Bakkalaureatsarbeit

Lie-Integrator

Verfasser: Christoph Saulder

Betreuer: Rudolf Dvorak

ABSTRACT

The aim of this work is to develop a modern and more flexible Lie-Integrator based on the work of A. Hanslmeier and R. Dvorak from 1983. The program shall be able to calculate the movement of objects in n-body systems with Newtonian gravitational interaction. Furthermore the user shall be able to evaluate the results of these calculations with the same program. In addition to that there shall be a function to do series of calculations with slightly different initial conditions automatically. The result of these considerations is a modern numerical integrator with a graphical surface and an easy handling. The integrator is programmed in Delphi6 and uses OpenGL for 3D-applications.

KEY WORDS: Lie-Integrator, celestial mechanics, n-body systems, computer program, numerical integration, Delphi6.

TABLE OF CONTENTS

1. INTRODUCTION

Lie-Integration is a method to solve numerically systems of differential equations by using lie-series. For this problem we are considering the Newtonian movement equations for objects with mutual gravitational interactions. The hard work to apply the Lie-Integration method on these equations and to develop an algorithm which can be processed into a computer program has already be done by A. Hanslmeier and R. Dvorak[i] in 1983 and I am very thankful being allowed to use the source code of their program. I translated the source code of their program (to which I'll be referring from now as "Old Lie-Integrator (OLI)") form Fortran77 into Delphi6 and improved it a little by using new features of my new programming language. The New Lie-Integrator (as I'll be referring to my program from now) uses a graphical interface with a clearly labelled input mask to get the input data instead of input files like the OLI, but you can also save and load your input data to and from files. In addition to the simple calculation of entered parameters of one system, the program can perform a sequence of calculation of systems with slightly different initial conditions, so that you can "scan" through the values of the orbit elements of one object. Moreover the NLI possesses a special tool to calculate artificial radial velocity curves of exoplanetary systems. This program feature can help to discover a deviation from the Keplerian fits which are usually used to find the orbit elements of multiple exoplanetary systems, but which won't be valid if there are strong dynamical interactions between the planets. The most important improvement of the NLI compared to OLI is that you won't need another program for most kinds of scientific evaluations of your data. The NLI is equipped with a couple of tools which are capable to produce the needed diagrams. You can get tables, line diagrams, stability maps and even 3D-animitions just by opening the right window in the program. Summing up we can say that the NLI is a modern program which unifies the high speed and accuracy of the OLI and the user friendliness of up-to-date professional software.

2. LIE-INTEGRATION

2.1 Sophus Lie[ii]

Sophus Lie was a Norwegian mathematician, who was born on the17th December 1842 in Nordfjordeid. He tried to study several different sciences (e.g. astronomy) but finally got to mathematics. There he published his first paper in 1869. Only a few years later he developed the Lie-Integration as a method to solve differential equations in the years 1871 and 1872. Furthermore Sophus Lie is known because of several works about geometry, transformation groups and commutator algebra. During his life he taught in Leipzig and Oslo. Personally Sophus Lie didn't seem to be the easiest man to get around with. It is told that he had a tendency to depressions and got in quarrel with his colloquies. In spite of this he was a very familiar human. Sophus Lie died on the 18th February 1899 in Oslo.

Fig. 1: Sophus Lie

2.2 Lie-Integration as method to solve differential equations

First of all it is to mention that the Lie-Integration, despite the fact it's still an integration method, has very little to do with the common Riemann-Integration. We start by defining a Lie-Operator:

$$(1) \quad D = \theta_1 \frac{\partial}{\partial z_1} + \theta_2 \frac{\partial}{\partial z_2} + ...\theta_n \frac{\partial}{\partial z_n}$$

This operator is a linear differential operator, where the functions θ_1 to θ_n may depend on all parameters z_1 to z_n as long as the function can be expanded into a converging power series. The operator can be applied on a function f, which must fulfil the same criterion. A multiple application of the Lie-Operator on the function f can be written like this:

$$(2) \quad \begin{aligned} D^2 f &= D(Df) \\ D^n f &= D(D^{n-1} f) \end{aligned}$$

In the next step we will define Lie-Series L(z,t) by using the Lie-Operator:

$$(3) \quad L(z,t) = e^{t.D} f(z)$$

By expanding the e-function into a Taylor-series we get:

$$(4) \quad L(z,t) = \sum_{i=0}^{\infty} \frac{t^i}{i!} D^i f(z)$$

$$(5) \quad L(z,t) = f(z) + t.Df(z) + \frac{t^2}{2!} D^2 f(z) + ...$$

Note that the Lie-Series can also depend on more than one parameter z like the Lie-Operator.

Now we are ready to have a look at systems of differential equations. The Lie-Integration method only works on systems of first order differential equations. All systems with higher

orders have to be reduced to a system of first order differential equations. Let's have a closer look at a system of first order differential equations:

$$(6) \quad \frac{dz_i}{dt} = \theta_i(z_1, z_2, ...z_n)$$

with following initial conditions:

$$(7) \quad z_i(t=0) = \xi_i$$

These initial conditions are for time t=0, but you can modify them easily all other times. Now we are going to use the Lie-Operator in following form:

$$(8) \quad D = \sum_{i=1}^{n} \theta_i(\xi_1, \xi_2, ..., \xi_n) \frac{\partial}{\partial \xi_i}$$

The solution of the differential equation system (6) with the initial condition (7) can be written as:

$$(9) \quad z_i = e^{tD} \xi_i$$

You can prove this relation by using the Vertauschungssatz of Lie-series and differentiating. A detailed proof can be found in "Chaos and Stability in Planetary Systems" by R. Dvorak, F. Freistetter and J. Kurths[iii]. It is more useful for numerical applications to expand the right-hand side of relation (9) into a Taylor-series:

$$(10) \quad z_i = \xi_i + t.D\xi_i + \frac{t^2}{2!} D^2 \xi_i + \frac{t^3}{3!} D^3 \xi_i + ...$$

2.3 Application of the Lie-Integration method on the 2-body and the n-body problem

This work has already been done by A. Hanslmeier and R. Dvorak when developing the OLI in 1983. I'll briefly summarize their paper which is based on the work of W. Gröbner[iv] from 1967.

First we will do a short review of the 2-body problem. The equations of motion for this problem written in relative coordinates are:

$$\ddot{r} + \frac{r}{|r|^3} = 0 \tag{11}$$

Due to the fact that the Lie-Integration method only works on systems of first order differential equations, we must separate the system (11) into:

$$\begin{aligned} \dot{r} &= u = \theta_1 \\ \dot{u} &= -\frac{r}{|r|^3} = \theta_2 \end{aligned} \tag{12}$$

Now we are able to define the Kepler-Operator:

$$D = u\frac{\partial}{\partial r} - \frac{r}{|r|^3}\frac{\partial}{\partial u} \tag{13}$$

$$r = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}; \quad u = \dot{r} = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \end{pmatrix}; \quad \rho = |r| \tag{14}$$

$$D = \sum_{i=1}^{3}\left(\eta_i \frac{\partial}{\partial \xi_i} - \frac{\xi_i}{\rho^3}\frac{\partial}{\partial \eta_i}\right) \tag{15}$$

By introducing new variables (14) we can also write the Kepler-Operator in a different form (15). The solution of this problem can be calculated by using the Kepler-Operator as Lie-Operator and solving it with the Lie-Integration method:

$$(16)\quad \begin{aligned} L(\xi_i,t) &= e^{tD}\xi_i(t=0) \\ L(\eta_i,t) &= e^{tD}\eta_i(t=0) \end{aligned}$$

In the next step you must find a recurrence formula for the multiple application of the Kepler-Operator (Lie-Operator) on these variables (initial conditions). I am skipping this long procedure by referring to the paper mentioned above and show you the final result:

$$(17)\quad D^n\xi_i = -\sum_{v=0}^{n-2}\binom{n-2}{v} D^v\rho^{-3}D^{n-2-v}\xi_i$$

Note that you can get nearly the same formula for η_i because of:

$$(18)\quad D\xi_i = \eta_i$$

Now we can use the methods we learned by solving the 2-body problem to find a numerical solution for general n-body problem. Therefore we start again with the equation of motion:

$$(19)\quad \ddot{x}_i = \sum_{\substack{j=1\\ j\neq i}}^{n} m_j \frac{x_j - x_i}{\left|x_j - x_i\right|^3} \qquad i = 1,2,...,n$$

We reduce this system of differential equations with the same substitution as we have done in the 2-body problem. After defining a Lie-Operator for the 3-body problem, introducing new variables and finding a recurrence formula for 3-body case, it is comparably simple to generalize it for the n-body-problem:

$$(20)\quad \begin{aligned} T_i(k,a) = D^n\xi_i^k &= -m_k\sum_{v=0}^{a-2}\binom{a-v}{v} D^v\Phi_{lk}T_i(k,a-2-v) - m_l\sum_{v=0}^{a-2}\binom{a-v}{v} D^v\Phi_{lk}T_i(k,a-2-v) \\ l &= 1,...,n \qquad k = 1,...,n \qquad l \neq k \end{aligned}$$

All these thoughts and calculations lead to a formalism which enables us to evaluate all terms ($T_i(k,a)$) of the Lie-series. The OLI was developed based on these mathematics and of course the NLI uses exactly the same algorithms to calculate the movement of planets and asteroids.

2.4 Advantages of the Lie-Integration method

The Lie-Integration method holds a couple of very useful advantages. It has a very high accuracy depending on the number of Lie-terms calculated. Furthermore it can operate with a large step length which makes it very fast. In addition to that the step length is flexible and can be adapted to the situation. So in case of problems of celestial dynamics it can calculate very fast when the planets are far away but it can also handle close encounters with a high accuracy by temporarily reducing the step length without loosing much calculation time.

3. PROGRAMMING LANGUAGE “DELPHI6”

3.1 Programming environment

The Borland Delphi 6.0 Personal Edition with Service Pack 2 has been used to develop the NLI. The programming editor has a visual surface to create program windows and other common tools and components by drawing (drag and drop) them up. Furthermore there is also a text editor to write the source code to control these components. A Compiler and a useful help are included too.

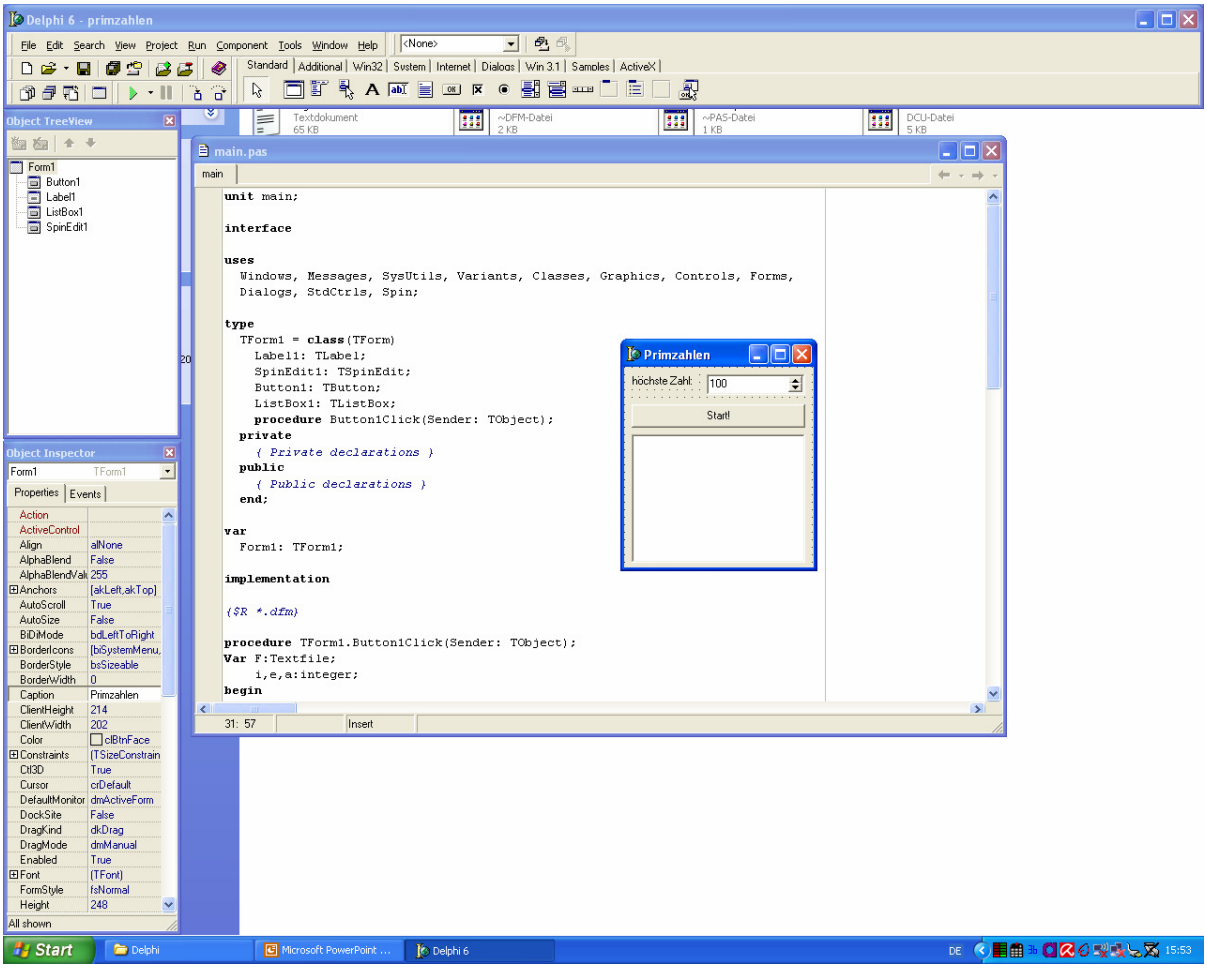

Fig. 2: Delphi6 programming environment

3.2 Features of Delphi6

First of all I've got to mention that programming language Delphi could also be called "Visual Pascal", because the syntax is of Delphi is based on Pascal like Visual C++ is based on C. Delphi6 is equipped with some very useful features: There is for example a 80-bit floating point variable called "extended" which covers values from $3.6\ 10^{-4951}$ to $1.1\ 10^{4932}$. Delphi6 can also handle dynamic arrays. This is one of the most important improvements compared to its predecessors and I've used this feature very often for the NLI to define the dimension of an array on runtime. Furthermore Delphi6 translates the source code directly into Assembler and not like many other programming editors first into C and then into Assembler. This is very important for the runtime speed of programs and fastness is essential especially for the usage of our program. In addition to that Delphi6 uses elements of object orientated programming languages as well as elements for structured programming languages. You can't say that Delphi can be classified as either the one or the other. This mixture gives you the opportunity to use the best features and elements of both types of programming languages. The graphical surface is very useful for developing user-friendly programs. Plenty of prefabricated components and libraries for all kinds of applications are another advantage of Delphi6. I've used some of them for the NLI, mainly for the graphical surface and the 3D-animations.

3.3 Syntax

I've already mentioned that the syntax strongly resembles Pascal. Now I am going more into detail by showing you an example source code in Fortran77 (the programming language of the OLI) and in Delphi6 of the same simple program. The following program (in both languages) can calculate prime numbers up to a certain number. I've chosen a very slow but simple algorithm for this showpiece program.

In Fortran77

```
integer a,i,e,max

      print*,'höchste Zahl eingeben'
      read*,max
      open(unit=11,file='ergebnis.txt',status='unknown')
      do 100 i=2,max
      a=0
      do 200 e=2,i-1
      if (MOD(i,e)==0) then
      a=a+1
      end if
200   continue
      if (a==0) then
      write(11,*) i
      print*, i
      end if
100   continue
      close(11)
      stop
      end
```

In Delphi6

```
procedure TForm1.Button1Click(Sender: TObject);
Var F:Textfile;
    i,e,a:integer;
begin
assignfile(F,'ergebnis.txt');
rewrite(F);
for i:=2 to spinedit1.Value do
begin
a:=0;
for e:=2 to i-1 do
begin
if (i mod e)=0
then
inc(a);
end;
if a=0
then
begin
writeln(F,inttostr(i));
listbox1.Items.Add(inttostr(i));
end;
end;
closefile(F);
end;
```

Some parts of the source code look similar, others don't. Writing to a file in Fortran77 consists of basically 3 commands:

```
open(unit=number,file='filename',status='unknown')   C opens file
write(number,*) variable   C writes a line
close(number)   C closes file
```

While in Delphi6 you've to use 4 commands:

```
assignfile(identifier,filename);   // opens file
rewrite(identifier);   //sets file ready to write(overwrite)
writeln(identifier,variable);   //writes a line
closefile(identifier);   //closes file
```

Let's have a look at another example: a for-loop in Fortran77 must be written like this:

```
do number variable=startvalue, stopvalue   C define and begin loop
COMMANDOS   C commandos that are whished to be done in the loop
number continue   C end loop
```

The same code lines in Delphi6 have to look like this:

```
for variable:=startvalue to stopvalue do   // define loop
begin   // begin bracket
COMMANDOS   // commandos that are whished to be done in the loop
end;   // end bracket
```

Let's compare an If-Case in Fortran77:

```
if (logical comparison) then   C make a comparison: if true then continue
COMMANDOS   C commandos that are whished to be done in this case
else   C if comparison is false then continue here
COMMANDOS   C commandos that are whished to be done in this case
end if   C comparison complete
```

with one in Delphi6:

```
if (logical comparison)   // C make a comparison
then   // if comparison is true then continue
begin   // begin bracket
COMMANDOS   // commandos that are whished to be done in this case
end   // end bracket
else   // if comparison is false then continue here
```

```
begin   // begin bracket
COMMANDOS   // commandos that are whished to be done in this case
end;   // end bracket
```

After these basic syntax elements I've to tell you some more differences between Delphi6 and Fortran77. Firstly variables must be declared at the beginning of a program or a procedure in Delphi6, while in Fortran77 they may be declared everywhere in the source code. Furthermore every code line has to be ended by a semicolon in Delphi6. This doesn't exist in Fortran77. There are many more differences and characteristics of both programming languages, but to mention and explain them all would blast the volume of this paper. I hope this overview has given you a small insight into the syntax of Delphi6. I am closing this chapter with a comparison of the program surface of the prime number program above in Fortran77 and Delphi6.

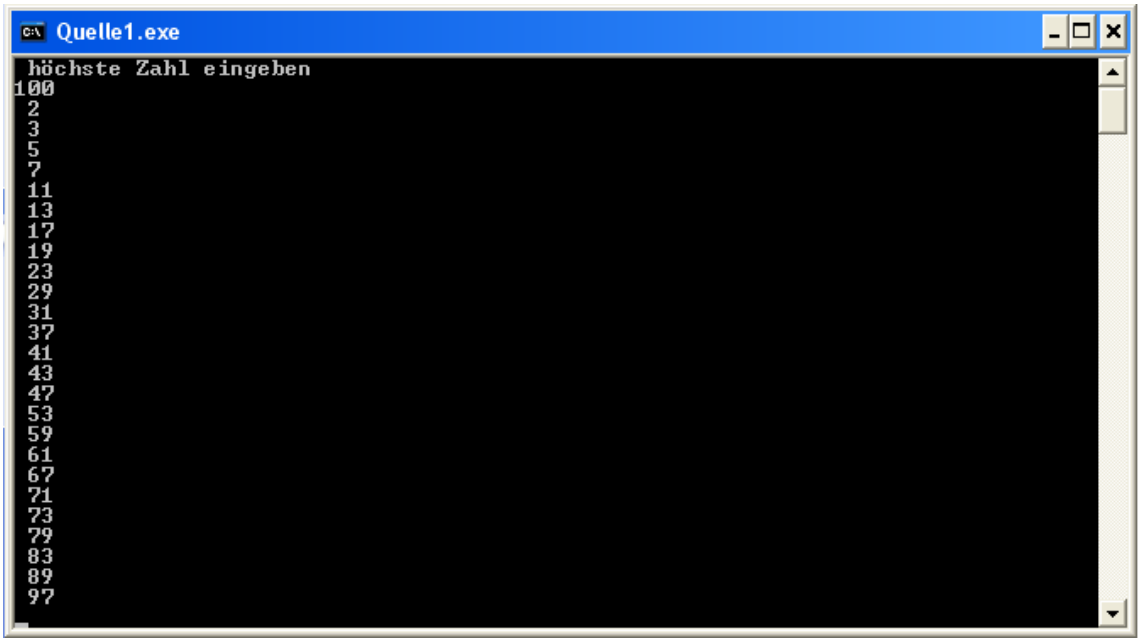

Fig. 3: the appearance of the running prime number program in Fortran77

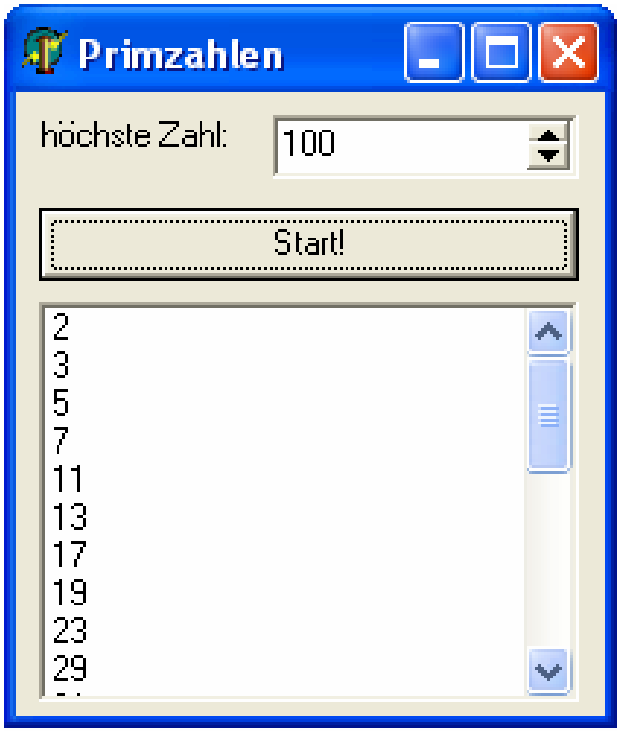

Fig. 4: the appearance of the running prime number program in Delphi6

4. MODIFICATIONS

In this part of the paper I will write about my modifications on the core program (the part that I've translated from the OLI). I will treat the program elements I've added to the original program in the next chapter. Due to the fact that the OLI all over all is a very efficient and well working program, I've avoided large changes in the mathematical functions. Most modification, I have done on the core program, were intended to make it more flexible. But I have also tried to improve the speed of the program by creating an option to turn off features which aren't always needed for the calculations. The number of objects (massive and mass less) has been limited in the OLI. The only possibility to change this limit has been to edit an include file and recompile the program. This isn't any longer necessary with the NLI, because of using dynamical arrays. In the NLI almost all parameters of the integrator are editable while running the program, even some parameters that couldn't be changed in the OLI without editing the source code of the program. For example: To indicate that an object gets very instable you can use its eccentricity. Of course it's not a perfect instability indicator, but this orbit parameters must increase up to greater or equal to1 to get an eject. Furthermore a high eccentricity also increases the probability of close encounters and that cause ejects too. Another argument to exclude high eccentric objects (as long as they are mass less) from calculations is that their extreme orbits decrease the step length of the integrator and slow it down. The limit of eccentricity in the OLI has been 0.5. This seems to be quite low, but simulations have shown that it is high enough (see Fig.5).

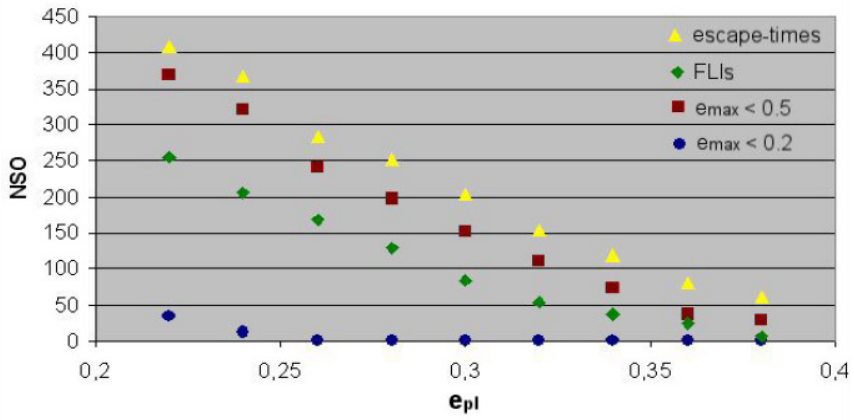

Fig. 5: number of objects left after a long-time calculation of the restricted three-body-problem; it shows little difference between the result of escape-time and e_{max} of 0.5.

But with respect to the discovery of many exoplanets during the last decade I must admit that a fix limit of eccentricity of 0.5 is no longer reasonable. 20 years ago nobody was expecting that we will discover massive planets with an eccentricity greater than 0.5, but now there are several known planets with stable orbits and an eccentricity much greater than 0.5. These mostly very massive planets can force other objects in the systems on orbits with a comparable high eccentricity.

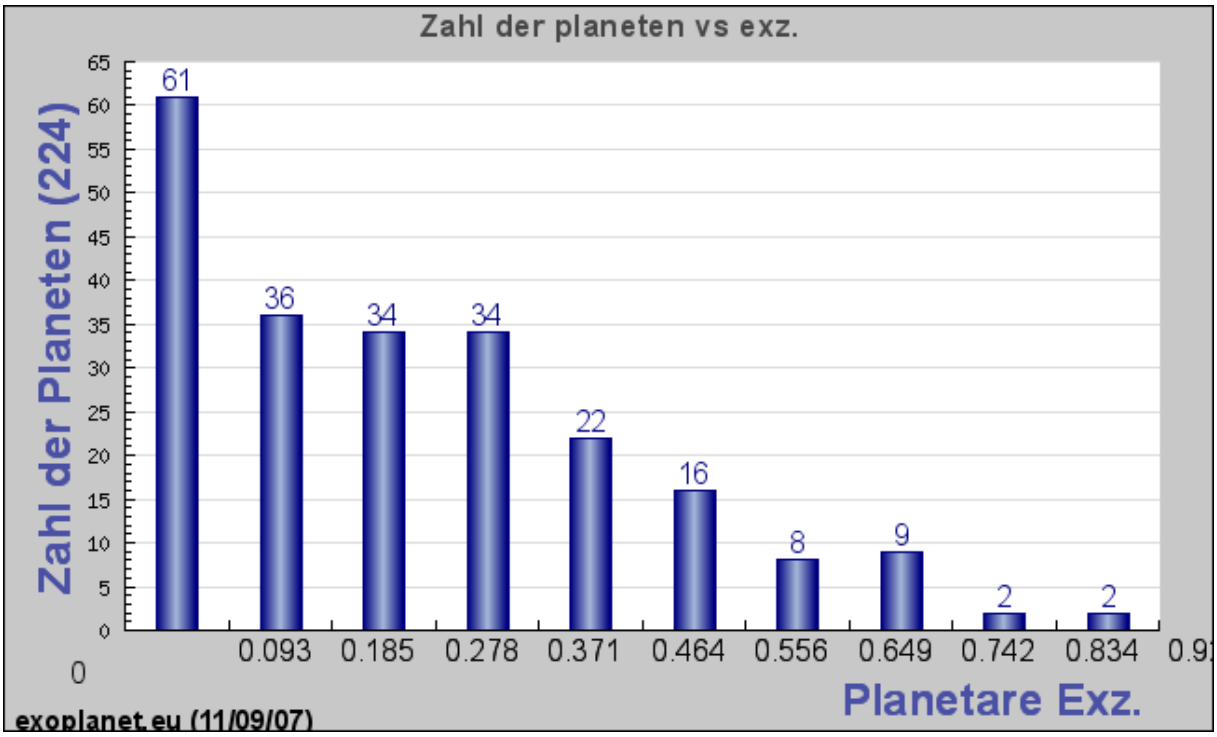

Fig. 6: number of known exoplanets ordered by eccentricity.

Increasing the limit up to a higher value isn't very practicable either, because most solar systems have planets with lower eccentricity and this would only lengthen the calculation time for these systems. So I have chosen to make the maximal eccentricity editable. Now the user is able to adjust this criterion for his cases. Other situations when you are allowed to exclude objects from the calculation are close-encounters. Within a certain distance from the planet a passing asteroid will either be captured or ejected. To investigate this case we have to ask ourselves the question, when the gravitational influence of a planet dominates the one of its host star. The Hill-Radius[v] has been defined for this situation:

$$(21) \quad R_{Hill} \approx a.(1-e)\sqrt[3]{\frac{m}{3M}}$$

In formula (21) the variable “a” is the semi-major axis of the smaller body (planet) and “e” its eccentricity. “M” stands for the mass of the larger body (star) and the variable “m” is the mass of the smaller body. Note that the formula for the Hill Radius can’t be derived rigorously. Despite it works fine as long as the mass of the third object is much less than the other two masses. For my calculations I’ve replaced the term a.(1-e) by the distance star-planet. The arguments to exclude object that are closer than a certain (in the NLI editable) fraction of the Hill-Radius to a planet are the same as for the eccentricity criterion, but here I would put a little more focus on the step length argument. In addition to these modifications I’ve also created the possibility to turn off or on several features of the integrator. For example you may disable that it puts out the heliocentric coordinates into a file, when you know that you only need the orbit elements, because writing to the hard disc takes some time. Furthermore I’ve made some minor improvements in the source code to speed the integrator up a little. In spite of all these modifications the largest part of the core source code of OLI is still the same in NLI.

5. PROGRAM ELEMENTS

In this chapter I will give an overview of the functions and the how-to-use of the program elements I've added to the integrator. Each of the following sub-chapters will treat a window of the program.

5.1 Main menu

The main menu is the part of the program which controls the integrator. It consists of five submenus, the large "calculate"-button and a menu bar.

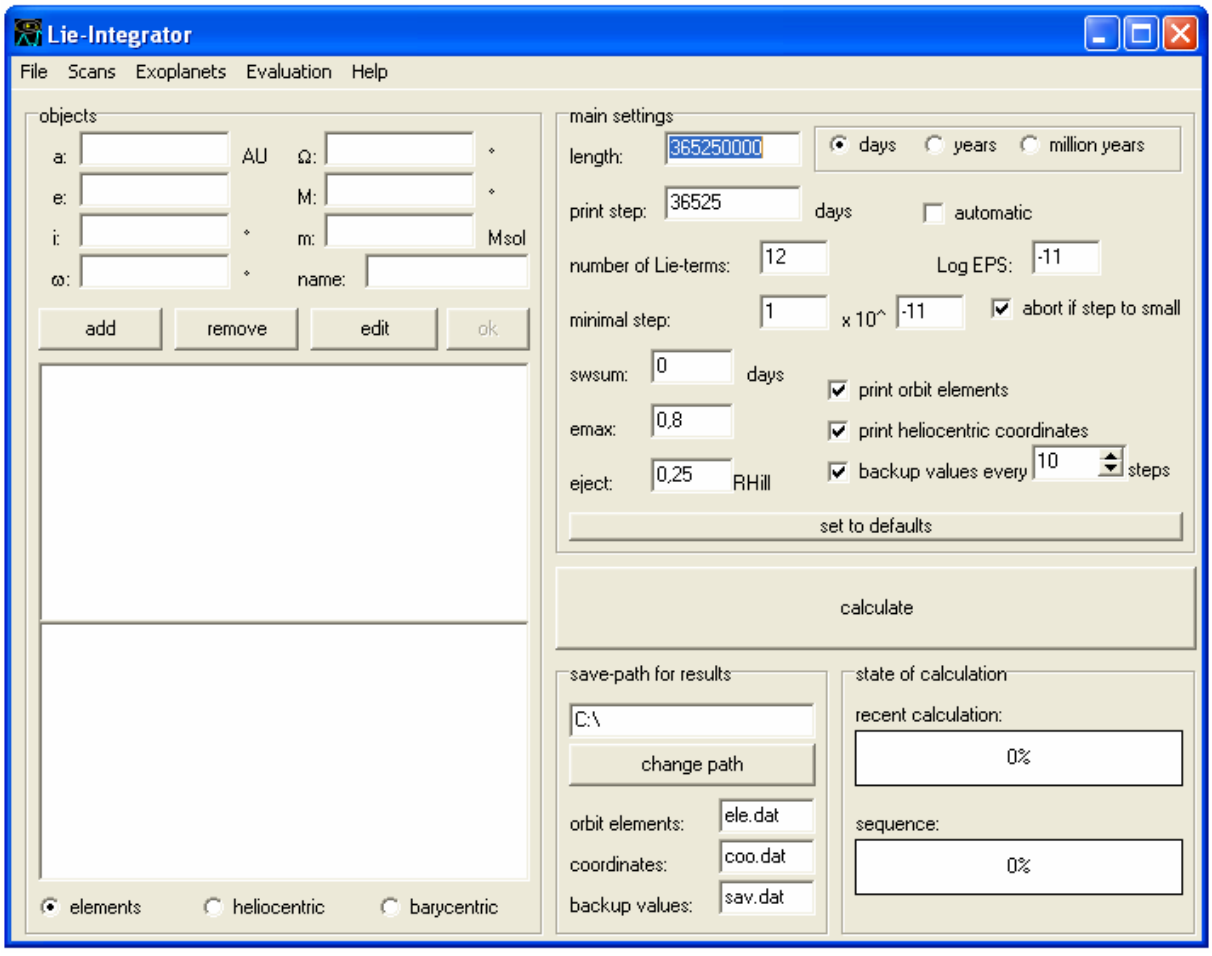

Fig. 7: the main menu of the Lie-Integrator

There are 8 "text edits" and 3 "radio buttons" in the submenu "objects". With the "radio buttons" you can select if you would like to enter orbit elements, heliocentric coordinates or

barycentric coordinates. The labels of the “text edits” will change automatically. I’m going to use following abbreviations in the program for the orbit elements (and masses):

abbreviation	**full name**
a	semi-major axis
e	eccentricity
i	inclination
ω	argument of perihelion
Ω	longitude of ascending node
M	mean anomaly
m	mass

Tab. 1: abbreviations of orbit elements

You can enter the orbit elements or the coordinates and the mass and name of the object you intend to add into the text edits. After this you only have to click the button “add” and your object will be displayed in one of the two “list boxes”. The upper “list box” is appointed to massive objects and in the other one only mass less objects will be displayed. Furthermore you can remove objects for these “list boxes” by clicking on them and using the button “remove” afterwards. In a similar way you can change your input with the button “edit”. After clicking “edit” the data of the selected object will be displayed in the “text edits” and you can confirm your changes by clicking the “ok”-button. Another submenu is called “main settings”. Into the first “text edit” you can enter the length of the calculation. On the right-hand side of it there are 3 “radio buttons” to select the unit of the calculation length. You can chose between days, years and million years. The next “text edit” is labelled with “print step”. The entered value sets the intervals in days in which the NLI makes a printout into a file. Beside this “text edit” you can find a “checkbox” named “automatic”. If you enable it by clicking the integrator will make a printout after every internal step. Note that this step length is variable. Into the next “text edit” you can enter the number of Lie-terms used for the calculation. The default setting of 12 has proved as most efficient for the OLI. On the right-hand side of this “text edit” there is another one to enter the so called “Log EPS”, which corresponds to accuracy. An average value for this is the default setting of -11. A setting for low accuracy would be -9 and one for a still reasonable high accuracy would be -13. The next 2 “text edits” are made to put in the minimal step length. Right beside it you can find a “checkbox” and with it, it’s possible to enable or disable an abort-function if the step length

gets to small. Another “text edit” is dedicated to the parameter “swsum”. The next “text edit” is labelled as “emax”. There you can enter the maximal allowed eccentricity for mass less bodies. If the eccentricity gets greater than this value, the object will be excluded from the calculation. Furthermore you can find another “text edit” below to set the Hill radius criterion. On the right hand-side of these “text edits” there are 3 “checkboxes” and one “spin edit”. With these controls you can enable and disable printout files for orbit elements, coordinates or backup. The “spin edit” gives you the opportunity to set the interval for backups. On the bottom of this submenu there is a button to reset all parameters in this submenu to default. Now we will focus on the third submenu, which is called “save path for results”. The first “text edit” in this submenu has been made to enter the path, where you intend to save your results. You can change it either by entering the path directly or by using the button below which opens a common Windows “save dialog”. In this submenu there are three more “text edits” where you can enter the filenames for orbit elements, coordinates and backups. The next submenu is labelled as “state of calculation”. You can find two “gauges” there. The first one is labelled as “recent calculation” and it indicates the progress of the recent calculation. The other gauge only shows overall progress if there is a queue of calculations (in the scan mode). The large button labelled with “calculate” is self-explaining. On the top of the main menu window there is a menu bar, where you can start all the other program features.

5.2 Help system

The second program window we are threatening is the help system. You can start the help in the program with the menu entry “Help” in the menu bar. The short manual is based on this part of the paper and explains how to use the program. You can start it with the common short cut “F1”. Beside this there are two more windows you can start from the help menu: the first one is called “Report bugs”. It gives you a short description how to report a bug you have found in the program to me, so that I can fix it. The other one is titled with “About” and only contains version number and copyright. Let’s focus on the short manual, which can be a real help if you know how to handle it.

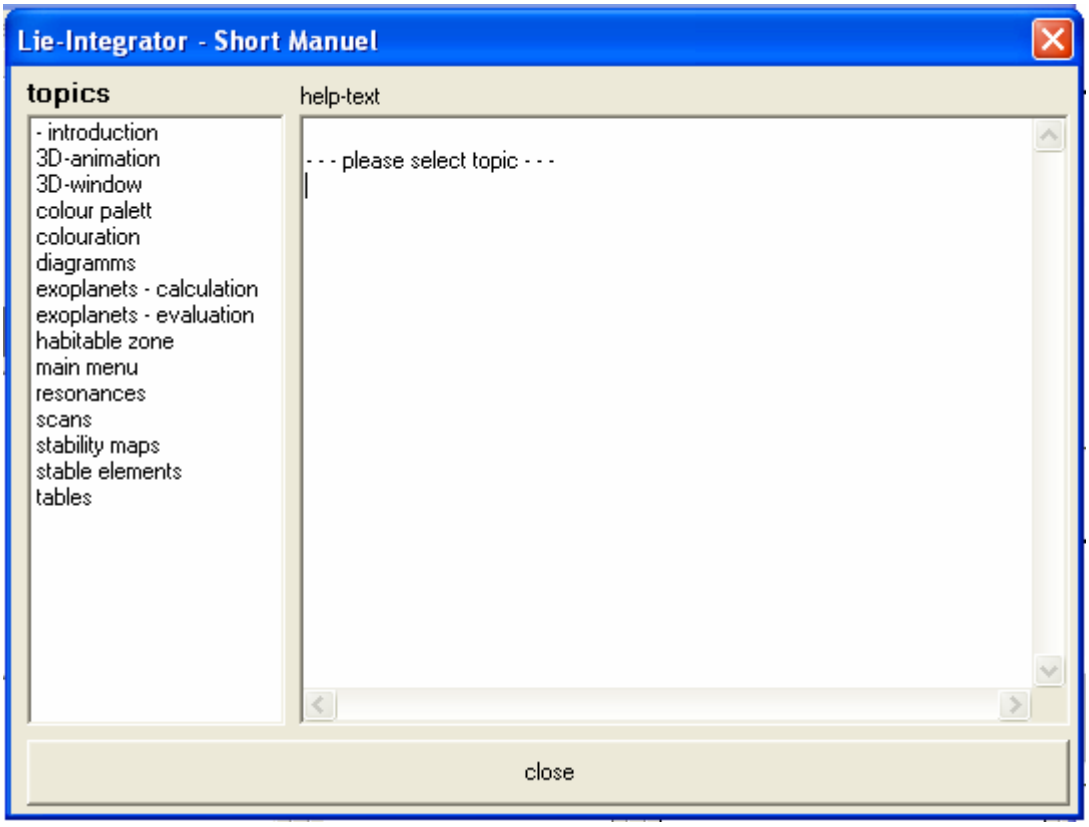

Fig. 8: the short manual for the Lie-Integrator

The short manual looks quite simple and it really is. There is one “list box” on the left, a “memo field” on the right and below these two objects I have placed a button called “close” to close this window. By clicking on a topic in the “list box” the corresponding help text will appear in the “memo field”. As already mentioned, these help texts are based on this chapter of this paper. This part of the program works in a very simple way. The titles of the topics in “list box” are as well the filename (without extension) of the corresponding text file with the help text. So it just loads the right file by using the title you’ve clicked on.

5.3 Scans

For many situations it’s very important to investigate several possibilities which are only slightly different. For these cases I’ve developed a scan procedure for the Lie-Integrator, which can be controlled by the scan window.

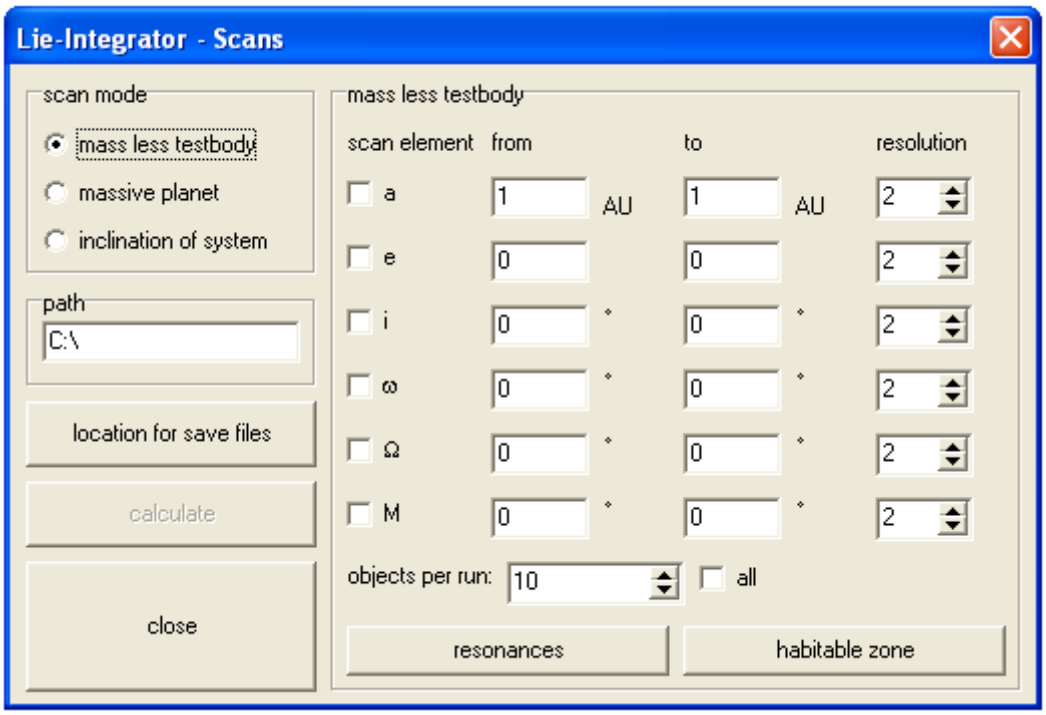

Fig. 9: the Scans window – scan mode: "mass less testbody"

The scan menu consists of 3 submenus and 3 buttons. In the first submenu which is labelled "scan mode" you are able to choose between 3 different kinds of scans: "mass less testbody", "massive planet" and "inclination of system". In the first mode there will be a mass less testbody added to system with variable initial conditions and in the second mode it's a massive planet or even star. The third mode is completely different from the other two. You can select the inclination of a system of exoplanets, so that the masses of the planets are multiplied with a factor depending on the angle of view. This feature treats a very up-to-date problem with systems of exoplanets discovered by radial velocity method. The next submenu is labelled "path" and contains a "text edit", where the save path of the result files is displayed. You can change it with the button below, which is labelled "location for save files". You find two more buttons below this one. The first one is labelled "calculate" and starts the calculation of the scan sequence while the other button is called "close". The last submenu changes its appearance due to the chosen scan mode.

Fig. 10: submenu from scans window – scan mode: "inclination of system"

In the scan mode “inclination of system” this submenu consist of only two “text edits” and one “spin edit” (see Fig.10). Into the first “text edit” you must enter the lowest angle you intend to scan and into the second “text edit” the highest angle for your scan. The “spin edit” has been made to select the resolution: how many runs shall be made? Note that only angles greater than 0° and less than or equal to 90° are reasonable. An angle of 90° means that you see the system edge on and the measured minimal masses are the real masses of the planets.

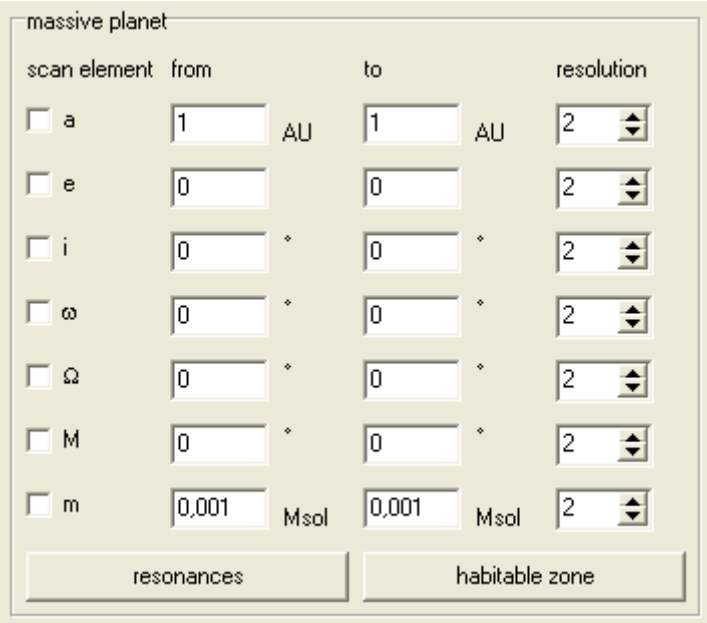

Fig. 11: submenu from scans window – scan mode: “massive planet”

If you select the mode „massive planet“, the submenu will change completely (Fig.11). Now it consists of 7 “checkboxes”, 14 “text edits”, 7 “spin edits” and 2 buttons. Each “checkbox” enables or disables a variable orbit element or variable mass for the additional planet. In the first “text edit” right of a “checkbox” you must enter the value you intend to start your scan with. In the next one you shall write the upper border of your scan area in this parameter. If the “checkbox” isn’t checked, the value in the first “text edit” will be the initial value of this orbit element for all runs. Finally on the right edge of this submenu there are the “spin edits”. With these controls you change the resolution (number of runs with different initial values for this orbit element) of the scan. On the bottom of this submenu you can find two buttons. The first one is called “resonances” and opens the resonances window, while the other, which is labelled “habitable zone”, displays the habitable zone window. The functions of these windows will be explained in the next two sub chapters. If you select the scan mode “mass less testbody”, you will see a very similar submenu (Fig.9). There is only one difference:

Instead of the possibility to select the mass of the object, which is of course in this case zero, you will find a “spin edit” that is entitled with “objects per run” and a “checkbox” with “all” written on it. Because mass less objects don’t influence each other by gravity you can calculate more than one object with different initial values per run. This “text edit” gives you the opportunity to choose how many you intend to calculate per run. The OLI has shown that the optimal number of objects in total per run is between 10 and 20. But you can also use the “checkbox” “all” and calculate all mass less testbodies in only one run.

Finally we are going to have a look on the orbit elements. They are all listed (with abbreviations) in Table 1, but what are they? The semi-major axis and the eccentricity are defining the shape of the orbit ellipse. You can calculate the eccentricity from the two semi axis of the ellipse:

$$(22) \quad e = \sqrt{1 - \frac{b^2}{a^2}}$$

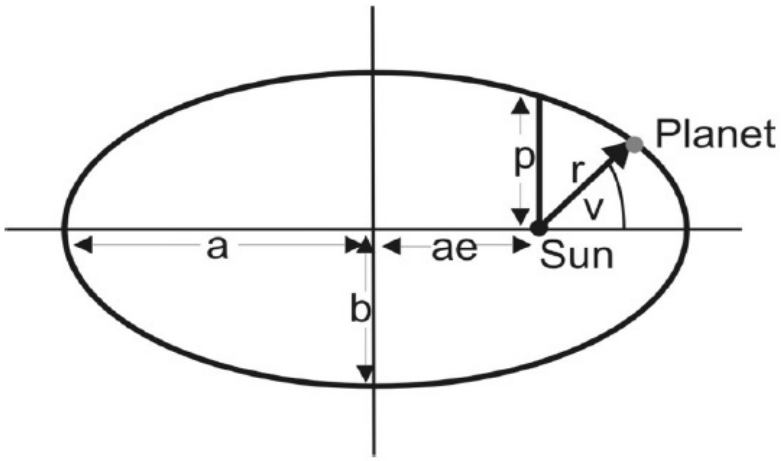

Fig. 12: an orbit ellipse viewed from above

The other 4 orbit elements are all angles and define the alignment of the ellipse in 3-dimensional space and the position of the planet on it. The inclination defines the gradient of the ellipse to a fix plane in space. The longitude of the ascending node refers to a fix point (direction) on this plane and the cutting point of the orbit plan. The argument of perihelion counts the angle from this cutting point to the point where the ellipse gets closest to one of its focal points. Now we have defined the alignment of the ellipse in space. Finally we have the

angle from the perihelion to the recent position of the planet. It is called the true anomaly (v) and can be transformed into the mean anomaly (M) with these formulas:

$$(23) \quad \tan(\frac{v}{2}) = \tan(\frac{E}{2}).\sqrt{\frac{1+e}{1-e}}$$

$$(24) \qquad E - e.\sin(E) = M(t_0) + n.(t - t_0)$$

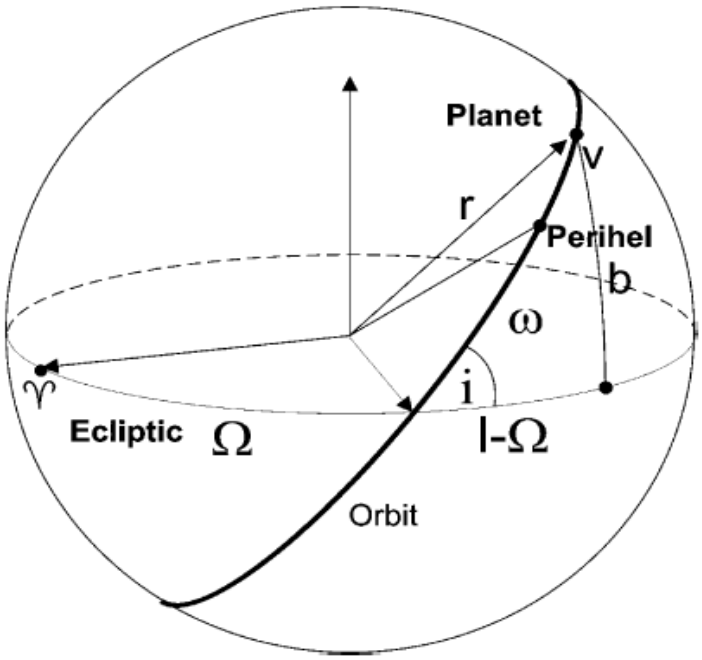

Fig. 13: an orbit ellipse in 3-dimensional space

5.4 Resonances

Investigating resonances is a very common work in Astrodynamics. Resonances are important in our own solar system (e.g. main asteroid belt and Kuiper belt) as well as in other multipanetary systems (in Classes Ia and Ib due to the classification of S. Ferraz-Mello[vi]). Because I intend to make the NLI as user-friendly as possible, I have created a feature which calculates resonances for scientific users so that they don't have to do side-calculations.

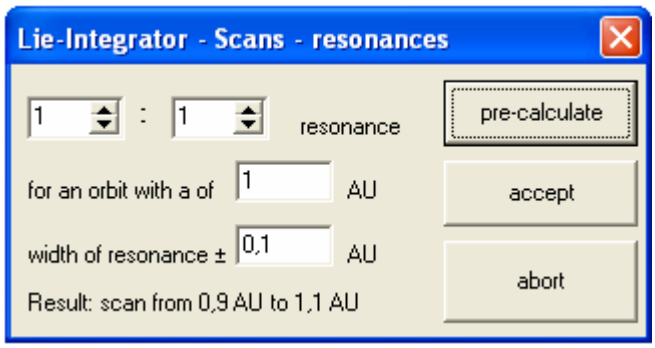

Fig. 14: the resonances window

The window only contains 2 “spin edits”, 2 “text edits”, 3 buttons and a couple of “labels”. With the 2 “spin edits” you are able to choose the resonance ratio. Furthermore you can enter the semi-major axis of the object you want to have a resonance with into the first “text edit”. The last “text edit” is designated to the width of resonance. In most cases it isn’t reasonable to investigate the exact resonance only. So you have to define a small area around it to see the structure of the resonance. The first button is titled “pre-calculate” and when clicking it calculates the resonance but only makes an output into the bottommost “label”. The button “accept” also calculates the resonance but closes the window and enters the result into the scan window. You can simply close this window with the last button.

5.5 Habitable Zone

The most interesting area around other stars is the habitable zone. Only within a certain distance from the host star a planet can support life (with a couple of other fitting parameters) comparable to ours. I’ve created this window with the same motivation as the window above.

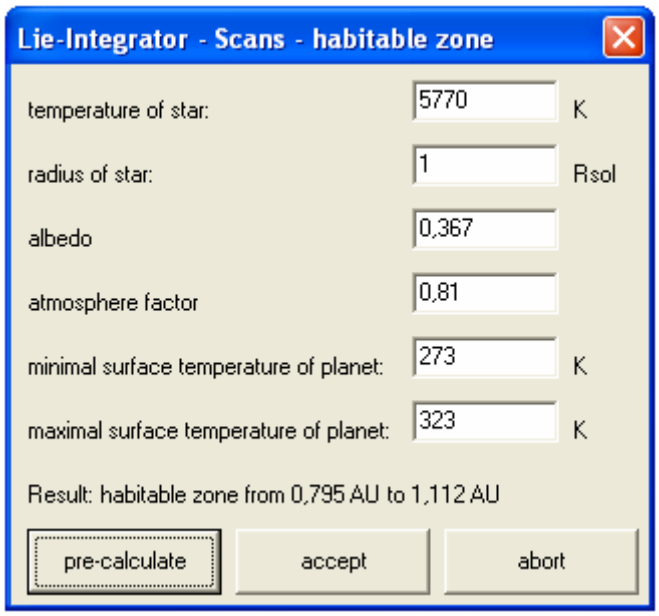

Fig. 15: the habitable zone window

The habitable zone window contains 6 "text edits", 3 buttons and a couple of "labels". The first "text edit" is reserved for the surface temperature of the host star in Kelvin. Into the next one you have to enter the radius of the star in solar radii. Below this "text edit" there is another one to enter the albedo of the planet. The fourth "text edit" is labelled "atmosphere factor" where you have to enter a constant (see formula (22)) which describes the heat storing of the planets atmosphere. Below this "text edit" you can find two more of them. Into the first one you have to enter the minimal average surface temperature of the planet and into the other one the maximal average surface temperature of the planet. All values in these "text edits" (except the bottommost two) are the standard values of Earth and Sun. The average surface temperature of is about 288K. The 3 buttons below fulfil the same functions as in the previous window. Let's have a closer look how the habitable zone is calculated. In this program I'm using a formula from J. Schneider[vii], which I've slightly altered:

$$(25) \quad a[AU] = \frac{1}{2}\frac{R_*}{R_\odot}\sqrt{1-A}\sqrt{F_{atm}+1}\left(\frac{T_*}{T_p}\right)$$

The formula I've found doesn't contain the term with F_{atm} (atmosphere factor) and leads for a non-zero albedo to completely wrong results for Earth. So I've added the atmosphere factor to this formula to get for a planetary temperature (T_p) of 288K a distance (a) of 1AU. An atmosphere factor of zero describes a planet without atmosphere but at a value of 0.81 it

simulates an Earth-like atmosphere. The other variables in this formula are radius of Sun ($R_\odot$), radius of the host star (R_*) and the surface temperature of the host star (T_*).

5.6 Tables

Up to now I've only described features of the program which helps you to calculate problems, but what are we going to do with the results? For this reason the NLI contains several different program elements to evaluate the results of previous calculations. The first and most simple of them is the table window. Here you can extracts the data you need from the result files and get it displayed in tables, which you can save for further external evaluation. In addition to that there is also a function, which can find extreme values in the results. Let's have a look on this window:

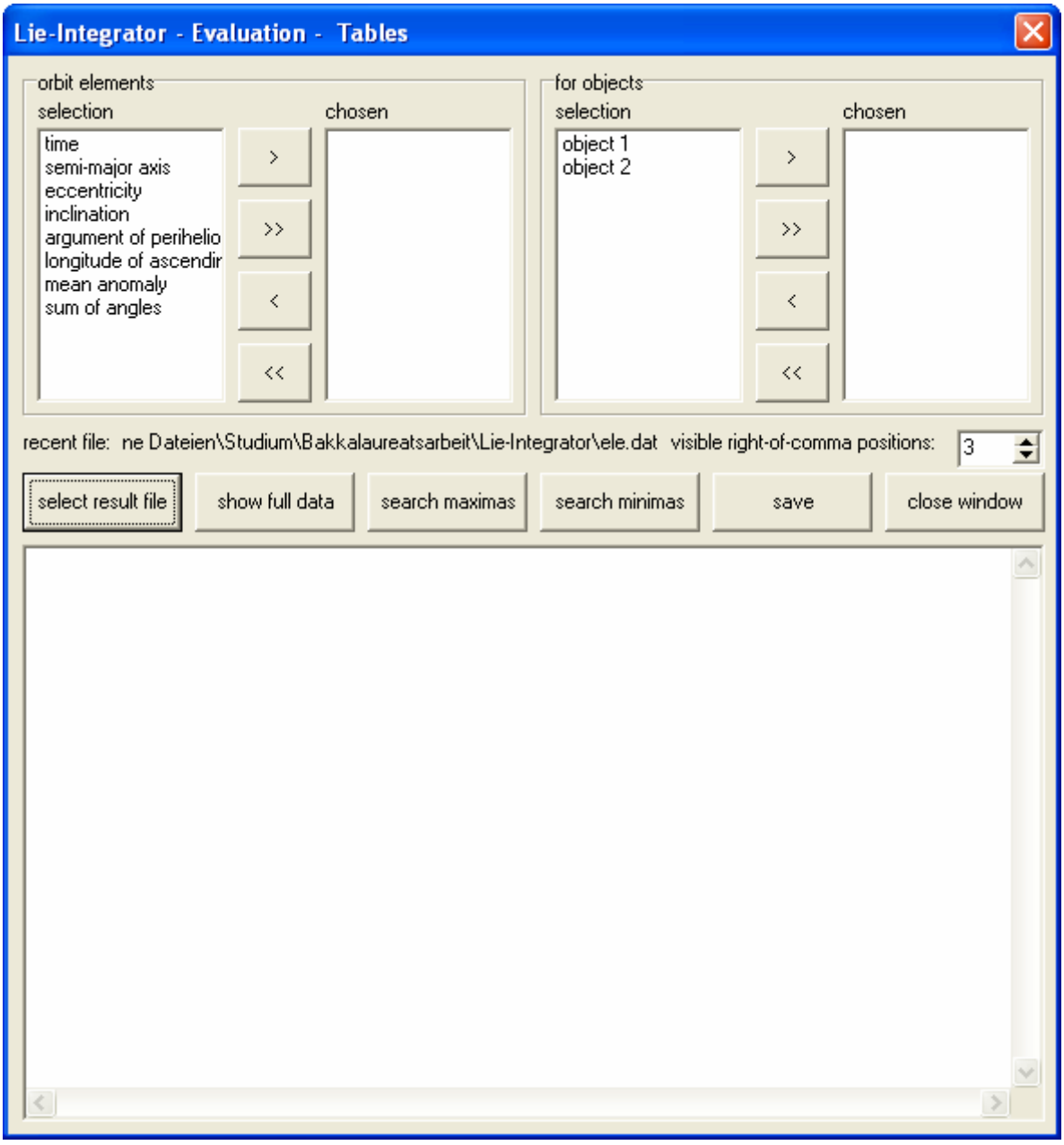

Fig. 16: the table window

You can see 2 submenus and each of them contains 2 “list box” and 4 buttons labelled with different arrows. Furthermore there is a long “label” and a “spin edit” below them. You can also find 6 buttons and a huge “memo field” on this window. The first submenu has been developed to select the orbit elements and parameters that you would like to display. By clicking on one you can move it into the “chosen” “list box” or back by using one of the 4 buttons. Their meaning is explained here in Table 2 and this convention is also used in other parts of the program:

symbol	meaning
>	move the selected element into the “list box” on the right-hand side
>>	move all elements into the “list box” on the right-hand side
<	move the selected element into the “list box” on the left-hand side
<<	move all elements into the “list box” on the left-hand side

Tab. 2: arrow symbols and their meaning

The other submenu works the same way. The only difference is that you have to choose the objects for which you intend to output their elements. The long “label” below the first submenu contains the path of the file you have selected with the first button that is labelled “select result file”. The “spin edit on the right edge of this window has been placed there to choose the number of right-of-comma-positions you intend to be shown in the table. By clicking the second button in the row the program will display a table due to the selected parameters above in the “memo field”. The next two buttons will put out a labelled list of extreme values of the orbit elements into the “memo field”. It is possible to save the content of the “memo field” to a text file with the button that is entitled with “save”. Finally the button “close window” does that what is written on it.

5.7 Diagrams

Results are usually display in diagrams and so I’ve decided to equip the program with a tool that can create line diagrams out of a result file. You can chose if you like to open a single “normal file” or a “scan file”. This tool creates a saveable diagram with labelled axis.

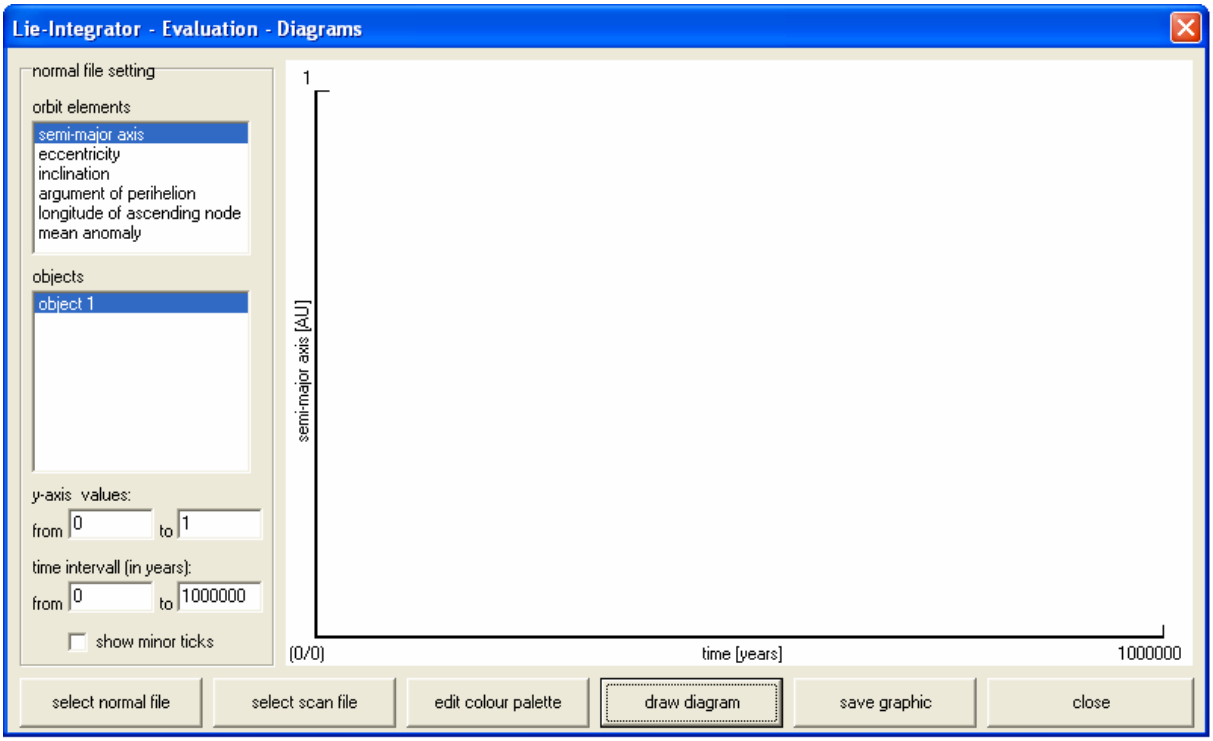

Fig. 17: the diagrams window- normal file mode

The diagram window consists of a submenu that changes its appearance due to the selected result file type. In addition to that you can find 6 buttons and a “paint box”. You can select a normal result file with the button “select normal file”, but you can also work with a scan file by clicking the button “select scan file”. The third button opens the colour palette window. You are able to draw a diagram by clicking the next button that is labelled “draw diagram”. The button “save graphic” saves the diagram in the “paint box” as a bitmap. To close this window you can use the button “close”. Let’s have a view on the submenu in case you’ve chosen a normal file. It consists of 2 “list boxes”, 4 “text edits” and a “checkbox” (Fig.17). The first “list box” contains all orbit elements and you select the one intend to display. In the other “list box” all objects are listed. You can select as many of them as you like by marking them by clicking. The first 2 “text boxes” are made to enter the borders of the visible y-axis. The displayed time interval can be selected by entering values into the other 2 “text boxes”. Finally the “checkbox” that is labelled “show minor ticks” enables additional tick marks in the diagram.

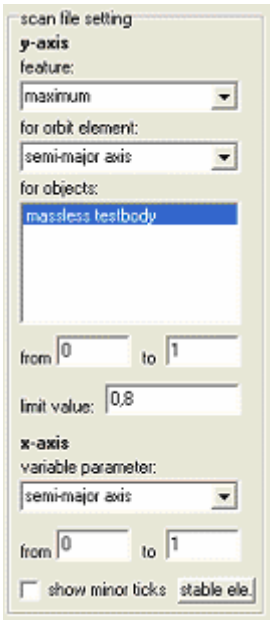

Fig. 18: the diagrams window- scan file mode

On the other hand you can select to evaluate a scan file. The submenu (see Fig.18) looks quite different and contains 3 “combo boxes”, 1 “list box”, 5 “text edits”, 1 “check box” and a button labelled “stable ele.”, which opens the window stable elements. You can select the parameters displayed on the y-axis with the first 2 “combo boxes” and the “list box”. The first “combo box” contains the so called “feature” of an orbit element. These features are listed in table 3 here:

maximum
minimum
time until maximum
time until minimum
time until greater than a limit
final value

Tab. 3: features of orbit elements that can be selected

Of course the same list of features is used other parts of the program. The next “combo box” gives you the opportunity to select the orbit element and with the “list box” you can chose the objects. You can define the borders of the visible area of the y-axis with the first 2 “text edits”. The third “text edit” is only needed in case you have selected the feature “time until greater than a limit”. In this case you can enter the limit value into it. For the x-axis you must

select a parameter that has been variable during the scan. This can be done with the third "combo box" and the border between the diagram line is displayed can be defined with the last 2 "text edits". Finally the "checkbox" remains to be explained, but it has the same function as in the case you've selected a normal file.

5.8 Stable elements

The program must know when it draws a diagram or stability map from a scan-file on what values the non-variable parameters are held. For this problem I've created the stable elements window, where you can select these values.

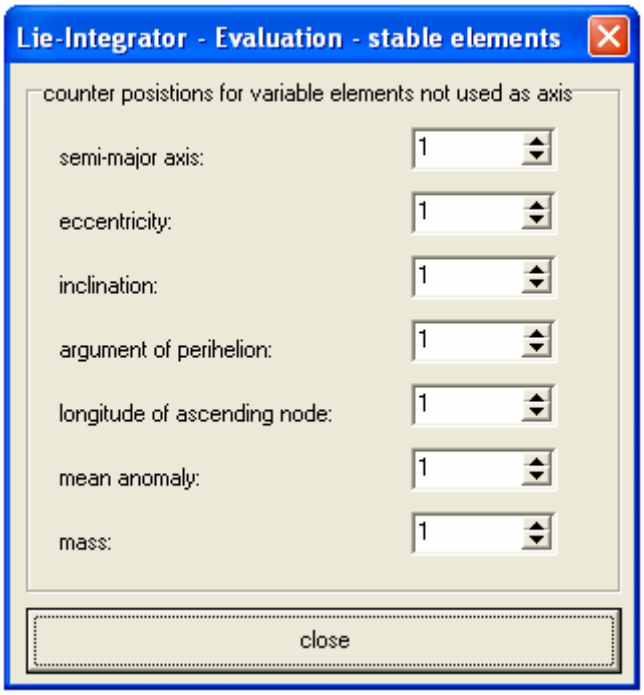

Fig. 19: the stable elements window

This window contains 7 labelled "spin edits" and a button to close this window again. With these "spin edits" you can set the counter position for the elements that have been variable during the scan, but are not displayed on an axis in the diagram. Reasonable values are from one up to the resolution of the orbit element. In case you've entered a not allowed value it will either take 1 or the highest reasonable value for this element.

5.9 Colour palette

It doesn't only look boring to have a monochromatic diagram; no it can even cause problems, because you will mix up the graphs if they cross over. So I've programmed a Colour palette to give the user the opportunity to select the colours of the individual curves. Furthermore I use this window in other parts of the program, where it's reasonable that the user can select colours. The colour palette is used to colour a stability map or the objects in the 3D-animation.

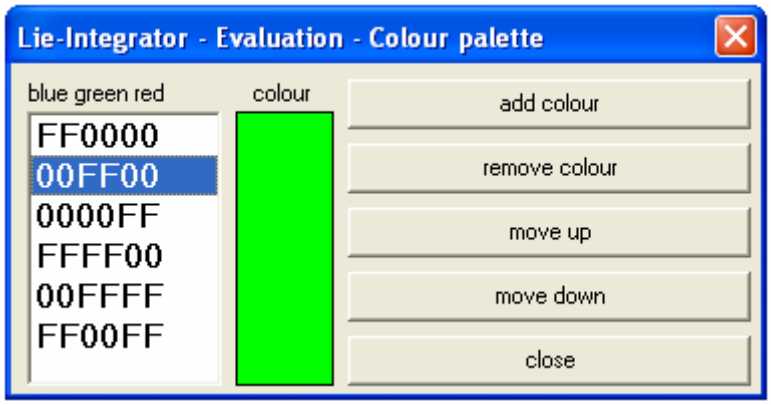

Fig. 20: the colour palette

This window consists of a "list box", 5 buttons and a "shape". The last object is usually used to display simple geometric figures, like in this case a rectangle. But it can also be coloured in any available 32bit colour and a "shape" can also react on an "on click-event". By clicking into the "list box" the "shape" will get the selected colour. When you click on the button "add colour" a simple colour dialog as known from programs like "Paint" will show up and you are able to choose a colour which will be added to the "list box". You can remove a colour from the "list box" by first clicking on the colour you want to get rid of and then clicking on the button which is labelled "remove colour". With the buttons "move up" and "move down" you can sort the "list box". The final button "close" closes the window so that you can continue with your work. Note that all colours are display in the "list box" in their hexadecimal RGB (red-green-blue) code (or better BGR as you can read in Fig.20).

5.10 Stability maps

A stability map is a very common way to display astrodynamical data of a solar system, because it contains as much information as a three dimensional diagram. On both axis you can display different variable orbit elements from a scan of a system while another parameter can be displayed at each point in a different colour.

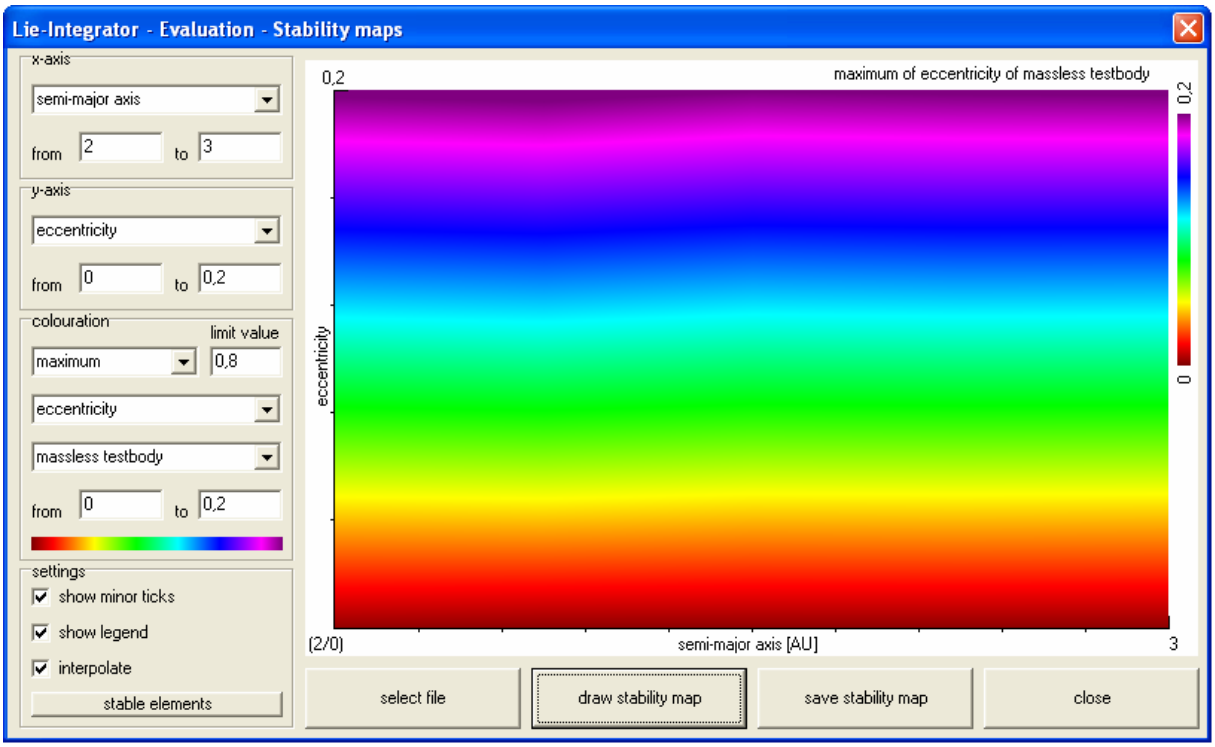

Fig. 21: the stability map window

This window contains of 4 submenus, a “paint box” and 4 buttons. The first of these buttons is labelled “select file” and with it you can of a result file from a scan. The next button draws a stability map with the parameter selected in the submenus to the “paint box”. You can save it to a bitmap with the third button that is labelled “save stability map”. The last button closes this window. Let’s have a look on the submenus. The first submenu is entitled with “x-axis” and contains a “combo box” and two “text edits”. With the “combo box” you can select a variable orbit element you want to display on the axis. The borders of the displayed area can be set with the two “text edits”. The next submenu is entitled “y-axis” and is completely equal to the previous. Colouration is the title of the submenu below these two and it consists of 3 “combo boxes”, 3 “text edits” and one image. The 3 “combo boxes” are similar to those in diagram window in the “scan file” mode. You can select a feature of an orbit element (also

selectable) of an object. The only difference is that you have a “combo box” instead of a “list box” like in the diagram window. The limit value for the feature “time until greater than a limit” can be entered into the uppermost “text edit”. The other “text edits” are placed there to enter the borders of colouration. Finally there is the image on the bottom of this submenu. It is coloured in the same way the stability map is going to be coloured. By clicking on it, the window colouration will open where you can edit the colouration. The last submenu is labelled “settings” and contains 3 “combo boxes” and a button that opens the stable elements window. The first “combo box” will show additional tick marks on the stability map if it’s enabled. To show a legend for the colouration you have to click the second “combo box”. The last “combo box” enables an interpolation mode for the stability map. In Fig.21 you can see the same stability map as in Fig.22, where the interpolation mode is disabled. The resolution is 5 times 5. The first one seems to be more elegant but mimics an illusory accuracy.

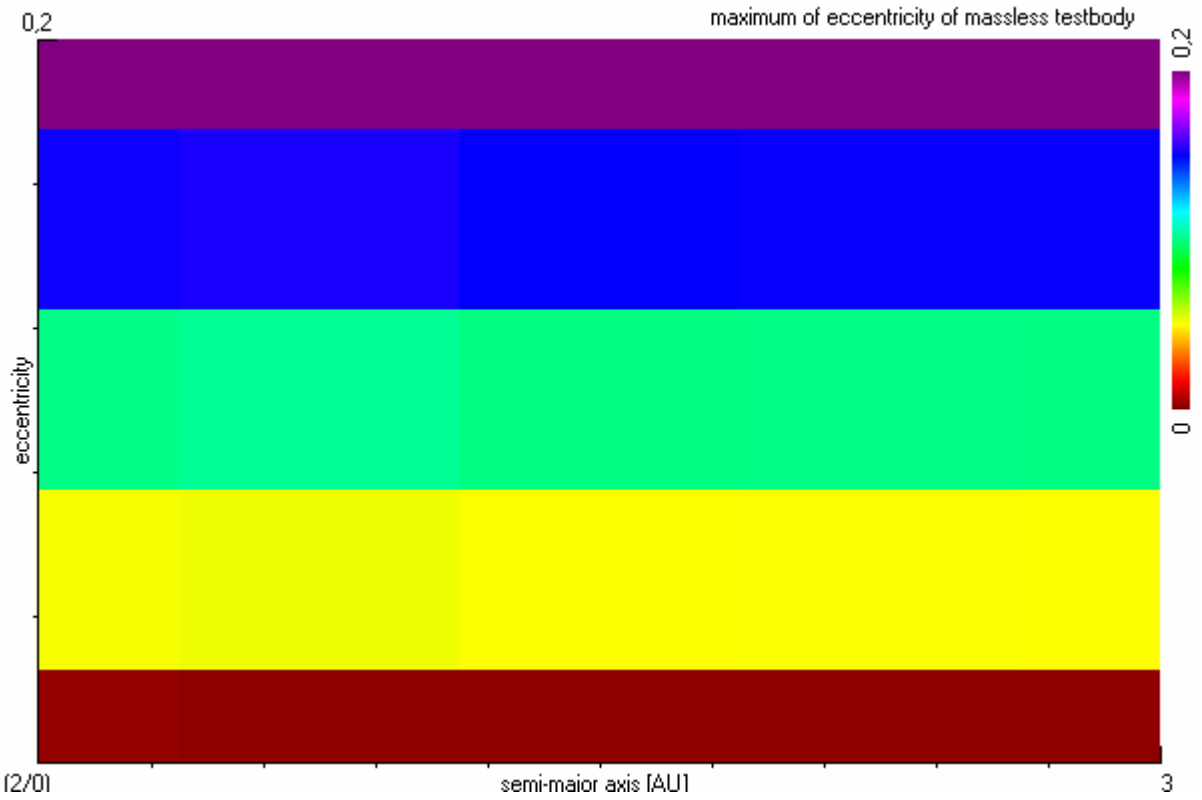

Fig. 22: a stability map from the NLI without interpolation

5.11 Colouration

I've developed a possibility to edit the colouration of a stability map. There are 4 different colouration modes, so that almost every user should be satisfied.

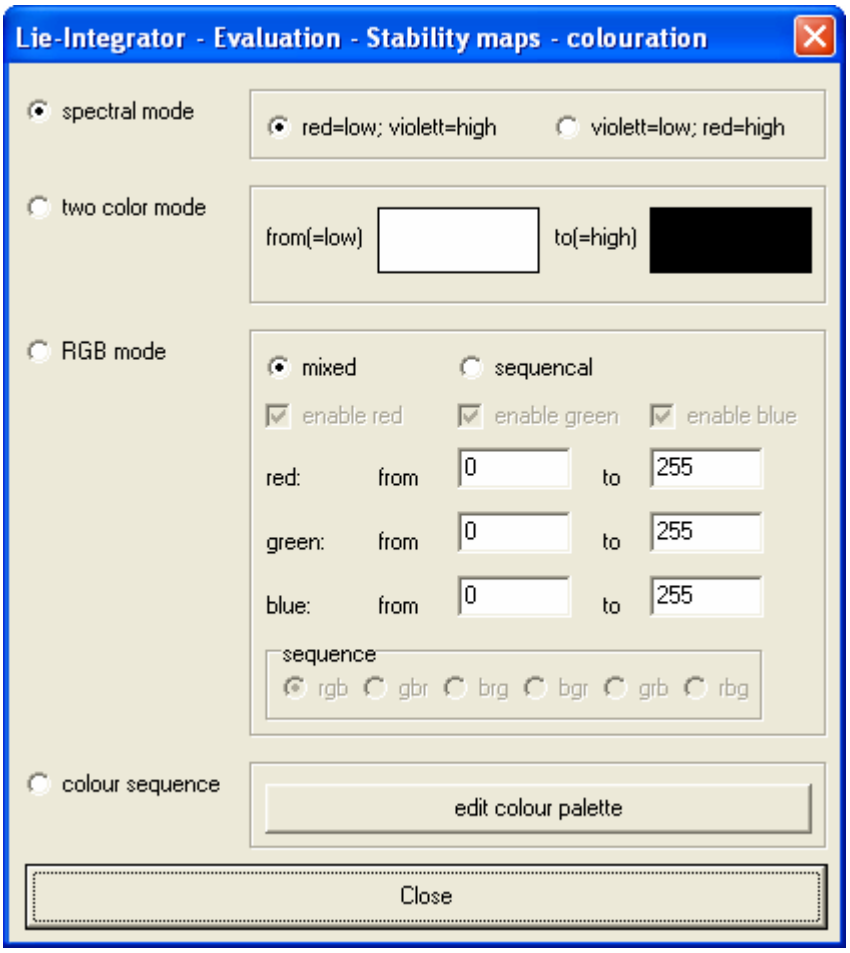

Fig. 23: the colouration window

There is a "radio button" and a submenu for each of the 4 colouration modes. In addition to that you can find a button to close this window on its bottom. By clicking on one of the "radio buttons" you can select the colouration mode. The first colouration mode is called "spectral mode" and you can chose between 2 sub-modes which are the same just mirrored. The title refers to the appearance of the mode, because it looks like a spectrum (see Fig.21). In the submenu for the second mode, that is called "two colour" you can find 2 "shapes". By clicking on them you can change their colour with a colour dialog. The colouration makes an RGB-interpolation between those two colours (see Fig.24).

Fig. 24: colouration mode – two colour

The next mode is called “RGB mode” and its submenu contains 2 “radio buttons”, 3 “check boxes”, 6 “text edits” and another submenu that consists of 6 “radio buttons”. With the first 2 “radio buttons” you can select the sub-mode: the first mode “mixed” is similar to the “two colour” mode, but the other mode draws a sequence of RGB colours with different intensity. You can enable and disable single colours (red, green or blue) with the 3 “check boxes”. With the “radio buttons” in the other submenu you can change the order of the sequence. In the 6 “text edits” you can enter the intervals for the main colours. You can see an example of this mode here (Fig.25):

Fig. 25: colouration mode – RGB mode – sequencal

The last colouration mode is quiet simple. It loads a colour sequence from the colour palette, we have already treated above. There is an example down here in Fig.26.

Fig. 26: colouration mode –colour sequence

5.12 3D-animations

The most impressing feature of the NLI is the ability to show a 3D-animation of the planets movement. But to get a proper animation you will have to do a small set-up which can be done in this window.

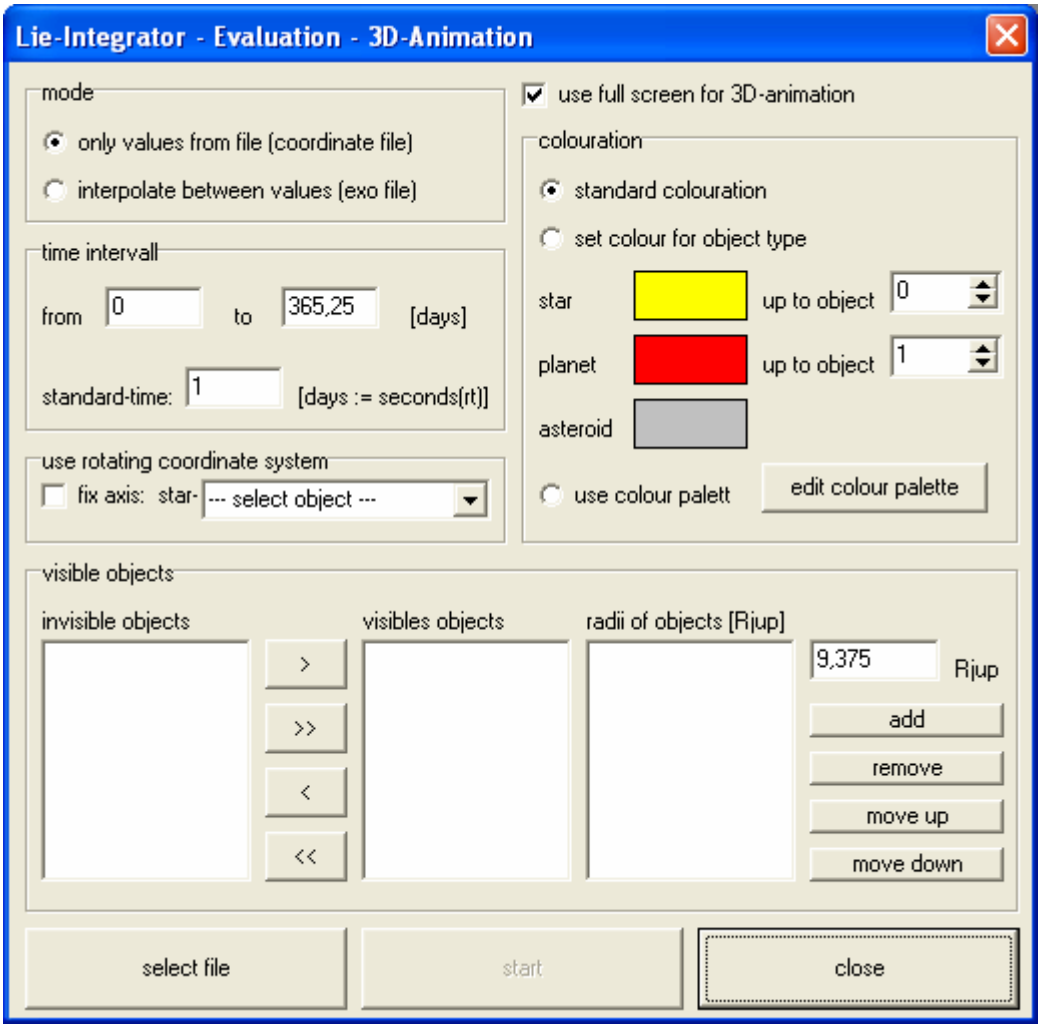

Fig. 27: the 3D-animation window

The 3D-animation window consists of 5 submenus, 3 buttons and one “check box”, that enables the users to decide if he wants to see the animation in full screen mode or not. With the first submenu that is labelled “mode” you can select the mode. It will be done automatically when you select a file with the first of the 3 buttons. The button on the right edge closes the window, while by clicking on the button in the middle the animation will start. There is another submenu that is labelled “time interval” and contains 3 “text edits”. Into the first 2 of them you can enter the time interval you want to get displayed in the animation. The other “text edit” has been placed there to enter the conversion factor for real time and simulation time. The submenu below is titled with “use rotating coordinate system” and consists of a “check box” and a “combo box”. With the “check box” you can enable this feature and with the “combo box” you can select the object which shall be stable in the rotating coordinate system. This feature is very useful to display horseshoe orbits, Trojans and orbits of exchange planets. The submenu in upper right part is called “colouration”. In it there

are 3 “radio buttons”, 3 “shapes”, 2 “spin edits” and a button to edit the colour palette. You can choose between the standard colouration, a colouration for object types and the colour palette by clicking on the “radio buttons”. The first mode uses a predefined list of colours that resembles the colours of the object in our system sorted by semi-major axis. The second mode paints objects up to a certain index with one colour and then up to another index with another colour and the rest with a third colour. You can set the border indexes with the two “spin edits”. The three colours can be change by clicking on the “shapes”. Finally the last mode uses the colours from the colour palette. The last submenu is labelled with “visible objects” and in it there are 3 “list boxes”, 8 buttons and one “text edit”. After you have selected a file a list of all objects of it will appear in the most left “list box”. Those objects you intend to see in the animation must be moved into the second “list box” with the 4 arrow buttons. The third “list box” contains the radii of the visible objects in Jupiter Radii. You can add a radius by entering a value into the “text edit” and clicking the “add”-button. With the next button you can remove an item from the radii-“list box”. Because the uppermost radius in the “list box” is assorted with the first object in the “list box” that is labelled with “visible objects” I’ve created two more buttons to change the order of the radii list. Note that the default value in the “text edit” is the radius of our Sun in Jupiter Radii.

5.13 3D-window

After you’ve clicked on the “start”-button in the 3D-animation window the 3D-window will appear. The 3D-animation here are mainly calculated on the computer’s graphic card and uses OpenGL, because it’s better implemented into the program language and there are many units (would be called classes in C) online. These facts make it comparable easy to develop a 3D-surface in Delphi. I’m using following units that are not part of the standard package of Delphi6: GL, GLu, GLext, TGA2, CgWindow, CgUtils, CgTypes, CgGeometry, CgLight, DotWindow, DotUtils, DotVideo, Glut. Despite so many different units I made the 3D animation quiet simple: all objects (stars, planets and asteroids) are spheres (or points). You can “fly” around by pressing some keys on the keyboard and I’ve only implemented one light source on the position of the host star. I’ve also developed a possibility to capture a video from the screen while watching the simulation, because else this feature would be just fun.

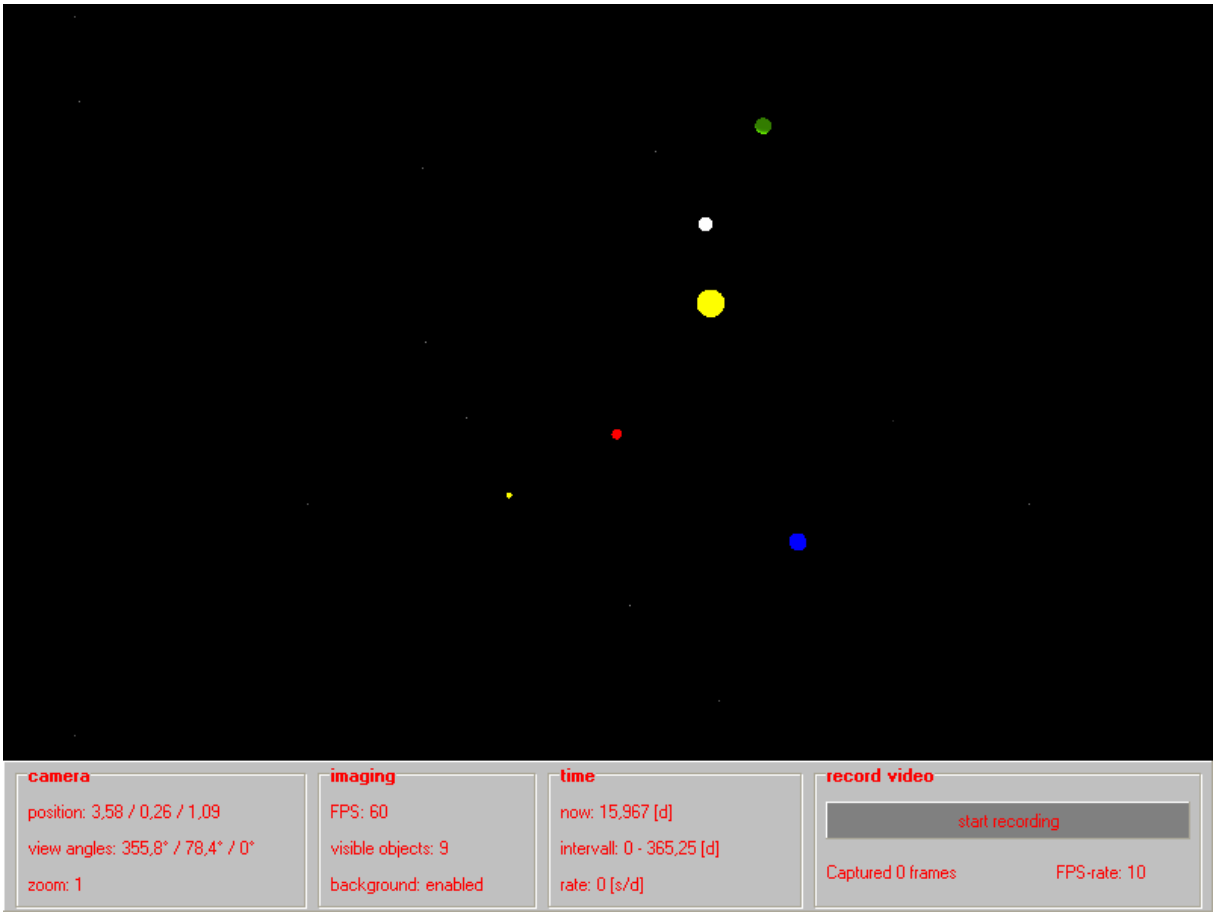

Fig. 28: the 3D-window

This window has a height of 600 pixels and a width of 800 pixels, but 100 pixels in height are lost for the control panel. So we have a resolution for the 3D-graphic of 800x500 pixels. Let's have a look on the control panel first: It contains 4 submenus. The first one is titled "camera" and contains 3 "labels". The coordinates (in AU and the origin is the host star) are written on the uppermost "label". Furthermore you can see the angle of view in the second "label" and the zoom factor in the last one. The second submenu is labelled "imaging" and contains 3 "labels" too. The first one gives you the recent frame rate in frames per second. The next "label" contains the number ob visible objects and the last one shows if the background stars are enabled. The next submenu, which also contains 3 "labels", is called "time". The uppermost "label" gives us the recent time in the simulation. The "label" below shows us the time interval for this simulation and on the last one you can read the recent conversion rate of real time and simulation time. The last submenu is different and it contains one button and two "labels". If you click on the button a menu will appear that asks you where to save the video. After this another menu will become visible and there you can make some settings for the .avi-video. Then the program will start recording at the current FPS-rate visible in the

right “label” and the number of frames that has been recorded yet are written into the other “label”. The process can be stopped by clicking on the button again. Although I’ve explained all visible features of this window one questions remains: “How can I fly around there?” This is controlled by the keyboard and the mouse. You can move at two different speeds in 6 directions and rotate around 3 angles. Furthermore you can control the time and recording of the video also by keyboard. The next table will give you a list of the shortcuts in this window:

key / mouse move	**action**
Click right mouse button and move up/down	moves view angle up and down
Click right mouse button and move left/right	moves view angle left and right
Q	rotates counter-clockwise
W	rotates clockwise
arrow up	moves camera forwards
arrow down	moves camera backwards
arrow left	moves camera left
arrow right	moves camera right
page up	moves camera upwards
page down	moves camera downwards
Shift	goes to higher speed while pressed
H	stops time
R	reverses time
F	increases time rate (faster)
S	decreases time rate (slower)
B	enables/disables background stars
I	zooms in
O	zooms out
M	starts/stops capturing a movie
X	increases FPS-rate for movie
Y	decreases FPS-rate for movie
Esc	closes window

Tab. 4: shortcuts for the 3D-window

5.14 Exoplanets - Calculations

As mentioned before a Keplerian fit to determine the orbit elements is not always valid. So I've decided a feature to check its validity by calculating an artificial radial velocity curve for exoplanetary systems. You just enter the parameters of the system you intend to investigate into the main menu and open this window.

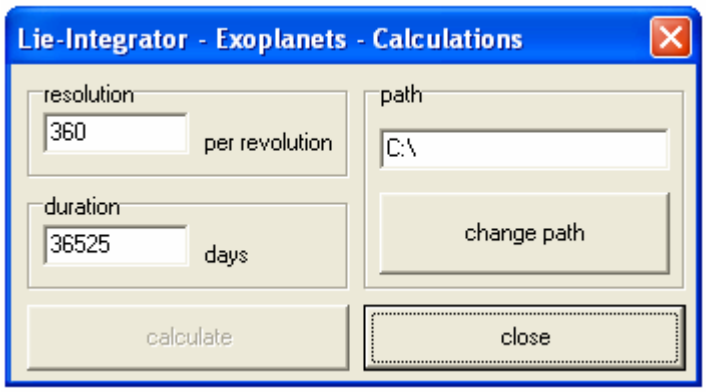

Fig. 29: the exoplanets – calculations window

This window is small and simple: into the first submenu you have to enter the resolution of your calculation. The program looks for the planet with the smallest semi-major axis and divides its orbit period by the value you have entered into this "text edit". The next submenu is designated for the duration of this calculation. I've decided to place this "text edit" here to avoid that the user makes the mistake to start this calculation at such a high resolution over a million years. This would simply take to much time. The last submenu contains a "text edit" and a button that is labelled "change path". Similar to the scan you have to select a path for this calculation which can be done by clicking on this button and the selected path will appear in the "text edit". Finally there are two more buttons on the bottommost edge of this window. The first one starts a calculation and the other one closes this window.

5.15 Exoplanets - Evaluation

Now we can evaluate the results of the previous calculation. For this I've developed another window, where you can investigate the exoplanetary system in two different ways.

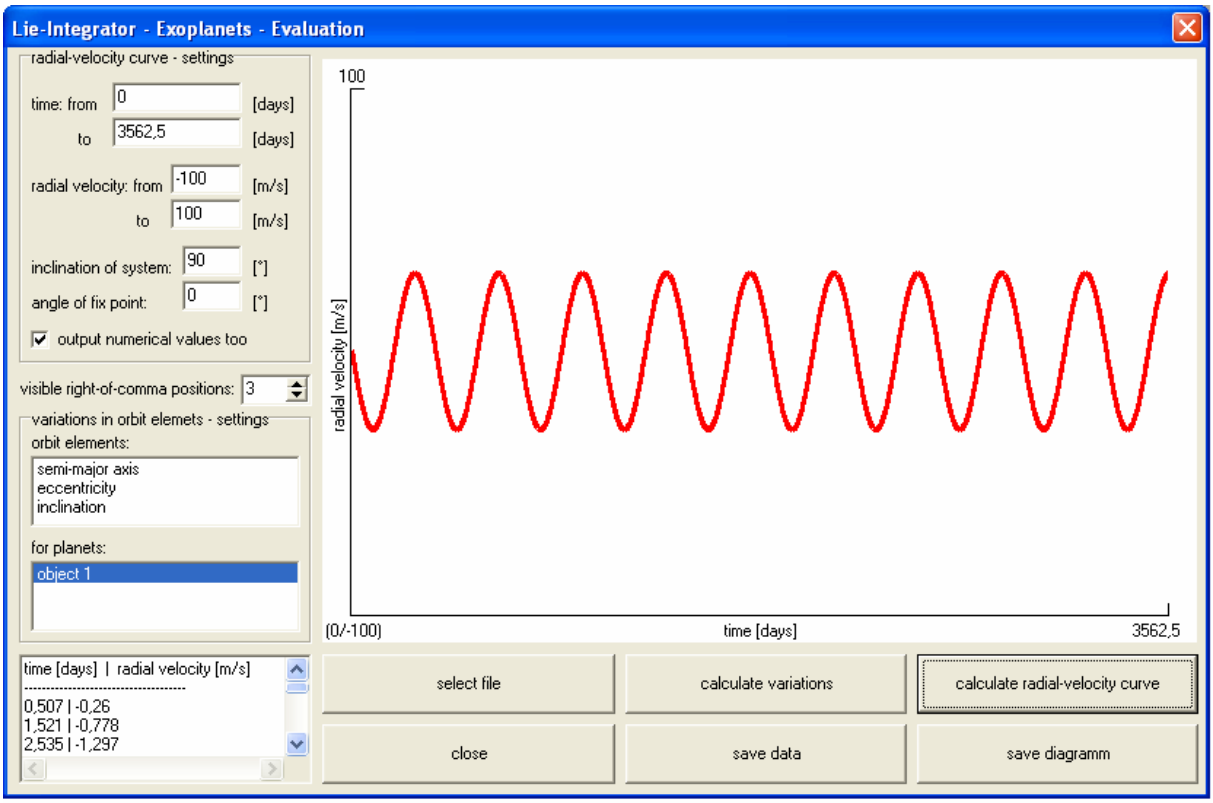

Fig. 30: the exoplanets – evaluation window

This window contains of 2 submenus, a "spin edit" to set the number of visible right-of-comma positions for the data that will be display in the "memo field" in the lower left corner, a "paint box" and 6 buttons. The first of these buttons has been placed there to open the result file of an exoplanet calculation. The other two buttons in the first row start the two evaluation procedures. In the second row the first button closes this window. The next one which is labelled "save data" saves the content of the "memo field" to a text file. The last button saves the diagram from the "paint box" to a bitmap by clicking. The first submenu is titled "radial-velocity curve – settings" and contains 6 "text edits" and a "check box". You can define the time interval that will be displayed with the first two "text edits". The next two "text edit" give the values for the borders of the y-axis (radial velocity). The inclination angle which influences the amplitude of the curve has to be entered into the "text edit". Into the last "text

edit" there can be the angle of the fix point entered. Finally if you click on the "check box" there will be an output of into the "memo field" as well as to the "paint box". The other submenu is titled "variation of orbit elements – settings". It consists of 2 "list boxes". The first one lists up the first three orbit elements, while the other contains a list of objects in this system. By starting the evaluation the program will search maxima and minima due to the selection and output the variation of the orbit elements into the "memo field".

5.16 Load, Save and Samples

In this sub-chapter I'll write about some features of the program, that haven't been mentioned yet. In the menu bar in the main menu there is also an items titled with "file". It contains 4 other items: New, Open, Save and Close. The first item deletes everything that has been entered into the main menu and resets to default values. The second item gives the user the opportunity to save the data from the main menu to a file, while the next one loads it from a file. In the order where you can find the .exe of the Lie-Integrator you can also find an order called samples. I have prepared some sample systems (e.g.: two body problem, restricted 3-body problem, our solar system …) there and you can load them by clicking on Open. The last item of course closes the program. There are also some shortcuts for the menu bar. I've decided to keep to the standards and use F1 for Help, Ctrl+N for New, Ctrl+S for Save, Ctrl+O for Open and finally Ctrl+F4 for Close.

6. CODE SAMPLES

6.1 The main procedure

This procedure is always called when the Lie-Integrator has to calculate the movements of planets and other objects. This can be in the main menu, in the scan window or in the exoplanet calculations window. Other important procedures are called in this procedure. Furthermore all parameters from the graphical surface are loaded into the program. In addition to that constants are initialized and coefficients are calculated. Moreover the printout is managed and the main lie series calculation procedure is called here too.

```
procedure TForm1.Lieberechnen;   // declaration of the procedure
var genauig:integer;   // declaration of local variables
begin   // begins procedure
genauig:=strtoint(edit14.Text);   // load accuracy from surface
assignfile(Elemente,pfad+edit19.text);   // opens file for orbit elements
rewrite(Elemente);   // sets file to write mode
assignfile(Koordinaten,pfad+edit20.text);   // opens file for coordinates
rewrite(Koordinaten);   // sets file to write mode
autoprt:=checkbox1.checked;   // gets a variable from surface
t:=0.0172020989500;   // sets constant: Gaussian gravitational constant
param;   // loads all parameters from surface and initializes arrays
sicher:=0;   // sets counter for backup file to zero
firstrun:=true;   // sets variable to initial value
steptosmall:=false;   // sets variable to initial value
swprt:=0;   // sets variable to initial value
swak:=0;   // sets variable to initial value
swakpr:=0;   // sets variable to initial value
konst;   // calls procedure konst to initialize constants
koeff;   // calls procedure koeff to calculate coefficients
if (ini=0)   // checks if input is given in orbit elements
then   // then
trnsko(2,NK);   // transforms orbit elements into heliocentric coordinates
```

```
if (ini<2)   // checks if input is given in orbit elements(that have already been transformed) or heliocentric coordinates
then   // then
heba;   // transforms into barycentric coordinates
mindis;   // calculates minimal distance between objects and checks Hill-criterion
print1;   // writes file header and initial values
repeat   // starts "repeat until" loop
lie_int;   // calls „Lie integration" procedure
mindis;   // calculates minimal distance between objects and checks Hill-criterion
swsum:=swsum+swak;   // adds current step length to total time count
swprt:=swprt+swak;   //  adds current step length to print step count
Gauge1.Progress:=round(100*swsum/stp);   // shows progress in %
if autoprt=true   // checks if automatic print step is selected
then   // then
prt:=swprt;   // stets print step to recent step length
if roundto(swprt,genauig)=roundto(prt,genauig)   // checks if print step is equal within a certain accuracy to recent print count
then   // then
begin   // begins bracket
print2;   // prints positions of planets into file
swprt:=0;   // stets print count back to zero
end;   // ends bracket
until (swsum>=stp) or (steptosmall=true);   // stops repeat loop if calculation is complete or step length too small
if swprt<>0   // checks if print count is unequal zero
then   // then
Print2;   // prints positions of planets into file
closefile(Elemente);   // closes file for orbit elements
closefile(Koordinaten);   // closes file for coordinates
end;   // ends procedure
```

6.2 Lie Integration procedure

This procedure is really the core of the Lie-Integrator. The lie series are calculated here.

```
Procedure Tform1.Lie_Int;   // declaration of the procedure
var maxddx,maxddt,AB:extended;   // declaration of local variables of floating
point type
norckw,nmak,k,l,i,i1,i2,j,j1,j2,ny:integer;   // declaration of local variables of
integer type
abbruch:boolean;   // declaration of local variables of boolean type
begin   // begins procedure
if firstrun=true   // checks if the procedure is running the first time
then   // then
begin   // begin bracket
phi:=0;   // sets variable to initial value
sig:=0;   // sets variable to initial value
x1:=0;   // sets variable to initial value
x2:=0;   // sets variable to initial value
x3:=0;   // sets variable to initial value
firstrun:=false;   // sets variable firstrun to false
end;   // end bracket
swmin:=999;   // sets variable to initial value
swmax:=0;   // sets variable to initial value
maxddx:=0;   // sets variable to initial value
maxddt:=0;   // sets variable to initial value
norckw:=0;   // sets variable to initial value
for k:=1 to nm do   // starts "for loop"
for l:=k+1 to nk do   // starts "for loop"
begin   // begins bracket
R2[l].spalte[k]:=-1/(RCUR[l].spalte[k]*RCUR[l].spalte[k]);   // calculates R2
array
DPHI[0].zeile[l].spalte[k]:=1/RCUR[l].spalte[k]*R2[l].spalte[k];   // calculates
DPHI array
ignore[k].spalte[l]:=ignore[l].spalte[k];   // makes ignore array symmetric
```

```
diffx1[0].zeile[l].spalte[k]:=DXB1[l].spalte[k];   // copies values into diffx1 array
diffx2[0].zeile[l].spalte[k]:=DXB2[l].spalte[k];   // copies values into diffx2 array
diffx3[0].zeile[l].spalte[k]:=DXB3[l].spalte[k];   // copies values into diffx3 array
end;   // ends bracket
for k:=1 to nk do   // starts "for loop"
begin   // begins bracket
DDX1[1].spalte[k]:=VB1[k];   // copies values into DDX1 array
DDX2[1].spalte[k]:=VB2[k];   // copies values into DDX2 array
DDX3[1].spalte[k]:=VB3[k];   // copies values into DDX3 array
end;   // ends bracket
for i:=0 to n2 do   // starts "for loop"
begin   // begins bracket
i1:=i+1;   // defines variable i1
i2:=i+2;   // defines variable i2
j1:=i1 div 2;   // defines variable j1
j2:=i1-j1;   // defines variable j2
for k:=1 to nm do   // starts "for loop"
begin   // begins bracket
nmak:=nm;   // copies value into variable nmak
if (k<=norckw)   // checks if variable norckw is greater than loop variable k
then   // then
nmak:=norckw;   // sets variable
abbruch:=false;   // sets variable
for l:=k+1 to nk do   // starts "for loop"
begin   // begins bracket
if (ignore[l].spalte[k]=true)   // checks if bodies are mass less
then   // then
abbruch:=true;   // sets variable
if abbruch=false   // checks if bodies aren't mass less
then   // then
begin   // begins bracket
if (i<>n2)   // checks loops variable i
then   // then
begin   // begins bracket
```

```
diffx1[i1].zeile[l].spalte[k]:=ddx1[i1].spalte[l]-ddx1[i1].spalte[k];   // calculates
difference
diffx2[i1].zeile[l].spalte[k]:=ddx2[i1].spalte[l]-ddx2[i1].spalte[k];   // calculates
difference
diffx3[i1].zeile[l].spalte[k]:=ddx3[i1].spalte[l]-ddx3[i1].spalte[k];   // calculates
difference
for j:=0 to j1 do   // starts "for loop"
begin   // begins bracket
sig:=sig+KO2[j].spalte[i]*(diffx1[j1-j].zeile[l].spalte[k]*
diffx1[j2+j].zeile[l].spalte[k]+diffx2[j1-j].zeile[l].spalte[k]*
diffx2[j2+j].zeile[l].spalte[k]+diffx3[j1-j].zeile[l].spalte[k]*
diffx3[j2+j].zeile[l].spalte[k]);   // calculates sig
end;   // ends bracket
Dsig[i].zeile[l].spalte[k]:=sig;   // gives value to Dsig array
for j:=0 to i do   // starts "for loop"
phi:=phi+KO1[j].spalte[i]*Dphi[i-j].zeile[l].spalte[k]*
dsig[j].zeile[l].spalte[k];   // calculates phi
dphi[i1].zeile[l].spalte[k]:=R2[l].spalte[k]*phi;   // calculates dphi array
end;   // ends bracket
for j:=0 to i do   // starts "for loop"
begin   // begins bracket
AB:=KO[j].spalte[i]*dphi[j].zeile[l].spalte[k];   // calculates AB
x1:=x1-AB*diffx1[i-j].zeile[l].spalte[k];   // calculates x1
x2:=x2-AB*diffx2[i-j].zeile[l].spalte[k];   // calculates x2
x3:=x3-AB*diffx3[i-j].zeile[l].spalte[k];   // calculates x3
end;   // ends bracket
DX1[k].spalte[l]:=-x1;   // copies value to DX1 array
DX2[k].spalte[l]:=-x2;   // copies value to DX2 array
DX3[k].spalte[l]:=-x3;   // copies value to DX3 array
DX1[l].spalte[k]:=x1;   // copies value to DX1 array
DX2[l].spalte[k]:=x2;   // copies value to DX2 array
DX3[l].spalte[k]:=x3;   // copies value to DX3 array
sig:=0;   // resets variable sig
phi:=0;   // resets variable phi
x1:=0;   // resets variable x1
```

```
x2:=0;   // resets variable x2
x3:=0;   // resets variable x3
end;   // ends bracket
end;   // ends bracket
abbruch:=false;   // sets variable
for l:=1 to nmak do   // starts "for loop"
begin   // begins bracket
if (ignore[l].spalte[k]=true)   // checks if objects are mass less
then   // then
abbruch:=true;   // sets variable
if abbruch=false   // checks if bodies aren't mass less
then   // then
begin   // begins bracket
x1:=x1+m[l]*dx1[l].spalte[k];   // calculates x1
x2:=x2+m[l]*dx2[l].spalte[k];   // calculates x2
x3:=x3+m[l]*dx3[l].spalte[k];   // calculates x3
end;   // ends bracket
end;   // ends bracket
DDX1[i2].spalte[k]:=x1;   // copies value to DDX1 array
DDX2[i2].spalte[k]:=x2;   // copies value to DDX2 array
DDX3[i2].spalte[k]:=x3;   // copies value to DDX3 array
x1:=0;   // resets variable x1
x2:=0;   // resets variable x2
x3:=0;   // resets variable x3
end;   // ends bracket
for k:=nm+1 to nk do   // starts "for loop"
begin   // begins bracket
for l:=1 to nm do   // starts "for loop"
begin   // begins bracket
x1:=x1+m[l]*dx1[l].spalte[k];   // calculates x1
x2:=x2+m[l]*dx2[l].spalte[k];   // calculates x2
x3:=x3+m[l]*dx3[l].spalte[k];   // calculates x3
end;   // ends bracket
DDX1[i2].spalte[k]:=x1;   // copies value to DDX1 array
DDX2[i2].spalte[k]:=x2;   // copies value to DDX2 array
```

```
DDX3[i2].spalte[k]:=x3;   // copies value to DDX3 array
x1:=0;   // resets variable x1
x2:=0;   // resets variable x2
x3:=0;   // resets variable x3
end;   // ends bracket
for k:=1 to nk do   // starts "for loop"
begin   // begins bracket
maxddx:=max(maxddx,max(abs(ddx1[i2].spalte[k]),
max(abs(ddx2[i2].spalte[k]),abs(ddx3[i2].spalte[k]))));   // finds the largest
DDX1 and copies it into maxddx
if ((maxddx-maxddt)>=0)   // checks if maxddx is greater than or equal to
maxddt
then   // then
NY:=I1;   // sets variable
maxddt:=maxddx;   // sets variable
end;   // ends bracket
end;   // ends bracket
SW:=Power((EPS*qfac[NY]/Maxddx),(1/NY));   // calculates step length
SWAK:=SW/T;   // converts step length into the correct units
if (SWAK<SWMINI)   // checks if step length is too small
then   // then
begin   // begins bracket
SWAK:=SWMINI;   // sets step length to minimal step length
if checkbox5.checked=true   // checks if program shall abort at a too small step
length
then   // then
begin   // begins bracket
steptosmall:=true;   // sets abort condition to true
MessageDlg('Step small than minimal step! - Program aborted.', mtInformation,
[mbOk], 0);   // output info message
end;   // ends bracket
end;   // ends bracket
SWMIN:=min(SWMIN,SWAK);   // redefine SWMIN
SWMAX:=max(SWMAX,SWAK);   // redefine SWMAX
if (swsum+swak>stp)   // checks if calculation have reached the end
```

```
then   // then
begin   // begins bracket
swakpr:=swak;   // secures value of swak
swak:=stp-swsum;   // redefines swak
sw:=t*swak;   // redefines sw
end;   // ends bracket
if (swprt+swak>prt)   // checks if there will be a printout
then   // then
begin   // begins bracket
swakpr:=swak;   // secures value of swak
swak:=prt-swprt;   // redefines swak
sw:=t*swak;   // redefines sw
end;   // ends bracket
fac1;   // calls procedure fac1
for k:=1 to nk do   // starts "for loop"
for j:=1 to n-1 do   // starts "for loop"
begin   // begins bracket
XB1[k]:=XB1[k]+TT[j]*DDX1[j].spalte[k];   // calculate new barycentric
coordinates
XB2[k]:=XB2[k]+TT[j]*DDX2[j].spalte[k];   // calculate new barycentric
coordinates
XB3[k]:=XB3[k]+TT[j]*DDX3[j].spalte[k];   // calculate new barycentric
coordinates
VB1[k]:=VB1[k]+TT[j]*DDX1[j+1].spalte[k];   // calculate new barycentric
velocities
VB2[k]:=VB2[k]+TT[j]*DDX2[j+1].spalte[k];   // calculate new barycentric
velocities
VB3[k]:=VB3[k]+TT[j]*DDX3[j+1].spalte[k];   // calculate new barycentric
velocities
end;   // ends bracket
end;   // ends procedure
```

6.3 Coefficient calculation procedure

The coefficients of the Lie series are calculated in this procedure. It will be only called once at the beginning of the calculation. This procedure is an essential part of the Lie-Integrator.

```
procedure Tform1.Koeff;   // declaration of the procedure
var i,j,i1,j1:integer;   // declaration of local variables
begin   // begins procedure
for i:=0 to n-2 do   // starts "for loop"
begin   // begins bracket
KO[0].spalte[i]:=1;   // sets KO array to initial value
KO[i].spalte[i]:=1;   // sets KO array to initial value
end;   // ends brackets
for i:=2 to n-2 do   // starts "for loop"
for j:=1 to i-1 do   // starts "for loop"
KO[j].spalte[i]:=KO[j-1].spalte[i-1]+KO[j].spalte[i-1];   // calculates KO array
for i:=0 to n-3 do   // starts "for loop"
KO1[i].spalte[i]:=3;   // sets values to KO1 array
for i:=0 to n-4 do   // starts "for loop"
KO1[0].spalte[i+1]:=KO1[0].spalte[i]+2;   // copies values into KO1 array
for i:=2 to n-3 do   // starts "for loop"
for j:=1 to i-1 do   // starts "for loop"
KO1[j].spalte[i]:=KO1[j-1].spalte[i-1]+KO1[j].spalte[i-1];   // calculates KO1
array
i1:=0;   // sets i1 to initial value
j1:=-1;   // sets j1 to initial value
i:=0;   // sets i to initial value
while i<=n-3 do   // starts "while loop"
begin   // begins bracket
i1:=i1+1;   // increments i1
j1:=j1+2;   // increases J1 by 2
for j:=0 to i1-1 do   // starts "for loop"
CO2[i].spalte[j]:=KO[i1+j].spalte[j1];   // copies value into CO2 array
i:=i+2;   // increases i by 2
```

```
end;   // ends brackets
j1:=-1;   // sets j1 to initial value
i:=1;   // sets i to initial value
while i<=n-3 do   // starts "while loop"
begin   // begins bracket
j1:=j1+1;   // starts "for loop"
CO2[i].spalte[0]:=KO[j1+1].spalte[i];   // copies values into CO2 array
CO2[i].spalte[j1+1]:=KO[i].spalte[i];   // copies values into CO2 array
for j:=1 to j1 do   // starts "for loop"
CO2[i].spalte[j]:=KO[j1+1+j].spalte[i+1];   // copies values into CO2 array
i:=i+2;   // increases i by 2
end;   // ends bracket
for j:=0 to n-3 do   // starts "for loop"
for i:=0 to ((n-2)div 2) do   // starts "for loop"
KO2[i].spalte[j]:=CO2[j].spalte[i];   // copies values into KO2 array
end;   // ends procedure
```

6.4 OpenGL Paint procedure

One also very interesting procedure is the one where the program draws the 3D-graphic on the window using OpenGL. There the planets are painted and the background stars. Furthermore the light is set and the calculation procedure to get the positions of all objects is called here too.

```
procedure TForm17.FormPaint(Sender: TObject);   // declaration of the
procedure
const    att: array [0..2] of Single = (0.25, 0, 1/60);   // declaration of local
constants
var i:integer;   // declaration of local variables of integer type
Q : PGLUQuadric;   // declaration of local variables of PGLUQuadric type
begin   // begins procedures
if lauf=true   // checks if simulations shall run
then   // then
begin   // begins bracket
```

```
rechneposi;   // calculates new positions of objects
glClear(GL_COLOR_BUFFER_BIT);   // clears the screen
glMatrixMode(GL_MODELVIEW);   // sets matrix mode
glLoadIdentity;   // loads some GL setup
glRotatef(-cam.pitch, 1, 0, 0);   // rotates camera
glRotatef(-cam.yaw, 0, 1, 0);   // rotates camera
glRotatef(-cam.rot, 0, 0, 1);   // rotates camera
glTranslatef(-cam.pos.x, -cam.pos.y, -cam.pos.z);   // moves camera
glLightfv(GL_LIGHT0, GL_POSITION, @LPOS);   // puts light to origin (star)
glLightfv(GL_LIGHT1, GL_POSITION, @LPOS);   // puts light to origin (star)
glLightfv(GL_LIGHT2, GL_POSITION, @LPOS);   // puts light to origin (star)
glLightfv(GL_LIGHT3, GL_POSITION, @LPOS);   // puts light to origin (star)
glLightfv(GL_LIGHT4, GL_POSITION, @LPOS);   // puts light to origin (star)
glLightfv(GL_LIGHT5, GL_POSITION, @LPOS);   // puts light to origin (star)
glenable(GL_CULL_FACE);   // enables a feature to save drawing time
glCullFace(GL_BACK);   // doesn't draw backside of objects
Q := gluNewQuadric;   // declaration of variable Q
gluQuadricDrawStyle(Q, GLU_FILL);   // sets draw style to solid
glPushMatrix();   // prepares painting of objects
glTranslatef(0, 0, 0);   // goes to origin
glPointsize(1);   // sets points size
glPointParameterfvEXT(GL_DISTANCE_ATTENUATION_EXT, @att);   // sets some
point parameters
glPointParameterfEXT(GL_POINT_FADE_THRESHOLD_SIZE_EXT, 1);   // sets
some point parameters
if bgstars=true   // checks if background stars shall be shown
then   // then
begin    //begins bracket
glColor3f(1, 1, 1);   // sets colour of stars to white
glBegin(GL_POINTS);   // begins to draw points
for i:=0 to 255 do   // starts "for loop"
glVertex3f(starx[i], stary[i], starz[i]);   // draws background stars
glEnd;   // ends to draw points
end;   // end bracket
for i:=0 to nsichtbar-1 do   // starts "for loop"
```

```
begin   // begins bracket
if (ausgabe[i].radius=0) or
(ausgabe[i].d>(314159*ausgabe[i].radius*ausgabe[i].radius))   // checks if
object should be drawn as a point
then   // then
begin   // begins bracket
glColor3f(ausgabe[i].rot, ausgabe[i].gruen, ausgabe[i].blau);   // defines colour
of points
glBegin(GL_POINTS);   // begins to draw points
glVertex3f(ausgabe[i].x, ausgabe[i].y, ausgabe[i].z);   // draws point at given
position in ausgabe
glEnd;   // ends to draw points
end   // ends bracket
else   // if object should be drawn as a sphere
begin   // begins bracket
glColor3f(ausgabe[i].rot, ausgabe[i].gruen, ausgabe[i].blau);   // defines colour
of sphere
glTranslatef(ausgabe[i].x,ausgabe[i].y,ausgabe[i].z);   // moves to position of
the centre of the sphere given in ausgabe
gluSphere(Q, ausgabe[i].radius, 16, 16);   // draws a sphere
glTranslatef(-ausgabe[i].x,-ausgabe[i].y,-ausgabe[i].z);   // move back to origin
end;   // ends bracket
end;   // ends bracket
glPopMatrix();   // resets matrix
glFinish;   // ends drawing process
gluDeleteQuadric(Q);   // destroys Q
gldisable(GL_CULL_FACE);   // ends mode
PageFlip;   // outputs to screen
end;   // ends bracket
if FRecording   // checks if recording is running
then   // then
begin   // begins bracket
if FVidRec.Snap   // checks if recording works fine
then   // then
```

```
Label11.Caption := Format('Captured %d frames', [FVidRec.NumFrames])   //
outputs number of frames captured yet
else   // if there are problems with recording
Label11.Caption := 'Error: couldn''t capture frame!';   // output error message
end;   // ends bracket
end;   // ends procedure
```

7. POSSIBLE APPLICATIONS OF THE LIE-INTEGRATOR

I see two main applications for this program: on the one hand numerical long-time integration of orbits of planets and asteroids in our own solar system or in other systems and on the other hand stability analysis of exoplanetary systems.

7.1 Long-time numerical integration of orbits

You are able to predict the movement of all planets in our solar system with the Lie-Integrator for millions of years. But you don't need to calculate into the far future. By choosing a very small print step and calculation time you can calculate the orbit of asteroids moving through our solar system. Some of them could hit Earth and this impact could cause depending on the size of the object minor up to global catastrophes. It's very important not only for academic scientific questions to be able to calculate the movement of objects in an interacting N-body system like our solar system. Another aspect is that the knowledge about the movement of our planets can give us information about the formation of our and other solar systems. Long-time numerical integration helps us to understand the structure of solar systems. Furthermore we can get also information about the stability of exoplanetary system (Fig.31) and investigate hypothetical arrangements of planets (Fig.32).

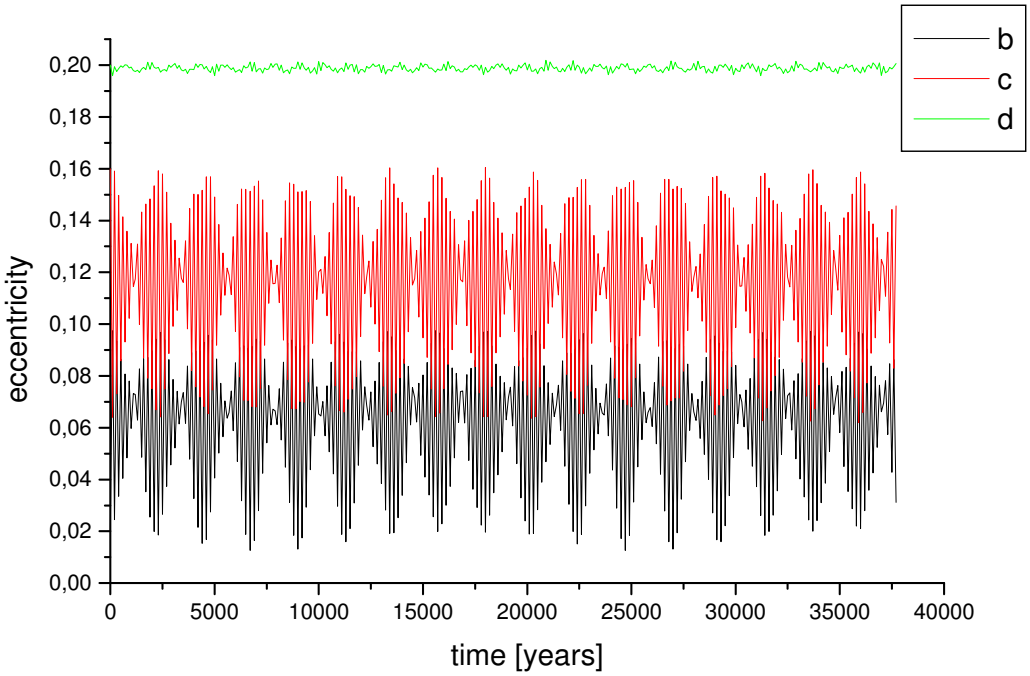

Fig. 31: eccentricity of planets in Gliese 581 for almost 40000 years (calculated with OLI)

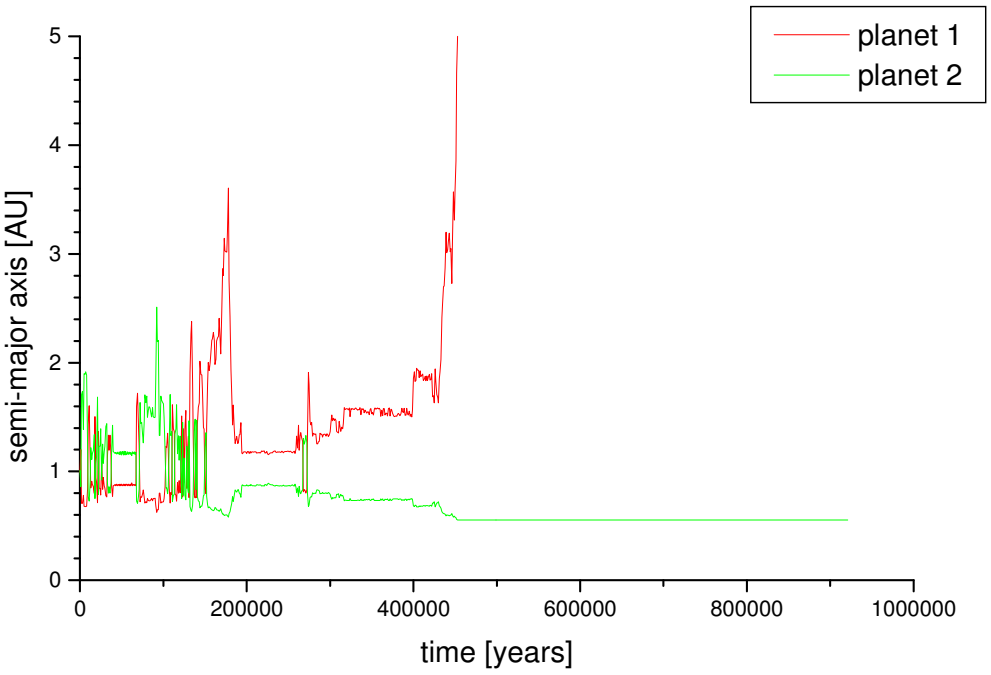

Fig. 32: the semi-major axis of a hypothetical but instable system with exchange orbits for almost 100000 years (calculated with OLI)

7.2 Stability analysis of exoplanetary systems

We know very little about exoplanetary systems for sure, because most of them have been detected with the radial-velocity method which doesn't give any information about the inclination of this system to our direction of view. Furthermore most orbit elements of the planets that have been discovered yet are often having a quiet large uncertainty. To narrow the possible parameters of a detected exoplanetary system we have to exclude some possible values by calculating the systems future. Most stars are billions of years old and their planets are as well. We know that a system we are observing now must have been stable (but dynamical interaction is possible) for this long time and so we except that it will stay stable for the next millions of years. Else it would be a quiet big random that we are nor observing a system drifting apart after billions of years remaining stable. To find out which parameters are reasonable we just calculate the same system over and over again with slightly different initial values. It's a simple try and error method. The results are often displayed in stability maps (Fig.33) and that's the reason why I've included such a feature into the NLI. Another aspect is the search for additional stable objects in a known exoplanetary system. Due to the fact that most exoplanets that have been discovered yet have masses of about Jupiter or more you often search for "mass less" objects with stable orbits in such systems. The "mass less" object could be an Earth-like planet that has of course a neglectable mass compared to a star and one or more Jupiter mass planets. These stability analyses help us on our quest to find life outside our own solar system.

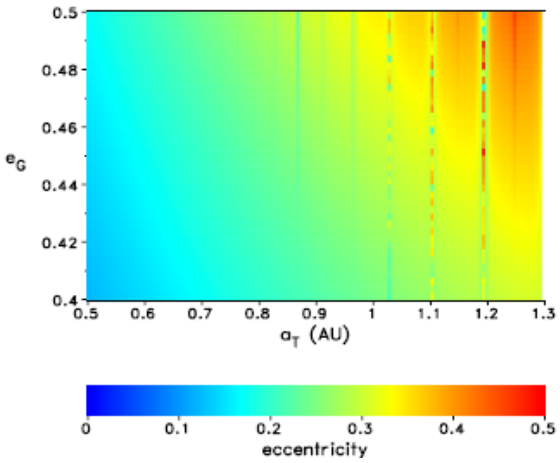

Fig. 33: a stability map, you can see some resonances between 0.9 and 1.2 AU.

7.3 Other imaginable applications

Beside these two main applications some more are possible. For example: you can create illustrative 3D-animations with one tool of the program. So an educational application is also possible, because you can show students the movement of planets and their interactions visual and not only in mathematics. Furthermore the exoplanet tool enables observing astronomers to check their results of the measurement of exoplanetary orbits. More over the high accuracy of the Lie-Integrator is also a possibility for Archeoastronomers to calculate the positions of the planets in our solar system thousands of years ago and compare them with archaeological discoveries. But they will have to do some side-calculations to project the orbits down to Earth. Maybe other fields of use will also open for the Lie-Integrator in the future.

8. CONCLUSION

The Lie-Integration is a method to solve differential equations that has been developed by Sophus Lie more than hundred years ago. You use this method to solve systems of first order differential equations numerically and the Lie-Integrator applies it on the Newtonian Gravitation Equations for an interacting n-body system. The mathematics for this has been done by A. Hanslmeier and R. Dvorak in 1983 and on that foundation they have developed the first Lie-Integrator for problems of n-body celestial mechanics. Due to the great advantages of the Lie-Integration method like the flexible step length and the high accuracy I have decided to create a more modern version of this program. For this task I have chosen Delphi as my programming language. I tried to fully exploit the possibilities of Delphi and developed a program that is not only capable to calculate the orbits of planet. It possesses some very powerful evaluation functions. The NLI can create tables with the needed data out of the result file as well as displaying elegant and fully labelled diagrams. Furthermore it is able to draw colourful stability maps that contain as much information as 3-dimensional diagram about a solar system. A very impression element of the NLI is the 3D-animation. The evaluation module uses OpenGL based 3D graphics to display a real time simulation of a solar system that has been previously calculated. You can fly around in a 3-dimensional space and even record your views. With another tool of the program it's possible to create artificial radial-velocity curves of exoplanetary systems. In addition to that you can find also a very useful program element in the Lie-Integrator to calculate scans in orbits elements. It gives the user the ability to perform a sequence of calculation with slightly different initial values. Furthermore the NLI contains a very useful help system and the content of the help files is based on chapter 5 of this paper. Series of test simulations have shown that this program works as well as the OLI. I've done many runs on the 2-body problem and the restricted 3-body problem, which have shown the expected results. Of course I have tested all new features as well. Finally I've done a long-time run on our own solar system and it stayed stable over a million years as expected.

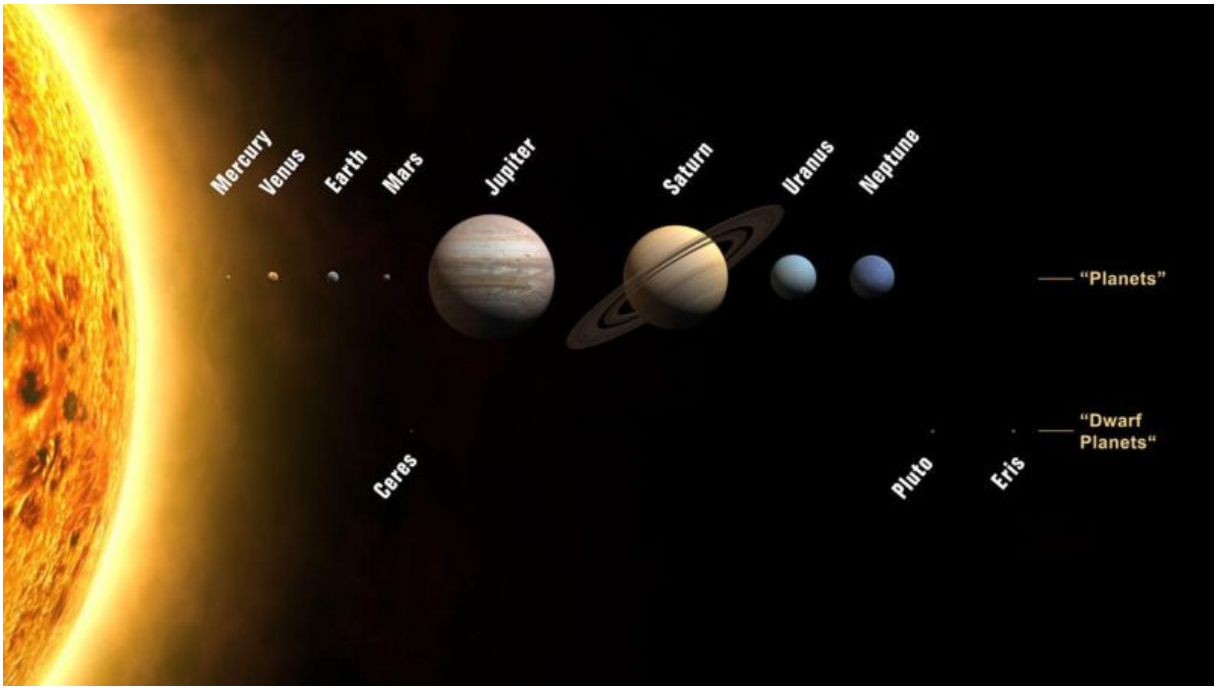

Fig. 34: Our solar system

Beside the elements I've added, I've also done some modifications on the core code of Lie-Integrator. The main modifications have made the instability criterion more flexible. You can now select the maximum eccentrity or the fraction of the Hill-radius when a mass less objects will be ejected. In addition to that I've also done some minor changes in the core code that increases the flexibility of program so that you can turn on or off not essential features. This program unites advantages of the OLI with those of modern program with graphical surface. So the NLI can calculate planetary orbits and evaluate the results in many different ways that seem to be useful in the possible fields of applications like predicting the orbit of asteroids or stability analysis of exoplanetary systems. This Lie-Integrator will certainly find its place as a useful program in numerical calculations for celestial mechanics.

REFERENCES

[i] A. Hanslmeier and R. Dvorak (1983): Numerical integration with Lie-series; A&A 132

[ii] online: http://turnbull.mcs.st-and.ac.uk/history/Mathematicians/Lie.html (10. 9. 2007)

[iii] R. Dvorak, F. Freistetter and J. Kurths (2005): Chaos and Stability in Planetary Systems; Springer (Berlin)

[iv] W. Gröbner: Die Lie-Reihen und ihre Anwendung(1967): VEB Deutscher Verlag der Wissenschaften (Berlin)

[v] online: http://en.wikipedia.org/wiki/Hill_sphere (10. 9. 2007)

[vi] C. Beaugé, N. Callegari, S. Ferraz-Mello and T.A. Michtchenko (2005): Resonances and stability of extra-solar planetary systems; Cambridge University Press

[vii] J. Schneider (2004);: The search for life outside the solar system; CNRS (Paris)