Java simulation of binaries

Frédéric Arenou

DMS-FA-03, Rev. 2.1, 11-01-02

Abstract

Java classes for the simulations of double or multiple systems (DMS, including BD or EP companions) are presented. Although all the complexity of real DMS has not been implemented, the classes allow to simulate the first order effects which will be noticeable in astrometry, radial velocity and photometry by Gaia.

1 Introduction

The Gaia simulator aims at reproducing the observational process of the Gaia satellite. The data model uses a "reference" universe made of solar system objects, extragalactic objects and stars, among them binaries and hosts of extrasolar planets.

The code which is described in this document is an all-purpose binary simulator with nothing specifically designed for Gaia, except that it is written in Java, and thus easily plugged in the current Gaia simulator. The only input to this simulator is the absolute magnitude of a primary star, which is easily given by any Galaxy model. Then, at any given date, the astrometric position, the radial velocity and photometry (in case of eclipses) of each component may be obtained.

A first version of this code had been sent to Barcelona early november. This second version implements now spectroscopic and photometric effects. A lot of simplifying assumptions make this code still crude, but it is unlikely that more sophisticated models are needed for the moment. The main goal of this simulator is to suggest calling conventions and to introduce in the current Gaia simulator a little more complexity than what is expected from single stars alone. Improvement of this code will come in future versions, and no science can be done with the code as is.

2 Class dms_orbit

Before describing the simulation itself, it is more logical to address first the class describing a binary or multiple star. In this simple class, we consider a primary star (with index 0) with several companions (index $i > 0$). Only a hierarchical Keplerian motion is implemented, with no interactions between companions. We assume also that the companions are on the main-sequence (down to BD/EP) for what concerns the mass-radius relations.

A system is either directly constructed by `dms_orbit()`, with several arguments indicating the number of companions to the primary star (so that the total number of components is `NumberOfCompanions+1`), the masses of all components and their orbital parameters. The radii of the components are optional (they are used in case of eclipses only, and computed from a mass-radius relation if not given as input). Alternatively, a system may be constructed by calling `dms_orbit()` with no arguments, followed by a call to `primary()`, then `companion()`, for each companion.

- **dms_orbit**: constructs all the components of the multiple system

Input1: number of companions to the primary `(int)`

Input2: masses ($M_\odot$) of all components `(double [numberofcomp+1])`

Input3: orbital parameters of all components `(double [numberofcomp+1][6])`, `OrbitalParameters[0]` being ignored

[[Input4: (optional) reference component `(int[6])`

[Input5 :]] (optional) radius (in solar radius) of all components `(double [numberofcomp+1])`

Note1: The orbital parameters are P (days), T (days), e, ω_2 (deg), i (deg), Ω (deg), in this order, with P in OrbitalParameters[0], etc.

Note2: The companions are implicitely numbered by increasing period. The reference component array is reserved for future versions where the true motion of a complex multiple system is taken into account.

- **primary**: constructs only the primary of the multiple system, and allocates data

Input1: number of companions to the primary `(int)`
Input2: mass (in $M_\odot$) of primary `(double)`
[Input3 :] (optional) radius (in $R_\odot$) `(double)`

- **companion**: creates only one of the companions of the multiple system

Input1: the number (> 0) of this companion `(int)`
Input2: mass (in ($M_\odot$)) of this companion `(double)`
Input3: orbital parameters (see above) `(double[6])`
[Input4: (optional) reference component `(int)`
[Input5 :]] (optional) radius (in $R_\odot$) `(double)`

- **printCompanionInfo**: prints what is known about the orbital parameters of the companion whose number is given as argument

Input: number (> 0) of this companion `(int)`

- **getCompanionInfo**: returns the orbital parameters of the companion

Input: number (> 0) of this companion `(int)`
Output: the orbital parameters (see above) of the companion `(double[6])`
Note: the angles are here in radian, not degrees, since this is the internal format which is used

- **getMass**: returns the mass of the component

Input: number (≥ 0) of this component `(int)`
Output: the mass ($M_\odot$) of the component `(double)`

- **getRadius**: returns the radius of the component

Input: number (≥ 0) of this component `(int)`
Output: returns the radius ($R_\odot$) of the component `(double)`

- **isEclipsing**: returns true if the companion will periodically eclipse the primary. This depends on the inclination and on the radius of this companion and of the primary.

Input: number (> 0) of this companion `(int)`
Output: returns true if transits will occur `(boolean)`

Once the system is constructed, methods are provided to predict the position, radial velocity and magnitude of each component at any given date. This is done by a call to `OrbitalPosition()` with this date as argument. Subsequent call to `getKsi(), getEta()` give the position offset (in A.U.) of a component, `getVra()` give the radial velocity offset with respect to the systemic velocity, `getFlx()` the flux factor, etc.

- **OrbitalPosition**: computes the position of all components at some given date

Input: date (days) `(double)`
Note: there is no explicit origin for the date. It can be given e.g. with respect to HJD 2440000, provided that it is consistent with the given T periastron date

- **getKsi**: returns the current ξ position on the tangent plane of a component

Input: number (≥ 0) of this component `(int)`
Output: x offset in A.U. wrt the barycenter of the system `(double)`

- **getEta**: returns the current η position of a component on the tangent plane

Input: number (≥ 0) of this component `(int)`
Output: y offset in A.U. wrt the barycenter of the system `(double)`

- **getRho**: returns the separation between the component and the reference component

Input: number (≥ 0) of this component `(int)`
Output: separation (A.U.) between component and primary `(double)`

- **getRad**: returns the distance offset along line of sight of a component

Input: number (≥ 0) of this component `(int)`
Output: distance offset in A.U. wrt the barycenter `(double)`

- **getVra**: returns the radial velocity offset of a component

Input: number (≥ 0) of this component `(int)`
Output: radial velocity offset in km/s wrt the barycenter `(double)`

- **getFlx**: returns the flux factor of a component. This factor is to be multiplied to the current flux of the object to get the actual flux observed at the given date.

Input: number (≥ 0) of this component `(int)`

Output: flux factor: will be [0-1[if this component is eclipsed and 1 outside eclipse `(double)`

Note: The components are assumed spherical, with no mass transfer, no limb darkening...

- **getLtt**: returns the light-time travel of a component

Input: number (≥ 0) of this component `(int)`

Output: LTT offset in days wrt the barycenter `(double)`

Note: added for completeness. This is needed to modify the light curve if the primary is variable; this is taken into account if the primary and secondary of the system are eclipsing.

3 Class dms_simu

The `dms_simu` class extends the `dms_orbit` class. For this reason, the construction of a `dms_simu` object gives also access to the methods described in the section above.

This class allows to create a random system: given a star with its absolute magnitude, the star will become single or double, etc, with some probability. If not single, the masses and orbital parameters of its companions are chosen at random. For the moment the simulation is rather crude and far from representing the actual properties of galactic binaries.

What is needed as input for the constructor `dms_simu` is at least the absolute magnitude of the primary. If colour and mass are not given, they are computed from the absolute magnitude, assuming a main-sequence star. If the system randomly drawn contains more than one star, the companion(s) have a mass drawn at random, which gives a colour and an absolute magnitude, still assuming main-sequence stars.

- **dms_simu**: construction of a random single, double or multiple system

Input1: V absolute magnitude of primary `(double)`

[Input2: (optional) $V-I$ colour of primary `(double)`

[Input3]:] (optional) Mass ($M_\odot$) of primary `(double)`

Note: It is likely that the age of the system should be introduced as optional parameter in next versions.

- **getNumberOfCompanions**: returns the number of companions

Output: the number of companions to the primary `(int)`

Note: The total number of components in this system is `getNumberOfCompanions()`+1

- **getAbsMag**: returns the V absolute magnitude of the component

Input: number (≥ 0) of this component `(int)`

Output: the V absolute magnitude of the component `(double)`

Note: This is obtained through a rough mass-luminosity calibration, with nothing serious for brown dwarfs and planets. For binaries what is needed now is just a rough indication of the light contamination from components. For BD/EP this is mostly negligible and has not be taken into account, and in any case this should wait for future versions where the age (and metallicity?) would be taken into account.

- **getVmICol**: returns the $V-I$ colour of the component

Input: number (≥ 0) of this component `(int)`

Output: the $V-I$ color of the component `(double)`

Note: Same note as above.

An rather obvious extension to this class would be a class "star" which would have a parallax and an extinction, so that `getAbsMag()` combined with `getVmICol()` and `getFlx()` could give the apparent magnitude in the desired band at the desired date; and the position given in A.U. by `getKsi()`, `getEta()` and `getRho()` could be converted into angular units.

4 Class dmstest

The purpose of this class is only to test the two classes above, using several examples, and is provided only to show how to implement the methods in practice. For the moment there are 5 examples, which can be tested putting a number between 1 and 5 on the command line:

1. creates a random system (will possibly be a single star)

2. creates a sample of random systems
3. astrometric motion of an intermediate period binary
4. radial velocity of Ups And due to its planets
5. R CMa, eclipsing binary with LTT effect

There is a lot to be done to render this code more realistic, and to thoroughly test it... but it can be used as a template.

The complete source code is available at `http://wwwhip.obspm.fr/~arenou/gaia/DMS-FA-03/`. Only the `dmstest` code is given below.

```
//
//+++++++++++++++++
//.IDENTIFICATION $Id: DMS-FA-03.tex,v 2.1 2002/01/11 15:55:49 arenou Exp $
//.TYPE class
//.LANGUAGE java
//.AUTHOR F.Arenou
//.ENVIRONMENT tested on MacOS X, should run anywhere ;-)
//.KEYWORDS orbits, Gaia
//.PURPOSE this is a test program of the classes dms_simu & dms_orbit
//.INPUT choose 1, 2, 3, 4 or  5 on command line to run one of the tests
//----------------
//
import java.text.* ;
import java.io.*;

public class dmstest {

    //
    // choose which test
    //
    public static void main (String args[]) {
        String myarg="0" ; if (args.length != 0) myarg=args[0] ;
        if (myarg.startsWith("1")) creates_one_random_system() ;
        else if (myarg.startsWith("2")) creates_a_random_sample() ;
        else if (myarg.startsWith("3")) test_a_true_star() ;
        else if (myarg.startsWith("4")) test_a_multiple_planet() ;
        else if (myarg.startsWith("5")) test_a_LTT_eclipsing() ;
        else System.out.println("Usage: dmstest #, with 1<=#<=5") ;
    }


    //
    // this test creates a random system (if the star is not single)
    // then prints the information about this system
    // and the position offset with respect to barycenter at some date
    //
    public static void creates_one_random_system () {
        // creates a random new system given the absolute mag of primary
        dms_simu my_system = new dms_simu(4.77) ; // abs magnitude of a G star
        int number_comp = my_system.getNumberOfCompanions() ;
        System.out.println("A system randomly chosen:") ;
        System.out.println("Mass of primary = " + my_system.getMass(0)) ;
        System.out.println("V-I of primary = " + my_system.getVmICol(0)) ;
        if (number_comp == 0) {
            System.out.println("This is a single star") ;
        } else {
            System.out.println("This is a system with a primary and " +
            number_comp + " companion(s)") ;
            for (int i=1; i<= number_comp; i++) {
                System.out.println("Companion " + i) ;
                System.out.println("\tMass = " + my_system.getMass(i)) ;
                System.out.println("\tRadius = " + my_system.getRadius(i)) ;
```

```
            System.out.println("\tV absolute magnitude = " +
            my_system.getAbsMag(i)) ;
            System.out.println("\tV-I color = " + my_system.getVmICol(i)) ;
            my_system.printCompanionInfo(i) ;
        }
        System.out.println("Position wrt barycenter at day "+
        "(e.g.) -100 will be (in A.U.):") ;
        my_system.OrbitalPosition(-100.) ; // computes position of all comp.
        for (int i=0; i<= number_comp; i++) {
            System.out.println("Component " + i + ", ksi=" +
            my_system.getKsi(i) + ", eta=" + my_system.getEta(i)) ;
        }
    }
}

//
// statistics for a large sample
//
public static void creates_a_random_sample () {
    DecimalFormat fmt = new DecimalFormat("##.####");
    int SampleSize = 1000 ; // of 1000 stars
    dms_simu [] sample = new dms_simu[SampleSize] ;
    try {
        String FileName = "sample.system"; // name to be changed...
        PrintWriter OutputFile=new PrintWriter(new
                BufferedWriter(new FileWriter(FileName)));
        System.out.println("Writing parameters of a simulated sample "+
        "in file "+FileName) ;
        System.out.println("Format: number of component, then "+
        "for each component: mass, absmag, V-I, radius, period");
        for (int star=0; star<SampleSize; star++) {
            sample[star] = new dms_simu(4.77) ;   // G-type primary
            int number_comp = sample[star].getNumberOfCompanions() ;
            OutputFile.print(number_comp) ;
            for (int i=0; i<= number_comp; i++) {
                OutputFile.print("\t" +
                fmt.format(sample[star].getMass(i)) + "\t" +
                fmt.format(sample[star].getAbsMag(i)) + "\t" +
                fmt.format(sample[star].getVmICol(i)) + "\t" +
                fmt.format(sample[star].getRadius(i)) + "\t" +
                fmt.format(sample[star].getCompanionInfo(i)[0])) ; // e.g.
                // abs mag and V-I should be ignored for BD and EP !!!
            }
            OutputFile.println() ;
        }
        OutputFile.close();
    }
    catch (Exception e) { e.printStackTrace(); } // in case of file IO error
}


//
// shows the astrometric motion of a 900 day period binary (HIP 39903)
//
public static void test_a_true_star () {
    // P (days), T (HJD-2440000), e, omega_2 (deg), i (deg), Omega (deg)
    double [] param39903 = { 899.2761,1845.6018,0.1117,136.49+180.,21.81,186.85 } ;
    double massprimary = 1.26, masssecond = 0.434 ; // masses primary& secondary
    double plx39903 = 48.941 ;  // parallax in mas
    double delta_alf0, delta_del0 ; // reflex motion on the sky
    double delta_alf1, delta_del1 ; // secondary motion on the sky
```

```
    double rho ; // angular separation
    DecimalFormat fmt = new DecimalFormat("######.###");

    System.out.println("This is the astrometric binary HIP 39903:") ;
    // We could have constructed in one instruction the system
    // but we show how it can be done in 3 steps
    dms_orbit double_star = new dms_orbit() ; // empty construction
    double_star.primary(1, massprimary) ;  // primary with 1 companion
    double_star.companion(1, masssecond, param39903) ; // then init secondary

    // foreach time (interval of Hipparcos observation in HJD-2440000),
    // the reflex motion of the primary in milliarcsec is printed
    try {
        String FileName = "HIP39903.pos"; // to be changed...
        PrintWriter OutputFile=new PrintWriter(new
                BufferedWriter(new FileWriter(FileName)));
        System.out.println("Writing the reflex motion alfa, delta and rho (mas) "+
        "versus time (HJD-2440000) in file "+FileName) ;
        for (double t=7800; t<9100; t+=10) {
            // compute position on tangent plane of all components at time t
            double_star.OrbitalPosition(t) ;
            // get position of primary and convert it to angular coordinates
            delta_alf0 = plx39903*double_star.getKsi(0) ; // in alpha
            delta_del0 = plx39903*double_star.getEta(0); // and delta
            rho = plx39903*double_star.getRho(1); // separation 0-1 (AB)
            OutputFile.println(t + "\t" + fmt.format(delta_alf0) + "\t" +
            fmt.format(delta_del0) + "\t" + rho) ;
        }
        OutputFile.close();
    }
    catch (Exception e) { e.printStackTrace(); } // in case of file IO error
}


//
// orbital parameters of star Ups And, assuming i=75, O=0 for all
//
public static void test_a_multiple_planet () {
    int number_comp = 3 ; // 3 planets
    // orbital parameters of star Ups And, assuming i=75, O=0 for all
    // P (days), T (HJD-2450000), e, omega_2 (deg), i (deg), Omega (deg)
    double [][] paramsUpsAnd = { {},  // nothing for the star then
    {    4.617, 2.24, 0.034, 83  +180,75,0 },// orbital param for the 3 planets
    {  241.2,  154.9, 0.18, 243.6+180,75,0 },
    { 1308.5,     44, 0.31, 255  +180,75,0 } } ;
    double [] massesUpsAnd = {1.3,  0.000678, 0.00196, 0.004097} ; // solar masses

    // creates the system: number of companions, masses, orbital parameters
    dms_orbit planet_star=new dms_orbit(number_comp,massesUpsAnd,paramsUpsAnd) ;

    System.out.println("This is the Ups And system with a primary and " +
        number_comp + " planets") ;
    for (int i=1; i<= number_comp; i++) {
        System.out.println("Companion " + i) ;
        System.out.println("\tMass = " + planet_star.getMass(i)) ;
        planet_star.printCompanionInfo(i) ;
    }

    try {
        String FileName = "upsand.rv"; // to be changed...
        PrintWriter OutputFile=new PrintWriter(new
```

```
                BufferedWriter(new FileWriter(FileName)));
        // the reflex motion of the primary in m/s is printed
        System.out.println("Writing the radial velocity (m/s) of primary "+
        "versus time (HJD-2450000) in file "+FileName) ;
        for (double t=0; t<3500; t+=0.5) {
            // compute position of all components at time t
            planet_star.OrbitalPosition(t) ;
            // get radial velocity
            double delta_vr = 1000*planet_star.getVra(0) ; // in m/s
            OutputFile.println(t + "\t" + delta_vr) ;
        }
        OutputFile.close();
    }
    catch (Exception e) { e.printStackTrace(); } // in case of file IO error
}


//
// star RCMa, assuming spherical stars, an eclipsing binary with a third comp.
//
public static void test_a_LTT_eclipsing () {
    int number_comp = 2 ;
    double [][] params = { {},
    { 1.13594197,-9563.4193, 0,3.612+180,79.47,0 }, // Omega ?, T=primary eclipse
    { 33892.569,9349.145,0.494,10.55+180,91.79,262.52 } } ; // T-2440000
    params[1][1] = Primary2Periastron(params[1][1],params[1][0],
        params[1][2],params[1][3]) ; // correct: periastron instead of primary eclipse
    double [] masses = {1.07, 0.17, 0.342} ; // in M_sun
    double [] radii = {1.57, 1.06, 0.5} ;  // in R_sun,
    double [] Vmag = {5.7, 9, 15} ;  // magnitudes (invented)
    int [] RefPrimary = {0,0,0} ; // anything else not yet implemented.
    DecimalFormat fmt = new DecimalFormat("######.####");

    // create the new system: number of companions, masses, orbital param
    dms_orbit eclipsing_star =
        new dms_orbit(number_comp,masses,params,RefPrimary,radii) ;
    System.out.println("This is (approx.) the RCMa system with an eclipsed "+
        "primary and " + number_comp + " companions, with LTT effect") ;
    for (int i=1; i<= number_comp; i++) {
        if (eclipsing_star.isEclipsing(i))
        System.out.println("Companion " + i + " (will eclipse primary)") ;
        else System.out.println("Companion " + i + " (will not eclipse primary)") ;
        System.out.println("\tMass = " + eclipsing_star.getMass(i) +
        "\tRadius = " + eclipsing_star.getRadius(i)) ;
        eclipsing_star.printCompanionInfo(i) ;
    }

    try {
        String FileName = "rcma.lc"; // to be changed...
        System.out.println("Writing the light curve (mag) versus "+
        "time (days) in file "+FileName) ;
        PrintWriter OutputFile=new PrintWriter(new
                BufferedWriter(new FileWriter(FileName)));
        double flux0 = Math.pow(10,-0.4*Vmag[0]) ;
        double flux1 = Math.pow(10,-0.4*Vmag[1]) ;
        for (double t=-9564.2; t<-9563; t+=0.0001) {
            eclipsing_star.OrbitalPosition(t) ; // around the primary eclipse T0
            double mag = -2.5*Math.log( eclipsing_star.getFlx(0)*flux0 +
                            eclipsing_star.getFlx(1)*flux1 )/Math.log(10) ;
            OutputFile.println(fmt.format(t) + "\t" + mag) ;
        }
```

```
            OutputFile.close();
        }
        catch (Exception e) { e.printStackTrace(); } // in case of file IO error

        try {
            String FileName = "rcma.ltt"; // to be changed...
            System.out.println("Writing light-Time Travel (s) for primary "+
            "vs time (year) in file "+FileName) ;
            PrintWriter OutputFile=new PrintWriter(new
                    BufferedWriter(new FileWriter(FileName)));
            for (double t=-30000; t<12000; t+=100) {
                // compute position of all components at time t
                eclipsing_star.OrbitalPosition(t) ;
                // get LTT
                double delta_ltt = 86400*eclipsing_star.getLtt(0) ;// in seconds
                double year = (t-8349.0625)/365.25+1991.25 ; // in julian year
                OutputFile.println(fmt.format(year) + "\t" + fmt.format(delta_ltt));
            }
            OutputFile.close();
        }
        catch (Exception e) { e.printStackTrace(); } // in case of file IO error

    }

}
```