*y13; double ir133=1.0/(r*sqrt(r));
		r=x23*x23+y23*y23; double ir233=1.0/(r*sqrt(r));
   double tD3x=x23*ir233-x13*ir133;
   double tD3y=y23*ir233-y13*ir133;
```

```
   double tE3x=Rm1*x13*ir133+Rm2*x23*ir233;
   double tE3y=Rm1*y13*ir133+Rm2*y23*ir233;
   r=y[1]*y[1]+y[2]*y[2];
   dd[1]=y[5];
   dd[2]=y[6];
   dd[3]=r*y[7];
   dd[4]=r*y[8];
   dd[5]=0.5*y[9]*y[1]+0.5*Rm3*r*(y[1]*tD3x+y[2]*tD3y);
   dd[6]=0.5*y[9]*y[2]+0.5*Rm3*r*(y[1]*tD3y-y[2]*tD3x);
   dd[7]=r*tE3x;
   dd[8]=r*tE3y;
   dd[9]=2*Rm3*(tD3x*(y[1]*y[5]-y[2]*y[6])+tD3y*(y[2]*y[5]+y[1]*y[6]));
   dd[10]=r;
}


void RegularizedToInertial(int collision, double yr[], double y[])
//collision mode : 1=P1+S, 2=P2+S, 3=P1+P2
//yr[1]=u1,yr[2]=u2,yr[3]=x3, yr[4]=y3, yr[5]=pu1, yr[6]=pu2, yr[7]=px3,
yr[8]=py3, yr[9]=h, yr[10]=t
{
      double rm12=Rm1+Rm2;
      double rmc1=Rm1/rm12; double rmc2=Rm2/rm12; double rmc3=Rm3/rm12;
      double rx1,rx2,rx3,ry1,ry2,ry3,drx1,drx2,drx3,dry1,dry2,dry3;
      double  r=yr[1]*yr[1]+yr[2]*yr[2];
      double xx=yr[1]*yr[1]-yr[2]*yr[2];  double yy=2*yr[1]*yr[2];
      double dxx=2*(yr[1]*yr[5]-yr[2]*yr[6])/r;       double
dyy=2*(yr[2]*yr[5]+yr[1]*yr[6])/r;
      //body 3 (collisionless)
      rx3=yr[3]; ry3=yr[4]; drx3=yr[7]; dry3=yr[8];
      //body 1
      rx1=-rmc3*rx3+rmc2*xx; ry1=-rmc3*ry3+rmc2*yy;
      drx1=-rmc3*drx3+rmc2*dxx; dry1=-rmc3*dry3+rmc2*dyy;
      //body 2
      rx2=-rmc3*rx3-rmc1*xx; ry2=-rmc3*ry3-rmc1*yy;
      drx2=-rmc3*drx3-rmc1*dxx; dry2=-rmc3*dry3-rmc1*dyy;
      //-----------------------------
      switch(collision){
            case 1: m1=Rm1; m2=Rm3; m3=Rm2;
                        y[1]=rx1;  y[2]=rx3;  y[3]=ry1;  y[4]=ry3;
                        y[5]=drx1; y[6]=drx3; y[7]=dry1; y[8]=dry3;
                        break;
            case 2: m1=Rm3; m2=Rm2; m3=Rm1;
                        y[1]=rx3;  y[2]=rx2;  y[3]=ry3;  y[4]=ry2;
                        y[5]=drx3; y[6]=drx2; y[7]=dry3; y[8]=dry2;
                        break;
            case 3: m1=Rm1; m2=Rm2; m3=Rm3;
                        y[1]=rx1;  y[2]=rx2;  y[3]=ry1;  y[4]=ry2;
                        y[5]=drx1; y[6]=drx2; y[7]=dry1; y[8]=dry2;
                        break;
            default : return;
      }
      y[0]=yr[10];
}

void InertialToRegularized(int collision, double y[], double yr[])
//collision mode : 1=P1+S, 2=P2+S, 3=P1+P2
{
      double rx1,rx2,rx3,ry1,ry2,ry3,drx1,drx2,drx3,dry1,dry2,dry3;
      double z[5];
      GetStarPos(y,z);
```

```
	switch(collision){
		case 1: Rm1=m1; rx1=y[1]; ry1=y[3]; drx1=y[5]; dry1=y[7];
			Rm2=m3; rx2=z[1]; ry2=z[2]; drx2=z[3]; dry2=z[4];
			  Rm3=m2; rx3=y[2]; ry3=y[4]; drx3=y[6]; dry3=y[8];
			  break;
		case 2: Rm1=m3; rx1=z[1]; ry1=z[2]; drx1=z[3]; dry1=z[4];
			Rm2=m2; rx2=y[2]; ry2=y[4]; drx2=y[6]; dry2=y[8];
			  Rm3=m1; rx3=y[1]; ry3=y[3]; drx3=y[5]; dry3=y[7];
			  break;
		case 3: Rm1=m1; rx1=y[1]; ry1=y[3]; drx1=y[5]; dry1=y[7];
			Rm2=m2; rx2=y[2]; ry2=y[4]; drx2=y[6]; dry2=y[8];
			  Rm3=m3; rx3=z[1]; ry3=z[2]; drx3=z[3]; dry3=z[4];
			  break;
		default : return;
	}
	//initial conditions
	//yr[1]=u1,yr[2]=u2,yr[3]=x3, yr[4]=y3, yr[5]=pu1, yr[6]=pu2,
yr[7]=px3, yr[8]=py3, yr[9]=h, yr[10]=t
	double rxx=rx1-rx2; double ryy=ry1-ry2; double drxx=drx1-drx2; double
dryy=dry1-dry2;
	double r=sqrt(rxx*rxx+ryy*ryy);
	yr[3]=rx3; yr[4]=ry3; yr[7]=drx3; yr[8]=dry3;
	yr[1]=ryy/sqrt(2*(r-rxx)); yr[2]=sqrt(0.5*(r-rxx));
	yr[5]=0.5*(yr[1]*drxx+yr[2]*dryy);
	yr[6]=0.5*(yr[1]*dryy-yr[2]*drxx);
	yr[9]=0.5*(drxx*drxx+dryy*dryy)-(Rm1+Rm2)/r;
	yr[10]=y[0];
	yr[0]=0;
}


//=========================================================================
//                      Integrals IN THE INERTIAL FRAME
//=========================================================================

double Energy(double y[])
{
	double z[5]; GetStarPos(y,z);
	double r12=(y[1]-y[2])*(y[1]-y[2])+(y[3]-y[4])*(y[3]-y[4]);
	double r13=(y[1]- z[1])*(y[1]- z[1])+(y[3]- z[2])*(y[3]- z[2]);
	double r23=(y[2]- z[1])*(y[2]- z[1])+(y[4]- z[2])*(y[4]- z[2]);
	r12=sqrt(r12); r13=sqrt(r13); r23=sqrt(r23); //squareroots

	double T=m1*(y[5]*y[5]+y[7]*y[7]);
	T+=m2*(y[6]*y[6]+y[8]*y[8]);
	T+=m3*(z[3]*z[3]+z[4]*z[4]);
	double V=-(m1*m2/r12+m1*m3/r13+m2*m3/r23);
	return 0.5*T+V;
}


double AngularMomentum(double y[])
{
	double L1=m1*(y[1]*y[7]-y[3]*y[5]);
	double L2=m2*(y[2]*y[8]-y[4]*y[6]);
	double z[5]; GetStarPos(y,z);
	double L3=m3*(z[1]*z[4]-z[2]*z[3]);
	double L=L1+L2+L3;
	return L;
}
```

```
void GetStarPos(double y[], double z[])
//y:systempos array , z[1..4]={x,y,xd,yd}
{
	z[0]=0.0;
	z[1]=-(m1*y[1]+m2*y[2])/m3;
	z[2]=-(m1*y[3]+m2*y[4])/m3;
	z[3]=-(m1*y[5]+m2*y[6])/m3;
	z[4]=-(m1*y[7]+m2*y[8])/m3;
}

void SetSystemPos(double X1[], double X2[], double Y[])
//get pos X1[4+1], X2[4+1] from bodies 1 and 2 and fill system pos data
array Y[24+1]
{
	Y[0]=X1[0];
	Y[1]=X1[1]; Y[2]=X2[1]; //x
	Y[3]=X1[2]; Y[4]=X2[2]; //y
	Y[5]=X1[3]; Y[6]=X2[3]; //xdot
	Y[7]=X1[4]; Y[8]=X2[4]; //ydot
}

double GetPPdistance(double X[])
//X:system pos
{
	double r=(X[1]-X[2])*(X[1]-X[2])+(X[3]-X[4])*(X[3]-X[4]);
	return sqrt(r);
}

double GetSP1distance(double X[])
//X:system pos
{
	double Z[NEQ];
	GetStarPos(X, Z);
	double r=(X[1]-Z[1])*(X[1]-Z[1])+(X[3]-Z[2])*(X[3]-Z[2]);
	return sqrt(r);
}

double GetSP2distance(double X[])
//X:system pos
{
	double Z[NEQ];
	GetStarPos(X, Z);
	double r=(X[2]-Z[1])*(X[2]-Z[1])+(X[4]-Z[2])*(X[4]-Z[2]);
	return sqrt(r);
}

double GetPPdistance(double X1[], double X2[])
//X1,X2 planets pos
{
	double r=(X1[1]-X2[1])*(X1[1]-X2[1])+(X1[2]-X2[2])*(X1[2]-X2[2]);
	return sqrt(r);
}
//================tools======================
void GetMasses(double *mass1, double *mass2, double *mass3)
{
	*mass1=m1; *mass2=m2; *mass3=m3;
}
void GetSysCoefs(double *d1,  double *d2, double *vmcf1,  double *vmcf2)
{
	*d1=db; *d2=da; *vmcf1=vm1; *vmcf2=vm2;
}
```