The Art of Computational Science

The Maya Open Lab Development Series,

volume 0

A Vision:

Dealing with Dense Stellar Systems

Piet Hut and Jun Makino

September 14, 2007

Contents

Preface

0.1 The ACS Initiative

This book is the first volume in the *Development Series*[1] of the *Maya Open Lab* project, for constructing a laboratory for simulations of dense stellar systems in astrophysics. The Maya project in turn is the first concrete example of projects developed in an initiative titled *The Art of Computational Science (ACS)*. The aim of the larger ACS series, of which the Maya Series will be a part, is to provide real-world guides to designing, developing, using, and extending computational laboratories.

In many areas of science, computer simulations of complex physical systems cannot be performed with off-the-shelf software packages. Instead, computational scientists have to design and build their own software environment, just as experimental scientists have to design and build their own laboratories, before being able to use them. For a long time, however, the know-how needed to construct computational laboratories has remained only a form of tacit knowledge.

Unlike explicit knowledge that can be found in manuals, this type of implicit knowledge has been alive in conversations among experts, and has been passed down in that way only as a form of oral tradition. This kind of knowledge has not been written down anywhere in sufficient detail to allow it to be passed down without direct personal instructions, or indirect osmosis through personal participation in a joint project.

We have started the ACS series with the aim of making explicit the implicit knowledge of experts of scientific simulations. Besides offering detailed explanations of the structure of the computer codes used, in an 'open source' style, we provide a deeper layer of knowledge. Besides the *what* and *how* for any computer code, we also provide the *why*: the motivation for writing it the way it was written, within the context in which it was conceived. This will give the user more appreciation for the background of the structure chosen, and most importantly, this will give the user the ability to easily modify and extend the

[1] `http://www.ArtCompSci.org/kali/development.html`

codes presented, without finding oneself at odds with the original style and aim.

The umbrella series within ACS is called *Open Knowledge*, to indicate the fact that we aim at far more than providing an 'open source' project. In that series we plan to publish what we have learned about the ACS approach, insofar as it is generally applicable, to any area in computational science.

0.2 The Maya Series, and the Kali Code

The Maya series will contain the first such fully worked-out example of building a computational lab. The first several volumes in this series will focus on one particular topic, the *gravitational N-body problem*, the oldest unsolved problem in mathematical physics, and an area of active research in astrophysics, with applications ranging from star formation to collisions between normal stars, neutron stars and black holes, with implication for the growth of structure in the universe on all scales, from that of clusters of galaxies down to the formation of planets and their satellites.

One reason to choose this topic is to provide an example of a computational lab where the underlying physics is very simple, just Newton's classical law of gravity. The complexity lies exclusively in the emergent properties that show up in the paradoxical thermodynamics of self-gravitating systems, where negative specific heat and other conundrums drive each system far out of equilibrium in ways that are theoretically still ill understood, yet can be modeled accurately in computational labs.

Given the simplicity of at least the underlying physics, we hope that the venerable N-body problem can function as a paradigm for the exploration of new principles and methods for a novel generation of computational laboratories. With this idea in mind, we are trying to present these first few books in sufficient detail for researchers from other areas of natural science to be able to follow and appreciate the approach to virtual lab building presented here.

The aim of the Maya series is to develop, in many small steps, the *Kali* code, a state-of-the-art code that can be used for research in dense stellar systems.

The name *Maya* seemed fitting for two reasons, one connected with Middle America and one with India. The Maya culture was very good at accurate calculations in astronomy. And the word *maya* in Sanskrit has the following meaning, according to the Encyclopedia Britannica: "Maya originally denoted the power of wizardry with which a god can make human beings believe in what turns out to be an illusion." Indeed, a simulation of the heavens is something virtual, an illusion of sorts, and a considerable feat of wizardry.

We borrowed the name *Kali* from the Sanskrit *kali*, meaning *dark*, as in the *kali yuga*, the dark ages we are currently in according to Hindu mythology. The same word also occurs in the name *Kali*, for the Hindu Goddess who is depicted as black. The term *dark* seemed appropriate for our project of focusing on forms of

tacit knowledge that have not been brought to light, so far, and perhaps cannot be presented in a bright, logical series of statements. Instead, we expect our dialogues to carry the many less formal and less bright shades of meaning, that pervade any craft.

Future volumes in this series are expected to cover wider and wider areas of science. Within the Maya series, we will move from stellar dynamics, associated with the gravitational N-body problem, first to stellar evolution and stellar hydrodynamics. It is our hope that this series will be followed by additional series for other areas of simulations in astrophysics and physics in general. And before too long, we hope to attract chemists and biologists, and well as scientists from other areas. If they find our approach useful, and like to try it out in their own field of expertise, we will welcome them to join this series, and to provide additional volumes discussing computational labs for their specialty.

The current volume is the first one in the *Development Series* of the Maya Open Lab project. In parallel, we are also writing a *School Series* for the same Maya project, as a more educational counterpart to, and introduction for, the full Development Series. In the School Series, also in dialogue form, we introduce the basic notions of physical forces and resultant motions, the associated mathematical description through differential equations, and numerical techniques for finding general solutions to those equations. Here, too, we welcome others to join us with similar educational approaches, applied to other topics.

0.3 A Narrative in Dialogue Form

The format we have chosen for our book series, is sufficiently unusual that it deserves some clarification. Most scientific text books summarize the state of the art of a field 'after the dust has settled.' Students are presented with a clean path of logic that bears little resemblance to the historical process of trial and error that underlies any scientific discovery, large or small. As a result, a graduate student facing his or her first major piece of independent research is in for a big shock, suddenly realizing how hard it is to derive anything new with little or no guidance.

In practice, the blow is softened in many cases by conversations with a range of helpers, from fellow students, somewhat more senior postdocs, as well as thesis advisers and other faculty members. However, not everyone is so lucky as to find sufficiently helpful advice that way, and in a world of increasing specialization, the advice of even the most well intended helpers may not suffice for the problem at hand. This is especially true in a rapidly evolving area such as computational science, where the most basic tools, from languages to packages to the very approach to programming, have changed so much even in the last ten years.

Nor is browsing in a book store is likely to turn up much that is helpful. While

there are many books that discuss algorithms and programming languages, those are necessary but far from sufficient ingredients for building a computational lab. What is needed is a practical form of know-how, of how to approach the whole process of lab building, from developing an overarching vision to connecting it to nitty-gritty details. To present this form of know-how, via case studies, is the central aim of our book series.

Given this wide-ranging challenge, a traditional approach of just summarizing our basic points would most likely be insufficient to let students apply our method to their own specific problem case. Instead, our approach is based on giving a hands-on feel for the many pitfalls and diversions on the road to the design and development of large and complex software systems. And the only real way to get such direct experience is to make mistakes, many of them, and then to learn from them, systematically.

Even the best researchers spent most of their time making mistakes; it is the fuel for the engine of exploration. Our aim in this series is to share this approach with our readers, providing them with a sample of the types of mistakes we have made in the the quarter century that we have been working in computational science. The best way to share this atmosphere of trial and error, we think, is through a dialogue between researchers who resemble ourselves, in their way of formulating plans, noticing problems, adjusting their aims, finding ways around stumbling blocks, in short, dealing with the joys and frustrations of every-day research.

One of us (J.M.) first applied this approach in a short book called *Pasokon Butsuri Jitchi Shidou (Practical Computer Physics)* [1999, Kyouritsu Shuppan, Tokyo, Japan]. Since we were both happy with the result, we decided to continue this style in the ACS series.

We could of course have written a fictional dialogue between the two of us, distilling our experience of working together for much of our professional life: since 1988 we have been coauthors on some fifty scientific publications, and together we have organized two symposia sponsored by the International Astronomical Union and edited the corresponding proceedings, as well as a number of workshops. However, such an literal approach would have been too narrow, and we decided it would be much more fun to use composite characters, drawing from history and imagination.

0.4 ACS versions

We use the name *The Art of Computational Science* not only for our book series, but more generally for the software environment for which the books provide the narrative. The environment includes the collection of computer codes discussed in the books, together with the infrastructure to make it all work together seamlessly. This implies extensive comments provided in the codes themselves,

as well as manual pages.

Our plan is to make successive stable versions of this software environment available, starting with ACS 1.0, which contains a small but self-sufficient core of simple N-body programs and accompanying documentation and narrative. These versions can be freely downloaded from our web site "http://www.ArtCompSci.org". They include all completed and partly completed volumes in our book series. Text, code, and everything else is presented as open source software under the conditions of the MIT license:

0.5 Background

The quickest way to obtain more background information about dense stellar systems is to consult the web site "http://www.manybody.org/modest.html" of the MODEST initiative (for *MOdeling DEnse STellar systems*). There you can find out about its many workshops and working groups, as well as teaching activities, including N-body Schools where we regularly present the current ACS material. You can also find literature references there, under working group 8.

Two specific recent references, close to the material presented in the *Maya* series, are:

- *Gravitational N-Body Simulations*, by Sverre Aarseth, 2003 [Cambridge University Press]
- *The Gravitational Million-Body Problem*, by Douglas Heggie and Piet Hut, 2003 [Cambridge University Press]

0.6 Historical Note

The current version of this introduction was rewritten in December 2006, to take into account the name change from Kali series to Maya series, and the introduction of a School series in addition to the Development series that was started in 2004.

0.7 Acknowledgments

Besides thanking our home institutes, the Institute for Advanced Study in Princeton and the University of Tokyo, we want to convey our special gratitude to the Yukawa Institute of Theoretical Physics in Kyoto, where we have produced a substantial part of our ACS material, including its basic infrastructure, during extended visits made possible by the kind invitations to both of us by Professor Masao Ninomiya. In addition, we thank the Observatory of Strasbourg, where the kind invitation of Professor Christian Boily allowed us to make a rapid beginning with the current volume.

Finally, it is our pleasure to thank Douglas Heggie, Stephan Kolassa, Ernest Mamikonyan, Bill Paxton, Michele Trenti and John Tromp for their comments on the manuscript.

Piet Hut and Jun Makino

Kyoto, June 2004 (revised in December 2006)

Chapter 1

Introduction

1.1 A Chat between Two Astronomers

Alice: Hi Bob, what are you up to?

Bob: Well, to be honest, I'm not quite sure yet. I'm in between projects right now. It's nice to sit back for a bit, looking at the whole field of astronomy, before plunging into a new project. But realistically speaking, it is likely I will continue working on star clusters. How about you?

Alice: Actually, I'm in a similar situation of having finished old projects and not yet taken on a major new one. Since I moved here, a month ago, I've been dealing with all kind of chores that had been accumulating, but now I have a clean desk, and I'm ready to start up.

Bob: The previous time I felt like this was when I had handed in my thesis. I was surprised to find myself suddenly in a pleasant vacuum, after rushing so much to get everything finished in time. It was soon afterward that I started to get involved in various parallel projects, which never seemed to come to an end. That was eight years ago.

Alice: I heard you just got tenure. I guess that has something to do with your finishing up your two latest projects?

Bob: how did you guess! While I enjoyed my work, I did feel a little constrained. There were times that I would have loved to go in some other directions, but that probably would not have been wise to do at that point.

Alice: I think you made the right decision. When I got tenure, about ten years ago now, I was in a different situation. We were in the middle of a large cosmological project, simulating the formation of large scale structure in the universe, working together with people from various teams. It was an exciting time, in which there were a number of basic questions that we could address for

the first time.

Bob: I guess cosmology has now become as detailed a modeling job as any other field in astronomy?

Alice: Yeah. Now that the standard model is pretty well understood, cosmology has become a rather ordinary field in astronomy. But I never specialized in cosmology. While most of my work has been in stellar dynamics, I seem to keep moving between different fields. For example, I just finished a paper in planetary dynamics on the dynamical formation of binaries in the Kuiper Belt.

Bob: I'm certainly familiar with your publications in cluster dynamics, especially the analytical treatments you have given. They were quite useful for my more applied simulations. And didn't I see a paper of yours on black hole dynamics recently?

Alice: Yes, that was on the question of whether two massive black holes spiral in within a Hubble time, after they have been brought into proximity by the merging of their parent galaxy. And indeed, I tend to work on more analytical questions, though I enjoy doing large simulations too. Comparing the two and figuring out from both sides why there are discrepancies is most fun: it keeps surprising me that you can use pen and paper to predict roughly what a computer will come up with after performing a trillions of floating point calculations.

Bob: Which of all those fields in stellar dynamics do you find most interesting, having worked on all length scales between asteroids and cosmology?

Alice: Right now I would say star clusters, especially those star clusters where there is an appreciable chance for stellar collisions in the densest regions. Dense stellar systems, in other words.

Bob: As you know, this is close to my interest and background. But let me play the devil's advocate for a moment. What is so interesting about dense stellar systems, in particular?

1.2 Dense Stellar Systems

Alice: A quarter century ago, when I was an undergraduate, astronomy was much less unified than it is today. People observing in different wavelength bands did not talk as much with each other as they do today, and theorists studying stars, star clusters, galaxies, and cosmology had even less overlap in their research. Someone simulating the birth or evolution of a star; someone simulating a star cluster; someone simulating a collision between two galaxies; or someone studying large scale structure of the universe – all four of them would be working separately, each on their own island. In contrast, we now see bridges everywhere.

When studying large scale structure simulations, we see how normal galaxies are built up through the process of merging smaller galaxies and protogalactic

gas clouds. And when we study a collision between two galaxies, we see how star clusters are formed through the shocks that occur in the bridges and tails connecting them. So this already connects the largest three categories from the largest scale down. Similarly, when we start at the bottom, studying the formation of an individual star, we have learned that we can only understand the formation process in an environmental way, taking into account the interactions between many protostellar clouds simultaneously. So this forces us to study a whole star forming region, a proto stellar cluster. In this way all four fields are now connected.

It is fair to say that dense stellar systems form a central topic in the current trend toward unification of astrophysics. Not only is the topic connected with structure from the scale of individual stars up to that of galaxies and beyond, it also is connected with the study of extreme forms of matter, including neutron stars and black holes. And finally, on the level of simulations, it combines the simplest and most elegant theory of complexity, that of self-gravitational systems, with a diverse mix of astrophysical effects, as seen in stellar evolution and stellar hydrodynamics, including the physics of accretion disks, common envelope evolution, and so on.

Bob: That is all nice and fine, from a high-level point of view. However, when I'm working in the trenches, so to speak, I do not have much use for the big picture, and my only job is getting the details straight, and to get the work done.

Alice: Of course, the details are what counts, in any real piece of research, if you want to get any real work done. But it is equally important to keep sight of a broader picture. If not, you'll still get a lot of work done, but not necessarily in an efficient way, and it will not necessarily lead to interesting results.

Bob: I agree that you have to be sensible. But you can be sensible without grand pictures and declarations, I would think. How can a grand picture help me to get my work done?

Alice: There are many examples. Just to name one in stellar dynamics: the introduction of tree codes in the mid eighties made it possible to do the type of cosmological simulations that we are all familiar with. Until then, there was a huge gap between P^3M codes and direct summation codes. The breakthrough came not by improving details in a particular piece of work in a given project, but by stepping back and rethinking the whole approach to large-scale simulations in stellar dynamics.

Bob: I guess you could call that a paradigm shift. But such shifts are few and far between. I don't think you can take that as a typical example. What I see as the future of dense stellar systems simulations doesn't require any paradigm shift. Computers are getting fast enough, and what is needed is to write more software, of the type that can handle not only stellar dynamics but also stellar evolution, on the fly. And we can throw in hydrodynamics as well. In short: what I see happening is a type of 'kitchen sink' approach to simulating dense

stellar systems.

Alice: That doesn't sound very attractive. At the very least I hope you come up with a better name! But more seriously, why would you want to do that? And even if you were to do that, why would anyone believe the results from such a monstrous combination of complicated codes?

Bob: I think we have no other choice. I grant you that idealized abstractions have their use, by now we have learned enough about a system of self-gravitating point masses, all of equal mass, and it is time to move on.

Alice: Well, moving on beyond equal-mass point masses, and doing your kind of kitchen sink simulation, adding stellar evolution and hydrodynamics, those are quite different concepts. Surely there is a middle ground between the two!

Bob: Not really, I would say. It may sound strange, but I think there are only two reasonable ways to simulate a star cluster: either you use the most extreme idealization, in which all stars have the same mass, and with radius zero; or you couple every star to its own stellar evolution program, to model its internal degrees of freedom, at least in principle.

Alice: Strange indeed. Please explain.

1.3 The Equal-Mass Point-Particle Approximation

Bob: Let us first look at the history. I presume you know how star cluster modeling got started. You have seen more of it than I have.

Alice: When it all got started, I had barely arrived on this planet, and I wasn't reading the Astrophysical Journal yet, or anything else for that matter. But yes, I know the rough history. It took some 25 to 35 years to understand the dynamical evolution of a star cluster, modeled as a collection of equal-mass point particles, depending on how you count, since the earliest N-body calculations were performed on modern computers, around 1960. But you know the details better than I do. I guess it is your turn to do some summarizing.

Bob: During the sixties, simulations by Aarseth, Wielen, and others showed how three-body interactions form the dynamical engine at the heart of an N-body system, the agents of change in their evolution. Even if no binary stars were present initially, they would be formed dynamically in simultaneous three-body encounters. And once there, encounters between these binaries and single stars would complicate the normally weak heat flow mediated by two-body relaxation: a single scattering encounter can suddenly release a large amount of energy when a binary increases its internal binding energy by a significant amount.

During the seventies, Henon and Spitzer and co-workers used approximate

Monte Carlo Fokker-Planck simulations to model the contraction of the core of a star cluster, on a time scale that is only an order of magnitude larger than that of the half-mass relaxation time scale. This so-called gravothermal catastrophe was predicted in the sixties, and actually observed by Henon in rudimentary form in the sixties as well. But it was seen much more clearly in the seventies in statistical simulations. While more difficult to observe in actual N-body calculations, because of the still low number of particles, a few hundred at most, they were seen there as well. In fact, one could even argue that in the late sixties, some N-body simulations already showed hints of this effect.

In the eighties, finally the behavior of an equal-mass point particle system after core collapse was elucidated, first by Sugimoto and Bettwieser, who showed that a post-collapse cluster can undergo so-called gravothermal oscillations, a series of local mini collapses and expansions in the very center of a star cluster.

Alice: And then Goodman gave a detailed stability analysis in which he predicted the minimal N value for which an equal-mass N-body system would show such behavior. At that point, theoretically the equal-mass evolution was well understood, wouldn't you say?

Bob: I only consider something understood if it comes out of my simulations, in a repeatable and robust way. I have seen too many semi-analytic predictions come and go in my young life to have too much trust in those!

Alice: What Goodman did was finding roots in the complex plane of relatively straightforward and certainly well defined equations; I wouldn't call that semi-analytic, and I certainly would trust the complex plane a lot more than the complexities of any complicated simulation.

Bob: I guess we're talking about matters of taste here, though many would think this an odd thing to say about 'hard' science. But moving right along, let me make my main point.

It would be another decade before the predictions would be tested in simulations. In the late eighties, various simulations based on Fokker-Planck approximations as well as gas models verified what Sugimoto and Bettwieser had seen in the early eighties, and in the mid nineties saw the first observation of gravothermal oscillations in a real N-body simulation, by Makino. In a way, this was the end of a chapter in stellar dynamics, and further progress had to come from more realistic systems.

Alice: But surely people had used a mass spectrum long before that.

Bob: Yes, in fact the very first paper by Aarseth already described a multi-mass simulation, in the early sixties. But it was only in the seventies that more detailed studies elucidated the main physical mechanisms of mass segregation.

By the way, the field of stellar dynamics owes a lot to the inspiration provided by Sverre Aarseth. For the last several decades he has shared his codes with anyone interested, operating in an 'open source' mode long before the term was invented. Not only that, with his constant readiness to help anyone interested in

using his code, he has set the tone for collaboration for at least two generations of stellar dynamicists. I believe that the attitude toward collaboration is more prevalent in stellar dynamics than in most areas in astrophysics, and as such this simple fact may be Sverre's single-handed accomplishment.

Chapter 2

Stellar Evolution

2.1 The Role of Stellar Evolution

Alice: I guess mass segregation had already been observed in equal-mass systems, as soon as hard binaries were formed, since most of the time these tight binaries behave as point masses that have twice as much mass as single stars.

Bob: Yes and no. If you would neglect three-body interactions, you would be right. And indeed, if you would start a simulation with a significant number of primordial binaries, that would definitely be the case. But those simulations were done later, for the first time in the late seventies, and more systematically in the early nineties, after observations had shown that globular clusters have indeed significant numbers of primordial binaries. In contrast, if you only deal with dynamically formed binaries, you don't see much subsequent mass segregation. First of all, such binaries are formed in the core, where the density is highest. Secondly, recoil from three-body reactions will tend to kick binaries away from the center, even while mass segregation will tend to let them move back toward the inner part of the core.

Alice: I see. So mass segregation was studied systematically only in the seventies?

Bob: Yes. Monte Carlo Fokker Planck simulations as well as direct N-body integrations showed in detail how heavier stars tend to wind up in or near the core of a star cluster. At the same time, lighter stars were seen to be more likely to escape from the system. However, the problem with these early multi-mass calculations is that they were inconsistent.

Alice: Inconsistent?

Bob: They were not very realistic, by and large, since most of them did not include the effects of stellar evolution. In the real world, while heavier stars do tend to sink to the center of a star cluster, roughly on a two-body relaxation

time scale, they also tend to burn up more quickly. As you know, a star 25 times more massive than the sun burns up a 1000 times faster. So it would be totally unrealistic to follow a multi-mass system, and to leave the heavier mass particles in the center, just sitting there. You really would have to remove them, or at least most of their mass, as soon as their age exceeds that dictated by stellar evolution. Few multi-mass simulations did that.

There were some exceptions. In the early eighties Aarseth and coworkers in their simulations began to remove some or all of the mass of stars after they reached the end of their life. But most multi-mass simulations in the seventies and eighties did not include stellar evolution.

Alice: But it can't be that hard to just diminish the mass of a point particle at the end of its nominal lifetime, from that of a main sequence star to that of a white dwarf, neutron star, or black hole, as the case may be.

Bob: Or put it equal to zero, for those stars where you believe that no remnant is left at all. Yes, that can be done, but that will not get you very far. Fiddling only with the masses of stars is not enough. You would then be faced with the question: what to do with binary stars? Take two stars in a binary with a semi-major axis that is much larger than the sum of their radii while they are on the main sequence. Soon before the heaviest star will end its life, it will want to evolve into a giant phase, and as a result, it may well dump much of its mass on its companion star. How are you going to model that? Not by simply removing the mass from the system. That would not do justice to a large fraction of close binary evolution – and close binaries are exactly the dynamical heat source of an N-body system.

In other words, the only way to make a meaningful simulation of a non-equal-mass system of particles is to add a realistic form of stellar evolution, for single stars and especially for binary stars. Nothing else makes sense to compare with observations.

2.2 The Case for Including Stellar Evolution

Alice: You like to make strong statements, but I can make a few too! We don't know much of anything about the quantitative evolution of close binaries, so what is the point of adding a largely unknown quantity to an otherwise very detailed stellar dynamical situation?

But let me answer my own question, if I may. We know that the dynamical evolution of a star cluster is not very sensitive to the nature of the central heat source. As long as we know the amount of heat lost at the half-mass radius, the core will adjust itself to whatever size it needs to have to let the central heat source produce the amount of energy needed. The situation is quite similar to the evolution of a single star, such as the sun: given the heat lost from the photosphere, the central temperature and density can adjust quite easily to

create the right amount of energy in nuclear reactions.

Bob: Well, that may give you a rough agreement for the overall structure, but for me that is not good enough. I am interested in predicting the number of X-ray binaries and millisecond pulsars and their binaries, as well as the physical characteristics of those systems. They are important from an observational point of view. As you know, a significant fraction of X-ray binaries are located in globular clusters, even though these clusters contain much less than 1 percent of all the stars in our galaxy. It is clear that the vast majority of those X-ray binaries in clusters are formed through dynamical effects, as a result of the high density of stars.

Alice: But you can do that in a two-step process. First you make a good enough dynamical model of the dynamical evolution of a star cluster. Since X-ray binaries and binary pulsars form only trace populations, neglecting them will not greatly influence the overall dynamical evolution, I would think. Now, having done that first step, you can then statistically sprinkle your exotic binaries into the outcome of your first step. This second step will have a Monte Carlo nature, but who cares? Your precise N-body calculation started of from random initial conditions, after all.

Bob: Not so fast. Have you ever looked at the HR diagrams based on recent observations of globular clusters?

Alice: Not recently, no. I do remember that astronomers traditionally talked about something like a second parameter effect. What was that again. I believe that metallicity was the first parameter, and that it was claimed that you need at least one more parameter. In other words, two clusters of the same metallicity can still show different types of HR diagrams.

Bob: Yes, that was a long time ago, even before the first observations with the Hubble Space Telescope. Since then we have learned that there is an amazing variety of morphologies in the HR diagrams. And here we are talking about large numbers of stars, not your occasional X-ray source or other exotic object. The only way we are ever going to explain the structure and evolution of globular clusters is by making our simulations detailed enough to at least be able to reproduce, say, the position and shape of the horizontal branch in each well observed globular cluster.

Alice: Okay, you make a strong case for including some form of stellar evolution right away, when performing a star cluster simulation, besides the basic stellar dynamics. The main question then is: how much? What is wrong with using a rather crude toy model?

Bob: I can't imagine modeling globular clusters on a cluster by cluster basis using such a rough approach. I'm afraid we have to do better than that. At the same time, I know that there are large uncertainties in our understanding of contact binary evolution. But what can we do? We do the best we can, and hopefully, 'the best' will get better every year, perhaps when our stellar evolution colleagues will use two- and three-dimensional models.

Alice: I won't hold my breath. But just to set the stage, why don't you continue your historical summary. When did people start using more detailed stellar evolution information in their dynamical simulations of globular clusters?

2.3 Tracks and Recipes for (Binary) Star Evolution

Bob: It was only in the nineties that a few groups began to use look-up tables for single star stellar evolution, and simple recipes to describe what would happen to the evolution of binary stars. While they followed the stellar dynamical evolution in the point mass approximation, as before, their stars would loose mass, and their binary stars would exchange mass, in a roughly realistic fashion.

Actually, it is pretty amazing that it took so long for stellar dynamics and stellar evolution to come together, even in this rather rudimentary way. It could easily have been done twenty years earlier. After all, in the early seventies, single star evolution was well understood, at least in outline, and plenty of evolution tracks had become available. And the main phases of binary star evolution, including the conditions for gradual and catastrophic mass transfer, had been analyzed as well.

Alice: Yes, I remember learning about cases A, B, and C of mass transfer in binary stars, during the undergraduate course in stellar evolution that followed. Why do you think people waited twenty years?

Bob: I don't quite know. But if I were to guess, I think the problem is that the bridge was made from stellar dynamics to stellar evolution. Stellar dynamicists had enough problems on their hands as it was, to figure out what happened during core collapse and afterward, with gravothermal oscillations. Only after all that had been sorted out, and especially with the observational discovery of primordial binaries, did an urgent need for stellar evolution treatments make itself felt.

Alice: But you had just convinced me that mass segregation simulations were unrealistic without stellar evolution. This would suggest that Aarseth should have added stellar evolution to his code in the early sixties!

Bob: Well, he was the first one to do so, as I mentioned, but I guess he too had more urgent problems to deal with early on, such as the treatment of close binaries, inventing and adapting regularization schemes to avoid numerical errors during close encounters. But in the end, I think astrophysicists have inertia, like all human beings: stellar dynamicists just won't get excited about bringing in stellar evolution in their codes until they really have to.

Alice: So why did stellar evolution folks not add dynamics to their simulations, or at least knock on the door of dynamicists?

Bob: They, too, could have and perhaps should have done so in the seventies.

They did start to do population synthesis studies around that time, Tinsley and others. But of course, they too had other worries to take care of first. It was already a big job to get single star evolution treated correctly in a statistical fashion, for a complete star cluster or a whole galaxy. Then, and even now, there are significant questions about what exactly happens to various types of stars in the late phases of their evolution. And when people started to add the signatures of binary star evolution, they quickly realized how little we know about many important phases of close binary evolution, and contact binaries in particular.

So it is perhaps not too surprising that they did not rush to put in dynamical effects of stellar collisions. Even though blue stragglers had been known to exist in star clusters since the fifties, it was not yet clear how important collisions were in producing those stragglers, notwithstanding the pioneering work by Hills in the seventies. In fact, the first conference dedicated to blue stragglers was held in the early nineties, and the first conference on stellar collisions took place in the year 2000!

In contrast, tidal capture of a neutron star by a main sequence had been studied in the mid seventies, because it seemed to be a promising route to explaining the large overabundance of X-ray binaries in globular clusters, which I mentioned before. But that type of analysis was largely local, studying individual encounters, rather than modeling the behavior of a whole cluster in much detail.

Alice: Given that it took twenty years for stellar dynamics and stellar evolution to meet, in evolution recipe mode, once they met it must have produced a major shift in star cluster modeling. Before that, most simulations were exercises in mathematical physics, a lot of fun in itself, but of limited use for comparison with observations, as you have just argued. But once you put in tracks and recipes, aren't you in a good position to start explaining the observational data?

Bob: No, not really. I consider the stage of using recipes as not much more than playing scales, as warm-up exercises before moving on to the type of kitchen sink simulations that I mentioned before.

2.4 Limitations of Tracks and Recipes

Alice: Let me come back to my previous objection. You admitted that we know rather little about the evolution of contact binaries. What is the point of computing detailed stellar evolution models if there are vast areas of stellar evolution where we don't even have a clue of what is going on, beyond a qualitative hand-waving hunch? Take the case of common envelope evolution. If a white dwarf starts to spiral in toward the core of a red giant, it seems plausible from an energetic argument that the envelope of the giant might be lost before the white dwarf reaches the core, leaving behind a tight pair of what will look like two white dwarfs. And it seems equally plausible that not enough mass is lost

not fast enough, and that the white dwarf will merge with the core. Detailed 3D calculations of this process are very hard to do, given the fact that the initial stages cover very many dynamical crossing times.

Given this fundamental uncertainty, why bother doing detailed calculations elsewhere in a simulation? A chain is as weak as its weakest link. It seems like a waste of computer time and software effort to build some very strong and detailed links as long as other links such as common envelope evolution are too weak to contemplate.

Bob: To some extent I agree with you. First of all, okay, I see no use for live stellar evolution codes to compute the evolution of single stars, within a stellar dynamical simulation. And as a second okay, I agree that even the evolution of primordial binaries can be treated adequately through a combination of recipes and stellar evolution tracks. Where I differ from you is in my view of the treatment of merger products.

Alice: Before we get to differences, let me point out that your first and second okay are very different types of okay. The first one applies to the use of relatively accurate and robust information. There is pretty good agreement between different stellar evolution experts as to the quantitative behavior of the tracks of normal stars, apart from perhaps the very most massive stars. Your second okay addresses the use of rather ad hoc and rough treatments of binary star evolution, where quantitative certainty is far less good, some would say almost absent.

Bob: Yes, I agree with all that, but what can a poor boy do? We do the best we can. And in order to get at least some new insight in the evolution of star clusters, I think those tracks and recipes are good enough, even though the latter are far from ideal, of course. But let me move on to my main difference with your view. Whenever two stars collide with each other and merge, you wind up with a merger product that is totally unlike the type of normal ZAMS (zero age main sequence) star that stellar evolution tracks all start with.

A merger remnant has not only a very different metallicity than ZAMS stars in the same cluster had, what is worse, the chemical composition is different at different radii in the star, due to incomplete mixing during the collision, which in general will be significantly off center. In addition, for a hundred million years or so, the merger product will be out of thermal equilibrium, and therefore will have a quite different structure from a normal star.

The only way you are getting even roughly close to determining the structure of such a star is to use not only a live stellar evolution code, but also a live hydrodynamics code to follow the collision. So what I envision is that when two stars come close together within the stellar dynamics part of the simulation, these point particles are handed over to a hydrodynamics code, which replaces them with blobs of, say, SPH particles, layered in the proper way as specified by the stellar evolution information in the simulation. From this point on, the power of SPH is let loose until we arrive at a dynamically settled merger

remnant. A stellar evolution code will then follow the thermal settling, as well as the subsequent more normal evolution.

But if this does not convince you, consider what happens subsequently with a merger remnant. It will be formed in the core of the cluster, most likely, since there the chance for collisions is highest. It will remain in the core, since on average it will be more massive than typical single stars. Therefore, it will stand a significant chance to undergo yet another collision, or be captured as a binary member in an exchange reaction. Even if it avoids collision through such a three-body dance, subsequent evolution is likely to lead to mass overflow. How can you possibly use recipes to treat mass overflow between stars that are not parametrized only by mass and chemical composition, but by the full functional dependence of weird composition gradients, and are possibly still out of thermal equilibrium?

To sum up, while it is possible to make tables of stellar evolution tracks for unperturbed stars, and while it is just possible to combine pairs of these tracks with elaborate prescriptions for any conceivable combination of two stars in orbit around each other, it is utterly impossible to prepare beforehand for all types of strange merger remnants that can be formed, let alone for their subsequent interactions.

Chapter 3

A Kitchen Sink Approach

3.1 The Need for Combining Codes

Alice: But as long as only a few strange stars appear in a whole cluster, why would you make your cluster evolution code so enormously more complicated in order to try to handle those? Surely the vast majority of stars in any globular cluster will never hit another stars.

Bob: But I am not only interested in the vast majority of the stars. I would like to know the nature of X-ray sources and binaries containing millisecond pulsars. By studying the properties of neutron stars and black holes in binaries, we can learn a lot of fundamental physics. We can learn about dense states of matter in a way we cannot possibly produce experimentally within a laboratory, and we can learn about general relativity in strong fields, right up to the horizon of a black hole. And since black holes are black when nothing falls into them, we have little choice but studying them in a binary context.

Alice: I am certainly interested in these objects as well. But wouldn't it be good enough to first neglect the stellar evolution complications, and later make up for them? As I suggested earlier, you can use a method of successive approximations. First you model the whole cluster history, and only then you take care of the special cases, given that you already know the state of the environment they are in.

Let me give an example. When you calculate the evolution of a single star, it is possible to calculate the evolutionary history using only a rather small network of nuclear reactions. Then, if you are interested in the amount of trace elements formed during the evolution, you can go over the whole history once again, in a second pass. You then take the time dependence of the structure of the star as given by the first pass, and against this background you compute the behavior of a much more detailed nuclear reaction network.

Bob: That wouldn't work here. While it is true that the total number of weird stars, affected by collisions, is small compared to the total number of stars, it may not be small compared to the number of stars in the core. The densest cores in globular clusters contain only about 1 percent of the total mass of a cluster, if that much. And a large fraction of stars in such a core may have undergone a close stellar encounter during its lifetime.

So for the overall cluster properties, you may or may not be right, that's an arguable point. But for the detailed core properties, you are certainly not right. How the merger products behave in the core affects the core parameters, and by changing the local environment, all stars in the core are affected.

Alice: Well, maybe that is true. And, come to think of it, there are other dense stellar systems where encounters are more dominant. If we look at star-forming regions, it now seems pretty clear that many if not most stars are produced by interactions between two or more protostellar gas clouds. So for those systems, you have no choice but combining hydrodynamics and stellar dynamics and stellar evolution. And if we look at galactic nuclei, we quickly realize that the stars in the immediate neighborhood of a central massive black hole have a mean time between collisions that is far shorter than the age of the universe.

So, yes, while I still don't like the idea of a kitchen sink approach, I must admit that there are certainly situations where one has no choice but to go that route. And the outcome of such calculations apply to the formation of all stars, and the final stages of the most interesting end products of stellar evolution. Okay, I'm convinced now!

Bob: Glad to hear that! And of course, I am not advertising a kitchen sink approach for all stars simultaneously. There is plenty of room for stellar evolution tracks and binary evolution recipes, as we saw. But that in itself is good and bad. It is good in that it is easier to check the accuracy of tracks and recipes than the accuracy of the outcome of kitchen sink combinations of codes. But it is bad in the sense that the software package you wind up with is maximally complicated.

Compare the situation with a mindless approach in which you do every stellar evolution from scratch, without using stellar evolution tracks. In addition, you could compute the behavior of each binary by letting two live evolution codes talk to each other, dump mass from the one onto the other if need be, and adjust the orbital parameters according to the mass and energy and angular momentum exchanged. It would cost a lot more computer time, perhaps, but it would certainly make the code simpler.

In contrast, what I have in mind is a code that combines everything: stellar dynamics, stellar hydrodynamics, stellar evolution tracks, recipes for binary star evolution, a live stellar evolution code, and yes, a live binary star evolution code as well. I'm not looking forward to putting all that together, but frankly, I see no other choice.

3.2 Parametrization

Alice: If you really succeed in putting together such a code, and you start doing simulations with it, then what?

Bob: publish the results.

Alice: I suppose so. But what have you learned from it? We have agreed now that such calculations should be done, but it is a different question how to interpret them.

Bob: I don't see what the problem is. After you do a detailed simulation, you just make a reasonable choice of what figures to publish, based on the most relevant physical parameters, and then you write the text around it. And for people who want to know more details, you can make the raw data available. What else can you do with the results of a simulation?

Alice: I mean more than that. You told me that you are interested in learning more about the properties of neutron stars and black holes. A theoretical physicist staring at your published figures of Lagrangian radii is not going to find much deep insight in those. In the end you want to come up with hard numbers describing some of the properties of these exotic objects.

Now in order to do that, you have to find a way to characterize the environment of these objects, at the very least the cluster core, as you stressed already. Since all stars interact with each other in the core, we can only believe the details of a simulation if we can believe the details of all physical processes that play a role in the environment.

Bob: But we have already seen that, regrettably, it will be a very long time indeed, before we will be able to give a quantitative treatment of common envelope evolution and similar such ill understood stellar evolution phases. I think we just have to live with what we can do.

Alice: I think we can do better. Let us take the example of the evolution of a single star. Imagine that you were the first person ever to attempt to do a computer calculation modeling the evolution of a single star. You want to write a computer code, but first you have to decide how to represent the physics. Now I walk by your office, ask what you are going to work on, and then I tell you that it cannot be done, since we don't know convection. Sure, we can make order of magnitude estimates, a form of dimensional analysis. We can talk about the typical size of a convective cell, but beyond that we know next to nothing. How can we possibly make quantitative calculations about a star in which convection plays an important role, if we don't even know the value of the most significant digit describing the local state of a fluid in convective motion?

Bob: I imagine that I would be unhappy, hearing that, but that I would argue that I would try anyway, just to see how good or bad the outcome of my calculation would be. After all, that is what happened historically, and we're all glad people didn't give up in the late fifties!

Alice: Well, yes, that is what happened, but there was an essential step that was taken first. Even without any detailed knowledge of convection, a brave step was taken by introducing the concept of convective scale length h. And while there really was no reason to believe that this notion had much physical meaning beyond an order of magnitude characterization, it did provide us with a quantitative handle, a number h that could be determined afterward, by comparing theory and observation.

It was altogether possible that no single good value for h would have reproduced the observed structure and evolution of stars. Perhaps convection should have been modeled by a function that is fully dependent on temperature and density and chemical composition. A priori you just don't know. The happy surprise was that a *single* number h turned out to be enough to find rather good agreement between the model produced and the stars observed.

What I would propose for a kitchen sink calculation of a globular cluster, or any dense stellar system for that matter, would be a parametrization of *all* the unknown physics, by however many parameters it would take.

3.3 Comparison with Observation

Bob: You'll wind up with dozens of parameters, do you realize that?

Alice: I do, and yes, that may present us with significant problems. But now it is my turn to say: what else can we do? But I can say more. First of all, any globular cluster simulation already requires you to choose dozens of parameters. There is the choice of initial model for the density distribution, if we assume the simplest case of isotropy. Then there is the choice of mass spectrum. And for the primordial binaries you have to give a prescription for the initial distribution function, as a function of mass ratios, separation and eccentricity.

Bob: And a prescription for the tidal field, which can be time dependent. And perhaps you'd like to take into account disk shocking, when the cluster passes through the galactic plane. Yes, I know, there are many numbers you have to specify already. And these may well be different for different globular clusters.

Alice: Exactly, that is my point. If you can make detailed realistic models for a number of different globular clusters, you have to provide all those initial conditions, and vary them until you get good fits with the observations. So the situation will not be much worse if you add a handful of parameters modeling the physical uncertainties. And the good thing is, a parameter describing the outcome of common envelope evolution, for example, can be expected to be the same for different clusters, at least in first approximation. So the more clusters you model, the less these extra parameters will cause problems.

Even so, it still seems worrisome to fit dozens of parameters to one set of observations. There is this nice quote about John von Neumann, who certainly knew how to use computers to fit data. He used to say: "with four parameters

I can fit an elephant, and with five I can make him wiggle his trunk."

Bob: It just shows that globular clusters are more complicated than elephants.

Alice: I don't think von Neumann would have agreed. But it all depends on the accuracy of the available data, and most importantly, upon the lack of degeneracy in the types of data available. And this brings me to my second point. If we would just try to fit density profiles and velocity dispersions, we would be hopelessly lost. The only thing that can save us is detailed observations of specific types of stars, such as binaries of different types. In that way, we can hope to obtain more independent constraints than there are independent initial conditions.

Bob: Are you thinking of X-ray binaries and binary pulsars? Given the many hundreds of such X-ray sources that have been found in galactic globular clusters, they can help constrain quite a few parameters already. And similarly, the wealth of pulsars that is being discovered in globulars is constantly growing as well.

Alice: Yes, but I am also thinking about more normal binaries. If future observations can pin down more of the parameters of the current population of binaries, we still will have to search around in parameter space to find the original distribution, but at least that will constrain the uncertainty quite a bit. Primordial binaries provide the largest number of free parameters in setting up a realistic set of initial conditions for a cluster evolution, I bet.

Bob: That is right. And yes, the observations are improving. Also, I don't expect to have a working kitchen sink code any time soon. I fully trust the observers to have quite a bit more data in hand by the time my code will work and is debugged.

Alice: Is a code ever fully debugged?

Bob: Well, good enough is good enough, I'd say. No person is perfect, and no code is perfect.

Alice: Given that you have convinced me that a kitchen sink code is a reasonable goal to work toward, I'm curious to hear a bit more about how you intend to write such a code.

Chapter 4

Software Architecture

4.1 Legacy Codes

Bob: The basic idea is to start with an N-body code, then to add stellar evolution tracks and binary evolution recipes, then to add a stellar evolution code, and possibly an extra version of it that can handle binary stellar evolution, then to put in a hydro code as well, and make it all work together.

Alice: That reminds me of a description of a human being as worth a few hundred dollars in terms of chemical ingredients. Just how to put the ingredients together to make a human being is not that simple.

Bob: That's not a fair comparison. That would be like saying that I had to start with ones and zeroes, or had to start at the byte level. Fortunately I can start with so-called legacy codes: codes for stellar dynamics, stellar evolution, and stellar hydrodynamics, that have been developed over many years by many people.

Alice: Okay, I admit, my example was too extreme. I should have said that you take a fish and a reptile and a bird, and put it together to make a primate.

Bob: Still not fair, but I won't argue with analogies. I can only repeat our mantra: what else can we do? You're not proposing that I'm going to write a kitchen sink code from scratch, do you? That would mean reproducing more than a hundred person-years work of code writing! In fact, that would be an apt use of your analogy of the chemical worth of a human being.

Alice: I'm not sure what the best solution is, but I am sure that there are various ways that are definitely not the best. For example, if you don't make your kitchen sink code modular, you are asking for trouble and you will become bogged down hopelessly. Also, if you do not insist that the legacy code providers give you completely robust codes, you're inviting disaster. Can you imaging handholding some of your stars, because they don't know how to complete

their full stellar evolution track through difficult phases along the giant branch or horizontal branch? With a few hundred thousand stars, you can't afford holding even 1 percent of them by the hand!

Bob: Your second point is certainly well taken. Until recently, in fact no stellar evolution codes could really run by themselves, in a fully automated fashion. However, there now are several experts in stellar evolution who have developed more robust codes, and who have shown an interest in sharing their codes with me. Still, it will be up to me to connect all the codes, from stellar dynamics to stellar evolution to hydrodynamics.

Alice: That brings me back to the notion of modularity.

Bob: What exactly do you mean?

Alice: If you can define a set of interfaces, between each of your legacy codes, through which a limited number of exactly defined variable values can be passed, your life will be a lot simpler.

Bob: Forcing yourself to do that will be very complicated, and frankly, unnatural. How can such an exercise in formality help you?

Alice: What is your alternative, what do you consider natural?

Bob: Any stellar dynamics legacy code I have ever seen has many different places where two stars can come in close proximity. Think about the various regularization techniques, as well as the ways in which perturbers are treated, not to mention stars with unusually high velocity that can suddenly show up unexpectedly. All these places in the code can and probably should be coupled to a hydro code, since in all these cases a stellar collision or close encounter can occur.

Alice: That will never work well. Even if you can get it to work ever, based on a particular legacy code, before you know it either the legacy code will change in small and subtle ways, throwing you off, or you may decide to ask slightly different questions, requiring you to revisit all these many connection points between the two codes. And you want to do this with at least three legacy codes and put in tracks and recipes to boot! This will be hopeless.

Bob: Do you have a *realistic* alternative? Surely your formal picture of limited well-defined interfaces can't possibly be implemented in a real stellar dynamics code with regularization and perturbers and all that. You'll have to rewrite everything from scratch, and even then, it will be far slower than your old code.

4.2 Modularity

Alice: Let us start with what modularity means. If a code is truly modular, you can understand and test each part separately. Then, to see how it all works together, you can make sure that you have at least two independent

implementations for each part, so that you can mix and match and see how swapping modules influences the outcome of a calculation.

Now, to answer your question about regularization and perturbers, I do not suggest that you rewrite the non-gravitational codes. You can put a wrapper around a stellar evolution code, with a well defined interface, and similarly you can put a wrapper around a hydrodynamics code. But the stellar dynamics code will have to be rewritten, I'm pretty sure, for the reasons you just gave. Let us take a bird's eye view of the situation.

Logically speaking, you could introduce four elements. A manager, and three legacy codes, all wrapped in an interface. The three codes will be like black boxes. If you open the hydro box or the evolution box, you will see lots of physics, and lots of wires connecting the various physics modules. But if you open the dynamics box, you will see very little physics, and almost only wires. The part of the code that solves Newton's equations of motion is literally only a few lines, occurring in only a few places. Everything else is bookkeeping, complex orchestration of coordinate transformations, updating of perturber lists, handling exceptions, shrinking or extending groups of particles that need special treatment and so on.

Bob: Ah, you know more about direct N-body codes that you have let me to believe, so far!

Alice: Yeah, it has been a while, but I did crawl through one of them, years ago. It wasn't easy. But you see my point. Logically speaking, you could introduce a manager module, which takes care of all the bookkeeping needed to let the three modules talk together. But you would be replicating much of the bookkeeping that is already happening in the stellar dynamics module. It would be much more natural to rewrite the stellar dynamics module, elevating it in the process to the status of manager module, and letting it do its interfacing with the two other modules. And it can take care of the interface timing between the two other modules as well, as a good manager should.

Bob: You're going way over my head here, talking about two different ways of doing your modular thing, while I have no interest whatsoever in being formal, or modular, or whatever other paradigm computer scientists may come up with. But as far as I can follow you, your second alternative goes more in the direction of what I would do. I would simply take an existing N-body code, and splice in all the connections that are needed with the stellar evolution code, and with the hydrodynamics code. The only difference I can see is that you want me to write the N-body code from scratch, while I maintain that it is a lot more efficient to stat from a legacy N-body code.

Alice: The main difference is what you just called this splicing thing. Where you would to make a few dozen connections between the dynamics code and the other codes, I want to limit the connections to a few, at most. And then I want to define and document those connections in a completely precise way, down to the byte level. That will require you to rewrite your N-body code. That will

take time, but you will soon gain time, over the years to come, when you want to make later extensions. As you know, several of the current legacy codes have a life time of twenty or thirty years, if not longer. It is well worth your time to spend a few years setting things up correctly, and then reaping the benefits for a long time afterward. After all, the time that is spent by experimentalists and observers building a laboratory or observatory is a fair fraction of the time they spend using it. If they wouldn't carefully plan in building a new lab or telescope, and just build new stuff haphazardly on top of old pieces, they would fail.

4.3 Interface specifications

Bob: Wait a minute: you said that modularity means flexibility, and that means that you can later extend things more easily. But then you said that you need well-defined interfaces. That to me sounds like the opposite of flexibility. We have seen plenty of examples where progress was hampered by rigid definitions.

To take a real life example: for decades now, the telephone system in the United States has been running up against the barrier of the ten-digit specification of telephone numbers, with three for an area code and seven for the number within an area code. As a result, they have split up existing regions for a given area code into two halves, one of which retained the old area code, and one of which got a new one. Think of all the effort in people having to reprint all the material that has their telephone number on it, having to repaint their delivery car, and so on.

Alice: Yes, that is a good example of the problems you can get with an inflexible specification. But think of the alternative: if any telephone number could have had any length, back in the days in which the telephone system was set up, based on mechanical relays, it would probably have been more cumbersome then. In any case, whatever solution they would have come up with would sooner or later have led to unforeseen problems.

The real mistake with the telephone system was not so much that they ran into a limitation of a specification, but rather that they did not change the specification.

Bob: What is the point of a specification if you can easily change it?

Alice: You don't want to change it in arbitrary ways, whenever there is a small problem for which it seems convenient to stretch the specification a bit. But you do want to make a change when it is unavoidable to do so, and you don't want to postpone such a change until you walk straight into a wall. In the case of the telephone system, it would probably have been much better to extend the number of digits a few decades ago, as soon as computers began to replace mechanical relays. That way there would have been only one change for everybody, rather than an annoying and interruptive change of area codes once

every few years or more.

I suspect that the real reason for the long delay in a change of specification for the telephone system has to do more with politics than with engineering. It is probably too difficult to get everyone involved to agree on all levels.

Bob: The same problem applies to the internet address protocol, of course. The limitations of a 32-bit address are being felt severely now, especially outside the United States, in countries that were given far fewer addresses than the U.S. had already taken, but where there are now a comparable number of users. The problem there, too, is politics: until the U.S. will agree to go to a wider address protocol, it is unlikely that it will change.

Alice: Fortunately, we don't have to worry too much about high level political pressure when developing star cluster simulation software. In our case we can learn from these and other mistakes. I suggest to allow a controlled update of the specifications, whenever you define a new major version number for your code, in a very well defined manner.

Bob: Why not just allow a common block, and let people pass whatever they want?

Alice: If you allow that, you may as well forget about modularity. In that case, there are so many ways that a small modification can cause a major perturbation that will be almost impossible to trace down. And if you want to parallelize your code, things will only get worse.

4.4 A Matter of Opinion?

Bob: I see that you are really serious with your modularity speak. I will give you the benefit of the doubt, for argument's sake. You have just shown that you are familiar with the basic lay-out of an N-body code, which I respect, since there are few astrophysicists who have actually gone through a direct N-body code that has all the bells and whistles to do collisional stellar dynamics. So let's play the game: you describe to me how I should modularize my code. Then I'll tell you whether it would even have a chance of being reasonable.

Alice: First of all, you should treat the dynamics code in exactly the same way as we have treated the overall kitchen sink code: it should be split up into smaller modules, connected with well-defined interfaces. For starters, the N-body code will be split into a global and local part. The global part will compute the particle orbits throughout the whole cluster, while the local part will handle all the close encounters, coordinate transformations, regularization, and so on.

Bob: But what about perturber lists?

Alice: You mean particles that are too far from a local clump to be part of the clump, but too close, so that their influence on the clump cannot be neglected,

unlike the bulk of stars that are further removed?

Bob: Exactly. Those perturbers will give you a bunch of spaghetti that cross wires between your two neat local and global modules.

Alice: No, they won't. I won't let them. One solution would be to introduce a separate perturber module, besides the local and global modules. Another solution would be to include the perturber lists within the local module. Yet another solution is to devise a different and novel algorithm that dispenses with the use of perturbers altogether.

Bob: The last case is just silly. Decades of experience has shown that you get an enormous penalty in computer time needed when you leave out the use of perturber lists. As for hiding the lists, in whatever module, and then passing their information through a few narrow interface bottlenecks, that is not much better. You'll get a huge data passing overhead.

Alice: The penalty may not be that bad. And you won't know until you try.

Bob: Until you try – that's easier said than done. What you just proposed does require a complete rewrite of a stellar dynamics code.

Alice: That, too, may not be as bad as you think. While it has taken decades to write a legacy code, to rewrite it will surely take a lot less time. You can learn from past insights as well as from past mistakes. You don't have to reinvent the wheel.

Bob: I strongly disagree. Writing a skeleton version, sure, but getting the details right is like fine-tuning a race car, it requires skill and patience.

Alice: Even so, inventing the concept of a race car must have taken far more time than any subsequent designing and building of race cars. In any case, it is clear that we have a different way of looking at things. Perhaps you are right, and I have too optimistic a view of the power of modularity. It is possible that the problem is just too dirty, and that any approach no matter how clean will get bogged down in details which force you to break the rules you started with. I doubt it, but it is a logical possibility. But perhaps I am right, and you have too pessimistic a view, colored by too much frustration with less modular approaches. How are we going to find out who is right? You said you wanted to give me the benefit of the doubt. Well, what type of benefit are you willing to give me?

4.5 Writing a toy model

Bob: I have a suggestion. How about working together for a while in writing a toy model version of your grand vision. You can have the satisfaction of seeing the first steps being taken toward your castle in the sky. And I will then have the satisfaction of your realizing how much more complicated it will be than you now think, to get from the toy model to reality.

Alice: But if you don't believe me, why would you even go to the trouble of writing a toy model?

Bob: Pragmatism. I have to teach a stellar dynamics class, and I had been thinking about developing some form of toy version of an N-body code, to give the students some hands-on experience without requiring them to walk through a legacy code, which they would never be able to do within the time frame of a single course.

Alice: So you will allow my ideas to spoil the young?

Bob: I'll take that risk – as long as I can then point out to them in the end, how very far they will still be from a competitive N-body code!

Alice: Good, we have a deal. I have thought for a long time about the question of how to structure a stellar dynamics code in a cleaner and more modular way, and still let it be able to handle close encounters and all that. I had not thought about combining it with stellar evolution and hydrodynamics, but now that we discussed these options, I feel even more convinced that a modular approach is called for.

If we can manage to construct a flexible modular framework for a general simulation code, getting the overall bookkeeping straight, we can then postpone any decision about what physics to put in, and what approximations to allow. You want stellar evolution? Fine, plug in your favorite module. You prefer to use Monte Carlo integration of orbits, rather than the more accurate but much more expensive direct N-body integration? Fine, replace the global dynamics module for a Monte Carlo module. You like perturbers? Use a local module in which perturbers lists are implemented. You think you can do without? Then design a local module built on an algorithms that doesn't require perturbers. Let a hundred flowers bloom!

Bob: I'd be happy to let one flower bloom, just writing one toy model, good enough for my students to play with. I'll leave you with the seeds for the other ninety nine flowers.

Alice: No need to try to convince you, at this point. Let's roll up our sleeves, and see where our toy model will lead us.

Chapter 5

Dense Stellar Systems

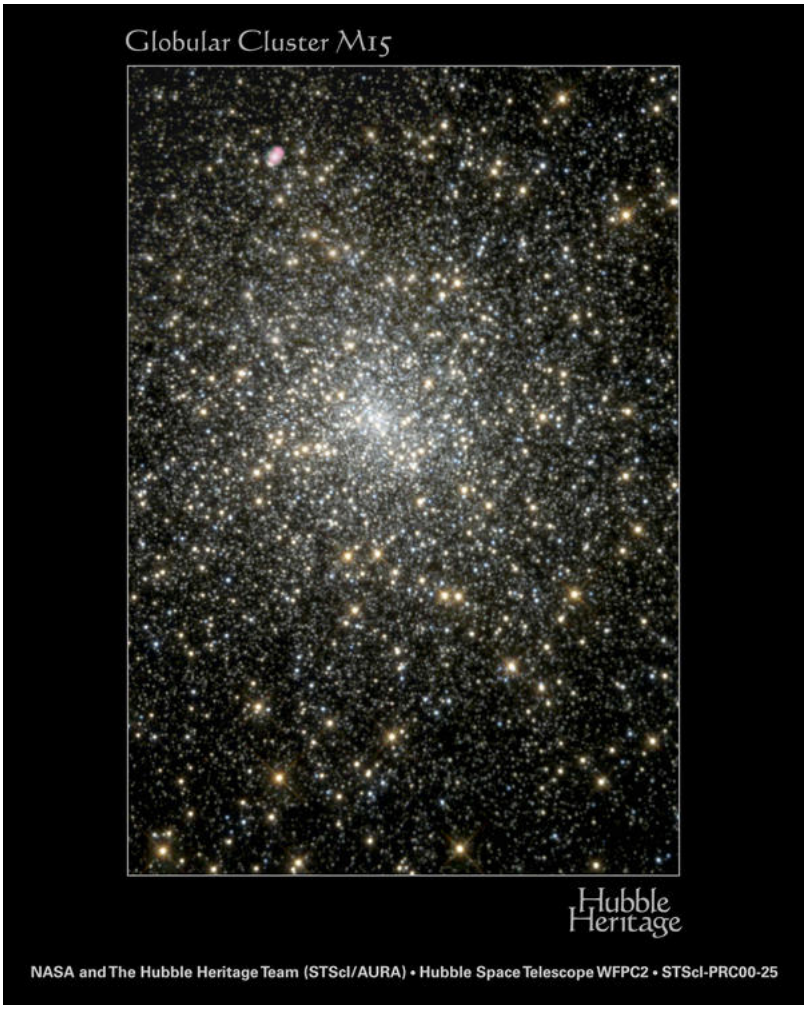
5.1 Galactic Suburbia

The sun is a star like any other among the hundred billion or so stars in our galaxy. It is unremarkable in its properties. Its mass is in the mid range of what is normal for stars: there are others more than ten times more massive, and there are also stars more then ten times less massive, but the vast majority of stars have a mass within a factor of ten of that of the sun. Our home star is also unremarkable in its location, at a distance of some thirty thousand light years from the center of the galaxy. Again, the number of stars closer to the center and further away from the center are comparable. Our closest neighbor, Proxima Centauri, lies at a distance of a bit more than four light years.

This distance is typical for separations between stars in our neck of the woods. A light year is ten million times larger than the diameter of the sun (a million km, or three light seconds). In a scale model, if we would represent each star as a cherry, an inch across, the separation between the stars would be many hundreds of miles. It is clear from these numbers that collisions between stars in the solar neighborhood must be very rare. Although the stars follow random orbits without any traffic control, they present such tiny targets that we have to wait very long indeed in order to witness two of them crashing into each other. A quick estimate tells us that the sun has a chance of hitting another star of less than 10^{-18} per year. In other words, we would have to wait at least 10^{18} years to have an appreciable chance to witness such a collision. Given that the sun is less than five billion years old, it is no surprise that it does not show any signs of a past collision: the chance that that would have happened was less than one in a hundred million. Life in our galactic suburbs is really quite safe for a star.

There are other places in our galaxy that are far more crowded, and consequently are a lot more dangerous to venture into. We will have a brief look at four types

of crowded neighborhoods.



A snapshot of the globular cluster M15, taken with the Hubble Space Telescope.

In the photo above we see a picture of the globular cluster M15, taken with the Hubble Space Telescope. This cluster contains roughly a million stars. In the central region typical distances between neighboring stars are only a few hundredth of a light year, more than a hundred times smaller than those in the solar neighborhood. This implies a stellar density that is more than a million times larger than that near the sun. Since the typical relative velocities of stars in M15 are comparable to that of the sun and its neighbors, a few tens of km/sec, collision times scale with the density, leading to a central time between collisions of less than 10^{12} years. With globular clusters having an age of more than 10^{10} years, a typical star near the center already has a chance of more than a percent to have undergone a collision in the past.

In fact, the chances are much higher than this rough estimate indicates. One reason is the stars spend some part of their life time in a much more extended state. A star like the sun increases its diameter by more than a factor of one hundred toward the end of its life, when they become a red giant. By presenting a much larger target to other stars, they increase their chance for a collision during this stage (even though this increase is partly offset by the fact that the red giant stage lasts shorter than the so-called main-sequence life time of a star, during which they have a normal appearance and diameter). The other reason is that many stars are part of a double star system, a type of dynamic spider web that can catch a third star, or another double star, into a temporary three- or four-body dance. Once engaged in such a dance, the local stellar crowding is enormously enhanced, and the chance for collisions is greatly increased.

A detailed analysis of all these factors predicts that a significant fraction of stars in the core of a dense globular cluster such as M15 has already undergone at least one collision in its life time. This analysis, however, is quiet complex. To study all of the important channels through which collisions may occur, we have to analyze encounters between a great variety of single and double stars, and occasional bound triples and larger bound multiples of stars. Since each star in a bound subsystem can be a normal main-sequence star, a red giant, a white dwarf, a neutron star or even a black hole, as well as an exotic collision product itself, the combinatorial richness of flavors of double stars and triples is enormous. If we want to pick a particular double star, we not only have to choose a star type for each of its members, but in addition we have to specify the mass of each star, and the parameters of its orbit, such as the semi-major axis (a measure for the typical separation of the two stars) as well as the orbital eccentricity.

The goal of our book series is to develop the software tools to make it possible to simulate an entire star cluster like M15, and to analyze the resulting behavior both locally and globally.

5.2 Galactic Nuclei

In photo below we see an image of the very center of our galaxy. This picture is taken with the Northern branch of the two Gemini telescopes, which is located in Hawaii on top of the mountain Mauna Kea.

An image of the central region of our galaxy, taken with the Gemini North telescope. The center is located on the right just above the bottom edge of the image.

In the very center of our galaxy, a black hole resides with a mass a few million times larger than the mass of our sun. Although the black hole itself is invisible, we can infer its presence by its strong gravitational field, which in turn is reflected in the speed with which stars pass near the black hole. In normal visible light it is impossible to get a glimpse of the galactic center, because of the obscuring gas clouds that are positioned between us and the center. Infrared light, however, can penetrate deeper in dusty regions. It is a false-color image, reconstructed from observations in different infrared wavelength bands.

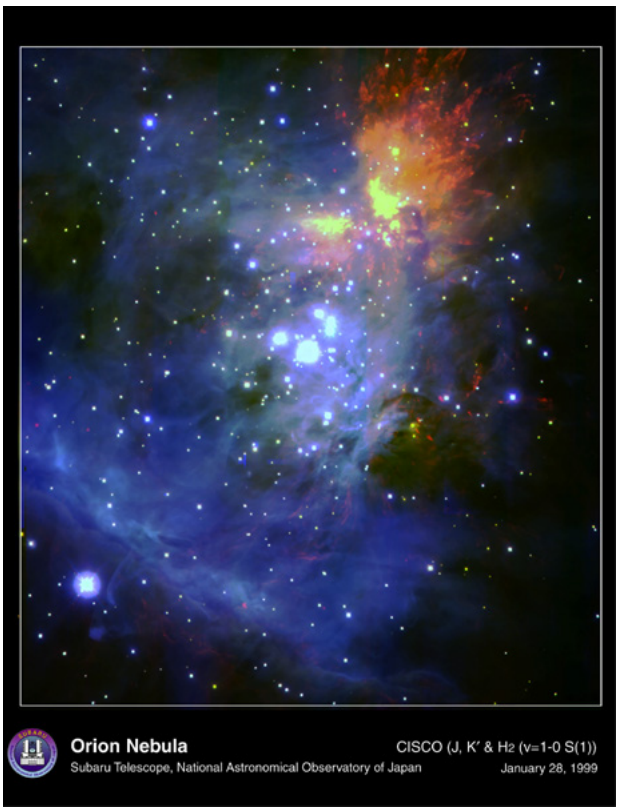
In the central few light years near the black hole, the total mass of stars is comparable to the mass of the hole. This region is also called the galactic nucleus. Here the stellar density is at least as large as that in the center of the densest globular clusters. However, due to the strong attraction of the black hole, the stars zip around at much higher velocities. Whereas a typical star in the core of M15 has a speed of a few tens of km/sec, stars near the black hole

in the center of our galaxy move with speeds exceeding a 1000 km/sec. As a consequence, the frequency of stellar collisions is strongly enhanced.

Modeling the detailed behavior of stars in this region remains a great challenge, partly because of the complicated environmental features. A globular cluster forms a theorist's dream of a laboratory, with its absence of gas and dust and star forming regions. All we find there are stars that can be modeled well as point particles unless they come close and collide, after which we can apply the point particle approximation once again. In contrast, there are giant molecular clouds containing enormous amounts of gas and dust right close up to the galactic center. In these clouds, new stars are formed, some of which will soon afterwards end their life in brilliant supernova explosions, while spewing much of their debris back into the interstellar medium. Such complications are not present in globular clusters, where supernovae no longer occur since the member stars are too old and small to become supernovae.

Most other galaxies also harbor a massive black hole in their nuclei. Some of those have a mass of hundreds of millions of solar masses, or in extreme cases even more than a billion times the mass of the sun. The holy grail of the study of dense stellar systems is to perform and analyze accurate simulations of the complex ecology of stars and gas in the environment of such enormous holes in space. Much of the research on globular clusters can be seen as providing the initial steps toward a detailed modeling of galactic nuclei.

5.3 Star Forming Regions



There are many other places in the galactic disk where the density of stars is high enough to make collisions likely, at least temporarily. These are the sites where stars are born. The above photo taken by the Japanese Subaru telescope in Hawaii shows the Orion Nebula, also known as M42, at a distance of 1500 light years from the sun. This picture, too, is taken in infrared light in order to penetrate the dusty regions surrounding the young stars. The four brightest stars in the center, collectively known as the Trapezium, form the most massive stars of a larger conglomeration of stars, all recently formed from the gas and dust that still surrounds them.

In order to study collisions in these star forming regions, we can no longer treat the stars as point masses. Many of the collisions take place while the stars are still in the process of forming. While a protostar is still in the process of contracting from the gas cloud in which it was born, it presents a larger target for collisions with other stars. In addition, a single contracting gas cloud may fission, giving rise to more than one star at the same time. In this way, the correlated appearance of protostars is even more likely to lead to subsequent collisions.

The proper way to model these processes is to combine gas dynamics and stellar dynamics. Much progress has been made recently in this area. One way to use stellar dynamics in an approximate fashion is to begin with the output of the gas dynamics codes, which present the positions and velocities of a group of newly formed stars, and then to follow and analyze the motions of those stars, including their collisions.

5.4 Open Clusters

Although stars are formed in groups, these groups typically do not stay together for very long. Perturbations from other stars and gas clouds in their vicinity are often enough to break up the fragile gravitational hold they initially have over each other. Some of the more massive groups of newly formed stars, however, are sufficiently tightly bound to survive their environmental harassment. They form the so-called open clusters, where their name indicates that they have central densities that are typically less than what we see in globular clusters.

The open star cluster M67, in a picture taken by the Anglo-Australian Observatory.

The above photo shows one of the richest and densest open clusters, M67, as observed by the Anglo-Australian Observatory. Since this cluster is old enough to have lost its gas and dust, all stars are visible at normal optical wavelengths, at which this image is taken. In the central regions of this cluster, there are indications that some of the stars have undergone close encounters or even collisions. Consequently, this star cluster qualifies as a dense stellar system.

Open clusters typically have fewer members than globular clusters. Also, they are younger. Both facts makes it easier to simulate open clusters than globular

clusters, in terms of computer time needed, when we model the stars on a one-to-one basis in a direct N-body simulation.

On the other hand, the densest globular clusters show a higher frequency and a far richer variety of stellar collisions, making them a more interesting laboratory. In that sense, a dynamical simulation of an open cluster can be seen as providing preparatory steps toward the modeling of globular clusters, just as a study of the latter forms a stepping stone toward the investigation of galactic nuclei.

Chapter 6

An Educational Project

6.1 A Toy Model on the Web

Alice: I really like your idea of writing a toy model for a code to simulate dense stellar systems. While it will be useful for your students, I'm sure we'll learn from it ourselves.

Bob: I don't doubt it. For one thing, you'll see how hard it is to then mature the code into a real research tool.

Alice: I'm not so sure. Given that there is a great demand for such a code, and precious little supply so far, I wouldn't be surprised if even a toy model would turn out to be useful already, if we do a good job.

Bob: I would be very surprised, but as we already concluded, no sense in arguing. We'll see soon for ourselves. And it will be interesting to get feedback from my students. By now I've been working in this field for so long that it is difficult for me to guess what is and what isn't doable for a student, without much guidance.

Alice: I remember very well how difficult it was to get started with N-body codes. There was little practical material available to tell you how to set up and run and analyze experiments. Plenty of articles about algorithms and about programming, but very little about how to put it all together. I basically learned by knocking on doors here and there, working with fellow students who were further along than I, and asking my thesis adviser.

Bob: Yeah, it would be nice to have a more detailed hands-on introduction available. Well, I was planning to make my course notes available on the web in any case, so why not put our toy model on the web as well. Who knows, students at other places might find it useful as well.

Alice: And researchers who are not yet specialists in this area. And perhaps

amateur astronomers, who may have a good background in programming, but don't have the information available about how to get started in modeling a star cluster.

Bob: Good point. You know, it so often happens, when I hear someone talk about the first computer programs they wrote by themselves, that they mention two standard examples: calculating the digits of pi and writing a quick and dirty N-body code, often just a simple forward-Euler implementation.

Alice: I calculated prime numbers, as my first application, and indeed, my second one was an 3-body code. I wanted to model sending a spacecraft from an orbit around Earth to Mars. Was I surprised to learn how tricky it was to deal with Kepler singularities and large scale factors! Writing a simple code wasn't that hard, but I never got it to work properly, the few days that I tried. Not surprising, in retrospect.

Bob: Indeed. It may seem simple, but handling such an Earth-Mars orbit is a rather tricky 4-body calculation, much more difficult numerically than, say computing the orbits of a few planets or even asteroids or comets around the Sun.

Alice: What was your first N-body calculation?

Bob: I wrote a simple video game for myself, inspired by what I had seen in officially packaged games. It seemed natural to me to put in real gravitational forces. It was fun to see how much space ships speed up all of a sudden when they pass close to a star.

Alice: Glad to hear you were more successful than I was in your first venture!

Bob: In any case, it is likely that there is a universal interest in playing with stars on a computer. So let's put our toy model on the web, and see what reactions we get.

6.2 Free and Open

Alice: If we do that, I suggest that we make it freely available, without restrictions.

Bob: I agree. That seems the most natural thing to do. I have benefited so much from all kind of tools that are freely available on the web that it only seems fair to contribute my own tools in return. Besides, it offers a great form of quality control: when we write something which is unclear or has bugs, chances are that we will soon hear about it from others.

Alice: I feel the same. I have been using Unix as long as I have been working with computers, and I was delighted when the Linux operating system became available.

Bob: Same here. And without all its GNU tools, Linux wouldn't exist. Take

the gcc compiler for example. And I certainly wouldn't want to live without emacs.

Alice: I wonder how we should present our toy model. Call it 'open source' or 'free software' or something like that?

Bob: Both terms are in wide use, and there are huge political discussions about what exactly is and is not open or free in what way. I'm not sure whether we want to get into all that.

Alice: But if we don't write anything along those lines, there may be drawbacks as well. What if someone uses our integrator to design a bridge, and then the bridge falls down because there was a bug in our code, and we will be held liable for damages?

Bob: Are you serious?

Alice: Not really. I mean, us introducing a bug?

Bob: Very funny. But I guess in this world you never know who will knock on your cyber door and copy your software. Do you have a suggestion as to how to prevent liability?

Alice: The easiest way would be to use a type of open source license, preferably a rather simple one, which at least includes the usual disclaimers.

Bob: Perhaps it is time to have a look at the web, and do a search for 'open source.'

Alice: Here is something: "http://www.opensource.org" with a long list of open source licenses.

Bob: That was quick! Let's see what they say. Hmmmm. Most of them are far too long to read, let alone figure out, for my taste. Isn't there a simple one?

6.3 Open Source License

Alice: I remember someone mentioning the X window system as having a straightforward license. Here, that is probably the MIT license. It is short and sweet:

```
The MIT License

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any
person obtaining a copy of this software and associated
documentation files (the "Software"), to deal in the
Software without restriction, including without
limitation the rights to use, copy, modify, merge,
```

```
publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software
is furnished to do so, subject to the following
conditions:

The above copyright notice and this permission notice
shall be included in all copies or substantial portions
of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF
ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR
IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

Bob: Great! We have our disclaimers, at the end in capitals, and the first paragraph sounds reasonable too. But does it really cover enough ground? I don't mind if someone else would sell our software, unlikely as that may be, but I sure would be unhappy if they would not include a pointer back to our own web site, and if they would sell only the executables, while hiding the source code. The real source will be on our web site, but who would be able to find it? They would even be allowed to change the name of the package, so it may then become impossible for a user to find our web site. Hmm.

Alice: Yes, that wouldn't be much fun. Hmm indeed. But if the X window people have lived with this license for all that time, there is perhaps not too much danger involved.

Bob: Not if our toy model will become as famous as X, but I doubt that we will see the day of an N-body model for each man, woman and child.

Alice: Maybe the best solution for now is to start with the MIT license. I presume that we can always later replace it with a more elaborate and more restrictive version. What do you think?

Bob: Fine with me.

Chapter 7

Stellar Evolution and Hydrodynamics

7.1 A Minimal Vision

Alice: It's time to decide what to put into our toy model. It would be good to have a clear vision of what we want to accomplish, before we get started. At the same time, it should be a minimal vision, to keep the software simple enough to serve as a toy model. We can have some simple stellar dynamics as a backbone, but we should at least include some stellar evolution and hydrodynamics.

Bob: That's asking a lot. The simplest stellar evolution code will be huge, much larger than a small N-body program. For hydrodynamics we can use SPH, that won't be that large. But in both cases we'll have a lot of explaining to do, if we really want to describe to the students what is going on in those codes.

Alice: I'm not sure that a stellar evolution code would have to be that large. It may be possible to write a code that is not longer than the stellar dynamics code. However, we really don't have to be that fancy. All that we need to do is to implement the correct interfaces for a real evolution code and a real hydro code. Having done that, for our toy model we can then finish the job by adding something very simple at the other side of the interface. In other words, we can write a minimal skeleton version for stellar evolution; and similarly for stellar hydrodynamics.

Bob: A look-up table for stellar evolution, for a few simple tracks?

Alice: Even simpler. For example, we could let the evolution program determine everything just by giving it a mass. It will then compute the ZAMS radius, as well as the lifetime, by a simple power law formula. It can keep the radius constant for the first 90 percent of its lifetime, and then linearly increase it to a value that is 100 times larger during the giant phase. After that, we can let the

radius go to zero, since the actual radius of a white dwarf is a hundred times smaller than the original ZAMS radius. The mass we can reduce to that of a typical white dwarf, assuming that the remaining mass is lost in last stages of stellar evolution, through AGB and horizontal branch evolution. This is all at least qualitatively correct for relatively low-mass stars.

Bob: Simple indeed. Now what would be the equivalent for hydrodynamics? Obviously something simpler than SPH. Well, how about giving each star a radius, according to a recipe what you just outlined, and then stipulating that two stars stick and merge when they pass to within the sum of their radii?

Alice: Good idea. That is the right level of approximation: almost trivially simple, but still it will allow you to study the effect of collisions. And it can even produce run-away mergers of many stars if the stars are crowded together sufficiently.

Bob: Yes, especially if we let the radius grow with mass in an appropriate way. For starters, we could just take a radius that is simply proportional to mass, which is not a bad approximation for low mass stars anyway. We might even use this toy model to mimic the formation of intermediate mass black holes!

Alice: What is an intermediate mass black hole?

Bob: A black hole that is significantly heavier than stellar remnants, but still a lot lighter than the supermassive black holes in the nuclei of galaxies. There is some observational evidence, suggesting that these types of black holes may be present in young dense star clusters, and perhaps also in older clusters. One possible explanation involves repeated mergers of stars. If you start with a young and very dense star cluster, where collisions occur frequently in the core already within the first million years, it is possible to get a run-away merger going before even the most massive stars undergo a supernova, something which takes a few million years. The most massive stars are most likely to merge, since they provide more gravitational focusing, and the more massive a merger product becomes, the more likely it is that it will eat yet another star, more and more rapidly, as a snowball effect.

Alice: Eating snow? I'm not sure about your mixed metaphors, and it sounds like a rather speculative scenario. But you make an interesting point, that students will already be able to start playing with such scenarios right away, even with our simple toy model, right on their own laptops.

Bob: Good. So now we have specified the nongravitational astrophysics part of the code. That was easy! Of course we'll have to implement it, but these simple approximations shouldn't be too difficult.

Alice: Wait a minute, we haven't done our specifications yet.

7.2 A Top Down Approach

Bob: What more would you like to specify?

Alice: The main reason to write a toy model was to show the importance of modular programming, with interface specifications between each module, remember?

Bob: Sure I remember. But we have decided upon the stellar evolution and hydro approaches, and as soon as we decide how to do the stellar dynamics, we can start coding. It will become clear pretty quickly how to call the stellar evolution from the stellar dynamics module, for example. As soon as we see how that works, we make a precise list of all the parameters in the subroutine that calls the stellar evolution, and voila, there we have your formal specification. Wasn't that the idea?

Alice: No, no, no. That is a completely bottom-up approach. Yes, that is the way most programs are written in theoretical astrophysics. But no, that is not what is meant by an interface specification, which is a top-down approach. The main reason to have an interface specification is that the different parts of the program can be written by different people, in different styles, in different languages, and at different times. You cannot expect them to wait for each other, or to know the details of each other's codes. The whole point is that one module is like a complete black box, as seen from the other module. This can only be accomplished if you specify exactly what is being passed through the interface in what way.

Bob: That sounds unnatural, and quite a bit of overkill. But since you are so serious about all this, and it is only a toy model, I don't mind trying it. Once it works, I'd be more than happy to point out to you how inefficient such an approach will be. But first I have to get a better idea of what you have in mind. What you said so far sounds just too abstract.

Alice: Okay, let us do the exercise. First let us see what happens when two stars come close together. The stellar dynamics (SD) module should be careful to flag such an occurrence, since this is something that the stellar evolution (SE) module cannot know about. In our case, we have a stellar hydrodynamics (SH) module that will have the responsibility to follow up on a close encounter. So when the SD module notices a close encounter, it notifies the SH module. If the SH module decides to merge two stars, it has to tell this to both the SE and SD modules. The SE module then has to construct a new star for the merger product, and independently the SD module will replace the two previous particles with one new particle, with a mass equal to that of the merger mass: the sum of the previous two masses minus the mass that may be lost in the process.

Bob: I'm happy to leave out the hydro part of the story, for now, and just discuss the interface between SD and SE. How do you see this?

Alice: The first question is: who is knocking on the door from which side, in order to request data to be passed through the interface. There are several ways to decide this. In the SD case, we will have individual time steps, so different stars will be updated at different times. In the SE case, in general each star will have its own natural internal time step as well. Given all this diversity, it may be complicated to give either module the responsibility to trigger the other module to advance in time, at the right time. It may be better to introduce a separate scheduler module, that takes care of all the timing management.

So the scheduler will inform the SD module which stars have to be updated next. In this case, this will involve taking one integration time step forward. At the same time, the scheduler will inform the SE module which stars need to update their internal state.

Bob: I should have known it! In your able hands we have a quick proliferation of modules. I wonder how many more you will introduce. But let me just follow your logic. Right here I can immediately see all kind of exceptions to this nice little rule you are specifying here 'from above'. For example, as soon as the SD module gives the green light to one star to step forward, there is a chance that it will run into another star immediately, on a time scale that can be arbitrarily shorter than the time step you thought it would take. This means that all the other stars that happily stepped forward are now ahead of their time, and we should call them back to their previous position, to make sure that they were not affected by the collision.

Alice: That is true if your time step criterion does not include the possibility of a collision. Normally the time step in an N-body code is determined by a concern for the accuracy of the numerical orbit integrator, but it is perfectly possible to include an extra criterion, to predict when a collision will take place. But there are many other options. For example, you could decide to postpone the actual collision treatment until the next time step. After all, two stars will not suddenly jump on top of each other. They may approach each other in such a way that their mutual distance decreases at a rate of a few percent per time step. And given other approximations, you might decide that starting the collision treatment a few percent nearer or further does not make much of a difference.

Similarly, if the SE module would want to let a star explode into a supernova, affecting all stars nearby, and potentially all stars in the system, we could ask the module to postpone the explosion until the next time step. Or we could include a prior warning mechanism, letting the SE module tell the SD not to take a larger time step than warranted before the supernova explosion. Either choice would be fine, but that type of choice has to be made before the interface specification. Once you have made that decision, you cannot tinker and play with it as you like, on a daily basis.

Bob: What if you want to use one criterion under one set of circumstances, and another one in other situations?

Alice: If it really turns out to be important to do both, we can of course incorporate that choice within the interface, indicated by a special flag, say. But I would want to be careful before making such decisions. Before we know it, throwing in more and more options will make the interface as complicated as many of the codes currently in use. The idea is to use the interface specification to protect you from spaghetti, not to create more.

Bob: I still don't see clearly what you may want to pass and how.

7.3 Choosing Ingredients

Alice: Okay, let us be more precise. We can think of passing data across the interface, or of making data available at the interface. Notice how the first formulation implies an active version, in the form of a question and answer type of handshaking, and the second one is more passive, like looking something up in memory.

Clearly, a lot of thought, as well as experimentation, will have to go into the writing of a detailed interface specification. But for now, to give a more concrete idea, let us take the first model, in terms of handshaking. We also have to say which side initiates the handshake. Let us put that responsibility for now at the side of the stellar dynamics module. It can ask a particle to evolve its internal state, or tell it that it is no longer needed, in the case of a merger.

We can find out what type of instructions should be passed through the interface, by looking at the physical requirement of the system that we are modeling. Let us make a list of what the stellar dynamics module would like to have. It needs to tell a star to evolve itself for a certain amount of time; it needs to tell a star to destroy itself, in a collision; and it needs to ask a star for its mass and radius. That already gives us a list of four necessary functions:

- a star evolution function
- a star destruction function
- a function providing the mass of a star
- a function providing the radius of a star

Bob: So these will be subroutine calls from the SD module, which can be evaluated by the SE module, after both modules have been compiled and linked, correct? I like to translate your abstract reasoning in concrete terms that I am familiar with.

Alice: Yes, you can think of them as subroutines. You are not allowed to add more arguments, unless you start a new version of the software, and document how and why you made that particular change. And under no circumstances are common blocks allowed in the interface.

Bob: Your list is far from complete. Don't you need a star constructor as well?

Alice: Perhaps. I was thinking that you start the N-body calculations with as input initial conditions that come from a file, or directly from another program. Presumably that other program knows how to create stars.

Bob: But after a merger occurs, what do you do? You can destroy one of the two stars, changing the other star into the merger, but that would be rather arbitrary. Wouldn't it be better to destroy both of the pre-merger stars, and create a new star instead, for the merger product?

Alice: Yes, you are right, that is much cleaner. See, you are starting to think in a more modular way already!

Bob: Hmm, you still haven't defined exactly what modular means, but I won't push for yet more specifications. Coming back to the functions, we also need a way to pass time information. Even though your nifty scheduler module will give wake up calls, I presume that we need to have a local way within each module to keep track of time. And then there is a question of synchronizing that time, or at least to know whether the information you get across this interface barrier is up to date or slightly out of date, stemming from the last time step.

Alice: I agree. So we already get six functions then:

- a star creation function
- a star evolution function
- a star destruction function
- a function providing the mass of a star
- a function providing the radius of a star
- a function providing the current time for a star

Are you happy with this list?

Bob: From an astrophysical point of view, this is still not enough. The reason that mergers are interesting is that they evolve in a very different way from ZAMS stars. A merger product will be far from thermal equilibrium, it will most likely be in a state of high rotation, and there will be strong gradients in chemical composition. In order to convey all this information, your interface specification will have to include sets of tables and functions with many parameters!

Alice: For the time being, while we are developing our toy model, we do not have to take all those effects into account. After we reach a stable state, we can give our software a version number 1.0, say, we can then update the interface specification in order to include more physics.

Also, most of the complexities you talk about are a matter of information transfer between the SE and SH modules, and the SD module doesn't have to know those details.

Bob: Fine, as long as it is understood that we will have to widen up those interfaces significantly later on, I'm happy to stay with the toy interface. But even so, we need to convey at least some information about chemical composition, to show that mergers have higher helium content, and higher metallicity in general, compared to freshly born stars. We don't need to worry about too many details, perhaps but at the very least we need to communicate Y and Z, helium and metallicity abundance.

7.4 An Interface Specification

Alice: Fine! So we get a list of eight functions then:

- a star creation function
- a star evolution function
- a star destruction function
- a function providing the mass of a star
- a function providing the radius of a star
- a function providing the current time for a star
- a function providing the total helium fraction of a star
- a function providing the total metallicity of a star

And since you asked for more concreteness in the examples, we can write the specifications for these functions in Fortran. Similar specifications can be prescribed for other languages, such as C or C++, and in many cases the interface will be used to connect two modules that may be written, say, in Fortran and C++. However, it is convenient to choose one language in which to give the specification. The alternative would be to specify the data passing on the byte level.

Bob: And in that case you would have to worry about architecture dependencies, such as big-endian versus little-endian byte ordering in integers and floating point numbers. But there would be much more involved. The subroutine passing would have to be specified on the level of assembly language, and that would in turn be architecture dependent. So yes, it is almost unavoidable to pick a particular language. Which language shall we choose? Why do you prefer Fortran; wouldn't C be more natural, and closer to the machine architecture?

Alice: The choice of language deserves a separate discussion; let's do that in a moment, for our toy model. For now, just to give you a concrete example of an interface specification, let's take Fortran. Starting with the first function on our list, here is how we can create a star.

The function

```
integer function
    CreateStar(M, Y, Z)
```

accepts as arguments `real*8` variables for the initial mass, the helium abundance and metallicity of a star created at the zero age main sequence. The return value is a unique integer that acts as the identifier for the particular star that has been created. A negative return value will signal an error (e.g. not enough storage left; unreasonable initial conditions provided; or some other internal error in the stellar evolution module).

Note that we don't have to give the position and velocity of the new star to the stellar evolution module. That is none of its business; it is information that the stellar dynamics module keeps track of. Position and velocity do not even have any meaning within the stellar evolution part of the software.

Bob: We'd better specify the units for the variables as well. How about:

```
M in solar masses

Y helium abundance fraction by weight; 0 <= Y <= 1

Z metallicity abundance by weight; 0 <= Y + Z <= 1
```

Alice: Fine. Moving right along to the second function, we can write:

```
real*8 function
    EvolveStar(id, dtmax, dMmax,
               dRmax, dYmax, dZmax)
```

for a function that accepts as first argument an integer variable for the identifier `id`, followed by five `real*8` variables that determine halting criteria. The stellar evolution code will start evolving the star, from the current time `tnow`, and it will stop as soon as one of the following halting criteria is satisfied:

- if the time exceeds that of the specified period: `t >= tnow + dtmax`;
- if the mass `M` obeys `| M - Mnow |` $>$ `dMmax`;
- if the radius `R` obeys `| R - Rnow |` $>$ `dRmax`;
- if the helium fraction `Y` obeys `| Y - Ynow |` $>$ `dYmax`;
- if the metallicity `Z` obeys `| Z - Znow |` $>$ `dZmax`.

This function then returns the new time `tnew`; a negative value for `tnew` indicates an error condition.

Bob: And we need additional units, for example:

```
t in millions of years

R in solar radii
```

Okay, that all gives me a better idea of what you had in mind. No need right now to write down the other six functions.

Chapter 8

Stellar Dynamics

8.1 Regularization

Bob: As for the stellar dynamics, how fancy shall we make our code?

Alice: At least we should be able to handle core collapse for a small number of particles, say a few hundred or so, so that we can model a small cluster, and see the inner parts shrink gravothermally.

Bob: In that case, it would be nice to be able to deal with close binaries as well. Otherwise we have to stop the simulation before core collapse is completed. In addition, we may want to study the effect of primordial binaries.

Alice: But I would prefer not to get into regularization and all that. Whatever I have seen from special treatment of close encounters convinced me that that whole topic is rather specialized, and not something we want to throw at the students right away. Wouldn't it be sufficient to use only one integration scheme globally, for all particles, without making any local exceptions?

Bob: As for the global dynamics, we are pretty much forced to introduce an individual time step scheme. That will require quite a bit of explaining to the students, since it goes way beyond what they can find in the standard text books on numerically solving differential equations, but we have little choice. If all particles would share the same time step size, the whole system would get down on its knees when somewhere two stars would come close together, or worse, would form a tight binary.

However, even with individual time steps, almost all of the computer time for a few hundred body system could go into just one tight binary. And what is worse, the integration of such a binary could lead to an unacceptable build-up of roundoff errors.

Here is an example. It can easily happen that two stars form a hard binary, with

a semi-major axis that is a thousand times smaller than the size of the system. The eccentricity e will fluctuate, due to the perturbing forces from encounters with other stars, and occasionally, e can get close to unity. If, say, $e = 0.999$, then at the distance of closest approach the stars will be a million times closer than the size of the system. In order to calculate the force between the stars, we have to first subtract the position vectors of both stars, and this will lead to a loss of precision of at least six digits, reducing an original 64-bit double precision calculation almost to 32-bit single precision. We simply cannot afford that, if we want to integrate the system in an accurate way.

One way to overcome this problem is to introduce a special coordinate system. Expressed in relative coordinates, with respect to the center of mass of the two stars, there is no longer any problem of round-off. An even better way is to go one step further: to replace the orbit of a very tight binary star by the analytic solution of the Kepler problem, an elliptical orbit for which we only have to solve Kepler's equation to know where the two particles are with respect to each other, at any given time. This will be an excellent approximation when there are no other stars anywhere in the area.

Now sooner or later, a third star may happen to come near. In that case we can temporarily switch back to numerical integration. Or, if the binary has a very eccentric orbit, we may want to avoid significant numerical errors by introducing a celestial mechanics perturbation treatment. In that case, we stick to an analytical Kepler orbit as the unperturbed solution, and we integrate the perturbation equations. But in general, the best solution is to use direct numerical integration in four dimenstions, using what is called a Kustaanheimo-Stiefel transformation, or other variants that have been introduced later. These of course are only the first steps. There are many other very nice tricks of the trade. If you use chain regularization and Stumpf . . .

Alice: Yes, yes, I'm sure there are many wonderful tricks, but we have to offer your students something that they can understand and apply, all within half a year. Whoever Stumpf is, or the other two gentlemen you mentioned, let's leave them out for now.

Bob: But we should do something, on the local level, to make the whole calculation meaningful. The question is what we can introduce that goes partway toward full regularization. One thing we cannot get around is to do the analytic regularization that I just mentioned. Perhaps that is good enough to get them started. It will actually be instructive for the students to see how things fail, and then to see what solutions can be found to repair the situation. The only way to do that, I think, would be to introduce the separate coordinate patches I talked about. While that will rapidly get too complicated, it would be fun, and it would give students a more realistic sense of the complexities that a real code has to deal with. Who knows, when we really get going, we might even want to give that a try.

Alice: Aha, do I detect the possibility of a smooth evolution from a toy model

to an actual production code?

Bob: No no, that would be far more complex. But it would be a step in the right direction, for sure.

Alice: If the main problem would be a loss of accuracy, how about use multiple precision? Instead of 64-bit word length for floating point numbers, using 128-bit word length?

Bob: You mean quadruple precision? Yes, that would be an option, but it will slow you down. Depending on the machine you use, it could slow you down anywhere from a factor of a few to a factor of a hundred or so, if you can do it at all: not every compiler has a quadruple precision option, and you don't want to write all the routines by hand for computing multiplication and division and square roots and all that.

In any case, yes, there are plenty of options. Right now, I wouldn't be able to guess what would and would not work well, under which conditions. So I'll be learning quite a bit from this exercise, I think, even though it's only a toy model. The more I think about it, the more it seems like a fun project.

8.2 Local versus Global

Alice: If we really cannot avoid introducing some local treatment in the stellar dynamics part of the code, it would be best to split the stellar dynamics itself into two modules, and a clear interface.

Bob: I told you, that you would want to introduce more modules, before long! How many do we have now? A scheduler (SC), a stellar evolution (SE) and a hydro (SH) module, and now a global stellar dynamics (GD) and a local stellar dynamics (LD) module, five in total!

Alice: Nothing wrong with having five modules in a code. This means that you have a lot of freedom! You can replace each module, rewrite it, experiment with it, all without affecting anything in any of the other modules. But let's not repeat our arguments. After we finish our first toy code, we can take up this discussion again, and by that time we'll have a lot of actual code to work with, to strengthen our arguments.

Bob: Fair enough. But now I'm puzzled. Passing information between the SD and SE module was natural enough, since we were dealing with two rather different types of physics. But the GD and LD modules both are dealing with the same physics, plain old Newtonian gravitational attraction. Putting an interface between them would seem like trying to draw a line on the water!

Alice: I want to draw a line in the sand, frankly, and insist that we divide the stellar dynamics into these two different modules. Look at it this way. From the point of view of the global dynamics, the local dynamics is a way to protect ourselves from the Kepler singularities, right? I presume that is the meaning of

the word regularization, removing the singularities.

Bob: Yes, I think that is where the term comes from.

Alice: In that case, the SH module is also a form of regularization. When two point particles come close, the SH module will replace them by, for example, a blob of SPH particles. We could call this a form of hydro regularization. It looks a bit like softening, where every point particle is replaced by a small Plummer model. Softening could also be called regularization.

Strictly speaking, the mathematical term regularization implies that you do not change the underlying equations of motion, so in that sense hydro treatment and softening should not be called regularization. But we are not mathematicians. In astrophysics, we start with extended objects. The idealization of replacing a star by a point mass is only a matter of convenience. And when this replacement turns out to be inconvenient, in close approaches between stars, we are free to use other idealizations. We are only 'regularizing' what had been 'singularized' as a too extreme idealization.

Bob: But you can't simulate a star cluster with a code that is only using softening. The code will run, but it will not give you a sufficient amount of two-body relaxation. And it will not admit hard binaries.

Alice: That is true if you use a large softening length, like people do when they collide two galaxies, and they want to suppress two-body relaxation, which would be unphysical anyway in that case, since each single particle represents thousands of stars, if not more. But in the case of a star cluster, you could give each star a softening length that is equal to the physical radius of a star. In that case, binaries could form and two-body relaxation will be just fine. You would have to do more, since you would have to prevent stars from passing through each other, as softened particles would. When you thus add a hard core and a friction term to let the particles stick together upon close approach, you have gone well above the simple softening recipe. But if you add those other prescriptions, using a small softening in itself is not necessarily wrong.

Bob: So you view hydro treatment and regularization as somewhat similar, from the point of view of the GD module, which only deals with point particles?

Alice: Yes. And the interface between GD and LD can be similar to what we saw before. We can specify a function to create a local clump of particles, and similarly we can specify a clump destructor function. We can have functions providing the mass and effective radius of a clump, as a function providing the current time. It all follows rather closely to what we just discussed for stellar evolution.

Bob: Hmmm. I still think it is like drawing a line in the water. But early on I agreed to do the experiment, to write a modular toy model, and if this is what you want, this is what we'll try to do. But allow me to laugh loudly if your approach will result in an unwieldy code, with many extra lines to circumvent your interface restrictions, and a very slooooow overall execution speed. And I

won't hesitate to tell my students whose brilliant idea it was to do all this ;¿).

Alice: If that is what will happen, you can use it as a case study in how not to follow the advice of computer scientists who tell you to use modular programming. I'm convinced that this will not happen, but there is only one way to find out: we will continue, and see who is right.

Bob: I can't wait!

8.3 The Name of the Game

Alice: I can't wait either, and whatever the outcome will be, it will be a useful experiment. Before we get started, though, there is one more thing we have to decide. As long as we put all this wonderful stuff soon-to-come on the internet, it would be good to give it a name. I wouldn't want to refer to it as 'our kitchen sink toy model' all the time.

Bob: A name. Hmm. Many options. We could use an acronym. Fortran comes from formula translator. How about Dentran, to translate your modular ideas for modeling dense stellar systems?

Alice: That sounds like something to do with dentists, perhaps a new type of tooth paste. How about using a Greek term? The word astrophysics comes from Greek after all.

Bob: I'm not very strong on languages. Do you have a specific suggestion?

Alice: A while ago I asked a Greek astronomer what the word for star cluster research would be. You know, geology is the study of the Earth, astronomy is the study of the stars, so what kind of something-logy or something-nomy would be the study of star clusters? After some thought, he suggested 'smenology'. I believe that 'smenos' means something like a swarm, like a swarm of bees.

Bob: I am often enough stung by bugs in N-body codes, so that would be quite appropriate, I'm afraid, although when I hear the word bug, I think of small little critters, not as big or ferocious as bees. But how do you turn that into a name?

Alice: We could call it the Smenology Code?

Bob: Too long, too difficult to pronounce, and besides, the logic doesn't work. You wouldn't want to call a code the Astrophysics Code, would you – as if there would be only one such thing.

Alice: And no comment. Okay, well, what else. You want a short name, I take it?

Bob: The shorter the better.

Alice: The A code?

Bob: Unless you will do all the writing, it should at least be the AB code,

for both of our initials. But seriously, it doesn't have to be quite that short. Hey, you like languages; it doesn't have to come from Greek – why not pick something from Chinese or Sanskrit or some other side street.

Alice: Sanskrit, now that's an idea. How about Kali? That means 'dark' in Sanskrit. As in the 'kali yuga', the dark ages we are currently in according to Hindu mythology. Or as in Kali, the Goddess who is depicted as black.

Bob: As least the name is conveniently short. And since the universe is by and large a rather dark place, the name is not inappropriate. We probably should include the option of modeling black holes too, with this name. Do you know the Sanskrit for 'hole'?

Alice: Beats me. But then again, we wouldn't want to only model holes. And yes, it would be nice if our code would be so robust, simple as it may otherwise be, that it could handle mass ratios of one to a billion, like in a brown dwarf circling around one of the most supermassive black holes in the nucleus of a central galaxy in a rich cluster of galaxies.

Bob: Okay, the Kali code it will be.

8.4 An Integration Scheme

Alice: So we have a name for our toy model, and we have decided how we will model the stellar dynamics, the stellar evolution and the stellar hydrodynamics: as simple as possible, but not simpler. Shall we get started?

Bob: Just a moment, we haven't yet decided what to do with the stellar dynamics.

Alice: You suggested an individual time step scheme, and I think that is fine. Actually, we may want to start with shared time steps, but that is a matter of presentation.

Bob: Yes, but the choice of time step is only one choice we have to make. We haven't chosen an integration scheme yet. If you want to start with shared time steps, you can do that using a leapfrog integrator, or more reasonably a fourth-order integrator of one type or another.

Alice: Ah, of course. And yes, plenty of choices. Well, to make it really accessible for students, with no prior background in numerical methods, we should really start with a first-order scheme such as forward-Euler, shouldn't we?

Bob: Really, not with the Hermite scheme?

Alice: What is a Hermite scheme?

Bob: This is the current workhorse of N-body methods in which close encounters can occur.

Alice: I admit that it is a long time ago since I looked at such an N-body code. My recollection was that a rather complicated predictor-corrector method was used, where the force calculation from various previous time steps was remembered. The book keeping was very complicated.

Bob: That must have been a long time ago indeed! For the first thirty years or so, this was indeed the method of choice, until in the late eighties Makino came up with a simpler self-starting scheme, which he called the Hermite integrator, since it uses some ideas put forward first by mister Hermite, a couple centuries ago, I think. This is what is now being used almost exclusively in stellar dynamics of dense stellar systems. And it's not that difficult to code. You can write it in such a way that it looks like an almost obvious generalization of a leapfrog integrator, apart from a few coefficients which would be hard to guess off-hand.

Alice: In that case, it would be good to incorporate that scheme in Kali code. Even so, I like to start simple. I don't want to scare the students by presenting complex integration schemes until they have gotten a sense of what the notion of integration means.

Bob: Wouldn't that be too much hand-holding? Why not let them jump in right away?

Alice: The drawback of giving them a fancy working tool right away would be that they would immediately run with it and move on to fun applications. Chances are they would never bother to think about how and why it can work in the first place. These days, students are all too much focused on learning how to use a package, rather than writing it and getting a feel for what is under the hood.

Bob: I'm afraid you're right. When I did my first programming, it was in basic, and soon afterward in C. Nowadays, when students talk about programming, they mean tweaking the parameters of a package, while using GUIs.

Alice: Perhaps we begin to sound like old folks, shaking our heads about the behavior of youngsters nowadays. But in any case, I really think it is best to start with the simplest possible integration scheme, forward Euler. That way they can get a feel of what it means to follow the tangent of a curve in literally the most straightforward way.

Bob: A clumsy way to start, for sure, but at least they will then learn to appreciate the power of higher-order integration schemes. Okay, let's do that.

Chapter 9

Environment

9.1 A Toolbox

Alice: Now that we know what we want to model, we have to address the question of the environment in which we want to work.

Bob: You mean what type of computer hardware, or operating system, or computer language?

Alice: All of these questions need to be addressed, and they are general concerns for any software project. However, in our particular case there is also the question of how to get data in and out of our integrator, and how to analyze the results.

Bob: Of course, you have to produce some initial conditions, and you want to look at the results, but basically we are dealing with one big program, and a bunch of small ad hoc programs, which could be some shell scripts or whatever you can put together quickly. The main point is to provide the students with a robust integrator. The rest they can take care of themselves.

Alice: But what about standard questions, such as deriving the potential energy for each particle, and the local density around each particle. You almost always want to know those quantities. Wouldn't it be better to build separate programs to compute those?

Bob: It would be much better to make that part of the integrator. The potential will already be calculated at regular intervals anyway, to provide accuracy diagnostics. And it wouldn't be too difficult to built in a density estimator as well.

Alice: Combining everything into one big program has been the traditional approach for a long time, but it is far from optimal. What I would prefer, certainly for a student demonstration project, is to provide something that is

more like a toolbox. I'm happy to let the calculation of quantities such as potential energy be done in the integrator, as part of the necessary on-the-fly analysis. But in addition, I would want to have a separate stand-alone program that can compute the potential as well.

Bob: Why?

Alice: Imagine that you want to calculate the total mass of a bound subsystem within a larger N-body system. You take the whole system, and remove the particles that our not bound to the subsystem. Perhaps you want to iterate a few times, removing more particles until you really have a self-gravitating subsystem. At that point you may want to check virial equilibrium, so you have to calculate the kinetic and potential energy of all particles.

Bob: That makes sense. And come to think of it, when you then want to calculate Lagrangian radii, it may be nice for the students to have yet another little program to do just that. Ah, we can follow the way of the Unix system: a large number of small tools, where you can pipe the results from one tool into the next one.

Alice: That would be one example of what I had in mind, but it is not the only one. There are other ways of combining a number of small tools. However, this type of decision we can make later. The main thing now is to agree upon the general approach: to provide a number of simple tools that can be used together to analyze the results of a calculation.

Bob: Fair enough. But you started out talking about choices we have to make with respect to the software environment. I thought you would talk about operating systems and languages and stuff like that. In short: what type of platform do we run our toy model integrator and toolbox on?

9.2 Operating System

Alice: Given that we are both used to working in Unix, there is not much of a question, as for as the operating system goes. Most of my work I do on a laptop, running Linux.

Bob: me too, but that is not necessarily true for most of the students. If we do not make our environment available for Windows and for the Mac, we may not have much of an audience.

Alice: I have no experience at all, porting software to a different platform.

Bob: I have some. For example, putting stuff on a Mac is not that hard, after they switched to Unix as the underlying environment. As for windows, a lot of the GNU tools have been ported there as well. If you use cygwin, basically any program written for Unix is likely to work.

Alice: Can you just type gcc and expect to find a GNU compiler, under cygwin?

Bob: Yes, you can.

Alice: Amazing. And how about emacs?

Bob: Yes, emacs is there, as well as all the standard Unix commands and programs.

Alice: Does that mean that when we develop our toy model under Linux, we can make a tar ball, copy that to a machine running Windows and cygwin, untar it, and expect everything to work right away?

Bob: Well, you can be lucky, but realistically speaking, you'll have to test it first, and you may well find some minor problems here and there. But of course, if you would switch from one Unix system to another, you would also have to make minor adjustments.

Alice: I know. Even switching from one Linux distribution to another has given me plenty of surprises in the past. Often it was just a question of tools being stored in different places.

Bob: Yes, but that's no big deal, and easy to fix.

Alice: For you perhaps, but if I had to do that myself I might be stuck for hours, without someone more experienced to help me. Well, I'm glad we're doing this project together. I'll count on you to do what you call the easy things.

Bob: We aim to please!

9.3 Choosing a Language

Bob: Finally we will have to make the decision which programming language to use. I have no strong preference among Fortran, C, or C++. They all have their strengths and weaknesses. C is nice and fast and straightforward. Fortran has by now caught up with C quite well, after lagging behind for a long time in a lack of appropriate data structures. C++ is an unwieldy beast of a language, but if you use an appropriate subset, and you don't try to become a language lawyer dealing with multiple inheritance and all that stuff, it's a good tool; it's fast enough, and you don't look at the unreadable stuff the compiler produces. Your choice: which of the three?

Alice: Perhaps none of the above?

Bob: Huh? Do you want to use Cobol? Or Lisp? Too late for Pascal, I'm afraid.

Alice: Frankly, I'd love to use Lisp, or the dialect Scheme to be more precise, but don't worry; I know that most non-Lispers have an aversion to too many parentheses. While I love the language, I don't have the illusion that I can single-handedly convert a critical mass fraction of the astrophysics community to start writing and thinking in terms of lambda calculus.

Bob: What calculus?

Alice: Forget I said that. No, I'm not thinking about Lisp or other more traditional languages. Rather, I have been considering using a scripting language.

Bob: You mean Perl?

Alice: Perl is one example of a scripting language, but there are others.

Bob: I've heard many good things about Perl. I've never used it myself, but I, too, have been thinking more and more about learning it. There are just too many times that I feel hampered by the limitations of C shell scripts, or whatever similar shells you can use in Unix. A little awking and grepping will help you in making C shell script more powerful, to analyze the output from N-body codes, for example. But still, I could easily be convinced to use Perl instead. I know several colleagues who swear by it.

Alice: I wasn't thinking about Perl. Nor Python.

Bob: Ah yes, Python, I couldn't remember the name, just now. What's the difference with Perl?

Alice: It's object oriented, like C++ but less unwieldy.

Bob: Hmm, that I like. I've grown quite fond of using classes in C++, for the types of problems where I know what I want to do. It would be nice to have a more playful language in which to prototype classes, and throw them away if you don't like them, without having to redeclare everything all over the place to make the gods of the C++ compiler happy.

Alice: Exactly, that was my thought. So given the two, I would prefer Python. But in fact, there is another choice, even better in my eyes. It is called Ruby.

Bob: Never heard of it. What's the difference with Perl and Python?

Alice: Since it was developed after Perl and Python had already come into being, the author of the language, Yukihiro Matsumoto, had the advantage of being able to learn from his experience with those other two languages. It is in fact more fully object oriented than Python. It has iterators and all kind of neat stuff. And very importantly, it is simple and natural. It seems to be built on what is called the principle of least surprise: immediately after you get a quick and rough familiarity with Ruby, you will be able to guess in most cases what a given expression will do. That sounds like a strong claim, but I know several computer science friends who have told me that this is really the case.

Bob: Well, since I have never yet learned any scripting language, I don't care which one I will learn. You choose.

Alice: Same for me, it will be my first one too. Okay, let's take Ruby. It is freely available on the web, all fully open source, and there are already a number of good books available. The first book to introduce it in the English language is in fact fully available on the web, so we can get started immediately.

Bob: Okay, Ruby it will be.

Alice: I'll have to warn you, though: a scripting language is slower than a compiled language. Ruby especially: besides being an interpreted language, almost everything goes through two layers of indirection, which makes for a marvelous flexibility, but a large penalty in performance.

Bob: There you go again, choosing flexibility over performance. But this time I don't care, as long as we have agreed to write only a toy model. My students will be slower in understanding stellar dynamics than the computer will be in running their programs, even if they are ten times slower.

Alice: Hmm, would you settle for a hundred times slower?

Bob: What?!? Surely you're joking!

Alice: I'm not joking. But the good news is: Ruby defines a way to interface with C code. So you can write the most time-critical part of a code directly in C, if you want to make your code run faster.

Bob: That sounds much better already. But as long as we want students to study the cold collapse of, say, a 25-body system, we can even live with a factor of 100. What do I care? People ran 25-body systems back in the sixties, when computers were much more than a million times slower than they are now, so we can loose a couple zeroes in factors of speed. And it will prevent you from ever claiming that your toy model has much to do with the reality of production runs!

Alice: We'll see about that.

9.4 Graphics

Bob: One more thing to decide, that is just as essential as choosing an operating system and a programming language. We have to pick a plotting package. The students must have a way to display the results of their calculations on the screen, and also a good way to print the pictures they produce.

Alice: I am using pgplot for my own work. That seems to be doing all that I want, normally.

Bob: Same here. I have some familiarity with Matlab, as well as a number of other packages, but none of them are open source. And since we decided to go the route of free software, we cannot rely on proprietary packages. So pgplot would be a reasonable choice. However, I'm not hundred percent sure that it is open source.

Alice: No? I was convinced it was open source. I've never paid for it, or signed any license.

Bob: I know. But when I checked on their web site, it was not really clear to me what their status was. Yes, they are open source in the sense that their source is open: you have free access to all the source code. However, in their

case this does not automatically imply that you can distribute it freely to third parties.

Alice: That would be inconvenient. It would be far easier if we can bundle our own environment together with the libraries that we invoke. And graphics will indeed form an essential part of our whole package. Are you sure we can't make it available? If we and everybody else can copy it freely from the web, what difference would it make if we put up our own copy?

Bob: That's the point: not everybody can pick it up freely. If you read the fine print, you will see that it is only really free for educational purposes.

Alice: That may be good enough for us, since we are only dealing with students.

Bob: For our classes, yes, we can make pgplot available. But it seems that we cannot make it available on the web, since anybody can access our web site, also people who are not officially within the educational sphere.

Alice: In that case, we can put a pointer on our web site to the place where one can pick up the source code. That's not as convenient as providing everything in a bundled way, though.

Bob: I know. They have a line on their web site, saying that you can only copy their software to your web site after getting official permission from them. Right now may be premature, but once we have a reasonable package installed on our web site, we may want to send them an email, asking for permission to put a copy of pgplot directly on our web site.

Alice: That seems like a good idea. Are there any alternatives?

Bob: The only thing I can think of is Gnuplot, but I have found pgplot to be far more convenient for scientific plotting.

Alice: I have tried Gnuplot too, and I agree. So let's at least get started with pgplot. But here is an idea. Since we may want to switch to a different graphics package in the future, it would be very unpleasant if we then had to translate all pgplot commands into the equivalent commands in that other package. We could instead define our own virtual plotting commands, and write a simple wrapper to translate them into a subset of pgplot commands that is large enough for our purposes.

Bob: Well, I don't like all those extra layers of code, and I'm afraid I've already given you too much leeway with your interface demands. Why not just write everything in pgplot for now. If and when we switch to a different plotting package, only then do we write a wrapper that emulates pgplot.

Alice: But it might be a lot of work to write a wrapper around a whole package. If we decide right from the beginning to use only a small subset of pgplot, and if we keep a list of the commands we use, we only have to write a wrapper around those commands, not around the whole package. Finally, the best way to insure that we really don't use anything else but our small list of commands is to give each command we use an alias. That way we cannot make any mistakes.

Bob: If a very good function is available in pgplot, why not use it?

Alice: Oh, it would be fine to add extra functions to our list of what we use. We can certainly extend the subset, whenever we like. My main point is that we keep close tabs on exactly what we use, and that we think carefully, before adding something to the subset.

Bob: The problem with providing a virtual layer between the user and pgplot is that we have to make a decision of what type of plotting model we want to use. Even if we restrict ourselves to two dimensional plots, there are several models. For example, in postscript you deal directly with a set of coordinates on paper, in terms of inches. And other graphics packages provide transformations between model coordinates, often called world coordinates, and device coordinates. So we are dealing with a situation that is much more complex than just providing a set of aliases.

Alice: I see. Well, perhaps you are right. Maybe it would be okay to go ahead with pgplot for now. But let us come back to this question before too long, after we have build a skeleton version of our toy model. By that time, we might have a better idea of how fancy our requirements will turn out to be.

Bob: I'm glad I could finally convince you not to add an extra layer of software somewhere. So now we can finally get started?

Chapter 10

Style

10.1 Documentation

Alice: Not quite. We have talked about writing programs and providing an environment, but we haven't said anything yet about documentation.

Bob: Well, my class notes will be the documentation, I thought. I will give them enough hints to get them started, and when they go through the code, it will become obvious what we intended. Of course, we'll put in enough comments to clarify our methods.

Alice: I would prefer a rather different approach. Comments in a code are only the first level of documentation. A good code needs to have manual pages, at the very least, and even better a form of introduction, a primer or that sort of thing. When you want to work with a new program, don't you look at the manual pages?

Bob: Not often. I prefer to just try things out. And if it is an important enough program, I am happy to go look at the source code directly. And while I agree that good manual pages can be of some help for a commercial product, I don't see why we have to be so fancy for an educational project.

Alice: There are several reasons why. For one thing, I like to make it attractive and easy for others to use my code. Once I spend a lot of time writing a good piece of code, I'm happy to spend a bit more time to polish it and make it available in a more user friendly way. And if more people will be using it, I will get more feedback which in turn can improve the product. Also, others are more likely to write extensions or complementary programs, and there is a better chance that a project will take off around what I initially wrote, which can then serve as the nucleus for something larger.

Bob: If you insist on more documentation than code comments, a good addition would be to include a help mechanism in each program. Typing `density`

`--help` could give you some help about the various command line arguments available for the program `density`, for example.

Alice: Yes, that would be a good addition, but it does not address the issue of a really comprehensive explanation of not only what a code does, but why it was written that way: the motivation for the overall approach and the choice of algorithm used, as well as the particular organization of the program itself.

You see, the most important reason to write good documentation is long-term efficiency. If you force yourself to reason out loud why you are doing what you just did, you will quickly find that there are many aspects you haven't thought through deeply enough. While writing, you will get new insights that will improve your program significantly. So writing detailed documentation is not only a good idea, it is especially a good idea of doing it right away, while you are writing your code. If you are doing it later, you may also discover better alternatives, but you are unlikely to implement those, because by then you have already invested so much time into the older approach.

Bob: that all sounds nice on paper, but in practice, when you deal with any complicated program, there always comes a time that you have to beat the thing into shape. You can start in a nice clean way, following this paradigm or that, but then when the program gets going, you find that this is missing or that, and that such and such is slowing down the execution much too much. You start throwing in this and that, and taking shortcuts here and there to get significant speedup. How are you going to document that? Spend a few hours every day rewriting the documentation, each time you have tinkered with the code? That seems hardly efficient!

Alice: What you just described is a sledge hammer approach. If you have only two weeks left before a conference where you have to present your latest results, I can imagine that you take that approach. But unless there is a serious emergency, I cannot see how that will gain you in the long run. But I don't think we can convince each other by arguing. We clearly have very different styles. Let us agree that we at least will be careful to write clear comments and a good help mechanism for each code. For myself I will keep track of what major and minor decisions we make while working on our project, so that I can document what happened, and why, and how we learned from initially wrong assumptions. We can later see what we do with that material.

Bob: I'm glad we agree to disagree. It's time to get our hands dirty doing some real programming. Hey, I see that you have a copy of Knuth's book 'The Art of Programming'. He is one of my heroes. Those books are great. And without TeX and LaTeX, where would we be in writing articles? That he singlehandedly invented TeX in the late seventies was very impressive. I bet he didn't let himself be side tracked by documentation.

10.2 Literate Programming and Coherent Programming

Alice: Wrong! Knuth was very keen on documentation. Have you heard of Literate Programming?

Bob: Literary Programming??

Alice: No, Literate Programming. This is a term that Knuth invented, for the process in which he wrote TeX.

Bob: Process? I presume he just wrote TeX, just like you write any programs. He just happens to be a lot better in programming than most of us. What do you mean with process?

Alice: He did not 'just write programs.' On the contrary, he put a lot of thought into the process of designing programs, coming up with a view first, then with a way to map out the various pieces needed, and so on. And most importantly, he came up with the idea that computer code should first of all be human readable, and only machine readable as an afterthought. You should read one of his bundles of collected essays on algorithms and programming. Since he is one of your heroes, perhaps he will convince you more than I can, about the value of documentation.

Bob: You haven't told me yet what he meant with Literature Programming.

Alice: *Literate* Programming. The idea is that code and documentation are interwoven in one book. You write a piece of code, and at the same time you write a page of text explaining your motivation for writing that piece of code, what it is supposed to do, what were the reasons going into the decisions of writing this part in this way and that part in that way. So the code parts and text parts are literally interwoven in one long manuscript.

Bob: But how can you run the code if it is spread through the text of a manuscript?

Alice: He wrote special programs to extract both the book part and the code part of what he wrote. He called those `weave` and `tangle`. The program `weave` went through the original, and produced the book text. The program `tangle` also went through the same original, extracting the bits and pieces of code, putting it all together as one large code, the source code for TeX, that then could be compiled in order to produce the TeX executable.

Bob: Why did he call the second one `tangle`?

Alice: He literally wrote `tangle` in such a way as to entangle the source code, in such a way that it was no longer human readable.

Bob: Why would he do that??

Alice: He wanted to force himself and others not to make quick changes in the source code alone, without touching the book. For him documentation was so

essential that he forced himself to change the code only within the context of the original manuscript, where code and text were living together. In that way, the threshold was lowered to explaining in the text part what you just changed in the source code part, and thus the two could grow together harmoniously.

So this answers directly your objection that when you want to get something done, you reach a stage where you drop documentation and just beat things into shape. It seems that Knuth more or less tied his hand on his back, to prevent himself for using a sledge hammer approach – or so it must have seemed to many people. But I can easily believe that in the long run, or even in the medium run, his approach was more efficient.

Bob: I did not know that. I'm really surprised. And now that you caught me with a story about Knuth that I didn't know about, are you going to force me to use literate programming?

Alice: No, don't worry. I don't think that literate programming is still the right answer, in this day and age. It was a good thing at the time, a visionary move really. But it was developed in the early days of line editors, even before screen editors came into being, let alone window systems. Nowadays it is easy to keep an eye on several files at once, but in those days the best you could hope for is to see a few dozen lines on a screen at the same time. And therefore the only way to achieve a close coupling between text and code was to literally weave them together.

Bob: What would be a more modern equivalent?

Alice: I have been thinking about that, off and on. I think that the most important aspect of Knuth's idea, an aspect that will survive his particular implementation, was the notion of coherence. With his trick, he could keep source code and documentation text coherent. Everything was updated in step, and he leaned way over backward to guarantee that the two remained in lockstep. That part I would like to preserve, the notion of coherence. My proposal is to introduce what I would call *coherent programming*.

The idea would be to allow different files for source code and for documentation. However, the two are linked through frequent pointers, in a hypertext kind of way, perhaps as it is done on web pages. I must say, I do not yet have very clear notions of how to implement this, and what to choose, but the main two points will be to, a), make it very natural to update code and text in unison, and b), provide some sort of penalty against violating that simultaneous type of update. I'm not sure whether I would go to the extreme that Knuth went, of making his source code unreadable, even to himself, but perhaps even that might not be a bad idea. I'll let you know if and when I get more inspiration in that direction.

Bob: I'm glad you let me off the hook so easily, without converting me into a coherent programmer! We can talk about those ideas later, over a drink, whenever you like, and yes, I'll probably have a look at those essays by Knuth. I must say that you have peaked my curiosity. But for now, let's get started

doing some programming ourselves!

10.3 A Lab Note Mechanism

Alice: One more thing.

Bob: I was afraid you would say that. What is it now? Don't we have all the pieces we need to get started, and more?

Alice: This will be the last ingredient, I promise. I won't insist on documentation, literate or coherent or otherwise. But what I do insist on is a systematic way of keeping lab notes.

Bob: Lab notes? For which lab? What notes?

Alice: Our toy model and its environment will form a kind of lab for your students. They can perform experiments, by setting up initial conditions, running a simulation, and analyzing the results. It is all virtual of course, but apart from that, the procedures and the skills needed are not that different from doing a lab experiment.

Bob: I remember burning my fingers while learning how to blow glass during my freshman physics class. After glass stops to glow, it is still incredibly hot. I sure wished that I had infrared eyesight then. But okay, if you want to stretch metaphors, I've burned myself with hasty programming as well. And there are parallels between lab work and working with software. Or in astronomical terms, between working with a simulation or with a telescope. If you walk into an astronomer's office, and see a pretty picture on his or her screen, you have no way of knowing, *a priori*, whether you're looking at an observation or a simulation. So I grant you that a lab is not such a bad metaphor, after all. But what about your notes?

Alice: I suggest that each time we get together to work on our toy model software, we keep some notes about the main decisions we make, and the reasons behind them. That will not take much time, and it will help us writing documentation later on. What is more, it will enable us to go back and trace down why we did what we did, when we come back to a piece of code a few weeks later, trying to debug it, and wondering why we ever wrote this or that.

Bob: Now that sounds easy. We can just scribble down some notes while we go along. Probably better to write that in a file somewhere, rather than on scraps of paper.

Alice: Ideally, we should do better than that. As part of a coherent programming approach, it would be good to combine computer code AND documentation text AND lab notes, all in one environment, with many pointers from each of the two to the third element in the triangle. But again, I won't try to convince you. All I'm asking for is that we both keep notes, together when we are working jointly, and individually when one of us extends something in our

toy model environment.

Bob: I have no trouble with that. How shall we send each other the notes we write down? In the form of emails?

Alice: The best way would be to use a form of source code control. The standard package to do this would be CVS, but an even better alternative would be to use subversion.

Bob: Yeah, I've heard about that approach, from people who use it when they work with big teams on forever-running projects. But for the two of us, for a toy project? You accused me of a sledgehammer approach, but I'm afraid you've just taken up quite a sledgehammer yourself!

Alice: I bet you will like it, once you get used to it. I started to use it a few years ago, and now I use it even for things I'm working on myself, like when I write a lengthy review article, and I want to make sure I have all the information at hand, from previous revisions of chapters. But not to worry. It's easy to set up a source code control system, so I'll do it for us. I'll show you a few commands you can use to commit changes from your side, and to update your side, to get the changes I have made. These two commands are the most important, in fact. We should put everything under this source code control system: the codes we write, whatever documentation I will add, and the lab notes we will both keep writing regularly.

Bob: Is this really your last requirement, as you have promised, before we can start to do some *real* programming?

Alice: Yes, this is it! We can get started now.

Bob: I thought I never would see the day. Well, as long as you take care of this code control business – what did you call it?

Alice: Source code control.

Bob: As long as you take care of it, and it really is as simple as you said, I'm willing to try it. Anything to get to the point of starting our programming!

Chapter 11

Literature References

Two specific recent references, close to the material presented in the *Kali* series, are:

Gravitational N-Body Simulations, by Sverre Aarseth, 2003 [Cambridge University Press]

The Gravitational Million-Body Problem, by Douglas Heggie and Piet Hut, 2003 [Cambridge University Press]

A lot of material on dense stellar systems can be found on the *MODEST web site*[1]

Two early review papers on MODEST, which stands for MOdeling DEnse STellar systems, are:

Hut, P., Shara, M.M., Aarseth, S.J., Klessen, R.S., Lombardi, J.C., Makino, J., McMillan, S., Pols, O.R., Teuben, P.J. and Webbink, R.F.; 2003, *MODEST-1: Integrating Stellar Evolution and Stellar Dynamics*, *New Astronomy* **8**, 337. See also the online *preprint*[2]

Sills, A., Deiters, S., Eggleton, P., Freitag, M., Giersz, M., Heggie, D., Hurley, J., Hut, P., Ivanova, N., Klessen, R.S., Kroupa, P., Lombardi, J.C., McMillan, S., Portegies Zwart, S. and Zinnecker, H., 2003, *Modest-2: A Summary*, *New Astronomy*, **8**, 605. Or: *preprint*[3].

[1] `http://www.manybody.org/modest.html`
[2] `http://arXiv.org/abs/astro-ph/0207318`
[3] `http://arXiv.org/abs/astro-ph/0301478`