BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-93-M7

"A Comparative Study of Relational and Object-Oriented Database Technology using The INGRES and ObjectStore Database Management Systems"

by

Seth H. Rosenzweig

A Comparative Study of Relational and Object–Oriented Database Technology using The INGRES and ObjectStore Database Management Systems

by

Seth H. Rosenzweig

A. B., University of Chicago, 1985

Sc. M., Brown University, 1989

Submitted in partial fulfillment of the requirements for the
Degree of Master of Science in the Department of Computer Science
at Brown University.

March 1993

This research project by Seth H. Rosenzweig is accepted in its present form
by the Department of Computer Science at Brown University
in partial fulfillment of the requirements for the Degree of Master of Science.

Date 4/1/93

Stanley B. Zdonik

Abstract

In this project, I use the ObjectStore object-oriented database management system (OODBMS) and the INGRES relational database management system (RDBMS) to conduct an empirical study of object-oriented database technology and its merits and drawbacks with regard to relational database systems. My assessment is both qualitative and quantitative: I develop an application to compute stock-option values from a financial database using each DBMS in order to get a qualitative feel for each system, and I design and run a series of benchmark tests on each DBMS in order to quantify the performance of each system. I have also explored the logistics of changing an INGRES-based application over to ObjectStore by converting the INGRES version of my option pricing program to be used with ObjectStore. Both quantitative and qualitative results show that ObjectStore is faster and more powerful than INGRES. On the other hand, INGRES is free, easy to use, well documented, and well supported. In addition, conversion from INGRES to ObjectStore was found to be quick and easy.

Preface

The *Option Pricing Case Study* portion of this project will eventually be incorporated into another research project conducted under the supervision of Professor Peter Falb in the Division of Applied Mathematics. Although I have developed this application in order to compare the two database management systems, it will subsequently be used to test the statistical validity of the various option-pricing models which I have derived and studied under Professor Falb. The relationship between the two projects will be mutually advantageous: The use of a real-life application will enhance the results of the database study, while the results of the database project will help to ensure the performance and reliability of the pricing model study. A brief synopsis of this application follows: A detailed presentation is given in appendix A.

My work in Applied Math focuses on deriving models for stock options and designing, performing, and analyzing tests on these (and other) models. A stock option is a contract which gives the holder the right to buy (or sell) one share of a certain stock at a fixed price, on or before a certain date. There may be several different options on the same stock. These options could include both call (right to buy) and put (right to sell) options, with different exercise prices and dates.

As a derivative instrument, an option's value can be modeled in terms of its underlying constituents, which include the stock price, the risk-free interest rate (Treasury Bill rate), time until expiration, etc. I can logically deduce certain relationships between the option value and the underlying variables. For example, the price of a call option can not exceed the value of the underlying stock; nor should it drop below its immediate exercise value (the current stock price minus the exercise price). There are a host of other such conditions, collectively called

arbitrage conditions, which must hold true for the relationship between an option and its underlying components.

One can empirically model the option value by finding a multivariate function which satisfies the arbitrage conditions and fitting it to data. One can also proceed by making a few additional assumptions about the stochastic nature of the stock market and deriving a theoretical model. In this case, data would still be required in order to estimate the appropriate statistical parameters (mean, variance, etc.).

Since data is needed in either case, in developing an option pricing application, I will need to construct a database. This database will include many different options on various stocks. Several intermingled data objects will be involved.

The main data object class will be a collection of stocks. Each stock will contain a collection of derivative securities (options, warrants, etc.), a price history object (containing date, closing price, and trading volume), and various other data. Each option will, in turn, contain its own price history object, a modeling parameter object, and other data specific to that trading instrument. There will also be an interest rate object which will be used by all of the securities.

Also to be considered are currency options, bond options, primes, scores, and compound options (i.e., options on options). The last may be particularly well suited to an object-oriented database system.

Acknowledgments

Above all, I would like to thank my parents, Rebecca and Charles Rosenzweig, without whose patience and support this work would not have been possible. I would also like to thank Professor Stanley Zdonik, who supervised this project, and Professor Peter Falb, from the Division of Applied Mathematics, whose careful guidance was invaluable to the *Option Pricing Case Study* portion of this project. In addition, I would like to thank Nikos Chloros, who helped me to think through many facets of modeling stock options, and Gregory Bucher, who patiently read and commented on many previous drafts of this document.

Contents

List of Tables

List of Figures

List of Abbreviations

DB	Database
DBMS	Database Management System
RDBMS	Relationally-Oriented Database Management System
OODBMS	Object-Oriented Database Management System
Ostore	ObjectStore
CUSIP number	A unique number which the Committee on Uniform Security Identification Procedures assignes to each publicly offered security
EQUEL	Embedded QUEry Language: Standard C code with the addition of query language commands which allow the program to access INGRES

Chapter 1

Introduction

1.1 Prolegomena

All database management systems have certain features in common: The ability to store and retrieve data, manage transactions, etc.[1] A DBMS should also perform its tasks with as little visibility to the end-user as possible, thus leaving the user free to concentrate on higher-level problems while the DBMS takes care of the details. Both relationally-oriented and object-oriented database management systems do these things; however, a relational DBMS is designed to manage simple tabular data, whereas an object-oriented package is designed to deal with more complex data and can also manage the code associated with that data.

With a relational system, for example, I could query all Fords from an automobile database, without worrying about the details of the actual file operations. An object-oriented system, on the other hand, moves the end-user to an even higher level of abstraction. For example, with an OODBMS, I might be able to instruct a "robot arm" to "move forward," without worrying about the low-level details of how to define a "robot arm" or how that data object would be manipulated to effect a "move forward." Thus, a DBMS allows the user to focus on what he wants done with the data, rather than how it will be done.

Abstracting to a higher level involves more overhead and may consequently

[1] For a complete description of the features of a DBMS, see [Ullman 1988].

cause things to run slower than they would under a lower-level implementation. On the other hand, managing and manipulating complex data with a system which is ill-equipped to do so can be awkward, inefficient, slow, and sometimes impossible. Thus, there are competing considerations when choosing a DBMS. Even if it were known that the old relational technology would be inferior for a given application, it might not be worthwhile to upgrade it to a newer object-oriented system.

1.2 Objectives

Given these two DBMS paradigms, it would be useful to conduct a comparative study to determine the relative strengths and weaknesses of each. It would also be useful to explore the logistics of upgrading an existing RDBMS-based application to be used with the newer OODBMS technology.

Thus, the purpose of this project is twofold: First, I wish to make a general comparison of OODBMS and RDBMS technology using the ObjectStore OODBMS and the INGRES RDBMS. Besides directly comparing the two systems, I also wish, as a second objective, to explore the logistics of converting an INGRES-based application to run using ObjectStore.

1.3 Strategy

There are two things which will reveal a lot about a system: A case study in which the system is used for a real-life purpose will yield qualitative information, while a series of benchmark tests which push the system to its limits will give specific quantitative performance data which will highlight the strengths and weaknesses of the system. Thus, I will carry out a controlled case study, as well as run benchmark tests using both ObjectStore and INGRES.

I have chosen a stock option pricing system for my case study. Since this application has elements of both the relational and object-oriented paradigms,

it should provide insight into the characteristics of each without excessively favoring either. I have successfully built this application, as well as a database of stock, option, and interest-rate data to be utilized, under both INGRES and ObjectStore. In constructing the Ostore version, I was also able to explore the logistics and potential benefits and drawbacks of converting an INGRES-based application to be used with ObjectStore. For each case, I was able to get a general qualitative feel for the flexibility, power, speed, and performance of the DBMS being used.

In order to quantitatively compare the two systems, I have directly measured the performance of each in a series of benchmark tests. These tests consist of timing various access operations on each of two test databases. Test database one, which is cast in the relational mold, is a single large table of integers, while test database two, which is characteristically object-oriented, contains a single large nested object of complex type. By benchmarking under these extreme conditions, I hope to reveal the strengths and weaknesses of each system.

Chapter 2

Procedure

2.1 Introduction

I proceed to construct both INGRES and ObjectStore versions of the stock option pricing application, the relationally-oriented benchmarking system, and the object-oriented benchmarking system. Each of these consists of a database and a program to manipulate that database. For the options pricing application, I made a financial database and wrote the associated options pricing program. For each benchmarking system, I constructed a test database and wrote software to clock various database operations. With INGRES and ObjectStore versions of each of these three applications, I should be able to make an educated comparison of the two systems.

2.2 Option pricing case study

2.2.1 Overview

I built the INGRES-based version of my option pricing application first. I proceeded by creating the database, making the required relations, and then filling those relations with data. When the database was completed, I wrote the application program. I then built the ObjectStore version by converting the INGRES system: First, I converted the INGRES database to ObjectStore format and then

I converted the program to interface with the ObjectStore database.

2.2.2 INGRES implementation

Database construction I used the UNIX command *creatdb* to create the INGRES version of the financial database and the command *create*, from within INGRES, to create the following relations for that database: 1) An *interest* relation which has records of three, six, and nine month treasury bill rates with their corresponding dates, 2) a *program* relation containing one record with the default model name, window size, trend length, tolerance, and limit, 3) a collection of *price* relations (one for each instrument including stocks) with records of closing prices and trading volumes with their associated dates, and 4) an *instrument* relation with each record containing name, underlying instrument, CUSIP number, ticker symbol, type, expiration, conversion value, strike price, outstanding shares, number of data points, number of model parameters, dividend payment, dilution factor, time parameter, interest parameter, trend parameter, dividend parameter, dilution parameter, and constant parameter.

Once the desired relations were created, I procured the necessary data. I then proceeded to convert the data from non-standard IBM PC format into standard ASCII files. Since SUN SPARCstations read IBM PC disks, I was able to access these ASCII files directly as standard UNIX text files. Using the INGRES *copy* command, I read each text file into the appropriate relation, and thus my database was built.

Program development With the database completed, I wrote my application program. I wrote the main option pricing program in C and the database interface in EQUEL, which is standard C code with the addition of Embedded QUEry Language commands which allow the program to access INGRES.

2.2.3 ObjectStore implementation

When the INGRES version was finished, I turned my attention to developing the application under ObjectStore. I strove first and foremost to explore the feasibility and ease of converting the INGRES-based version to run using ObjectStore. Thus, even though I could have designed a more optimal database schema, I adopted the INGRES layout, improving it only if the changes would not require an extensive overhaul of the main program or the database interface to become unduly complicated. This led to an ObjectStore database schema which was suboptimal by ObjectStore standards but still improved over the INGRES setup.

The main improvement was that ObjectStore could store each instrument as an object which consists in part of its own price history object. INGRES, on the other hand, must have a separate relation for each price history, and the name of that relation must be stored in the instrument relation. This removes the main inefficiency from the INGRES schema which was that it had to read the name of the price history relation from the *instrument* relation and then access that price history table. With Ostore, the instrument's price history could be accessed directly as part of that instrument's object. This improvement also eliminates the possibility that an error in the name of the price history would cause the database to be inconsistent.

Using this improved schema, I constructed the ObjectStore database by writing an EQUEL program which read the data from INGRES and then wrote it to ObjectStore. I was able to effect this conversion and otherwise complete the ObjectStore system without altering the main option pricing code and still keep the Ostore interface simple and straightforward. In fact, Ostore required only 220 lines of code to access the database, while INGRES needed 293 lines.

2.3 Benchmark testing

2.3.1 Overview

A meaningful set of benchmark tests should measure those aspects of performance which are most important to end users. In the case of a DBMS, access speed, ease of use, transparency, and flexibility are all important. Of these attributes, only access speed is easily quantified. However, if I design my tests carefully, I can indirectly gauge the other qualities as well.

By benchmarking access speeds for diverse types of data, I can reveal the flexibility and power of each DBMS. Since I am comparing relational and object-oriented systems, I will test each DBMS at the limits of the relational–object-oriented spectrum. This should give us performance data under a variety of circumstances which an end user may encounter.

Thus, my results should be useful for matching a DBMS to a particular application or in determining if it would be worthwhile to upgrade to a different DBMS.

2.3.2 Experimental design

With this strategy in mind, I created two test databases: The first is relationally-oriented, while the second is object–oriented. Following the maxim that simplicity is a virtue, I constructed my test databases entirely from random integers in the range $[0, 99]$. Test database one contains a single table of integers, whereas test database two consists of a main table and several subsidiary tables. For test database two, however, the individual entries would be either integers or other tables, which would, in turn, consist of integers and yet other tables. The lowest level would contain tables with only integers. Thus, test database one would consist of simple relationally-oriented tabular data, while database two, having tables nested within tables, is object-oriented.

For both cases, the simple nature of the basic integer data allows for straightforward query testing. I can neatly divide my test queries into four categories: 1) Simple, 2) Intermediate, 3) Complex, and 4) Very Complex. A simple query

might be to find all records whose third attribute is an integer in the range $10 <= x <= 20$. A complex query might be to find all records whose third attribute is in the range $10 <= x <= 20$, and whose fifth attribute is less than 70, and sixth attribute is an even number.

This kind of testing setup has the advantage of being simple in concept and construction, straightforward to benchmark, generic, and universal: The benchmark results should be relevant to many diverse problems.

2.3.3 Implementation

Test databases

Design I created both INGRES and ObjectStore versions of my two test databases. Test database one is a single large table which has one thousand records of eight integers, while test database two consists of top, middle and bottom levels, nested within one another. The top level contains ten entries of eight fields. The first seven fields are integers, and the last field is a middle-level table. Each middle level table contains four entries of eight fields. The first seven fields are integers, and the last field is a bottom-level table. Each bottom table has one thousand entries containing eight integers.

Creation In order to make these databases, I wrote a program which produced tables of random integers on an interval. The dimensions of the table as well as the the interval of the numbers are command-line options. By varying the dimensions and piping the output to different files, I was able to produce all of the required tables of integers in text-file format.

As with the options pricing application, I used the UNIX command *creatdb* to create the INGRES database, and the command *create*, from within INGRES, to create the relations from each table of data. Using the INGRES *copy* command, I was able to read each text file into a standard INGRES relation. With the relations in place, I used EQUEL to write the data to the INGRES database with a C program.

I made the ObjectStore versions by reading the data into memory using

EQUEL and then writing the data to the new ObjectStore database, just as with the options pricing example.

Test programs

I wrote benchmark application programs for each version of each test database. Under controlled conditions, these programs measured the time required to perform the following access operations on each test database:

1. Read the entire database into RAM.
2. Perform several simple queries (e.g., retrieve all records where column four is greater than 57).
3. Perform several intermediate-level queries (e.g., retrieve all records where col4 is greater than 57 and col3 is less than 45)
4. Perform several complex queries (with more retrieval constraints).
5. Perform several more complex queries (with even more (and more varied) retrieval constraints).

On object-oriented data, in order to get to the data which is nested on the bottom, INGRES must first read in the top-level data, then loop through this data in order to read in the middle-level data, and then loop through both the top and then the middle levels in order to retrieve the data from the bottom level. This is both awkward and inefficient. ObjectStore, on the other hand, can simply access the bottom level directly by navigating through the appropriate pointers. this is both quick and efficient, and it takes much less code to implement. It should also be noted that it is less prone to error.

It should be noted that for test database two, I did queries from the bottom level, since this puts the database at its most object-oriented, and thus tests the system at its limit by forcing the DBMS to "dig down" to the bottom of the nested structure. This should reveal how well each DBMS performs under object-oriented conditions.

I note that with INGRES, there are delays in processing: I need to read from the top-level relations in order to get the names of the middle-level relations, and then, in turn, I must read from the middle-level relations in order to get the names of the bottom-level relations. Only at that point will I be able to access data from the bottom level with INGRES.

With ObjectStore, on the other hand, I can directly address the bottom level (or any level) immediately. This is faster, neater, and not awkward as is the INGRES approach.

An additional consideration

Rather than making very large test databases and timing various access operations once, I will make the test databases smaller, run each query multiple times, and compute the average time. This technique has several advantages: I can use a smaller, more manageable database; I can easily adjust the accuracy of the benchmark by adjusting the number of trials; on average, I am likely to get more accurate results; and the chance of getting grossly erroneous results are drastically reduced.[1]

This method of repeated trials is the so-called *Monte Carlo Method.* The accuracy and reliability of this kind of technique is well known and has been thoroughly analyzed. The Monte Carlo error is given by $\sqrt{D/N}$, where D is a fixed constant, and N is the number of trials.[2] A typical Monte Carlo simulation might be accurate to about 5 percent.[3]

Thus, I can use a smaller test database to get measurements which are as accurate as those for a much larger one simply by increasing the number of trials. The relationship is inverse, such that, for example, ten thousand trials on a one hundred thousand kilobyte database would yield results as accurate as one trial on a one gigabyte database. I should also bear in mind that I can much more

[1]This method is analogous to determining the weight of a penny by weighing a sack of pennies and dividing by the total number, as opposed to weighing a single coin. Averaging over repeated trials smoothes the data so that the final averaged result is much less likely to be way off the mark than would be a single measurement.

[2]See [Sobol' 1974], p. 3.

[3]See [Sobol' 1974] for a general discussion of Monte Carlo Methods and their applications.

easily vary the number of trials than the size of the test database. Thus, using a smaller test database and repeated trials is also more flexible than using a large database and a single trial.

Chapter 3

Results

3.1 Option pricing case study

I strove first and foremost to explore the feasibility and ease of converting the INGRES-based version to run using ObjectStore. Thus, even though I could have designed a more optimal database schema, I adopted the INGRES layout, improving it only if the changes would not require either an extensive overhaul of the main program or the database interface to become unduly complicated. This led to an ObjectStore database schema which was suboptimal by ObjectStore standards but still improved over the INGRES setup.

With INGRES, my choices for data schemata were limited, and none of the few options possible was completely satisfactory. For Ostore, on the other hand, there were were several good schemata possible, but since I sought to convert as quickly and easily as possible, I adapted a modified version of the INGRES schema to that of ObjectStore. The conversion was quick and easy, the minor changes made to the schema proved to be important, and access speed was noticeably faster.

Thus, while ObjectStore appears to be better able to solve the problem from scratch by using a better database schema, it also proved to be powerful and flexible enough to quickly and quietly step in and replace INGRES as an application DBMS and boost performance, while preserving nearly all of the original code.

DBMS	lines of code required	disk space used	amount of data stored
INGRES	183	69 Kb	8 Kb
Ostore	320	336 Kb	8 Kb

Table 3.1: Specifications for test database one (relationally-oriented) benchmarks.

3.2 Benchmark testing

The benchmarks showed ObjectStore to be much faster (about thirty times), more powerful, and more flexible than INGRES, though it uses more disk space (see tables and graphs).

While certain results were expected, others were surprising: I expected Ostore to be more powerful, but I also thought that it would incur more overhead and thus possibly be slower for relationally-oriented data. This was not the case; In fact, ObjectStore accessed relational data much faster than did INGRES. More predictably, ObjectStore outperformed INGRES on object-oriented data. Thus, while Ostore can easily handle relational data – much better than INGRES, in fact – INGRES has only a limited facility for coping with object-oriented data.

I found certain things in common for both INGRES and ObjectStore. I would expect to find these things for any DBMS.

1. Query time increases with the number of hits. That is the retrieval will take longer if there are more hits.

2. Query time increases with the complexity of the query. That is, more conditions that are put on the retrieval, the longer it will take.

3.2.1 Test database one

3.2.2 Test database two

As I noted before, INGRES is unable to access the bottom level tables directly. It must first retrieve all of the middle table names from the top level relation and then retrieve all of the bottom table names from the middle level relations. An

DBMS	time per access	# of hits	# of trials	total time
total read				
INGRES	0.9 sec.	na	10	9 sec.
Ostore	0.034 sec.	na	1000	34 sec.
simple query one				
INGRES	0.8 sec.	978	10	8 sec.
Ostore	0.033 sec.	978	1000	33 sec.
simple query two				
INGRES	0.8 sec.	735	10	8 sec.
Ostore	0.033 sec.	735	1000	33 sec.
simple query three				
INGRES	0.7 sec.	481	10	7 sec.
Ostore	0.031 sec.	481	1000	31 sec.
simple query four				
INGRES	0.5 sec.	241	10	5 sec.
Ostore	0.030 sec.	241	1000	30 sec.
simple query five				
INGRES	0.5 sec.	0	10	5 sec.
Ostore	0.029 sec.	0	1000	29 sec.
intermediate query				
INGRES	0.5 sec.	4	10	5 sec.
Ostore	0.029 sec.	4	1000	29 sec.
complex query				
INGRES	0.4 sec.	0	10	4 sec.
Ostore	0.029 sec.	0	1000	29 sec.
very complex query one				
INGRES	0.5 sec.	0	10	5 sec.
Ostore	0.029 sec.	0	1000	29 sec.
very complex query two				
INGRES	0.9 sec.	720	10	9 sec.
Ostore	0.034 sec.	720	1000	34 sec.
very complex query three				
INGRES	0.9 sec.	730	10	9 sec.
Ostore	0.035 sec.	730	1000	35 sec.

Table 3.2: Benchmarks for test database one (relationally-oriented).

DBMS	lines of code required	disk space used	amount of data stored
INGRES	587	475 Kb	160 Kb
Ostore	562	1,548 Kb	160 Kb

Table 3.3: Specifications for test database two (object-oriented) benchmarks.

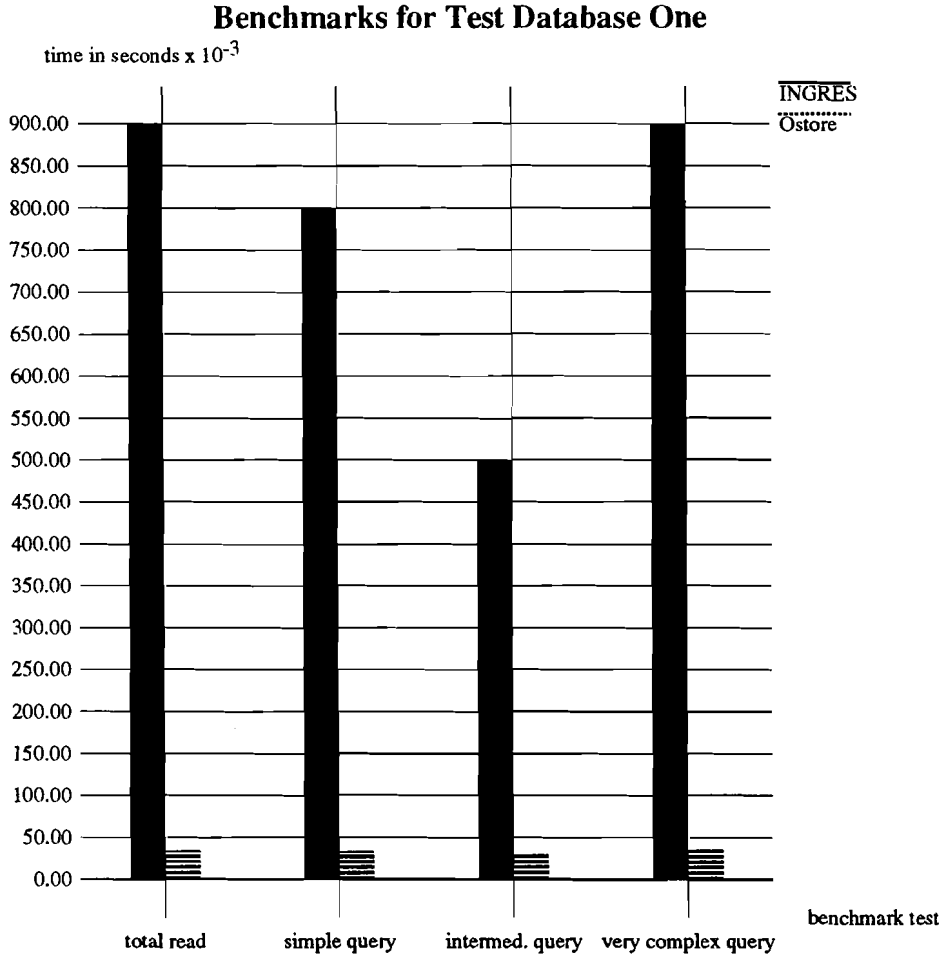

Figure 3.1: Benchmarks for test database one (relationally-oriented).

DBMS	time per access	# of hits	# of trials	total time
total read				
INGRES	18.5 sec.	na	10	185 sec.
Ostore	0.24 sec.	na	100	69 sec.
simple query one				
INGRES	19.5 sec.	38221	10	195 sec.
Ostore	0.26 sec.	38221	100	67 sec.
simple query two				
INGRES	16.0 sec.	28861	10	13 sec.
Ostore	0.23 sec.	28861	100	58 sec.
simple query three				
INGRES	12.3 sec.	19104	10	123 sec.
Ostore	0.20 sec.	19104	100	49 sec.
simple query four				
INGRES	8.7 sec.	9366	10	87 sec.
Ostore	0.19 sec.	9366	100	40 sec.
simple query five				
INGRES	4.9 sec.	0	10	49 sec.
Ostore	0.17 sec.	0	100	30 sec.
intermediate query				
INGRES	5.6 sec.	146	10	56 sec.
Ostore	0.17 sec.	146	100	31 sec.
complex query				
INGRES	5.5 sec.	0	10	55 sec.
Ostore	0.17 sec.	0	100	30 sec.
very complex query one				
INGRES	5.7 sec.	0	10	57 sec.
Ostore	0.17 sec.	0	100	31 sec.
very complex query two				
INGRES	21.4 sec.	27815	10	214 sec.
Ostore	0.25 sec.	27815	100	63 sec.
very complex query three				
INGRES	21.3 sec.	28233	10	213 sec.
Ostore	0.26 sec.	28233	100	63 sec.

Table 3.4: Benchmarks for test database two (object-oriented).

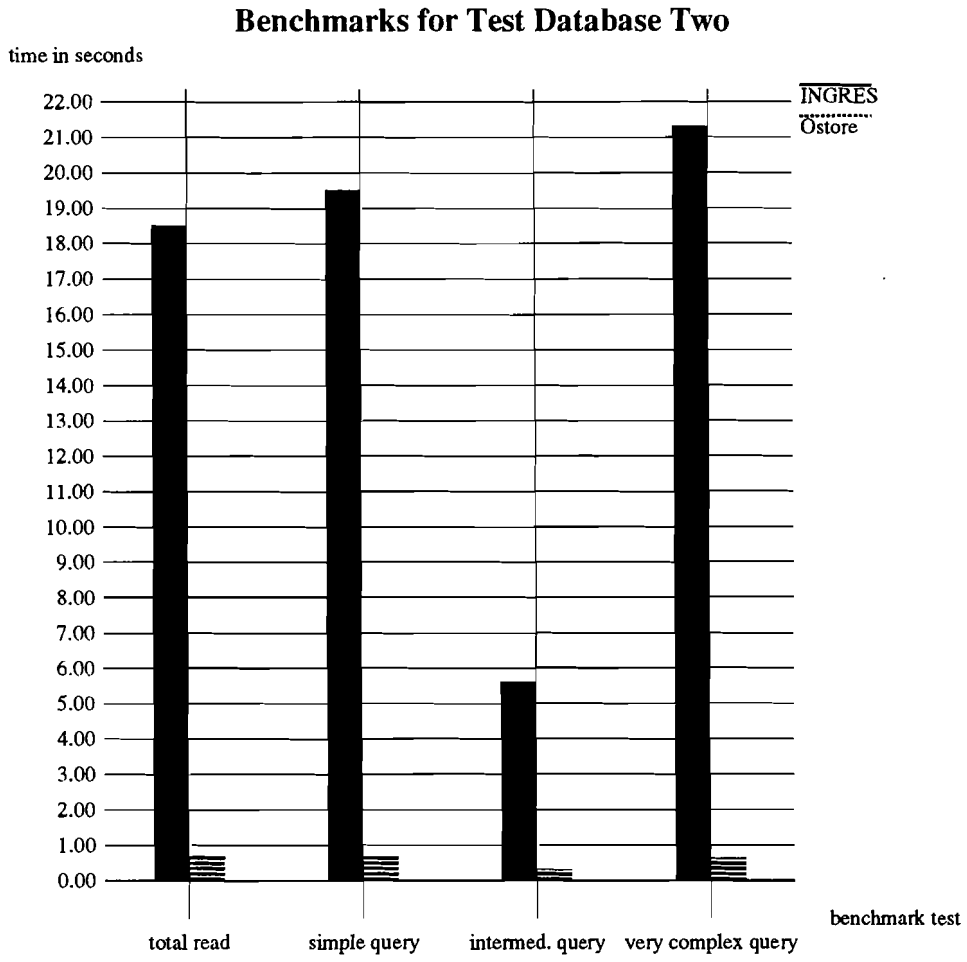

Figure 3.2: Benchmarks for test database two (object-oriented).

OODBMS, on the other hand, is capable of addressing the bottom level directly through the top level. This illustrates one example of how an OODBMS is able to more efficiently and gracefully navigate complex data. As the data gets more complex, the differences become more pronounced.

Chapter 4

Conclusions

General observations One would expect ObjectStore to excel in performance whenever navigational access is needed (e.g., with test database two), whereas INGRES might be a better choice for relational data (e.g., test database one). However, it turns out that ObjectStore is far superior in both realms. It is much faster (about thirty times) for both kinds of access, and it requires relatively simple and transparent coding for both cases. INGRES, on the other hand, is reasonable in the associative case, but cumbersome and complex for navigational access. Ostore is also considerably more flexible in how it can approach a problem and in how it can be custom tailored to specific needs. For example, it can be used with either the C or C++ programming languages, it has a built-in query optimizer, and it can be easily adapted to an existing DBMS application.

ObjectStore is a sophisticated program with a host of facilities to perform intricate navigation of complex object-oriented data. However, it is also adept at associative type queries, and can even be customized to search by user-defined keys through the query optimizer. It is almost invisible to the user in both C and C++.

Like INGRES, Ostore has all of the facilities to do simple relational queries. In fact it can do more sophisticated manipulations, and it can do them faster. At the same time, it can also perform set operations on any number of large collections of complex objects. It can do it in an optimized fashion, almost invisible to the user, with just a few lines of code. It would be difficult, if not impossible, to do

this sort of stuff with INGRES, even with many lines of code, and even if it were to be possible, it would be slow and inefficient.

In addition, I noted that it was relatively quick, easy, and painless to upgrade an application to be used with ObjectStore. I had no problem converting my option-pricing program from INGRES to ObjectStore. Database access speed was boosted noticeably

Recommendations If the data is simple and speed and performance are not critical, then INGRES is a reasonable package: It is free, easy to use, well-documented, and reliable. ObjectStore, on the other hand is more complex to master, though easy to use, and it is not free. However, it is much faster than INGRES even on relationally-oriented data and much more able to cope with complex object-oriented data setups elegantly, quickly, efficiently, and nearly transparently.

Thus, I would recommend INGRES if one is on a limited budget, performance is not important, and the data is simple. However, if the money is available, performance is important, or the data is complex, than ObjectStore is the clear choice for both relational and object-oriented data.

Further research Further research in this area would be interesting. This would include testing against a more modern relational DBMS as well as other kinds of benchmark tests, although I would not expect the results of additional benchmark tests to differ significantly.

Appendix A

Empirical Modeling Techniques for Stock Options and Other Convertible Instruments

A.1 Introduction

A stock option is a contract which gives the holder the right to buy (or sell) one share of a certain stock at a fixed price, on or before a certain date. There may be several different options on the same stock. These options could include both call (right to buy) and put (right to sell) options, with different exercise prices and dates.

As a derivative instrument, an option's value can be modeled in terms of its underlying constituents, which include the stock price, the risk-free interest rate (Treasury Bill rate), time until expiration, etc. I can logically deduce certain relationships between the option value and the underlying variables. For example, the price of a call option can not exceed the value of the underlying stock; nor should it drop below its immediate exercise value (the current stock price minus the exercise price). There are a host of other such conditions, collectively called arbitrage conditions, which must hold true for the relationship between an option and its underlying components.

I can empirically model the option value by finding a particular multivariate function which satisfies the arbitrage conditions and fitting it to data. I could also proceed by making a few additional assumptions about the stochastic nature of the stock market and deriving a theoretical model. In the latter case, I would still need data in order to estimate the appropriate statistical parameters (mean, variance, etc.).

A.2 The empirical modeling approach

The method of choice is to curve-fit the data to a multi-parameter function which behaves appropriately with respect to each contributing variable. This is a good approach because theoretically I know a great deal about how such a function should look, and how it should act with respect to each variable. Furthermore, such a model could statistically take into account the prior trading history, while making no implicit assumptions about the nature of such trading. In effect, this approach is interpolating/extrapolating the option value based on its past historical relationship to its underlying constituents (stock price, interest rate, time until expiration, etc.). Of course, the interpolation is not linear, but rather it is determined by the form of the function which I choose to fit. In addition, if I use a reasonably large set of data and choose a reasonable function to fit, then my model's values should converge to the values which would be found with the true density function.

Giguère, Kassouf, Shelton, and others have derived formulas for pricing warrants.[1] Giguère's formula, $W = S^2/4k$, is a simple parabola, which has no adjustable parameters and is only useful as a rule of thumb.[2] Shelton's formula is more sophisticated and more accurate, but it does not reflect all of the requisite properties of a warrant's value with respect to its determining variables. There is

[1] See [Giguère 1958], [Kassouf 1965, Kassouf 1969], and [Shelton 1967a, Shelton 1967b] respectively. Also, note that in my notation, W is the price of the warrant, S is the price of the stock, and k is the strike price.

[2] Empirical studies have shown Giguère's formula to work best for warrants with 3–5 years of life. The formula tends to overvalue those instruments with less than one year remaining, and to undervalue those with more than 3–5 years until expiration ([Gastineau 1988], 178).

also the drawback that the equation is not continuously differentiable, and thus the delta, or hedge ratio, can not always be computed. Kassouf has derived two different models. The first, $W = \sqrt{S^2 + k^2} - k$, is similar to Giguère's. It is the positive branch of a hyperbola, with no parameters, and, like the Giguère formula, it is only useful for rough estimates of Warrant values. Kassouf's second model is described in section A.4. With this background, I will proceed to develop an original formula, as well as to explore Kassouf's model.[3]

A.2.1 The arbitrage conditions

Before I can find a proper functional form for a model, I must first state the arbitrage conditions which determine the option's behavior with respect to each variable.[4] These conditions will form a set of constraints on my formula. They are as follows (see Figures A.1 and A.2):

Calls

Stock Price

$$\lim_{S \downarrow 0} C = 0 \tag{A.1}$$

$$\lim_{S \uparrow \infty} C = \infty \tag{A.2}$$

$$\frac{\partial C}{\partial S} > 0 \tag{A.3}$$

$$\frac{\partial C}{\partial S} \leq 1 \tag{A.4}$$

$$\frac{\partial^2 C}{\partial S^2} > 0 \tag{A.5}$$

[3]My derivation and formulation of Kassouf's model differ from his own (*cf.* [Kassouf 1965]).

[4]Note that in the arbitrage conditions, the strike price, k, is not discounted to be ke^{-rt}, as is sometimes done. If k were discounted, then this would imply that perpetual warrants would be worth S. Empirically, this is not the case. In addition, I feel that k should remain a constant throughout time and varying interest rates, and so, the strike price is not discounted, and the option's behavior with respect to both t and r will be expressed completely by the fitted parameters. This has the added benefit of relaxing the assumption that I can borrow or lend at some fixed risk-free interest rate, which is not always true.

Strike Price

$$\lim_{k \downarrow 0} C = S \tag{A.6}$$

$$\lim_{k \uparrow \infty} C = 0 \tag{A.7}$$

Time

$$\lim_{t \downarrow 0} C = \max\{0, S - k\} \tag{A.8}$$

$$\lim_{t \uparrow \infty} C \leq S \tag{A.9}$$

Interest Rate

$$\lim_{r \downarrow 0} C \geq \max\{0, S - k\} \tag{A.10}$$

$$\lim_{r \uparrow \infty} C \leq S \tag{A.11}$$

Relationship between calls and puts

$$P = C + k - S \tag{A.12}$$

$$\frac{\partial C}{\partial S} - \frac{\partial P}{\partial S} = 1 \tag{A.13}$$

$$\frac{\partial^2 C}{\partial S^2} - \frac{\partial^2 P}{\partial S^2} = 0 \tag{A.14}$$

Puts

Stock Price

$$\lim_{S \downarrow 0} P = k \tag{A.15}$$

$$\lim_{S \uparrow \infty} P = 0 \tag{A.16}$$

$$\frac{\partial P}{\partial S} < 0 \tag{A.17}$$

$$\frac{\partial P}{\partial S} \leq 1 \tag{A.18}$$

$$\frac{\partial^2 P}{\partial S^2} > 0 \tag{A.19}$$

Strike Price

$$\lim_{k \downarrow 0} P = 0 \tag{A.20}$$

$$\lim_{k \uparrow \infty} P = \infty \tag{A.21}$$

Time

$$\lim_{t \downarrow 0} P = \max\{0, k - S\} \tag{A.22}$$

$$\lim_{t \uparrow \infty} P \leq k \tag{A.23}$$

Interest Rate

$$\lim_{r \downarrow 0} P \geq \max\{0, k - S\} \tag{A.24}$$

$$\lim_{r \uparrow \infty} P \leq k \tag{A.25}$$

where t is time until expiration, r is the current risk free interest rate, S is the price of the underlying instrument, k is the strike price, C is the value of the call, and P is the value of the put.

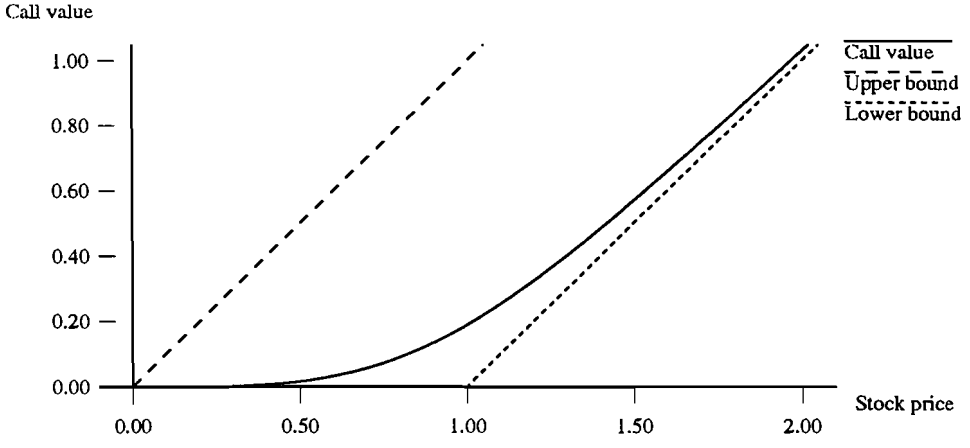

Figure A.1: Theoretical relationship between call and stock price.

Clearly, condition (A.1) must be true; i.e. if the stock has no value, then the right to buy it has no value. Similarly, if the stock has infinite value, then the

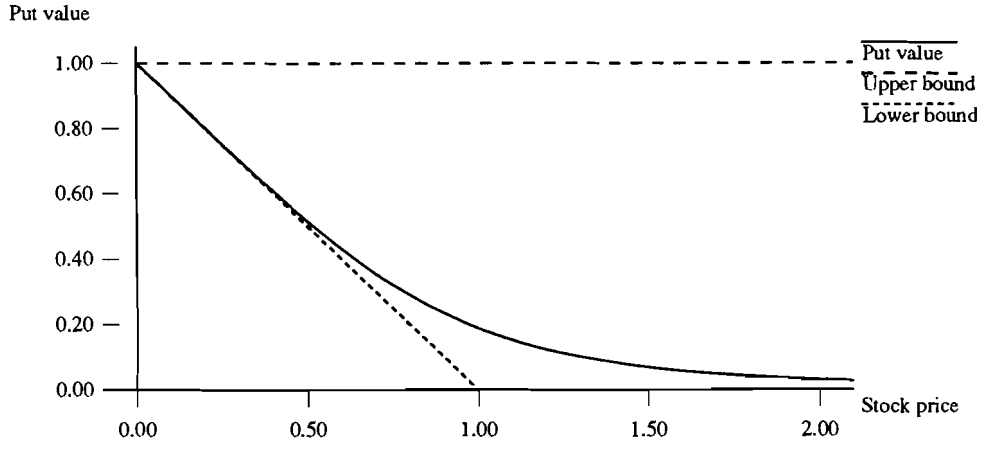

Figure A.2: Theoretical relationship between put and stock price.

right to purchase it at a fixed price has infinite value (condition A.2). I also know that the value of a call must increase with the stock's price (condition A.3), but it must not increase at a rate greater than the rate of change of S (condition A.4). And since the call's value must asymptotically approach both zero and infinity with the stock's price (conditions A.1 and A.2), I hypothesize that the call price must be a convex function of the stock price, hence condition (A.5).

As for conditions (A.6) and (A.7), if the strike price goes to zero, then the stock is to be purchased for nothing; therefore the call's value approaches that of the stock, and if the strike price goes to infinity, then the stock is to be purchased for an infinite sum, so the call is worth nothing. In addition, as t approaches zero, I approach expiration, and therefore, the call approaches intrinsic value (condition A.8). Similarly, as the interest rate approaches zero, the call value decreases, being lower-bounded by the intrinsic value (condition A.10). I must also have that as t or r get very large, C will be upper bounded by S (conditions A.9 and A.11).

Condition (A.12) states that owning a call and shorting a put of the same strike is equivalent to selling stock at the market value and buying it back at the strike. In theory, this condition must hold. At expiration, either the call will be

in the money or the put will be in the money. If $S > k$, then the put expires worthless, and I exercise the call, buying stock at k. If $S < k$, then the call expires worthless, and the put is exercised on us, forcing us to buy the stock at k. In either scenario, when I buy a call and sell a put, I buy stock at k. Buying stock at k when the market value is S is worth $S - k$. Therefore, $C - P = S - k$, or rewritten, $P = C + k - S$.[5] Condition (A.13) follows directly from condition (A.12), and condition (A.14) follows directly from condition (A.13).

I notice that condition (A.12) together with all of the arbitrage conditions for calls implies all of the arbitrage conditions for puts, and visa versa. This is not surprising since equation (A.12) completely determines the functional form for puts given calls, and for calls given puts. Since I can express calls completely in terms of puts and stock, and puts completely in terms of calls and stock, I conclude that calls, puts, and stock constitute only two independent instruments: options and their underlying stock.

A.2.2 Generic option model

For any option, I can easily convert between calls and puts by (A.12). All of the other convertible securities can similarly be expressed as either a call or a put. Thus, I need only to consider a generic option type and I can express a call, put, warrant, convertible bond, etc. in terms of this generic convertible.

My goal is to derive a model for Ω, a generic convertible. Ω will be a function of several variables, including S, t, r, and others, depending upon the specific security.[6] The function will also include adjustable parameters, $a_1, \ldots, a_n$, in order to fit the model to actual data. In general, Ω will not be a linear function of these parameters. I therefore seek a linear function, $g(\mathbf{v}) = \mathbf{a} \cdot \mathbf{v}$ such that

[5] In practice, I may occasionally wish to early exercise a call to collect a dividend or a put to collect interest from the cash. Thus, I have not accounted for all possible scenarios, and so, condition (A.12) will not always hold. However, condition (A.12) allows us to easily determine the proper functional form for puts given the formula for calls, and visa versa, even though the relation is not strictly true. Once I have the proper form, I can account for early exercise and other possibilities by fitting each call and put separately.

[6] Other variables, for example, might include dividend yield, trend, potential dilution, etc. A complete discussion of this is in section A.3.

$\Omega = f(g(\mathbf{v}))$, where $\mathbf{v}$ is the vector of basis functions. Each basis function, $V_i(t, r, \ldots)$, is a fixed function of one or more variables.

Once I have appropriate functions g and f, I can perform a linear least-squares regression on $g(\mathbf{v}) = \mathbf{a} \cdot \mathbf{v} = f^{-1}(\Omega)$ and solve for the parameter vector $\mathbf{a}$. I can use this estimate of $\mathbf{a}$ as the initial guess for a nonlinear regression on $\Omega = f(\mathbf{a} \cdot \mathbf{v})$.

A.2.3 Parameter estimation

Linear least-squares regression

I am left with the function $f^{-1}(\Omega) = g(\mathbf{v}) = \mathbf{a} \cdot \mathbf{v}$ which I must fit to a set of data.[7] Rewritten, I have

$$f^{-1}(\Omega) = g(\mathbf{v}) = \mathbf{a} \cdot \mathbf{v} = a_1 V_1(t, r, \ldots) + a_2 V_2(t, r, \ldots) + \cdots + a_n V_n(t, r, \ldots)$$

or,

$$f^{-1}(\Omega) = \sum_{i=1}^{n} a_i V_i(t, r, \ldots)$$

I want to fit this function to a set of m data points. Plugging the data points into my equation yields a set of m linear equations:

$$\begin{aligned}
f^{-1}(\Omega_1) &= a_1 V_1(t_1, r_1, \ldots) + \cdots + a_n V_n(t_1, r_1, \ldots) + \epsilon_1 \\
f^{-1}(\Omega_2) &= a_1 V_1(t_2, r_2, \ldots) + \cdots + a_n V_n(t_2, r_2, \ldots) + \epsilon_2 \\
&\vdots \\
f^{-1}(\Omega_m) &= a_1 V_1(t_m, r_m, \ldots) + \cdots + a_n V_n(t_m, r_m, \ldots) + \epsilon_m
\end{aligned}$$

where ϵ_i is the difference between the model's value and the actual value

[7] For a general description of the linear least-squares fitting techniques outlined in this section, see [Harvey 1981].

of $f^{-1}(\Omega_i)$.

I want to compute the parameter vector $\mathbf{a}$ such that $\chi^2 = \sum_{i=1}^{m} \epsilon_i^2$ is minimized. Restated, I want to minimize χ^2 with respect to $\mathbf{a}$ where

$$\chi^2 = \sum_{j=1}^{m} \left[f^{-1}(\Omega_j) - \sum_{i=1}^{n} a_i V_i(t_j, r_j, \ldots) \right]^2$$

This can be done by computing the partial derivatives of χ^2 with respect to each parameter a_i and then setting the result equal to zero. This yields a set of n linear equations:

$$0 = \sum_{j=1}^{m} \left[f^{-1}(\Omega_j) - \sum_{i=1}^{n} a_i V_i(t_j, r_j, \ldots) \right] V_k(t_j, r_j, \ldots) \quad \text{for} \quad k = 1, \ldots, n$$

To solve for the parameter vector, $\mathbf{a}$, I define the following:

$$\mathbf{v}_i = \begin{pmatrix} V_1(t_i, r_i, \ldots) \\ V_2(t_i, r_i, \ldots) \\ \vdots \\ V_n(t_i, r_i, \ldots) \end{pmatrix}, \quad \mathbf{y}_m = \begin{pmatrix} f^{-1}(\Omega_1) \\ f^{-1}(\Omega_2) \\ \vdots \\ f^{-1}(\Omega_m) \end{pmatrix}, \quad \mathbf{a}_m = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

and

$$\mathbf{V}_m = \begin{pmatrix} V_1(t_1, r_1, \ldots) & V_2(t_1, r_1, \ldots) & \ldots & V_n(t_1, r_1, \ldots) \\ V_1(t_2, r_2, \ldots) & V_2(t_2, r_2, \ldots) & \ldots & V_n(t_2, r_2, \ldots) \\ \vdots & & & \\ V_1(t_m, r_m, \ldots) & V_2(t_m, r_m, \ldots) & \ldots & V_n(t_m, r_m, \ldots) \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_m^T \end{pmatrix}$$

where $\mathbf{a}_m$ is the parameter vector computed from a sample of m data points.

In this notation, I can write

$$\mathbf{y}_m = \mathbf{V}_m \mathbf{a}_m$$

Solving for $\mathbf{a}_m$, I get:

$$\mathbf{a}_m = (\mathbf{V}_m^T \mathbf{V}_m)^{-1} \mathbf{V}_m^T \mathbf{y}_m \tag{A.26}$$

If I have computed $\mathbf{a}_m$, and a new data point is added, then I can compute the new parameter vector, $\mathbf{a}_{m+1}$, without inverting another matrix. Using the identity

$$(\mathbf{V}_{m+1}^T\mathbf{V}_{m+1})^{-1} = (\mathbf{V}_m^T\mathbf{V}_m)^{-1} - (\mathbf{V}_{m+1}^T\mathbf{V}_{m+1})^{-1}\mathbf{v}_{m+1}\mathbf{v}_{m+1}^T(\mathbf{V}_m^T\mathbf{V}_m)^{-1}/h_{m+1}$$

where

$$h_{m+1} = 1 + \mathbf{v}_{m+1}^T(\mathbf{V}_m^T\mathbf{V}_m)^{-1}\mathbf{v}_{m+1}$$

I get the following:

$$\mathbf{a}_{m+1} = \mathbf{a}_m + (\mathbf{V}_m^T\mathbf{V}_m)^{-1}\mathbf{v}_{m+1}(\mathbf{y}_{m+1} - \mathbf{v}_{m+1}^T\mathbf{a}_m)/h_{m+1} \qquad \text{(A.27)}$$

Thus I have an efficient algorithm for computing the parameters. [8] Once this computation is done, I can use these estimates as initial guesses for my nonlinear fitting routine. The nonlinear routine will actually fit parameters to the option price.

Nonlinear least-squares regression

Given an initial estimate for $\mathbf{a}$, I must now minimize the function $\chi^2(\mathbf{a}) = \sum_{i=1}^n [\Omega_i - f(\mathbf{a}\cdot\mathbf{v}_i)]^2$ with respect to $\mathbf{a}$. Setting the partial derivatives of χ^2 with respect to each parameter equal to zero gives the following set of n equations:

$$\frac{\partial\chi^2(\mathbf{a})}{\partial a_j} = \frac{\partial}{\partial a_j}\sum_{i=1}^m [\Omega_i - f(\mathbf{a}\cdot\mathbf{v}_i)]^2 = 0, \quad \text{for} \quad j = 1, 2, \ldots, n.$$

I must now solve for each a_j in order to form the parameter vector, $\mathbf{a}$. In general, this will not be possible. I must therefore use numerical methods to find the parameters,[9] $a_1, a_2, \ldots, a_n$ which minimize $\chi^2(\mathbf{a})$.

[8] Since I will only use linear parameter estimation to initialize my parameters, I should not in general need the recursive formula, (A.27). Under unusual circumstances, however, when the nonlinear search fails, it may be needed. In any case, it has been included for completeness.

[9] For a general description of the nonlinear least-squares fitting techniques outlined in this

Linearization of the fitting function I begin by expanding the function to be fitted, $f(\mathbf{v})$, into a Taylor series.[10]

$$f(\mathbf{v}) = f_0(\mathbf{v}) + \sum_{j=1}^{n}\left[\frac{\partial f_0(\mathbf{v})}{\partial a_j}\delta a_j\right] + \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n}\left[\frac{\partial^2 f_0(\mathbf{v})}{\partial a_j \partial a_k}\delta a_j \delta a_k\right] + \ldots$$

where $f_0(\mathbf{v})$ denotes $f(\mathbf{v})$ evaluated at $\mathbf{a}_0$, the current estimate of $\mathbf{a}$, and the parameter increment, δa_j, is given by $\delta a_j = a_j - a_{j,0}$, where $a_{j,0}$ is the current estimate of a_j.

If I am sufficiently close to the minimum, then I can drop the higher order terms and approximate $f(\mathbf{v})$ by

$$f(\mathbf{v}) = f_0(\mathbf{v}) + \sum_{j=1}^{n}\left[\frac{\partial f_0(\mathbf{v})}{\partial a_j}\delta a_j\right] \tag{A.28}$$

Since this is linear in the parameter increments, δa_j, I can use linear-least squares regression to solve for the parameter increment vector, $\delta\mathbf{a}$. With this approximation, I have

$$\chi^2 = \sum_{i=1}^{m}\left[\left(f(\mathbf{v}_i) - f_0(\mathbf{v}_i) - \sum_{j=1}^{n}\left[\frac{\partial f_0(\mathbf{v}_i)}{\partial a_j}\delta a_j\right]\right)^2\right]$$

Differentiating with respect to each parameter increment and setting the result equal to zero gives

$$\frac{\partial \chi^2}{\partial \delta a_k} = -2\sum_{i=1}^{m}\left[\left(f(\mathbf{v}_i) - f_0(\mathbf{v}_i) - \sum_{j=1}^{n}\left[\frac{\partial f_0(\mathbf{v}_i)}{\partial a_j}\delta a_j\right]\right)\frac{\partial f_0(\mathbf{v}_i)}{\partial \delta a_k}\right] = 0$$

for $k = 1, 2, \ldots, n$.

This gives n linear equations:

$$b_k = \sum_{j=1}^{n}\delta a_j A_{jk}, \quad \text{for} \quad k = 1, 2, \ldots, n$$

section, see [Bevington 1969].

[10]Computing the Taylor series of the function $\chi^2(\mathbf{a})$ and following the same line of reasoning will give the same results. In that case, it is called parabolic extrapolation of χ^2.

where

$$b_k \equiv -\frac{1}{2}\frac{\partial \chi_0^2}{\partial a_k} = \sum_{i=1}^{m}\left([f(\mathbf{v}_i) - f_0(\mathbf{v}_i)]\,\frac{\partial f_0(\mathbf{v}_i)}{\partial a_k}\right) \tag{A.29}$$

and[11]

$$A_{jk} = \sum_{i=1}^{m}\left[\frac{\partial f_0(\mathbf{v}_i)}{\partial a_j}\frac{\partial f_0(\mathbf{v}_i)}{\partial a_k}\right] \tag{A.30}$$

The vector of parameter increments is given by[12]

$$\delta\mathbf{a} = \mathbf{b}\mathbf{A}^{-1} \tag{A.31}$$

and, finally, I have

$$\mathbf{a} = \mathbf{a}_0 + \mathbf{A}^{-1}\cdot\mathbf{b}$$

or

$$\mathbf{a}_{\text{next}} = \mathbf{a}_{\text{curr}} + \mathbf{A}^{-1}\cdot\mathbf{b} \tag{A.32}$$

Gradient search In the event that $\mathbf{a}_{\text{curr}}$ (i.e., $\mathbf{a}_0$) is far from the minimum, then the higher order terms of the Taylor series, which I have discarded, become significant. Therefore, (A.28) gives a poor approximation of $f(\mathbf{v})$ when $\mathbf{a}_{\text{curr}}$ is far from the minimum. If this is the case, then I must conduct another kind of iterative search of the $\chi^2(\mathbf{a})$ hypersurface to find the parameters, $a_1, a_2, \ldots, a_n$ which minimize $\chi^2(\mathbf{a})$. In this case, where I am far from the minimum, I would like my search to propagate down the path of steepest descent (i.e., negatively along the gradient of the χ^2 hypersurface) in order to progress towards $\mathbf{a}_{\text{min}}$. Searching along the path of steepest-descent gives

[11]The first order Taylor series gives an expression for A_{jk} which is only a linear approximation. As I said before, when I am close to the minimum, this should be adequate.

[12]This method is often called fitting by linearization or the inverse-Hessian method ($\mathbf{A}$ is sometimes called the Hessian or curvature matrix).

$$\mathbf{a}_{\text{next}} = \mathbf{a}_{\text{curr}} - \text{constant} \times \nabla\chi^2(\mathbf{a}_{\text{curr}}) \tag{A.33}$$

where

$$\nabla\chi^2(\mathbf{a}_{\text{curr}}) = \sum_{i=1}^{n} \hat{a}_i \frac{\partial\chi^2}{\partial a_i}$$

where $\hat{a}_i$ is the unit vector along the $\hat{a}_i$–axis.

Noticing the similarity between the gradient, $\nabla\chi^2(\mathbf{a})$, and the vector $\mathbf{b}$, I can, without loss of generality, rewrite (A.33) as

$$\mathbf{a}_{\text{next}} = \mathbf{a}_{\text{curr}} - \text{constant} \times \mathbf{b} \tag{A.34}$$

The drawback of the steepest-descent method is that it is inefficient, requiring many small steps in order to converge on the minimum. The inverse-Hessian method, on the other hand, is plagued by the fact that far from the minimum, when the higher order derivatives are significant, (A.28) is a poor approximation, and thus, the method falls apart.[13] The best approach is to combine the two methods.

One such combination, the Levenberg-Marquardt Method, is the algorithm of choice. This approach will iteratively find the parameters which give the best χ^2 fit for the data points.[14]

Levenberg-Marquardt method First, I define the matrix $\mathbf{A}'$ such that

$$A'_{ij} \equiv \begin{cases} A_{ij} & \text{if } i \neq j \\ A_{ij}(1+\lambda) & \text{if } i = j \end{cases}$$

Substituting $\mathbf{A}'$ for $\mathbf{A}$ in equation (A.31), I get

$$\delta\mathbf{a} = \mathbf{b}\mathbf{A}'^{-1} \tag{A.35}$$

[13]It can even move in the wrong direction, causing convergence to the maximum χ^2.
[14]See [Marquardt 1963].

Solving this as before, I get

$$\delta \mathbf{a} = \mathbf{b}\mathbf{A}'^{-1} \tag{A.36}$$

and finally

$$\mathbf{a}_{\text{next}} = \mathbf{a}_{\text{curr}} + \mathbf{A}'^{-1} \cdot \mathbf{b} \tag{A.37}$$

Notice that when λ is zero, this is identical to (A.32), the inverse-Hessian estimate. However, as λ becomes large, this approaches (A.34), the gradient search estimate. So, changing λ varies (A.37) between the limits of the inverse-Hessian and gradient iterates. I use the following algorithm, put forth by Marquardt.

1. Compute $\chi^2(\mathbf{a})$.
2. Set λ equal to 0.001.
3. Compute $\delta\mathbf{a}$ and $\chi^2(\mathbf{a} + \delta\mathbf{a})$.
4. If $\chi^2(\mathbf{a} + \delta\mathbf{a}) \geq \chi^2(\mathbf{a})$, then increase λ by a factor of 10 and goto step (3).
5. If $\chi^2(\mathbf{a} + \delta\mathbf{a}) < \chi^2(\mathbf{a})$, then decrease λ by a factor of 10, set $\mathbf{a} = \mathbf{a} + \delta\mathbf{a}$, and goto step 3.

I will stop the procedure when the value of $\chi^2(\mathbf{a})$ decreases negligibly for one or two consecutive iterations. This indicates convergence of the parameter vector, $\mathbf{a}$, to the minimum.

A.3 The hyperbolic model

A.3.1 Derivation

Call options

From the arbitrage conditions, I know that $C \geq \max\{0, S - k\}$. From this, I can write $C = \max\{0, S - k\} + f(S, t, r, \tau, d)$, where τ is a trend indicator, and d is

the dividend yield during the life of the option. If I rewrite $\max\{0, S-k\}$ as $(|S-k|+S-k)/2$, then I have

$$C = \frac{|S-k|+S-k}{2} + f(S,t,r,\tau,d)$$

Substituting $\sqrt{(S-k)^2}$ for $|S-k|$ gives

$$C = \frac{\sqrt{(S-k)^2}+S-k}{2} + f(S,t,r,\tau,d)$$

where $f(S,t,r,\tau,d)$ gives the call premium over parity (i.e., value in addition to its intrinsic value). I now have to determine the functional form of f.

In order to satisfy arbitrage conditions (A.1), and (A.8) through (A.11), f will need to obey the following:

$$\begin{aligned}
&\lim_{S\downarrow 0} f = 0 \\
&\lim_{t\downarrow 0} f = 0 \\
&\lim_{t\uparrow \infty} f \leq \min\{S,k\} \\
&\lim_{r\downarrow 0} f \geq 0 \\
&\lim_{r\uparrow \infty} f \leq \min\{S,k\}
\end{aligned}$$

I can satisfy these conditions, along with arbitrage conditions (A.2), (A.6), and (A.7), by writing

$$f(S,t,r,\tau,d) = \min\{S,k\} \cdot \theta(t,r,\tau,d)$$

where[15]

[15]There are other forms for θ which would also work, such as $\theta(t,r,\tau,d) = 1 - 1/(a_1 t + a_2 tr + a_3 t\tau + a_4 td + 1)$.

$$\theta(t, r, \tau, d) = \frac{1}{a_1/t + a_2/(1+r) + a_3\tau + a_4 d + 1}$$

and $a_1, \ldots, a_4$ are parameters to be fitted.

I note that $\min\{S, k\} = Sk/\max\{S, k\} = 2Sk/(\sqrt{(S-k)^2} + S + k)$, and so, in analytic form, I have

$$f(S, t, r, \tau, d) = \frac{2Sk}{\sqrt{(S-k)^2} + S + k}\theta(t, r, \tau, d)$$

which gives

$$C = \frac{\sqrt{(S-k)^2} + S - k}{2} + \frac{2Sk}{\sqrt{(S-k)^2} + S + k}\theta(t, r, \tau, d)$$

The problem with this form is that it is not continuously differentiable with respect to S. There exists a discontinuity in the derivative at $S = k$. This makes it impossible to compute the delta of my option (the hedge ratio). Thus, I need to modify the form.

I know that at $t = 0$ the equation behaves correctly. Also, at $t = 0$ the equation assumes the form of a hyperbola, rotated 22.5° and translated along the S-axis by k. If I preserve the hyperbolic form for $t > 0$, then I will have resolved the discontinuity problem. Since θ makes the equation nonhyperbolic, except at $t = 0$, I seek to modify the way that θ is incorporated into the equation so that I preserve the hyperbolic form.

I begin with the general form for a hyperbola, rotate by 22.5° and translate along the S-axis by k. This will give us asymptotes of $C = 0$ and $C = S - k$ as required. I start with the unrotated, untranslated hyperbola in my primed coordinate system:

$$\frac{C'^2}{a^2} - \frac{S'^2}{b^2} = d$$

where $a = \sin 22.5°, b = \cos 22.5°$, and a_4 is a constant. By solving for C', I obtain $C' = a\sqrt{d + S'^2/b^2}$. I now apply the coordinate transformation matrix, given by:

$$\begin{pmatrix} b & -a \\ a & b \end{pmatrix} \begin{pmatrix} S' \\ C' \end{pmatrix} = \begin{pmatrix} S-k \\ C \end{pmatrix}$$

This yields

$$S' = b(S-k) + aC$$

and

$$C' = -a(S-k) + bC$$

Substituting these back into $C' = a\sqrt{d + S'^2/b^2}$ and solving for C as a function of S, I obtain

$$C = \frac{ab\sqrt{(S-k)^2 + (b^2-a^2)d} + ab(S-k)}{b^2-a^2}$$

Noting that $a = \sin 22.5°$ and $b = \cos 22.5°$, I can use the double angle formulas to obtain $2ab = \sin 45°$ and $(b^2 - a^2) = \cos 45°$. Since $\sin 45° = \cos 45°$, then $(b^2 - a^2) = 2ab$ and my formula reduces to:

$$C = \frac{\sqrt{(S-k)^2 + h} + S - k}{2}$$

where $h = (b^2 - a^2)d$.

Clearly, this formulation is similar to my previous expression, with the difference being that the premium-over-parity term is under the radical. This makes the expression continuously differentiable, and hyperbolic for all t, thus indicating that I should move $\theta(t, r, \tau, d)$ under the radical. Proceeding accordingly, I obtain the form:

$$C = \frac{\sqrt{(S-k)^2 + \theta(t, r, \tau, d)} + S - k}{2}$$

I must now insure that $C = 0$ whenever $S = 0$. I can do this, without otherwise affecting the behavior of my model, by translating my hyperbola, by an amount w, negatively along both the C and S-axes. So, I have

$$C + w = \frac{\sqrt{(S + w - k)^2 + \theta(t, r, \tau, d)} + S + w - k}{2}$$

Solving for w, I get

$$w = \frac{\theta}{4k}$$

Hence my new formula is

$$C = \frac{\sqrt{(S + \theta/4k - k)^2 + \theta} + S - k - \theta/4k}{2}$$

where $\theta = \theta(t, r, \tau, d)$.

I now must solve for θ and determine $\theta(t, r, \tau, d)$. It would be easiest to solve for my parameters if θ were linear in t, r, and τ.

First, I define:

$$u = 4k$$

$$v = S - k$$

$$w = \frac{\theta}{4k}$$

So, my equation becomes

$$C = \frac{\sqrt{(v + w)^2 + wu} + v - w}{2}$$

$\Rightarrow$

$$\frac{(2C + w - v)^2}{wu} = \frac{(v + w)^2}{wu} + 1$$

$\Rightarrow$

$$\frac{2C+w-v}{\sqrt{wu}} = \sqrt{\left(\frac{v+w}{\sqrt{wu}}\right)^2+1}$$

$\Rightarrow$

$$\frac{2C+2w}{\sqrt{wu}} = \sqrt{\left(\frac{v+w}{\sqrt{wu}}\right)^2+1}+\frac{v+w}{\sqrt{wu}} \tag{A.38}$$

Now let $(v+w)/\sqrt{wu} = (e^y - e^{-y})/2 = \sinh y$.
This gives

$$2\left(\frac{v+w}{\sqrt{wu}}\right) = e^y - e^{-y}$$

$\Rightarrow$

$$2e^y\left(\frac{v+w}{\sqrt{wu}}\right) = e^{2y} - 1$$

$\Rightarrow$

$$e^{2y} - 2e^y\left(\frac{v+w}{\sqrt{wu}}\right) - 1 = 0$$

By using the quadratic formula, I get

$$e^y = \frac{2(v+w)/\sqrt{wu} + \sqrt{4[(v+w)/\sqrt{wu}]^2+4}}{2}$$

$\Rightarrow$

$$e^y = \frac{v+w}{\sqrt{wu}} + \sqrt{\left(\frac{v+w}{\sqrt{wu}}\right)^2+1}$$

$\Rightarrow$

$$y = \ln\left[\frac{v+w}{\sqrt{wu}} + \sqrt{\left(\frac{v+w}{\sqrt{wu}}\right)^2+1}\right]$$

Since $(v+w)/\sqrt{wu} = \sinh y$, I have that $y = \sinh^{-1}[(v+w)/\sqrt{wu}]$.
Substituting back into (A.38), I get

$$\ln\left[\frac{2C+2w}{\sqrt{wu}}\right] = \ln\left[\sqrt{\left(\frac{v+w}{\sqrt{wu}}\right)^2+1}+\frac{v+w}{\sqrt{wu}}\right] = \sinh^{-1}\left(\frac{v+w}{\sqrt{wu}}\right)$$

$\Rightarrow$

$$\sinh\ln\left[\frac{2C+2w}{\sqrt{wu}}\right] = \frac{v+w}{\sqrt{wu}}$$

Recall that $\sinh y = (e^y - e^{-y})/2$
So, I get

$$(e^{\ln[(2C+2w)/\sqrt{wu}]} - e^{-\ln[(2C+2w)/\sqrt{wu}]})/2 = \frac{2C+2w}{2\sqrt{wu}} - \frac{\sqrt{wu}}{4C+4w} = \frac{v+w}{\sqrt{wu}}$$

$\Rightarrow$

$$C + w - \frac{wu}{4(C+w)} = v + w$$

Solving for w, I obtain

$$w = \frac{\theta}{4k} = \frac{C(C-S+k)}{S-C}$$

and finally

$$\theta = \frac{4kC(C-S+k)}{S-C} \tag{A.39}$$

I now search for a functional form for θ given my arbitrage conditions. Conditions (A.1) through (A.7) are always satisfied. From condition (A.8), as t approaches zero, the premium-over-parity term, θ, should vanish. From condition (A.9), as t becomes infinite, θ should increase without bound at the same rate. So, I derive the form $\theta(t) = a_1 t$.

Now, by condition (A.10), as r approaches zero, θ is lower bounded by 0. Also, by condition (A.11), as r becomes infinite, θ should increase without bound

at the same rate. So I arrive at $\theta(r) = a_2(1 + r)$.

For the trend term, I assume that when the last price of S, denoted by S_l, is greater than the average price over the previous n days, denoted by $\overline{S_n}$, then there is an upward trend. Thus the premium should be increased. When the opposite is true, the premium should be decreased. I decide on a trend term of $\theta(\tau) = a_3 \ln(S_l/\overline{S_n})$.

I must also incorporate dividends into my model. I note that when the dividend is zero, its contribution to θ is zero, and that dividends have a negative effect on the value of a call. That is, the higher the dividend, the more valuable the stock is relative to the call.[16] The simplest form to satisfy these conditions is $\theta(d) = a_4 d$, where the parameter a_4 should be negative.

My final form for θ is:

$$\theta(t, r, \tau, d) = a_1 t + a_2 tr + a_3 t \ln(S_l/\overline{S_n}) + a_4 td$$

where each term is multiplied by t to insure that as $t \to 0, \theta \to 0$. I can find the parameters a_1, a_2, a_3, and a_4 with standard least-squares fitting techniques.

Put options

From the call formula, and condition (A.12), I can find the hyperbolic formula for puts. Condition (A.12) states that $P = C - S + k$. Thus, I have

$$P = \frac{\sqrt{(S + \theta/4k - k)^2 + \theta} + S - k - \theta/4k}{2} - S + k$$

and so, I get

$$P = \frac{\sqrt{(S + \theta/4k - k)^2 + \theta} + k - S - \theta/4k}{2} \qquad \text{(A.40)}$$

where

[16] [Kassouf 1965], 43–44.

$$\theta = \theta(t, r, \tau, d) = a_1 t + a_2 tr + a_3 t\tau + a_4 td = \frac{4kP(P + S - k)}{k - P} \qquad \text{(A.41)}$$

This follows from substituting (A.12) into (A.39).

Other convertible instruments

Since stock options are the simplest type of convertible security, my model will need to be custom tailored to each of the other types of securities to be evaluated. The other instruments which I will consider are warrants, convertible bonds, currency options, primes, and scores, although it should be possible to use this basic model to evaluate any type of convertible security. Of this group, warrants are the easiest to model.

Warrants A warrant is virtually identical to a long-term call option. The difference, aside from the longer life, is the fact that the warrant is issued by the company, and, as a result, if the warrant is called, sometimes the company will issue additional shares of stock. These additional shares tend to dilute the value of those shares already outstanding. Thus, there is a dilution factor to consider for warrants. I define the dilution as $\delta = w/(v + w)$, where v is the number of outstanding shares of stock, and w is the number of shares which the outstanding warrants in the series can purchase.

As the dilution increases, the value of the stock, and consequently the value of the warrant, decreases. This can be incorporated into θ as $\theta(\delta) = a_5\delta$, where a_5 is another parameter. My functional form of θ for warrants becomes

$$\theta(t, r, \tau, d, \delta) = a_1 t + a_2 tr + a_3 t \ln(S_l/\overline{S_n}) + a_4 td + a_5 t\delta \qquad \text{(A.42)}$$

where $\delta = w/(v + w)$ is the dilution factor.

A.3.2 Model summary

A hyperbolic conic section, appropriately rotated and translated, will satisfy all of the conditions. I arrived at the following function:

$$C = \frac{\sqrt{(S + \theta/4k - k)^2 + \theta} + S - k - \theta/4k}{2} \tag{A.43}$$

where

$$\theta = \theta(t, r, \tau, d) = a_1 t + a_2 tr + a_3 t\tau + a_4 td + a_5 \delta = \frac{4kC(C - S + k)}{S - C} \tag{A.44}$$

and

$$\tau = \ln(S_l / \overline{S_n}) \tag{A.45}$$

where S_l is the last closing stock price, and $\overline{S_n}$ is the average price over the previous n days.

The delta of the call, or the hedge ratio, is given by:

$$\frac{\partial C}{\partial S} = \frac{\sqrt{(S + \theta/4k - k)^2 + \theta} + S - k + \theta/4k}{2\sqrt{(S + \theta/4k - k)^2 + \theta}} \tag{A.46}$$

In addition, I have the following:

$$\begin{aligned}
\frac{\partial C}{\partial a_1} &= \frac{\partial C}{\partial \theta} t \\
\frac{\partial C}{\partial a_2} &= \frac{\partial C}{\partial \theta} tr \\
\frac{\partial C}{\partial a_3} &= \frac{\partial C}{\partial \theta} t\tau \\
\frac{\partial C}{\partial a_4} &= \frac{\partial C}{\partial \theta} td \\
\frac{\partial C}{\partial a_5} &= \frac{\partial C}{\partial \theta} t\delta
\end{aligned}$$

where

$$\frac{\partial C}{\partial \theta} = \frac{S + \theta/4k + k - \sqrt{(S + \theta/4k - k)^2 + \theta}}{8k\sqrt{(S + \theta/4k - k)^2 + \theta}} \tag{A.47}$$

The formula for puts is:

$$P = \frac{\sqrt{(S + \theta/4k - k)^2 + \theta} - k + S - \theta/4k}{2} \tag{A.48}$$

where

$$\theta = \theta(t, r, \tau, d) = a_1 t + a_2 tr + a_3 t\tau + a_4 td + a_5 \delta = \frac{4kP(P + S - k)}{k - P} \tag{A.49}$$

The delta of the put is given by:

$$\frac{\partial P}{\partial S} = \frac{-\sqrt{(S + \theta/4k - k)^2 + \theta} + S - k + \theta/4k}{2\sqrt{(S + \theta/4k - k)^2 + \theta}} \tag{A.50}$$

I also have the following:

$$\begin{aligned}
\frac{\partial P}{\partial a_1} &= \frac{\partial P}{\partial \theta} t \\
\frac{\partial P}{\partial a_2} &= \frac{\partial P}{\partial \theta} tr \\
\frac{\partial P}{\partial a_3} &= \frac{\partial P}{\partial \theta} t\tau \\
\frac{\partial P}{\partial a_4} &= \frac{\partial P}{\partial \theta} td \\
\frac{\partial P}{\partial a_5} &= \frac{\partial P}{\partial \theta} t\delta
\end{aligned}$$

where

$$\frac{\partial P}{\partial \theta} = \frac{S + \theta/4k + k - \sqrt{(S + \theta/4k - k)^2 + \theta}}{8k\sqrt{(S + \theta/4k - k)^2 + \theta}} \tag{A.51}$$

Thus, I have a complete set of equations for modeling options. These equations can also be used to model other convertible instruments.

A.3.3 Parameter estimation

Linear estimate

In the notation of section A.2.2, I have (for warrants),

$$\Omega = C = \frac{\sqrt{(S + \theta/4k - k)^2 + \theta} + S - k - \theta/4k}{2}$$

and by equation (A.44),

$$f^{-1}(\Omega) = \mathbf{a} \cdot \mathbf{v} = a_1 V_1 + \ldots + a_n V_n = a_1 t + a_2 tr + a_3 t\tau + a_4 td + a_5 t\delta = \frac{4kC(C - S + k)}{S - C}$$

where I have

$$V_1 = t$$

$$V_2 = tr$$

$$V_3 = t\tau$$

$$V_4 = td$$

$$V_5 = t\delta$$

and

$$\mathbf{v}_i = \begin{pmatrix} t_i \\ t_i r_i \\ t_i \tau_i \\ t_i d_i \\ t_i \delta_i \end{pmatrix}, \quad \mathbf{y}_m = \begin{pmatrix} 4kC_1(C_1 - S_1 + k)/(S_1 - C_1) \\ 4kC_2(C_2 - S_2 + k)/(S_2 - C_2) \\ \vdots \\ 4kC_m(C_m - S_m + k)/(S_m - C_m) \end{pmatrix}, \quad \mathbf{a}_m = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

and

$$\mathbf{V}_m = \begin{pmatrix} t_1 & t_1 r_1 & t_1 \tau_1 & t_1 d_1 & t_1 \delta_1 \\ t_2 & t_2 r_2 & t_2 \tau_2 & t_2 d_2 & t_2 \delta_2 \\ \vdots & & & & \\ t_m & t_m r_m & t_m \tau_m & t_m d_m & t_m \delta_m \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_m^T \end{pmatrix}$$

where $\mathbf{a}_m$ is the parameter vector computed from a sample of m data points.

Recall from equation (A.26) that $\mathbf{a}_m$ is given by

$$\mathbf{a}_m = (\mathbf{V}_m^T \mathbf{V}_m)^{-1} \mathbf{V}_m^T \mathbf{y}_m$$

The formulation is identical for puts, except that C is replaced by $P + S - k$, according to equation (A.12). Note that for some instruments, I may not have dividends or dilution, etc. In these cases, I simply eliminate those factors from the model.

Nonlinear estimate

From equations (A.29) and (A.30), I have

$$b_k = \sum_{i=1}^{m} \left(\left[C_i - \frac{\sqrt{(S_i + \theta_{i,\text{curr}}/4k - k)^2 + \theta_{i,\text{curr}}} + S_i - k - \theta_{i,\text{curr}}/4k}{2} \right] \frac{\partial C}{\partial \theta_{i,\text{curr}}} \right)$$

and

$$A_{jk} = \sum_{i=1}^{m} \left[\left(\frac{\partial C}{\partial \theta_{i,\text{curr}}} \right)^2 \frac{\partial \theta_{i,\text{curr}}}{\partial a_j} \frac{\partial \theta_{i,\text{curr}}}{\partial a_k} \right]$$

and

$$A'_{jk} \equiv \begin{cases} A_{jk} & \text{if } j \neq k \\ A_{jk}(1+\lambda) & \text{if } j = k \end{cases}$$

where

$$\theta_{i,\text{curr}} = a_{1,\text{curr}} t_i + a_{2,\text{curr}} t_i r_i + a_{3,\text{curr}} t_i \tau_i + a_{4,\text{curr}} t_i d_i + a_{5,\text{curr}} t_i \delta_i$$

and

$$\frac{\partial C}{\partial \theta_{i,\text{curr}}} = \frac{S_i + \theta_{i,\text{curr}}/4k + k - \sqrt{(S_i + \theta_{i,\text{curr}}/4k - k)^2 + \theta_{i,\text{curr}}}}{8k\sqrt{(S_i + \theta_{i,\text{curr}}/4k - k)^2 + \theta_{i,\text{curr}}}}$$

and

$$\frac{\partial \theta_{i,\text{curr}}}{\partial a_{1,\text{curr}}} = t_i$$

$$\frac{\partial \theta_{i,\text{curr}}}{\partial a_{2,\text{curr}}} = t_i r_i$$

$$\frac{\partial \theta_{i,\text{curr}}}{\partial a_{3,\text{curr}}} = t_i \tau_i$$

$$\frac{\partial \theta_{i,\text{curr}}}{\partial a_{4,\text{curr}}} = t_i d_i$$

$$\frac{\partial \theta_{i,\text{curr}}}{\partial a_{5,\text{curr}}} = t_i \delta_i$$

From equation (A.37), I have

$$\mathbf{a}_{\text{next}} = \mathbf{a}_{\text{curr}} + \mathbf{A}'^{-1} \cdot \mathbf{b}$$

With this, I can adjust λ by the algorithm described in section A.2.3 to iteratively fit the parameters.

A.4 The Kassouf model

A.4.1 Derivation

Call options

From arbitrage conditions (A.8) and (A.9), I require that

$$\lim_{t \downarrow 0} C = \max\{S, k\} - k,$$

and

$$\lim_{t \uparrow \infty} C \leq S$$

I therefore seek a function, $f(S, t, r, \tau, d)$ such that

$$\lim_{t \to n} C = f(S, t, r, \tau, d) - k.$$

Where, when $n = 0$, $f(S, t, r, \tau, d) = \max\{S, k\}$, and when $n = \infty$, $f(S, t, r, \tau, d) = S + k$.

Thus I need a function which satisfies the following two properties:

$$\lim_{n \to 0} f(S, t, r, \tau, d) = \max\{S, k\}$$

and

$$\lim_{n \to \infty} f(S, t, r, \tau, d) = S + k.$$

I recall from functional analysis that the L_p norm of $(S, k) \in \mathbb{R}$ approaches $\max\{S, k\}$ as $p \to \infty$, and it approaches $S + k$ as $p \to 1$. Restated,

$$\lim_{p\to\infty} \|(S,k)\|_{L_p} = \max\{S,k\}$$

and

$$\lim_{p\to 1} \|(S,k)\|_{L_p} = S + k$$

I write the L_p norm of (S,k) as $\|(S,k)\|_{L_p} = (S^p + k^p)^{1/p}$. I relabel p as γ, and write

$$C = (S^\gamma + k^\gamma)^{1/\gamma} - k$$

where

$$\gamma = \gamma(t, r, \tau, d).$$

I normalize the formula by setting S equal to S/k, and C equal to C/k. This gives

$$C = (S^\gamma + 1)^{1/\gamma} - 1, \tag{A.52}$$

Now, I must solve for γ. Let $\alpha = \ln(C+1)$ and $\beta = \ln S$. So,

$$C + 1 = (S^\gamma + 1)^{1/\gamma}$$

$\Rightarrow$

$$(C+1)^\gamma = S^\gamma + 1$$

$\Rightarrow$

$$e^{\alpha\gamma} = e^{\beta\gamma} + 1$$

$\Rightarrow$

$$e^{\alpha\gamma} - e^{\beta\gamma} - 1 = 0$$

I use Newton's method to solve for the root, γ, of this equation.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$\Rightarrow$

$$\gamma_{n+1} = \gamma_n - \frac{e^{\alpha\gamma_n} - e^{\beta\gamma_n} - 1}{\alpha e^{\alpha\gamma_n} - \beta e^{\beta\gamma_n}}$$

This becomes

$$\gamma_{n+1} = \gamma_n - \frac{(C+1)^{\gamma_n} - S^{\gamma_n} - 1}{(C+1)^{\gamma_n}\ln(C+1) - S^{\gamma_n}\ln S} \tag{A.53}$$

Thus, I compute γ by successive iterations.

I must now determine a functional form for $\gamma(t, r, \tau, d)$, which satisfies the arbitrage conditions. The first seven conditions always hold true. From condition (A.8) and the properties of the L_p norm, as t approaches 0, γ must approach ∞; and from condition (A.9), as t approaches ∞, γ is lower bounded by 1. I satisfy these conditions with the form $\gamma(t) = a_1/t$.

From conditions (A.10), and (A.11), as r goes to 0, γ approaches some finite constant; and as r approaches ∞, γ is upper bounded by 1. To satisfy these conditions, I use the form $\gamma(r) = a_2/(1+r)$.

For the trend term, I have $\gamma(\tau) = a_3 \ln(S_l/\overline{S_n})$. The parameter a_3 should be negative, since γ should be inversely proportional to the trend.

As with the Hyperbolic model, the dividend term should vanish when d is zero. Since both γ and d are inversely proportional to the premium over parity, the dividend term should contribute positively to γ. The simplest form to meet this criterion is $\gamma(d) = a_4 d$, where the parameter, a_4, should be positive.

Thus, my final form for γ is:

$$\gamma(t, r, \tau, d) = \frac{a_1}{t} + \frac{a_2}{1+r} + a_3 \ln(S_l/\overline{S_n}) + a_4 d + \gamma_0, \tag{A.54}$$

where a_1, a_2, a_3, a_4, and γ_0 are the parameters to be fitted.

Put options

Using condition (A.12), and the formula for calls, I find the formula for puts:

$$P = (S^\gamma + 1)^{1/\gamma} - S, \tag{A.55}$$

where

$$\gamma = \gamma(t, r, \tau, d) = \frac{a_1}{t} + \frac{a_2}{1+r} + a_3 \ln(S_l/\overline{S_n}) + a_4 d + \gamma_0 \tag{A.56}$$

I compute γ for puts by substituting (A.12) into (A.53). This gives the following formula for iteratively computing γ for puts:

$$\gamma_{n+1} = \gamma_n - \frac{(P+S)^{\gamma_n} - S^{\gamma_n} - 1}{(P+S)^{\gamma_n} \ln(P+S) - S^{\gamma_n} \ln S}$$

Other convertible instruments

As with the Hyperbolic model, I can use Kassouf's model to describe warrants, convertible bonds, currency options, primes, and scores. The model has to be modified in a different way for each type of convertible instrument. Again, I begin with the simplest case, warrants.

Warrants Since both dilution and γ are inversely proportional to the value of the warrant, I can write $\gamma(\delta) = a_5\delta$, where a_5 is a parameter to be fitted. The final form of γ for warrants becomes

$$\gamma(t, r, \tau, d, \delta) = \frac{a_1}{t} + \frac{a_2}{1+r} + a_3 \ln(S_l/\overline{S_n}) + a_4 d + a_5\delta + \gamma_0 \tag{A.57}$$

A.4.2 Model summary

By using the fact that the limit of the L_p norm of $(x, y) \in \mathbf{R}$ goes to $\max\{x, y\}$ as $p \to \infty$, and that it goes to $x + y$ as $p \to 1$, I can write

$$C = \|(S, k)\|_{L_p} - k$$

for some $p < \infty$, where

$$\|(S,k)\|_{L_p} = (S^p + k^p)^{1/p}$$

If I substitute γ for p, I get

$$C = (S^\gamma + k^\gamma)^{1/\gamma} - k,$$

I normalize my formula and get

$$C = (S^\gamma + 1)^{1/\gamma} - 1, \tag{A.58}$$

where S is now S/k, and C is now C/k, and

$$\gamma = \gamma(t,r,\tau,d) = \frac{a_1}{t} + \frac{a_2}{1+r} + a_3 \ln(S_l/\overline{S_n}) + a_4 d + a_5 \delta + \gamma_0 \tag{A.59}$$

γ can be computed iteratively by Newton's method, where each iteration is given by

$$\gamma_{n+1} = \gamma_n - \frac{(C+1)^{\gamma_n} - S^{\gamma_n} - 1}{(C+1)^{\gamma_n}\ln(C+1) - S^{\gamma_n}\ln S}$$

The hedge ratio is given by:

$$\frac{\partial C}{\partial S} = S^{\gamma-1}(S^\gamma + 1)^{1/\gamma - 1} \tag{A.60}$$

In addition, I have the following:

$$\frac{\partial C}{\partial a_1} = \frac{\partial C}{\partial \gamma}\frac{1}{t}$$
$$\frac{\partial C}{\partial a_2} = \frac{\partial C}{\partial \gamma}\frac{1}{1+r}$$
$$\frac{\partial C}{\partial a_3} = \frac{\partial C}{\partial \gamma}\tau$$

$$\frac{\partial C}{\partial a_4} = \frac{\partial C}{\partial \gamma} d$$
$$\frac{\partial C}{\partial a_5} = \frac{\partial C}{\partial \gamma} \delta$$
$$\frac{\partial C}{\partial \gamma_0} = \frac{\partial C}{\partial \gamma}$$

where

$$\frac{\partial C}{\partial \gamma} = -(S^\gamma + 1)^{1/\gamma} \left[\frac{1}{\gamma} \ln(S^\gamma + 1) - \frac{S^\gamma \ln S + 1}{\gamma(S^\gamma + 1)} \right] \tag{A.61}$$

The formula for puts is:

$$P = (S^\gamma + 1)^{1/\gamma} - S, \tag{A.62}$$

where

$$\gamma = \gamma(t, r, \tau, d) = \frac{a_1}{t} + \frac{a_2}{1+r} + a_3 \ln(S_l/\overline{S_n}) + a_4 d + a_5 \delta + \gamma_0 \tag{A.63}$$

and γ is computed iteratively by the following formula:

$$\gamma_{n+1} = \gamma_n - \frac{(P+S)^{\gamma_n} - S^{\gamma_n} - 1}{(P+S)^{\gamma_n} \ln(P+S) - S^{\gamma_n} \ln S}$$

The hedge ratio is given by:

$$\frac{\partial P}{\partial S} = S^{\gamma-1}(S^\gamma + 1)^{1/\gamma - 1} - 1 \tag{A.64}$$

In addition, I have the following:

$$\frac{\partial P}{\partial a_1} = \frac{\partial P}{\partial \gamma} \frac{1}{t}$$
$$\frac{\partial P}{\partial a_2} = \frac{\partial P}{\partial \gamma} \frac{1}{1+r}$$
$$\frac{\partial P}{\partial a_3} = \frac{\partial P}{\partial \gamma} \tau$$

$$\frac{\partial P}{\partial a_4} = \frac{\partial P}{\partial \gamma} d$$
$$\frac{\partial P}{\partial a_5} = \frac{\partial P}{\partial \gamma} \delta$$
$$\frac{\partial P}{\partial \gamma_0} = \frac{\partial P}{\partial \gamma}$$

where

$$\frac{\partial P}{\partial \gamma} = -(S^\gamma + 1)^{1/\gamma} \left[\frac{1}{\gamma} \ln(S^\gamma + 1) - \frac{S^\gamma \ln S + 1}{\gamma(S^\gamma + 1)} \right] \tag{A.65}$$

This gives us another complete set of formulas for option pricing.

A.4.3 Parameter estimation

Linear estimate

I have

$$\Omega = C = (S^\gamma + 1)^{1/\gamma} - 1$$

and

$$f^{-1}(\Omega) = \mathbf{a} \cdot \mathbf{v} = a_1 V_1 + \ldots + a_n V_n = \frac{a_1}{t} + \frac{a_2}{1+r} + a_3 \tau + a_4 d + a_5 \delta + \gamma_0 = \gamma$$

where I have

$$V_1 = \frac{1}{t}$$
$$V_2 = \frac{1}{1+r}$$
$$V_3 = \tau$$
$$V_4 = d$$
$$V_5 = \delta$$

$$V_6 = 1$$

and

$$\mathbf{v}_i = \begin{pmatrix} 1/t_i \\ 1/(1+r_i) \\ \tau_i \\ d_i \\ \delta_i \\ 1 \end{pmatrix}, \quad \mathbf{y}_m = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_m \end{pmatrix}, \quad \mathbf{a}_m = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

where γ is computed iteratively by Newton's method as given in formula (A.53).

I also have

$$\mathbf{V}_m = \begin{pmatrix} 1/t_1 & 1/(1+r_1) & \tau_1 & d_1 & \delta_1 & 1 \\ 1/t_2 & 1/(1+r_2) & \tau_2 & d_2 & \delta_2 & 1 \\ \vdots & & & & & \\ 1/t_m & 1/(1+r_m) & \tau_m & d_m & \delta_m & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_m^T \end{pmatrix}$$

where $\mathbf{a}_m$ is the parameter vector computed from a sample of m data points.

Recall from equation (A.26) that $\mathbf{a}_m$ is given by

$$\mathbf{a}_m = (\mathbf{V}_m^T \mathbf{V}_m)^{-1} \mathbf{V}_m^T \mathbf{y}_m$$

As with the Hyperbolic model, the formulation is identical for puts, except that C is replaced by $P + S - k$, according to equation (A.12).

Nonlinear estimate

From equations (A.29) and (A.30), I have

$$b_k = \sum_{i=1}^{m} \left(\left[C_i - (S^{\gamma_{i,\text{curr}}} + 1)^{1/\gamma_{i,\text{curr}}} - 1 \right] \frac{\partial C}{\partial \gamma_{i,\text{curr}}} \right)$$

and

$$A_{jk} = \sum_{i=1}^{m} \left[\left(\frac{\partial C}{\partial \gamma_{i,\text{curr}}} \right)^2 \frac{\partial \gamma_{i,\text{curr}}}{\partial a_j} \frac{\partial \gamma_{i,\text{curr}}}{\partial a_k} \right]$$

and

$$A'_{jk} \equiv \begin{cases} A_{jk} & \text{if } j \neq k \\ A_{jk}(1+\lambda) & \text{if } j = k \end{cases}$$

where

$$\gamma_{i,\text{curr}} = \frac{a_{1,\text{curr}}}{t_i} + \frac{a_{2,\text{curr}}}{1+r_i} + a_{3,\text{curr}}\tau_i + a_{4,\text{curr}}d_i + a_{5,\text{curr}}\delta_i + \gamma_{i,\text{curr}}$$

and

$$\frac{\partial C}{\partial \gamma_{i,\text{curr}}} = -(S^{\gamma_{i,\text{curr}}} + 1)^{1/\gamma_{i,\text{curr}}} \left[\frac{1}{\gamma_{i,\text{curr}}} \ln(S^{\gamma_{i,\text{curr}}} + 1) - \frac{S^{\gamma_{i,\text{curr}}} \ln S + 1}{\gamma_{i,\text{curr}}(S^{\gamma_{i,\text{curr}}} + 1)} \right]$$

and

$$\frac{\partial \gamma_{i,\text{curr}}}{\partial a_{1,\text{curr}}} = \frac{1}{t_i}$$

$$\frac{\partial \gamma_{i,\text{curr}}}{\partial a_{2,\text{curr}}} = \frac{1}{1+r_i}$$

$$\frac{\partial \gamma_{i,\text{curr}}}{\partial a_{3,\text{curr}}} = \tau_i$$

$$\frac{\partial \gamma_{i,\text{curr}}}{\partial a_{4,\text{curr}}} = d_i$$

$$\frac{\partial \gamma_{i,\text{curr}}}{\partial a_{5,\text{curr}}} = \delta_i$$

$$\frac{\partial \gamma_{i,\text{curr}}}{\partial a_{6,\text{curr}}} = 1$$

From equation (A.37), I have

$$\mathbf{a}_{\text{next}} = \mathbf{a}_{\text{curr}} + \mathbf{A}'^{-1} \cdot \mathbf{b}$$

With this, I can adjust λ by the algorithm described in section A.2.3 to iteratively fit the parameters.

Appendix B

Option Pricing Database Specifications

B.1 Kinds of securities to be considered

Although there are many different types of derivative securities, I will limit myself to the following five types:

1. **Long-term stock options** A long-term stock option is a contract which gives the holder the right to buy (or sell) one share of common stock at a fixed price, on or before a certain date. There may be several different options on the same stock. These options could include both call (right to buy) and put (right to sell) options, with different exercise prices and dates. Long-term options are identical to regular options, except that they have a longer lifespan (normal options have a maximum lifespan of nine months, whereas long-term option may last for several years.) Both regular and long-term options are issued by the various exchanges.

   (a) **Calls** A call option gives the holder the right to buy common stock.

   (b) **Puts** A put option gives the holder the right to sell common stock.

2. **Warrants** A warrant is a contract which gives the holder the right to buy one share of a certain stock at a fixed price, on or before a certain date.

There may be several different warrants on the same stock (with different exercise prices and expiration dates). A warrant is identical to a call option, except that a warrant has a longer life, and it is issued by the underlying company.

3. **Americus Trust Units** The Americus Trust unit, which is sponsored by the Americus Shareowner Service Corporation, separates the risk and the reward of a stock into two option-like components.

   (a) **PRIMEs** This is an acronym for *Prescribed Right to Income and Maximum Equity.* The holder gets dividends as well as all appreciation up to the strike. This is similar to a covered call (i.e., owning one share of stock and being short one call.).

   (b) **SCOREs** This is an acronym for *Special Claim On Residual Equity.* The holder gets no dividends but all appreciation above the strike. This is similar to a call option.

4. **Convertible Bonds or Debentures** This is a bond contract which can be converted into a specified number of common shares.[1]

5. **Convertible Preferreds** These are preferred shares which can be converted to a specified number of common shares.

I have chosen these five derivative securities since they all have long lifespans, are actively traded, and have many different issues outstanding. It should be noted that all outstanding Americus Trust units expire by the end of 1992. Despite this, I will include these instruments since their high volume and long lifespan provide an excellent source of historical data for testing, even thought they may no longer be useful for predictive modeling.

[1]Sometimes a convertible bond may be exchanged for preferred stock, warrants, or various other securities. Of course, this depends upon the specific terms of the contract.

B.2 Data requirements

The following outline illustrates the relationship of the data in the database.

1. **Array of stocks** This will be an array of the stocks in the database. Each stock will contain the following information.

   (a) **Name** Stock name.

   (b) **CUSIP** Alpha-numeric string which is unique to the security.

   (c) **Symbol** Stock exchange symbol.

   (d) **Number of shares outstanding**

   (e) **Dividend** Annual dividend payment.

   (f) **Data array** This will contain all of the daily (or weekly) data for the stock.

      i. **Date**

      ii. **Closing price**

      iii. **Volume**

   (g) **Derivative Securities** This will be an array of derivative securities. Of course, each stock will have its own array of derivatives.

      i. **Name**

      ii. **CUSIP**

      iii. **Symbol**

      iv. **Number of outstanding contracts**

      v. **Type**

      vi. **Expiration date**

      vii. **Conversion Ratio**

      viii. **Strike price**

      ix. **Yield** This is the Yield on the convertible bond.

      x. **Data array** This will contain all of the daily (or weekly) data for the derivative security.

A. **Date**

B. **Closing price**

C. **Volume**

D. **Conversion value** This is for convertible bonds and convertible preferreds.

E. **Investment value** This is the estimated value of the convertible bond if it were not convertible. Together with the conversion value, the investment value forms a theoretical lower limit for the the value of a convertible bond. I have the following theoretical arbitrage relationship: convertible bond value $\geq$ max{Investment Value, Conversion Value}. Note that this is analogous to the arbitrage relation $C \geq \max\{0, S - k\}$ for call options.

xi. **Model array** This contains sets of parameters or weights. Each set of weights is associated with one particular model. Each individual weight is associated with one particular determinant within that model. Note that these are the values which I compute when I fit my function to the data.

A. **Time weight**

B. **Interest weight**

C. **Trend weight**

D. **Dividend weight**

E. **Dilution weight**

F. **Constant weight**

2. **Array of interest rates** Daily risk-free interest rates.

(a) **Date**

(b) **Three month Treasury Bill rate**

(c) **Six month Treasury Bill rate**

(d) **Nine month Treasury Bill rate**

Appendix C

Code for Options Pricing Case Study

C.1 Program flowchart

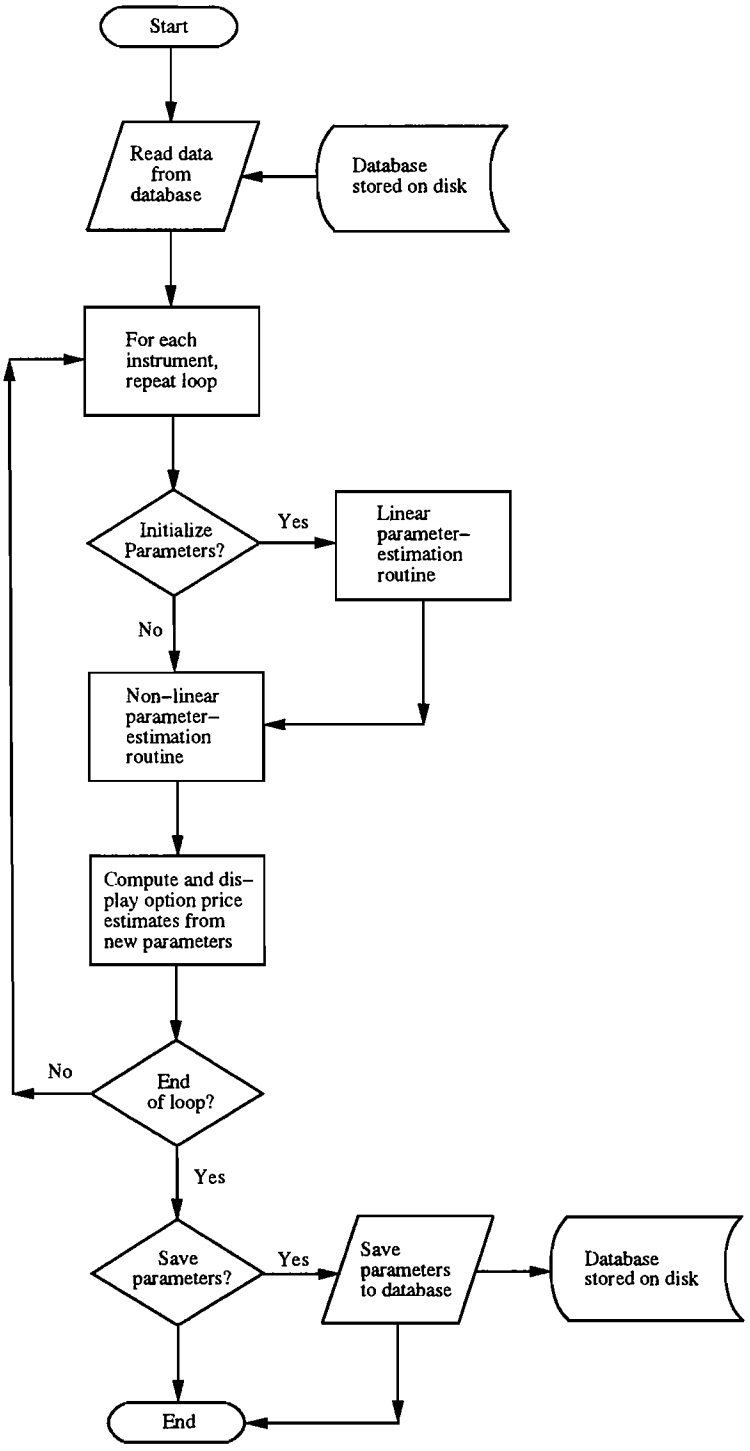

Figure C.1: Convertible Instruments Trading Program Flowchart.

C.2 Sample report output by program

```
Enter Date: 06/01/1990

Enter Interest: .0845


Date: 06/01/1990
Interest: 0.0845
Model: Kassouf
Database: test

Fordmotor  Fjan105c  expiration: 01/15/1991  type: call  strike: 105.00
ptime=4.827418   pinterest=7.457493   ptrend=-0.848223   p0=3.849772
S= 96.88  98.88 100.88 102.88 104.88 106.88 108.88 110.88 112.88 114.88 116.88
O=  1.30   1.79   2.42   3.20   4.15   5.26   6.53   7.94   9.47  11.10  12.82

Fordmotor  Fdec110c  expiration: 12/15/1990  type: call  strike: 110.00
ptime=-1.100803   pinterest=14.396595   ptrend=-0.431441   p0=0.891321
S= 96.88  98.88 100.88 102.88 104.88 106.88 108.88 110.88 112.88 114.88 116.88
O=  1.94   2.40   2.96   3.60   4.34   5.19   6.14   7.19   8.35   9.60  10.95

Navistar  NAVaw  expiration: 12/15/1993  type: warnt  strike:   5.00
ptime=0.857610   pinterest=2.979983   ptrend=-0.103293   p0=-1.223230
S=  3.25   3.38   3.50   3.62   3.75   3.88   4.00   4.12   4.25   4.38   4.50
O=  1.35   1.43   1.51   1.59   1.68   1.76   1.85   1.94   2.03   2.12   2.21
```

C.3 General purpose header files

C.3.1 instrument.h

```
#include <string.h>

struct PROGRAM
{
   char model[20];
   int window_size;
   int trend_length;
   float tolerance;
   int limit;
   int no_of_instruments;
   int no_of_interests;
};

struct INTEREST
{
   float three_month;
   float six_month;
   float nine_month;
   char date[11];
};

struct PRICE
{
   char date[11];
   float open;
   float high;
   float low;
   float close;
   float volume;
};

struct INSTRUMENT
{
   char name[15];
```

```
    char underlying[15];
    char cusip[15];
    char symbol[5];
    char type[8];
    char expiration[11];
    float conversion;
    float strike;
    float shares;
    int ndata;
    int nparam;
    float dividend;
    float dilution;
    float ptime;
    float pint;
    float ptrend;
    float pdiv;
    float pdil;
    float p0;
    struct PRICE *prices;
};
```

C.3.2 malloc.h

```
#ifndef __TURBOC__
extern  char *  malloc();
extern  char *  realloc();
void            free();
#endif

/* Allocates N objects of a given type, using malloc() */
#define MALLOCN(P,T,N,M) \
    if (((P) = (T *) malloc((unsigned) (N) * sizeof(T))) ==  NULL) { \
        fprintf(stderr, "Malloc failure: (%d bytes) %s\n", (N) * sizeof(T), M); \
        exit(1); \
    } \
    else

/* Reallocates N objects of a given type, using realloc() */
```

```
#define REALLOCN(P,PTEMP,T,N,M) \
   if ((P) == NULL) {  \
      MALLOCN((P),T,(N),M);   \
   }                       \
   else if (((PTEMP) = (T *) realloc(P,(unsigned)(N)* sizeof(T))) == NULL) {\
      fprintf(stderr, "Realloc failure: (%d bytes) %s\n", (N) * sizeof(T), M); \
      exit(1); \
   } \
   else (P) = (PTEMP)

#define FREE(P)          free((char *) P)
```

C.4 General option pricing program code (`options.c`)

```
/*************************************************************************
/*
/*                  EVALUATING CONVERTIBLE INSTRUMENTS
/*                        USING MULTIPLE REGRESSION
/*
/*
/* The purpose of this program is to evaluate convertible trading
/* instruments based on past data in order to spot and profit from
/* pricing misalignments in the market. The program is as flexible as
/* possible so that it can be easily modified and expanded to improve
/* its usefulness as its performance is evaluated.
/*
/* The program evaluates calls, puts, and warrants by regressing past
/* prices against:
/*
/*
/*                          1. time until expiration
/*                          2. interest rate
/*                          3. price trend in the underlying
/*                          4. dividend
/*                          5. dilution
/*
/*
```

```
/* Both linear and nonlinear regression are implemented. The linear
/* estimate is used to initialize the parameters (if necessary), and
/* the nonlinear estimate is used otherwise (with the previous
/* extimate as the seed). The four data structures mirror the
/* relations from which they read.
/*
/* The program is invoked by typing "options -ddatabase -mmodel -s -c"
/* where 'database' is the name of the database and 'model' is the
/* name of the option pricing model to be used (either the
/* 'hyperbolic' or 'kassouf' models). The '-s' flag tells the program
/* to save the newly computed model parameters to the database. The
/* '-c' flag tells the program to clear (and thus reinitialize) the
/* parameters.
/*********************************************************************/


#include <stdio.h>

#ifdef __TURBOC__
#include <stdlib.h>
#endif

#include <string.h>
#include <math.h>


/*********************************************************************
Include numerical recipes header files.
*********************************************************************/

#include "malloc.h"
#include "nr.h"
#include "nrutil.h"

/*********************************************************************
Define STREQ, which compares two strings.
*********************************************************************/
```

```
#define STREQ(A, B)     (*(A) == *(B)  && strcmp((A), (B)) == 0)

/*********************************************************************
Allocates N objects of a given type, using malloc().
*********************************************************************/

#define MALLOCN(P,T,N,M) \
   if (((P) = (T *) malloc((unsigned) (N) * sizeof(T))) ==  NULL) { \
      fprintf(stderr, "Malloc failure: (%d bytes) %s\n", (N) * sizeof(T), M); \
      exit(1); \
   } \
   else

#define FREE(P) free((char *) P)


/*********************************************************************
The following declaration creates the new variable type "Program."
Later, we declare a variable of this type which will hold all of the
information which governs the operation of the program. This
information will be read out of a single file and stored in this
variable. The variable type is composed of five other types:

1. model is a string (20 characters long) which specifies which model
we are fitting (Kassouf or Hyperbolic, although others could be
added).

2. window_size is an integer which specifies the number of data points
which we regress.

3. trend_length is the number of data points which we use to compute
the trend.

4. tolerance is a float which specifies the computational precision
tolerance (this is useful for the computation of gamma using Newton's
method in the routine computed_gamma).
*********************************************************************/
```

```
typedef struct {                                                    100
    char model[20];
    int window_size;
    int trend_length;
    float tolerance;
    int limit;
} Program;


/*********************************************************************
The next declaration creates the variable type "Interest." This     110
variable type is composed of four types:

1. date is an 11 character string which is the date of the associated
interest rates.

2. three_month is a float which is the three month T-Bill rate.

3. six_month is a float which is the six_ month T-Bill rate.

4. nine_month is a float which is the nine month T-Bill rate.       120
*********************************************************************/

typedef struct {
    char date[11];
    float three_month;
    float six_month;
    float nine_month;
} Interest;


                                                                    130
/*********************************************************************
This declaration creates the variable type "Prices," which is composed
of three other types:

1. date is a 11 character string which is the date of the associated
price and volume measurements.
```

2. price is a float which is the price of an instrument on that date.

3. vol is a float which is the trading volume of the instrument on that date.
**********/

```
typedef struct {
    char date[11];
    float price;
    float vol;
} Prices;
```

```
/**********
```

The Last type declaration creates the new variable type "Instrument." This type is composed of 20 other types:

1. name is a string of 15 characters which is the name of the instrument (i.e., Ford Jul110C).

2. underlying is a 15 character string which is the name of the underlying security (i.e., Ford Motor Co.). Note that this information is only useful for convertible instrument entries, although Instrument will need to also include stock entries.

3. cusip is a 15 character string which is the instruments cusip identification number (it may include characters).

4. symbol is a 5 character string which is the instruments exchange symbol (i.e., F).

5. type is a 8 character string which specifies the type of the instrument (i.e., call, put, warrant, stock, etc.)

6. expiration is a 11 character date string which gives the instruments expiration date.

7. conversion is a float which specifies the number of shares which

one convertible covers.

8. strike is the strike price of the convertible (float).

9. shares is the number of outstanding contracts for the issue (float).

10. ndata is an integer which is the number of data points which we have for the instrument

11. nparam is an integer which is the number of data points which we have for the instrument

12. dividend is the dividend yield during the life of the option (float).

13. dilution is the potential dilution of the stock (float).

14. ptime is the parameter associated with time (i.e., a1).

15. pint is the parameter associated with interest rate (i.e., a2).

16. ptrend is the parameter associated with stock trend (i.e., a3).

17. pdiv is the parameter associated with dividend yield.

18. pdil is the parameter associated with potential dilution.

19. p0 is the parameter which is a constant.

20. prices is an array of type "Prices," and it contains all of the chronological (usually daily) price information for the instrument.
**********/

```
typedef struct {
    char name[15];
    char underlying[15];
    char cusip[15];
    char symbol[5];
```

```
    char type[8];
    char expiration[11];
    float conversion;
    float strike;
    float shares;
    int ndata;
    int nparam;                                                    220
    float dividend;
    float dilution;
    float ptime;
    float pint;
    float ptrend;
    float pdiv;
    float pdil;
    float p0;
    Prices *prices;
} Instrument;                                                      230

/*********************************************************************
The following variable declarations are global to the program. We make
8 global declarations:

1. program of type program contains information which governs the
operation of the program.

2. interest is an array of type Interest which will contain all of the   240
interest rate data.

3. prices is an array of type Prices which will contain all of the
price data.

4. instrument is an array of type Instrument with each entry
containing all of the information about one specific instrument.

5. no_of_instruments is the total number of instruments in the
instrument array.                                                  250
```

```
6. no_of_interests is the total number of dates for which we have
interest rate data (stored in the interest array).

7. today is the current date.

8. current_interest is the current interest rate.
**********************************************************************/

Program program;                                                        260
Interest *interest;
Prices *prices;
Instrument *instrument;

int no_of_instruments;
int no_of_interests;
char today[9];
float current_interest;


                                                                        270
/**********************************************************************
This is the main routine which enters the current date and interest
rate; instantiates the data from the appropriate database relations
into the variables program, interest, instrument, etc.; computes the
new parameter estimates; and, finally, computes the model values.
**********************************************************************/

   int
main(argc , argv)
   int argc;                                                            280
   char ** argv;
{
   void read_data();
   void write_data();
   void clear_parameters();
   void compute();
   char s[20], database[20], model[20];
   int clear, save;
   int i;
```

```
                                                                    290
save = 0;

while (--argc > 0 && (*++argv)[0] == '-')
{
    strcpy(s,++*argv);

    if (s[0] == 'd')
        strcpy(database,&s[1]);

    if (s[0] == 's')                                                300
        save = 1;

    if (s[0] == 'c')
        clear = 1;

    if (s[0] == 'm')
        strcpy(model,&s[1]);
}

read_data(database);                                                310

if (clear == 1)
    clear_parameters(database);

if (STREQ(model,"hyperbolic"))
    strcpy(model,"Hyperbolic");

if (STREQ(model,"kassouf"))
    strcpy(model,"Kassouf");
                                                                    320
if (! STREQ(model,"Hyperbolic") && ! STREQ(model,"Kassouf"))
    printf("Invalid model -- using default model\n");

if (STREQ(model,"Hyperbolic") || STREQ(model,"Kassouf"))
    strcpy(program.model,model);

printf("Enter Date: ");
```

```
    scanf("%s", today);
    printf("\n");

    printf("Enter Interest: ");
    scanf("%f", &current_interest);
    printf("\n");

    compute(database);

    if (save == 1)
        write_data(database);
}


/*************************************************************************
This routine computes the basis function vector, p[], from the raw
data. This is for the Kassouf model.
*************************************************************************/

    void
linear_gamma(x, p, np, index)
    float x[], p[];
    int np, index;
{
    int j;

    p[1] = 1.0 / x[1];
    p[2] = 1.0 / (1.0 + x[2]);
    p[3] = x[3];

    if (instrument[index].dilution == 0 && instrument[index].dividend == 0)
        p[4] = 1.0;

    if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
    {
        p[4] = x[4];
        p[5] = 1.0;
    }
```

```
    if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
    {
        p[4] = x[4];
        p[5] = 1.0;                                                    370
    }

    if (instrument[index].dilution != 0 && instrument[index].dividend != 0)
    {
        p[4] = x[4];
        p[5] = x[5];
        p[6] = 1.0;
    }
}
                                                                       380

/*************************************************************************
This routine computes the basis function vector, p[], from the raw
data. This is for the hyperbolic model.
*************************************************************************/

    void
linear_theta(x, p, np, index)
    float x[], p[];
    int np, index;                                                     390
{
    int j;

    p[1] = x[1];
    p[2] = x[1] * x[2];
    p[3] = x[1] * x[3];
    p[4] = x[1] * x[4];
    p[5] = x[1] * x[5];
}
                                                                       400

/*************************************************************************
Computes the number of days between two dates. Uses the numerical
```

```
recipes routine julday.
*************************************************************************/

   float
days_difference(date_1, date_2)
   char date_1[12], date_2[12];
{                                                                        410
   int mm1, id1, iyyy1, mm2, id2, iyyy2;
   float difference;

   sscanf(date_1, "%d/%d/%d", &mm1, &id1, &iyyy1);
   sscanf(date_2, "%d/%d/%d", &mm2, &id2, &iyyy2);

   difference = (julday(mm2, id2, iyyy2) - julday(mm1, id1, iyyy1)) / 365.25;
   return difference;
}
                                                                         420

/*************************************************************************
This routine computes the implied gamma from s (stock price), k
(strike) and c (option price) for the Kassouf model. Since there is no
analytic form for gamma, we compute it iteratively using Newton's
method.
*************************************************************************/

   float
computed_gamma(c, s, k, old_gamma, i)                                    430
   float c, s, k, old_gamma;
   int i;
{
   float new_gamma;
   float logs, logc1;
   float s_2_gamma, c_2_gamma;

   s = s / k;
   c = c / k + 1.0;
                                                                         440
   logs = log(s);
```

```
    logc1 = log(c);

    new_gamma = old_gamma;

    i = 0;

    do {

        old_gamma = new_gamma;                                          450
        s_2_gamma = pow(s, old_gamma);
        c_2_gamma = pow(c, old_gamma);

        new_gamma = old_gamma -
            (c_2_gamma - s_2_gamma - 1.0) / (c_2_gamma * logc1
                                             - s_2_gamma * logs);

        i++;
    } while (fabs((new_gamma - old_gamma) / old_gamma) >= program.tolerance &&
             i <= program.limit);                                       460

    return new_gamma;
}


/*********************************************************************
This routine computes the implied theta from s (stock price), k
(strike) and c (option price) for the hyperbolic model.
*********************************************************************/
                                                                        470
    float
computed_theta(c, s, k)
    float c, s, k;
{
    return 4 * k * c * (c - s + k)/(s - c);
}


/*********************************************************************
```

```
This routine explicitly computes the Kassouf model option value by                480
computing gamma, and then plugging it into the Kassouf formula.
Note that p[] is the parameter vector.
**********************************************************************/

   float
computed_gamma_value(s, k, t, r, trend, div, dil, p, index)
   float s, k, t, r, trend, div, dil, p[];
   int index;
{
   float gamma;                                                           490

   if (instrument[index].dilution == 0 && instrument[index].dividend == 0)
      gamma = p[1] / t + p[2] / (1.0 + r) + p[3] * trend + p[4];

   if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
      gamma = p[1] / t + p[2] / (1.0 + r) + p[3] * trend +
         p[4] * dil + p[5];

   if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
      gamma = p[1] / t + p[2] / (1.0 + r) + p[3] * trend +                500
         p[4] * div + p[5];

   if (instrument[index].dilution != 0 && instrument[index].dividend != 0)
      gamma = p[1] / t + p[2] / (1.0 + r) + p[3] * trend +
         p[4] * div + p[5] * dil + p[6];

   return k * (pow(pow(s / k, gamma) + 1.0, 1.0 / gamma) - 1.0);
}

                                                                          510
/**********************************************************************
This routine explicitly computes the hyperbolic model option value by
computing theta, and then plugging it into the hyperbolic formula.
Note that p[] is the parameter vector.
**********************************************************************/

   float
```

```
computed_hyperbolic_value(s, k, t, r, trend, div, dil, p, index)
   float s, k, t, r, trend, div, dil, p[];
   int index;                                                          520
{
   float theta;

   if (instrument[index].dilution == 0 && instrument[index].dividend == 0)
      theta = p[1] * t + p[2] * t * r + p[3] * t * trend;

   if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
      theta = p[1] * t + p[2] * t * r + p[3] * t * trend + p[4] * t * dil;

   if (instrument[index].dilution == 0 && instrument[index].dividend != 0)   530
      theta = p[1] * t + p[2] * t * r + p[3] * t * trend + p[4] * t * div;

   if (instrument[index].dilution != 0 && instrument[index].dividend != 0)
      theta = p[1] * t + p[2] * t * r + p[3] * t * trend
         + p[4] * t * div + p[5] * t * dil;

   return (sqrt((s + theta/4/k - k) * (s + theta/4/k - k) + theta) + s -
            theta/4/k - k) / 2;
}
                                                                       540

/*********************************************************************
This routine is the function called by the numerical recipes nonlinear
fitting routine mrqmin. This routine returns the vector of derivatives
of the function (Kassouf in this case) with respect to the parameters.
It corresponds to the numerical recipes routine "funcs."
*********************************************************************/

   void
non_linear_gamma(s, k, x, a, y, dyda, na, index)                       550
   float x[], *y, a[], dyda[];
   float s, k;

   int na;
   int index;
```

```
{
    float t, r, trend, div, dil, gamma, s_to_the_g, dydg;

    t = x[1];
    r = x[2];                                                          560
    trend = x[3];
    div = 0;
    dil = 0;

    if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
        dil = x[4];

    if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
        div = x[4];
                                                                       570
    if (instrument[index].dilution != 0 && instrument[index].dividend != 0)
    {
        div = x[4];
        dil = x[5];
    }

    *y = computed_gamma_value(s, k, t, r, trend, div, dil, a, index);

    if (instrument[index].dilution == 0 && instrument[index].dividend == 0)
        gamma = a[1] / t + a[2] / (1.0 + r) + a[3] * trend + a[4];     580

    if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
        gamma = a[1] / t + a[2] / (1.0 + r) + a[3] * trend +
            a[4] * dil + a[5];

    if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
        gamma = a[1] / t + a[2] / (1.0 + r) + a[3] * trend +
            a[4] * div + a[5];

    if (instrument[index].dilution != 0 && instrument[index].dividend != 0)  590
        gamma = a[1] / t + a[2] / (1.0 + r) + a[3] * trend +
            a[4] * div + a[5] * dil + a[6];
```

```
    s_to_the_g = pow(s/k, gamma);

    dydg = -pow(s_to_the_g + 1.0, 1.0 / gamma)
        * (1.0 / gamma * log(s_to_the_g + 1.0) -
            (s_to_the_g * log(s/k) + 1.0) / (gamma * (s_to_the_g + 1.0)));

                                                                        600
    dyda[1] = dydg / t;
    dyda[2] = dydg / (1.0 + r);
    dyda[3] = dydg * trend;

    if (instrument[index].dilution == 0 && instrument[index].dividend == 0)
        dyda[4] = dydg;

    if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
    {
        dyda[4] = dydg * dil;                                           610
        dyda[5] = dydg;
    }


    if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
    {
        dyda[4] = dydg * div;
        dyda[5] = dydg;
    }
                                                                        620
    if (instrument[index].dilution != 0 && instrument[index].dividend != 0)
    {
        dyda[4] = dydg * div;
        dyda[5] = dydg * dil;
        dyda[6] = dydg;
    }
}

/*********************************************************************** 630
```

This routine is the function called by the numerical recipes nonlinear

```
fitting routine, mrqmin. This routine returns the vector of
derivatives of the function (Hyperbolic in this case) with respect to
the parameters. It corresponds to the numerical recipes routine
"funcs."
**************************************************************************/

    void
non_linear_theta(s, k, x, a, y, dyda, na, index)
    float x[], *y, a[], dyda[];                                          640
    float s, k;
    int na, index;
{
    float t, r, trend, div, dil, dydtheta;
    float radical, theta;

    t = x[1];
    r = x[2];
    trend = x[3];
    div = 0;                                                             650
    dil = 0;

    if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
        dil = x[4];

    if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
        div = x[4];

    if (instrument[index].dilution != 0 && instrument[index].dividend != 0)
    {                                                                    660
        div = x[4];
        dil = x[5];
    }

    *y = computed_hyperbolic_value(s, k, t, r, trend, div, dil, a, index);

    if (instrument[index].dilution == 0 && instrument[index].dividend == 0)
        theta = a[1] * t + a[2] * t * r + a[3] * t * trend;
```

```
    if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
        theta = a[1] * t + a[2] * t * r + a[3] * t * trend + a[4] * t * dil;

    if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
        theta = a[1] * t + a[2] * t * r + a[3] * t * trend + a[4] * t * div;

    if (instrument[index].dilution != 0 && instrument[index].dividend != 0)
        theta = a[1] * t + a[2] * t * r + a[3] * t * trend +
            a[4] * t * div + a[5] * t * dil;

    radical = sqrt((s + theta/4/k - k) * (s + theta/4/k - k) + theta);

    dydtheta = ((s + theta/4/k + k)/radical - 1)/8/k;

    dyda[1] = dydtheta * t;
    dyda[2] = dyda[1] * r;
    dyda[3] = dyda[1] * trend;

    if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
        dyda[4] = dyda[1] * dil;

    if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
        dyda[4] = dyda[1] * div;

    if (instrument[index].dilution != 0 && instrument[index].dividend != 0)
    {
        dyda[4] = dyda[1] * div;
        dyda[5] = dyda[1] * dil;
    }
}

/*********************************************************************
This routine computes the trend term.
*********************************************************************/

    float
compute_trend(stock_index, first_stock_j)
```

```
    int stock_index, first_stock_j;
{
    int i, start;                                                    710
    float s_total;

    s_total = 0;

    start = first_stock_j - program.trend_length - 1;

    if (start <= 0)
        start = 0;

    for (i = start; i < first_stock_j - 1; ++i)                      720
        s_total = s_total + instrument[stock_index].prices[i].price;

    return program.trend_length *
        instrument[stock_index].prices[first_stock_j - 1].price /s_total;
}


/*********************************************************************
This routine computes the parameter vector for each instrument, and
then loops through and computes the model values.                    730
*********************************************************************/

    void
compute(database)

    char database[20];

{
    void parameter_estimate();
    int i, index, k, j, w, first_stock_j, last_j;                    740
    int interest_index, stock_index;
    int first_j;
    int last_stock_j;
    float strike, s, t, r, trend, div, dil, c;
    float incr, bound;
```

```
float p[6], last_stock_price;

parameter_estimate();

printf("\nDate: %s\n", today);                                          750
printf("Interest: %5.4f\n", current_interest);
printf("Model: %s\n",program.model);
printf("Database: %s\n\n",database);

for (i = 0; i < no_of_instruments; i++) {
    index = i;
    if (! STREQ(instrument[index].type, "stock"))
    {
        strike = instrument[index].strike;
        first_j = instrument[index].ndata - program.window_size;          760
        last_j = instrument[index].ndata;

        p[1] = instrument[i].ptime;
        p[2] = instrument[i].pint;
        p[3] = instrument[i].ptrend;

        if (instrument[i].dividend == 0 && instrument[i].dilution == 0)
            p[4] = instrument[i].p0;

        if (instrument[i].dividend == 0 && instrument[i].dilution != 0)   770
        {
            p[4] = instrument[i].pdil;
            p[5] = instrument[i].p0;
        }

        if (instrument[i].dividend != 0 && instrument[i].dilution == 0)
        {
            p[4] = instrument[i].pdiv;
            p[5] = instrument[i].p0;
        }                                                                 780

        if (instrument[i].dividend != 0 && instrument[i].dilution != 0)
        {
```

```
    p[4] = instrument[i].pdiv;
    p[5] = instrument[i].pdil;
    p[6] = instrument[i].p0;
}


for (k = index; ;k--)                                                    790
    if (STREQ(instrument[index].underlying, instrument[k].symbol)) {
        stock_index = k;    break;
    }

first_stock_j = instrument[stock_index].ndata - program.window_size;
interest_index = no_of_interests - program.window_size;

printf("%s  %s  expiration: %s  type: %s  strike: %6.2f\n",
        instrument[stock_index].name,
        instrument[index].name,                                          800
        instrument[index].expiration,
        instrument[index].type,
        instrument[index].strike);

printf("ptime=%f    pinterest=%f    ptrend=%f",
        instrument[index].ptime,
        instrument[index].pint,
        instrument[index].ptrend);

if (instrument[index].dividend != 0)                                     810
    printf("    pdiv=%f", instrument[index].pdiv);

if (instrument[index].dilution != 0)
    printf("    pdil=%f", instrument[index].pdil);

if (STREQ(program.model, "Kassouf"))
    printf("    p0=%f", instrument[index].p0);

printf("\n");
                                                                         820
printf("S=");
```

```
incr = 2;
last_stock_j = last_j - first_j + first_stock_j - 1;
last_stock_price = instrument[stock_index].prices[last_stock_j].price;

if (last_stock_price <= 100)
    incr = 1.0;

if (last_stock_price <= 50)                                          830
    incr = .5;

if (last_stock_price <= 25)
    incr = .25;

if (last_stock_price <= 12.5)
    incr = .125;

bound = incr * 5;
                                                                     840
for (s = instrument[stock_index].prices[last_stock_j].price - bound;
      s <= instrument[stock_index].prices[last_stock_j].price + bound;
      s += incr)
    printf("%6.2f ", s);

printf("\n");

printf("O=");

trend = compute_trend(stock_index, instrument[stock_index].ndata);   850

t = days_difference(today, instrument[index].expiration);
r = current_interest;

for (s = instrument[stock_index].prices[last_stock_j].price - bound;
      s <= instrument[stock_index].prices[last_stock_j].price + bound;
      s += incr)
{
    if (STREQ(program.model, "Kassouf") &&
```

```
                ! STREQ(instrument[index].type, "put"))                    860
            printf("%6.2f ",
                    computed_gamma_value(s, strike, t, r,
                                         trend, div, dil, p, index));

        if (STREQ(program.model, "Kassouf") &&
            STREQ(instrument[index].type, "put"))
            printf("%6.2f ",
                    computed_gamma_value(s, strike, t, r, trend, div, dil,
                                         p, index) - s + strike);
                                                                           870
        if (STREQ(program.model, "Hyperbolic") &&
            ! STREQ(instrument[index].type, "put"))
            printf("%6.2f ",
                    computed_hyperbolic_value(s, strike, t, r,
                                              trend, div, dil, p, index));

        if (STREQ(program.model, "Hyperbolic") &&
            STREQ(instrument[index].type, "put"))
            printf("%6.2f ",
                    computed_hyperbolic_value(s, strike, t, r,             880
                                              trend, div, dil, p, index) -
                    s + strike);
        }
        printf("\n\n");
    }
  }
}


/*************************************************************************** 890
This routine loops through all of the convertible instruments in the
instrument array and computes the new parameter vector. If the
parameter vector has not been initialized, then the linear estimate is
used to initialize the parameters. Once we have an initialized
parameter vector, then the nonlinear estimate is used.
***************************************************************************/
```

```
   void
parameter_estimate()
{                                                                    900
   void linear_estimate();
   void non_linear_estimate();
   int i;

   for (i = 0; i < no_of_instruments; i++)
   {
      if (! STREQ(instrument[i].type, "stock") &&
           instrument[i].ptime == 0 &&
           instrument[i].pint == 0 &&
           instrument[i].ptrend == 0 &&                              910
           instrument[i].pdiv == 0 &&
           instrument[i].pdil == 0 &&
           instrument[i].p0 == 0)

         linear_estimate(i);

      if (! STREQ(instrument[i].type, "stock"))
         non_linear_estimate(i);
   }
}                                                                    920


/*********************************************************************
This routine sets up the appropriate vectors and matrices and then
calls the numerical recipes linear fitting routine, lfit. The
estimated parameter vector is then instantiated into the instrument
array.
*********************************************************************/

   void                                                              930
linear_estimate(index)
   int index;
{
   int first_j, first_stock_j;
   int i, j, m;
```

```
int nparam;
int last_j;
int stock_index;
int first_stock_index;
int next_index;                                                    940
int interest_index;

int lista[6];
float chisq;
float s, k;

float **covar;
float *sig;
float **x, *y;
float *a;                                                          950

nparam = 6;

if (STREQ(program.model, "Hyperbolic"))
    nparam = 5;

if (instrument[index].dilution == 0)
    nparam = nparam - 1;

if (instrument[index].dividend == 0)                               960
    nparam = nparam - 1;

x = matrix(1, program.window_size, 1, 6);
y = vector(1, program.window_size);
a = vector(1, nparam);

sig = vector(1, program.window_size);
covar = matrix(1, nparam, 1, nparam);

first_j = instrument[index].ndata - program.window_size;           970
last_j = instrument[index].ndata;

for (m = index; ; m--)
```

```
    if (STREQ(instrument[index].underlying, instrument[m].symbol)) {
        stock_index = m; break;
    }

first_stock_j = instrument[stock_index].ndata - program.window_size;
interest_index = no_of_interests - program.window_size;

                                                                        980
for (j = first_j; j < last_j; j++)
{

    x[j - first_j + 1][1] = days_difference(instrument[index].prices[j].date,
                                            instrument[index].expiration);
    x[j - first_j + 1][2] = interest[interest_index].three_month;
    x[j - first_j + 1][3] = compute_trend(stock_index, first_stock_j);
    x[j - first_j + 1][4] = instrument[index].dividend;
    x[j - first_j + 1][5] = instrument[index].dilution;
    x[j - first_j + 1][6] = 1.0;                                        990

    ++first_stock_j;
    ++interest_index;
}

if (instrument[index].dilution == 0 && instrument[index].dividend == 0)
{
    for (j = first_j; j < last_j; j++)
    {
        x[j - first_j + 1][4] = 1.0;                                    1000
    }
}

if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
{
    for (j = first_j; j < last_j; j++)
    {
        x[j - first_j + 1][4] = instrument[index].dilution;
        x[j - first_j + 1][5] = 1.0;
    }                                                                   1010
}
```

```
if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
{
    for (j = first_j; j < last_j; j++)
    {
        x[j - first_j + 1][4] = instrument[index].dividend;
        x[j - first_j + 1][5] = 1.0;
    }
}                                                                    1020

if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
{
    for (j = first_j; j < last_j; j++)
    {
        x[j - first_j + 1][4] = instrument[index].dividend;
        x[j - first_j + 1][5] = instrument[index].dilution;
        x[j - first_j + 1][6] = 1.0;
    }
}                                                                    1030


first_stock_j = instrument[stock_index].ndata - program.window_size;

for (j = first_j; j < last_j; j++)
{
    if (STREQ(program.model, "Kassouf"))
        y[j - first_j + 1] =
            computed_gamma(instrument[index].prices[j].price,
                           instrument[stock_index].prices[first_stock_j].price,   1040
                           instrument[index].strike, 1.0, 0);

    if (STREQ(program.model, "Hyperbolic"))
        y[j - first_j + 1] =
            computed_theta(instrument[index].prices[j].price,
                           instrument[stock_index].prices[first_stock_j].price,
                           instrument[index].strike, 1.0, 0);
    ++first_stock_j;
}
```

```
                                                                    1050
for (i = 1; i <= program.window_size; i++)
    sig[i] = 1.0;

for (i = 1; i <= nparam; i++)
    lista[i] = i;

if (STREQ(program.model, "Kassouf"))
    lfit(x, y, sig, program.window_size, a, nparam, lista, nparam,
         covar, &chisq, linear_gamma, index);
                                                                    1060
if (STREQ(program.model, "Hyperbolic"))
    lfit(x, y, sig, program.window_size, a, nparam, lista, nparam,
         covar, &chisq, linear_theta, index);

instrument[index].ptime = a[1];
instrument[index].pint = a[2];
instrument[index].ptrend = a[3];

next_index = 4;
                                                                    1070
if (instrument[index].dividend != 0)
{
    instrument[index].pdiv = a[next_index];
    next_index = next_index + 1;
}

if (instrument[index].dilution != 0)
{
    instrument[index].pdil = a[next_index];
    next_index = next_index + 1;                                    1080
}

if (STREQ(program.model, "Kassouf"))
    instrument[index].p0 = a[next_index];

free_vector(y, 1, program.window_size);
free_vector(a, 1, nparam);
```

```
    free_vector(sig, 1, program.window_size);

    free_matrix(x, 1, program.window_size, 1, 6);                    1090
    free_matrix(covar, 1, nparam, 1, nparam);
}


/*************************************************************************
This routine sets up the appropriate vectors and matrices and then
calls a modified version of the numerical recipes nonlinear fitting
routine, mrqmin. The estimated parameter vector is then instantiated
into the instrument array.
*************************************************************************/                                                    1100

    void
non_linear_estimate(index)
    int index;
{
    int first_j, first_stock_j;
    int i, j, m;
    int last_j;
    int nparam;
    int stock_index;                                                 1110
    int first_stock_index;
    int interest_index;
    int next_index;

    int lista[6];
    float chisq;
    float alamda;
    float k;

    float **covar;                                                   1120
    float **alpha;
    float *sig;
    float **x, *y;
    float *a;
    float *s;
```

```
nparam = 6;

if (STREQ(program.model, "Hyperbolic"))
    nparam = 5;                                                    1130

if (instrument[index].dilution == 0)
    nparam = nparam - 1;

if (instrument[index].dividend == 0)
    nparam = nparam - 1;

x = matrix(1, program.window_size, 1, 6);
alpha = matrix(1, nparam, 1, nparam);
covar = matrix(1, nparam, 1, nparam);                              1140

y = vector(1, program.window_size);
s = vector(1, program.window_size);
a = vector(1, nparam);
sig = vector(1, program.window_size);

a[1] = instrument[index].ptime;
a[2] = instrument[index].pint;
a[3] = instrument[index].ptrend;
                                                                   1150
next_index = 4;

if (instrument[index].dividend != 0)
{
    a[4] = instrument[index].pdiv;
    next_index = next_index + 1;
}

if (instrument[index].dilution != 0)
{                                                                  1160
    a[next_index] = instrument[index].pdil;
    next_index = next_index + 1;
}
```

```
if (STREQ(program.model, "Kassouf"))
    a[next_index] = instrument[index].p0;

first_j = instrument[index].ndata - program.window_size;
last_j = instrument[index].ndata;

for (m = index; ; m--)
    if (STREQ(instrument[index].underlying, instrument[m].symbol))
    {
        stock_index = m;
        break;
    }

first_stock_j = instrument[stock_index].ndata - program.window_size;
interest_index = no_of_interests - program.window_size;

for (j = first_j; j < last_j; j++)
{
    x[j - first_j + 1][1] = days_difference(instrument[index].prices[j].date,
                                            instrument[index].expiration);
    x[j - first_j + 1][2] = interest[interest_index].three_month;
    x[j - first_j + 1][3] = compute_trend(stock_index, first_stock_j);
    x[j - first_j + 1][4] = instrument[index].dividend;
    x[j - first_j + 1][5] = instrument[index].dilution;
    x[j - first_j + 1][6] = 1.0;

    s[j - first_j + 1] = instrument[stock_index].prices[first_stock_j].price;

    ++first_stock_j;
    ++interest_index;
}

k = instrument[index].strike;

if (instrument[index].dilution != 0 && instrument[index].dividend == 0)
{
    for (j = first_j; j < last_j; j++)
```

```
    {
        x[j - first_j + 1][4] = instrument[index].dilution;
        x[j - first_j + 1][5] = 1.0;
    }
}

if (instrument[index].dilution == 0 && instrument[index].dividend != 0)
    for (j = first_j; j < last_j; j++)
        x[j - first_j + 1][5] = 1.0;                                    1210

if (instrument[index].dilution == 0 && instrument[index].dividend == 0)
{
    for (j = first_j; j < last_j; j++)
    {
        x[j - first_j + 1][4] = 1.0;
    }
}

                                                                        1220
for (j = first_j; j < last_j; j++)
    y[j - first_j + 1] = instrument[index].prices[j].price;

for (i = 1; i <= program.window_size; i++)
    sig[i] = 1.0;

for (i = 1; i <= nparam; i++)
    lista[i] = i;

alamda = -1.0;                                                          1230

if (STREQ(program.model, "Kassouf"))
    mrqmin(s, k, x, y, sig, program.window_size, a, nparam,
            lista, nparam, covar, alpha, &chisq, non_linear_gamma,
            &alamda, index);

if (STREQ(program.model, "Hyperbolic"))
    mrqmin(s, k, x, y, sig, program.window_size, a, nparam,
            lista, nparam, covar, alpha, &chisq, non_linear_theta,
```

```
                &alamda, index);                                    1240

    instrument[index].ptime = a[1];
    instrument[index].pint = a[2];
    instrument[index].ptrend = a[3];

    next_index = 4;

    if (instrument[index].dividend != 0)
    {
        instrument[index].pdiv = a[next_index];                     1250
        next_index = next_index + 1;
    }

    if (instrument[index].dilution != 0)
    {
        instrument[index].pdil = a[next_index];
        next_index = next_index + 1;
    }

    if (STREQ(program.model, "Kassouf"))                            1260
        instrument[index].p0 = a[next_index];

    free_vector(y, 1, program.window_size);
    free_vector(a, 1, nparam);
    free_vector(sig, 1, program.window_size);

    free_matrix(x, 1, program.window_size, 1, 6);
    free_matrix(covar, 1, nparam, 1, nparam);
    free_matrix(alpha, 1, nparam, 1, nparam);
}                                                                   1270
```

C.5 INGRES interface code

C.5.1 Makefile

```
NAME     =  ioptions

C_FILES  =  ioptions.c gaussj.c mrqcof.c julday.c mrqmin.c covsrt.c lfit.c\
            nrutil.c

O_FILES  =  ioptions.o gaussj.o mrqcof.o julday.o mrqmin.o covsrt.o lfit.o\
            nrutil.o

INC_FILES =  malloc.h nr.h nrutil.h
SRC_FILES =    $(INC_FILES) $(C_FILES)                                    10

CC       =  cc
CFLAGS   =  -g
ALL_LIBS =    -lm -lc -lq
LIBS_DIR =


all:             $(NAME)

$(NAME):         $(O_FILES)                                               20
                 $(CC) $(CFLAGS) $(CPPFLAGS) -o $(NAME) $(O_FILES) \
                 $(ALL_LIBS) $(LIBS_DIR)

clean:   ; /bin/rm -f \#*\#* ~ $(NAME) $(O_FILES) core
purge:   ; /bin/rm -f \#*\#* ~ core
print:   ; enscript -2r -Pps1 $(INC_FILES) $(C_FILES)
```

C.5.2 Option pricing interface program (ioptions.q)

```
/*********************************************************************
This code interfaces INGRES and the INGRES version of the financial
database to the option pricing program.
*********************************************************************/

#include "options.c"

##      char    DATE[11];
##      float   THREE;
##      float   SIX;                                                    10
##      float   NINE;

##      char    MODEL[21];
##      int     WINDOW_SIZE;
##      int     TREND_LENGTH;
##      float   TOLERANCE;
##      int     LIMIT;

##      float   PRICE;
##      float   VOL;                                                    20

##      char    NAME[16];
##      char    UNDERLYING[16];
##      char    CUSIP[16];
##      char    SYMBOL[6];
##      char    TYPE[9];
##      char    EXPIRATION[11];
##      float   CONVERSION;
##      float   STRIKE;
##      float   SHARES;                                                 30
##      int     NDATA;
##      int     NPARAM;
##      float   DIVIDEND;
##      float   DILLUTION;
##      float   PTIME;
##      float   PINT;
```

```
##      float   PTREND;
##      float   PDIV;
##      float   PDIL;
##      float   P0;                                                    40

##      char    INST[16];
##      char    DB[21];

/*********************************************************************
This routine reads the data from the appropriate relations in the
database and instantiates the variables.
*********************************************************************/

   void                                                                50
read_data(database)

   char database[20];
{
   int   i, j, k;

/*********************************************************************
First, we read in the interest rate data.
*********************************************************************/
                                                                       60
   no_of_interests = 0;
   strcpy(DB,database);

##      ingres DB
##      retrieve (DATE=interest.date)
##      {
            no_of_interests = no_of_interests + 1;
##      }

   MALLOCN(interest, Interest, no_of_interests, "Malloc 1");           70

   i = 0;

##      retrieve (DATE=interest.date,THREE=interest.
```

```
##                          three,SIX=interest.six,NINE=interest.nine)
##      {
            strcpy(interest[i].date,DATE);
            interest[i].three_month = THREE;
            interest[i].six_month = SIX;
            interest[i].nine_month = NINE;                                    80
            i++;
##      }


/*************************************************************************
We read in the program data.
*************************************************************************/

##      retrieve (MODEL=program.model,
##              WINDOW_SIZE=program.window_size,                            90
##              TREND_LENGTH=program.trend_length,
##              TOLERANCE=program.tolerance,
##              LIMIT=program.limit)

   strcpy(program.model,MODEL);
   program.window_size = WINDOW_SIZE;
   program.trend_length = TREND_LENGTH;
   program.tolerance = TOLERANCE;
   program.limit = LIMIT;
                                                                            100
/*************************************************************************
Next, we read in the instument data.
*************************************************************************/

   no_of_instruments = 0;

##      retrieve (NAME=instrument.name)
##      {
            no_of_instruments = no_of_instruments + 1;
##      }                                                                   110

   MALLOCN(instrument, Instrument, no_of_instruments, "Malloc 2");
```

```
    i = 0;

##      retrieve (NAME=instrument.name,
##                UNDERLYING=instrument.underlying,
##                CUSIP=instrument.cusip,
##                SYMBOL=instrument.symbol,
##                TYPE=instrument.type,                                    120
##                EXPIRATION=instrument.expiration,
##                CONVERSION=instrument.conversion,
##                STRIKE=instrument.strike,
##                SHARES=instrument.shares,
##                NDATA=instrument.ndata,
##                NPARAM=instrument.nparam,
##                DIVIDEND=instrument.dividend,
##                DILLUTION=instrument.dillution,
##                PTIME=instrument.ptime,
##                PINT=instrument.pint,                                    130
##                PTREND=instrument.ptrend,
##                PDIV=instrument.pdiv,
##                PDIL=instrument.pdil,
##                P0=instrument.p0)
##      {
            for (j = 0; j < 14; ++j)
                if (NAME[j] != ' ')
                    instrument[i].name[j] = NAME[j];

            for (j = 0; j < 14; ++j)                                     140
                if (UNDERLYING[j] != ' ')
                    instrument[i].underlying[j] = UNDERLYING[j];

            for (j = 0; j < 14; ++j)
                if (CUSIP[j] != ' ')
                    instrument[i].cusip[j] = CUSIP[j];

            for (j = 0; j < 4; ++j)
                if (SYMBOL[j] != ' ')
                    instrument[i].symbol[j] = SYMBOL[j];                 150
```

```
            for (j = 0; j < 7; ++j)
               if (TYPE[j] != ' ')
                  instrument[i].type[j] = TYPE[j];

            for (j = 0; j < 10; ++j)
               if (EXPIRATION[j] != ' ')
                  instrument[i].expiration[j] = EXPIRATION[j];

            instrument[i].conversion = CONVERSION;                    160
            instrument[i].strike = STRIKE;
            instrument[i].shares = SHARES;
            instrument[i].ndata = NDATA;
            instrument[i].nparam = NPARAM;
            instrument[i].dividend = DIVIDEND;
            instrument[i].dilution = DILLUTION;
            instrument[i].ptime = PTIME;
            instrument[i].pint = PINT;
            instrument[i].ptrend = PTREND;
            instrument[i].pdiv = PDIV;                                170
            instrument[i].pdil = PDIL;
            instrument[i].p0 = P0;
            i++;
##       }

   for (i = 0; i < no_of_instruments; ++i)
   {
      MALLOCN(instrument[i].prices, Prices, instrument[i].ndata, "Malloc 3");

      strcpy(INST,instrument[i].name);                                180

      j = 0;

##       retrieve(DATE = INST.date,
##                PRICE = INST.price,
##                VOL   = INST.vol)
##       {
            strcpy(instrument[i].prices[j].date,DATE);
```

```
            instrument[i].prices[j].price = PRICE;
            instrument[i].prices[j].vol = VOL;                    190
            j++;
##      }
    }
}


/*********************************************************************
This routine writes the newly computed parameters into the database.
*********************************************************************/
                                                                  200
    void
write_data(database)

    char database[20];
{
    int i, j;

    for (i = 0; i < no_of_instruments + 1; ++i)
    {
##      {                                                         210

            strcpy(NAME,instrument[i].name);
            PTIME = instrument[i].ptime;
            PINT = instrument[i].pint;
            PTREND = instrument[i].ptrend;
            PDIV = instrument[i].pdiv;
            PDIL = instrument[i].pdil;
            P0 = instrument[i].p0;
##      }
                                                                  220
##      replace instrument(ptime=PTIME,
##                         pint=PINT,
##                         ptrend=PTREND,
##                         pdiv=PDIV,
##                         pdil=PDIL,
##                         p0=P0)
```

```
##          where instrument.name=NAME
            }
}
                                                                    230

/*********************************************************************
This routine resets all of the parameters in the database to zero.
This causes the parameters to be reiniitialized by the linear fitting
routine. This routine should therefore be invoked when the parameters
need to be reset. .
*********************************************************************/

    void
clear_parameters(database)                                          240

    char database[20];

{
    int i, j;

    for (i = 0; i < no_of_instruments; ++i)
    {
##      {
            strcpy(NAME,instrument[i].name);                        250
            PTIME = 0.0;
            PINT = 0.0;
            PTREND = 0.0;
            PDIV = 0.0;
            PDIL = 0.0;
            P0 = 0.0;

            instrument[i].ptime = 0.0;
            instrument[i].pint = 0.0;
            instrument[i].ptrend = 0.0;                             260
            instrument[i].pdiv = 0.0;
            instrument[i].pdil = 0.0;
            instrument[i].p0 = 0.0;
##      }
```

```
##      replace instrument(ptime=PTIME,
##                         pint=PINT,
##                         ptrend=PTREND,
##                         pdiv=PDIV,
##                         pdil=PDIL,                                    270
##                         p0=P0)
##      where instrument.name=NAME

        }
}
```

C.6 ObjectStore interface code

C.6.1 Makefile

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk

OS_COMPILATION_SCHEMA_DB_PATH= /$(LOGNAME)/db1.comp_schema
OS_APPLICATION_SCHEMA_DB_PATH= /$(LOGNAME)/db1.app_schema

LDLIBS = -los -losc -lq -lm -lc -lg

SOURCES = db_make.c ooptions.c gaussj.c mrqcof.c julday.c mrqmin.c covsrt.c\
 lfit.c nrutil.c options_schema.cc
                                                                          10
OBJECTS =  db_make.o ooptions.o gaussj.o mrqcof.o julday.o mrqmin.o covsrt.o\
lfit.o nrutil.o options_schema.o

EXECUTABLES = db_make ooptions

CPPFLAGS = -I$(OS_ROOTDIR)/include
CFLAGS = -g

LDFLAGS= $(OS_EXPORT)
CC = cc                                                                   20
```

```
all: $(EXECUTABLES)

db_make: db_make.o schema_standin
        $(OS_PRELINK) .os_options_schema.cc \
            $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH)
            db_make.o $(LDLIBS)
        OSCC -c .os_options_schema.cc
        $(CC) $(CFLAGS) $(LDFLAGS) -o db_make db_make.o .os_options_schema.o\
         $(LDLIBS)                                                          30

db_make.o: db_make.c
        $(CC) $(CPPFLAGS) $(CFLAGS) -c db_make.c

ooptions: ooptions.o ooptions.o gaussj.o mrqcof.o julday.o mrqmin.o covsrt.o\
        lfit.o nrutil.o schema_standin
        $(OS_PRELINK) .os_options_schema.cc \
            $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH)
            ooptions.o ooptions.o gaussj.o mrqcof.o julday.o mrqmin.o covsrt.o\
        lfit.o nrutil.o $(LDLIBS)                                           40
        OSCC -c .os_options_schema.cc
        $(CC) $(CFLAGS) $(LDFLAGS) -o ooptions ooptions.o gaussj.o mrqcof.o\
         julday.o mrqmin.o covsrt.o lfit.o nrutil.o .os_options_schema.o\
         $(LDLIBS)

ooptions.o: ooptions.c
        $(CC) $(CPPFLAGS) $(CFLAGS) -c ooptions.c

schema_standin: options_schema.cc
        OSCC -batch_schema $(OS_COMPILATION_SCHEMA_DB_PATH) options_schema.cc  50
        touch schema_standin

clean:
        osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
        rm -f $(EXECUTABLES) $(OBJECTS) schema_standin

depend:
        osmakedep .depend $(CPPFLAGS) -files $(SOURCES)
```

```
include .depend                                                    60
```

C.6.2 Database schema code (options_schema.cc)

```
#include <ostore/ostore.hh>
#include <ostore/manschem.hh>

#include "instrument.h"

static void dummy ()
{
    OS_MARK_SCHEMA_TYPE(PROGRAM);
    OS_MARK_SCHEMA_TYPE(INTEREST);
    OS_MARK_SCHEMA_TYPE(PRICE);                                    10
    OS_MARK_SCHEMA_TYPE(INSTRUMENT);
}
```

C.6.3 Database creation code

Code to read INGRES financial database into RAM (ingres_read.q)

```
/*************************************************************************
This code reads the INGRES version of the options database into RAM.
*************************************************************************/

#include <stdio.h>
#include <string.h>

#define STREQ(A, B)     (*(A) == *(B)  && strcmp((A), (B)) == 0)

#define MALLOCN(P,T,N,M) \                                         10
    if (((P) = (T *) malloc((unsigned) (N) * sizeof(T))) ==  NULL) { \
        fprintf(stderr, "Malloc failure: (%d bytes) %s\n", (N) * sizeof(T), M); \
        exit(1); \
    } \
    else
```

```
#define FREE(P) free((char *) P)

typedef struct {
    char model[20];
    int window_size;
    int trend_length;
    float tolerance;
    int limit;
} Program;

typedef struct {
    char date[11];
    float three_month;
    float six_month;
    float nine_month;
} Interest;

typedef struct {
    char date[11];
    float price;
    float vol;
} Prices;

typedef struct {
    char name[15];
    char underlying[15];
    char cusip[15];
    char symbol[5];
    char type[8];
    char expiration[11];
    float conversion;
    float strike;
    float shares;
    int ndata;
    int nparam;
    float dividend;
    float dilution;
```

```
    float ptime;
    float pint;
    float ptrend;
    float pdiv;
    float pdil;
    float p0;
    Prices *prices;                                          60
} Instrument;

Program program;
Interest *interest;
Prices *prices;
Instrument *instrument;

int no_of_instruments;
int no_of_interests;
char today[9];                                               70
float current_interest;

void read_ingres_data(database)

    char database[20];
{

##      char    DATE[11];
##      float   THREE;
##      float   SIX;                                         80
##      float   NINE;

##      char    MODEL[21];
##      int     WINDOW_SIZE;
##      int     TREND_LENGTH;
##      float   TOLERANCE;
##      int     LIMIT;

##      float   PRICE;
##      float   VOL;                                         90
```

```
##      char    NAME[16];
##      char    UNDERLYING[16];
##      char    CUSIP[16];
##      char    SYMBOL[6];
##      char    TYPE[9];
##      char    EXPIRATION[11];
##      float   CONVERSION;
##      float   STRIKE;
##      float   SHARES;                                          100
##      int     NDATA;
##      int     NPARAM;
##      float   DIVIDEND;
##      float   DILLUTION;
##      float   PTIME;
##      float   PINT;
##      float   PTREND;
##      float   PDIV;
##      float   PDIL;
##      float   P0;                                              110

##      char    INST[16];
##      char    DB[21];

    int   i, j, k;

/*************************************************************************
First, we read in the interest rate data.
*************************************************************************/
                                                                 120
    no_of_interests = 0;
    strcpy(DB,database);

##      ingres DB
##      retrieve (DATE=interest.date)
##      {
            no_of_interests = no_of_interests + 1;
##      }
```

```
    MALLOCN(interest, Interest, no_of_interests, "Malloc 1");                    130

    i = 0;

##      retrieve (DATE=interest.date,THREE=interest.
##                          three,SIX=interest.six,NINE=interest.nine)
##      {
            strcpy(interest[i].date,DATE);
            interest[i].three_month = THREE;
            interest[i].six_month = SIX;
            interest[i].nine_month = NINE;                                       140
            i++;
##      }


/*************************************************************************
We read in the program data.
*************************************************************************/

##      retrieve (MODEL=program.model,
##                WINDOW_SIZE=program.window_size,                           150
##                TREND_LENGTH=program.trend_length,
##                TOLERANCE=program.tolerance,
##                LIMIT=program.limit)

    strcpy(program.model,MODEL);
    program.window_size = WINDOW_SIZE;
    program.trend_length = TREND_LENGTH;
    program.tolerance = TOLERANCE;
    program.limit = LIMIT;
                                                                             160
/*************************************************************************
Next, we read in the instument data.
*************************************************************************/

    no_of_instruments = 0;

##      retrieve (NAME=instrument.name)
```

```
##      {
            no_of_instruments = no_of_instruments + 1;
##      }                                                                    170

    MALLOCN(instrument, Instrument, no_of_instruments, "Malloc 2");

    i = 0;

##      retrieve (NAME=instrument.name,
##                UNDERLYING=instrument.underlying,
##                CUSIP=instrument.cusip,
##                SYMBOL=instrument.symbol,
##                TYPE=instrument.type,                                      180
##                EXPIRATION=instrument.expiration,
##                CONVERSION=instrument.conversion,
##                STRIKE=instrument.strike,
##                SHARES=instrument.shares,
##                NDATA=instrument.ndata,
##                NPARAM=instrument.nparam,
##                DIVIDEND=instrument.dividend,
##                DILLUTION=instrument.dillution,
##                PTIME=instrument.ptime,
##                PINT=instrument.pint,                                      190
##                PTREND=instrument.ptrend,
##                PDIV=instrument.pdiv,
##                PDIL=instrument.pdil,
##                P0=instrument.p0)
##      {

            for (j = 0; j < 14; ++j)
                if (NAME[j] != ' ')
                    instrument[i].name[j] = NAME[j];

                                                                             200
            for (j = 0; j < 14; ++j)
                if (UNDERLYING[j] != ' ')
                    instrument[i].underlying[j] = UNDERLYING[j];

            for (j = 0; j < 14; ++j)
```

```
            if (CUSIP[j] != ' ')
                instrument[i].cusip[j] = CUSIP[j];

        for (j = 0; j < 4; ++j)
            if (SYMBOL[j] != ' ')                                    210
                instrument[i].symbol[j] = SYMBOL[j];

        for (j = 0; j < 7; ++j)
            if (TYPE[j] != ' ')
                instrument[i].type[j] = TYPE[j];

        for (j = 0; j < 10; ++j)
            if (EXPIRATION[j] != ' ')
                instrument[i].expiration[j] = EXPIRATION[j];
                                                                     220
        instrument[i].conversion = CONVERSION;
        instrument[i].strike = STRIKE;
        instrument[i].shares = SHARES;
        instrument[i].ndata = NDATA;
        instrument[i].nparam = NPARAM;
        instrument[i].dividend = DIVIDEND;
        instrument[i].dilution = DILLUTION;
        instrument[i].ptime = PTIME;
        instrument[i].pint = PINT;
        instrument[i].ptrend = PTREND;                               230
        instrument[i].pdiv = PDIV;
        instrument[i].pdil = PDIL;
        instrument[i].p0 = P0;

        i++;

##      }

    for (i = 0; i < no_of_instruments; ++i)
    {                                                                240
        MALLOCN(instrument[i].prices, Prices, instrument[i].ndata, "Malloc 3");

        strcpy(INST,instrument[i].name);
```

```
        j = 0;

##      retrieve(DATE = INST.date,
##               PRICE = INST.price,
##               VOL   = INST.vol)
##      {                                                          250
            strcpy(instrument[i].prices[j].date,DATE);
            instrument[i].prices[j].price = PRICE;
            instrument[i].prices[j].vol = VOL;
            j++;

##      }

    }
}
```

Code to create ObjectStore financial database (`db_make.q`)

```
/*********************************************************************
This program creates the ObjectStore version of the options database
from the INGRES original.
*********************************************************************/

#include <ostore/ostore.h>
#include "instrument.h"
#include "ingres_read.c"

database *db1;                                                       10
int i, j;
int no_of_interests;

main(argc , argv)
    int argc;
    char ** argv;
{
    os_typespec * PROGRAM_type;
```

```
os_typespec * INTEREST_type;
os_typespec * INSTRUMENT_type;                                          20
os_typespec * PRICE_type;
printf("reading in data from INGRES. . .\n");
read_ingres_data("mytest");
printf("done.\n\n");

printf("creating ObjectStore database. . . .\n");

start_objectstore ();

db1 = database_lookup_open(argv[1], 0, 0664);                           30

PROGRAM_type = alloc_typespec ("PROGRAM", 0);
INTEREST_type = alloc_typespec ("INTEREST", 0);
PRICE_type = alloc_typespec ("PRICE", 0);
INSTRUMENT_type = alloc_typespec ("INSTRUMENT", 0);

OS_BEGIN_TXN(t1,0,transaction_update)
{
   database_root * PROGRAM_root;
   struct PROGRAM * PROGRAM;                                            40
   PROGRAM_root = database_create_root (db1, "PROGRAM_entry");

   PROGRAM = (struct PROGRAM *)objectstore_alloc (PROGRAM_type,1,db1);

   strcpy(PROGRAM->model,program.model);
   PROGRAM->window_size = program.window_size;
   PROGRAM->trend_length = program.trend_length;
   PROGRAM->tolerance = program.tolerance;
   PROGRAM->limit = program.limit;
   PROGRAM->no_of_instruments = 5;                                      50
   PROGRAM->no_of_interests = 24;
   no_of_interests = PROGRAM->no_of_interests;
   no_of_instruments = PROGRAM->no_of_instruments;

   database_root_set_value (PROGRAM_root, (void *)PROGRAM, PROGRAM_type);
}
```

```
OS_END_TXN(t1);


OS_BEGIN_TXN(t2,0,transaction_update)                                    60
{
   database_root * INTEREST_root;

   struct INTEREST * INTEREST;
   struct INTEREST *INTEREST_ptr;

   INTEREST_root = database_create_root (db1, "INTEREST_entry");
   INTEREST_ptr = (struct INTEREST*) INTEREST_root;

   INTEREST = (struct INTEREST                                           70
               *)objectstore_alloc(INTEREST_type,no_of_interests,db1);
   i = 0;

   for (INTEREST_ptr = INTEREST;
        INTEREST_ptr < INTEREST + no_of_interests; INTEREST_ptr++)
   {
      strcpy(INTEREST_ptr->date,interest[i].date);
      INTEREST_ptr->three_month=interest[i].three_month;
      INTEREST_ptr->six_month = interest[i].six_month;
      INTEREST_ptr->nine_month = interest[i].nine_month;                  80
      i++;
   }

   database_root_set_value (INTEREST_root, (void *)INTEREST, INTEREST_type);
}
OS_END_TXN(t2);

OS_BEGIN_TXN(t3,0,transaction_update)
{
   database_root * INSTRUMENT_root;                                      90

   struct PRICE * PRICE;
   struct INSTRUMENT * INSTRUMENT;
```

```
struct PRICE *PRICE_ptr;
struct INSTRUMENT *INSTRUMENT_ptr;

INSTRUMENT_root = database_create_root (db1, "INSTRUMENT_entry");

INSTRUMENT_ptr = (struct INSTRUMENT*) INSTRUMENT_root;                    100
INSTRUMENT = (struct INSTRUMENT
        *)objectstore_alloc(INSTRUMENT_type,no_of_instruments,db1);
i = 0;

for (INSTRUMENT_ptr = INSTRUMENT; INSTRUMENT_ptr <
     INSTRUMENT + no_of_instruments; INSTRUMENT_ptr++)
{
   PRICE = (struct PRICE
           *)objectstore_alloc(PRICE_type,instrument[i].ndata,db1);
   INSTRUMENT_ptr->prices = PRICE;                                        110

   j = 0;

   for (PRICE_ptr = PRICE; PRICE_ptr <
        PRICE + instrument[i].ndata; PRICE_ptr++)
   {
      strcpy(PRICE_ptr->date,instrument[i].prices[j].date);
      PRICE_ptr->open = instrument[i].prices[j].price;
      PRICE_ptr->high = instrument[i].prices[j].price;
      PRICE_ptr->low = instrument[i].prices[j].price;                     120
      PRICE_ptr->close = instrument[i].prices[j].price;
      PRICE_ptr->volume = instrument[i].prices[j].vol;
      j++;
   }

   strcpy(INSTRUMENT_ptr->name,instrument[i].name);
   strcpy(INSTRUMENT_ptr->underlying,instrument[i].underlying);
   strcpy(INSTRUMENT_ptr->cusip,instrument[i].cusip);
   strcpy(INSTRUMENT_ptr->symbol,instrument[i].symbol);
   strcpy(INSTRUMENT_ptr->type,instrument[i].type);                       130
   strcpy(INSTRUMENT_ptr->expiration,instrument[i].expiration);
```

```
        INSTRUMENT_ptr->conversion = instrument[i].conversion;
        INSTRUMENT_ptr->strike = instrument[i].strike;
        INSTRUMENT_ptr->shares = instrument[i].shares;
        INSTRUMENT_ptr->ndata = instrument[i].ndata;
        INSTRUMENT_ptr->nparam = instrument[i].nparam;
        INSTRUMENT_ptr->dividend = instrument[i].dividend;
        INSTRUMENT_ptr->ptime = instrument[i].ptime;
        INSTRUMENT_ptr->pint = instrument[i].pint;                    140
        INSTRUMENT_ptr->ptrend = instrument[i].ptrend;
        INSTRUMENT_ptr->pdiv = instrument[i].pdiv;
        INSTRUMENT_ptr->pdil = instrument[i].pdil;
        INSTRUMENT_ptr->p0 = instrument[i].p0;
        i++;
    }

    database_root_set_value (INSTRUMENT_root,
                             (void *)INSTRUMENT, INSTRUMENT_type);
  }                                                                   150

  OS_END_TXN(t3);
  database_close(db1);
  printf("done.\n");
  exit(0);
}
```

C.6.4 Option pricing interface program (`ooptions.c`)

```
/*********************************************************************
This code interfaces ObjectStore and the ObjectStore version of the
financial database to the option pricing program.
*********************************************************************/

#include "options.c"
#include <ostore/ostore.h>
#include "instrument.h"

void read_data(dbase)                                                 10
```

```
    char dbase[20];
{
    int i, j;

    database *db1;
    start_objectstore ();

    db1 = database_lookup_open(dbase, 1, 0664);

    OS_BEGIN_TXN(t1,0,transaction_update)
    {
        database_root * PROGRAM_root;
        struct PROGRAM * PROGRAM;
        PROGRAM_root = database_root_find ("PROGRAM_entry",db1);

        PROGRAM = (struct PROGRAM *)database_root_get_value(PROGRAM_root,0);

        strcpy(program.model,PROGRAM->model);
        program.window_size = PROGRAM->window_size;
        program.trend_length = PROGRAM->trend_length;
        program.tolerance = PROGRAM->tolerance;
        program.limit = PROGRAM->limit;
        no_of_instruments = PROGRAM->no_of_instruments;
        no_of_interests = PROGRAM->no_of_interests;
    }
    OS_END_TXN(t1);

    OS_BEGIN_TXN(t2,0,transaction_update)
    {
        database_root * INTEREST_root;
        struct INTEREST * INTEREST;
        struct INTEREST *INTEREST_ptr;

        INTEREST_root = database_root_find ("INTEREST_entry",db1);
        INTEREST_ptr = (struct INTEREST*) INTEREST_root;

        INTEREST = (struct INTEREST *)database_root_get_value(INTEREST_root, 0);
```

```
    i = 0;                                                          50

    MALLOCN(interest, Interest, no_of_interests, "Malloc 1");

    for (INTEREST_ptr = INTEREST;
         INTEREST_ptr < INTEREST+no_of_interests; INTEREST_ptr++)
    {
        strcpy(interest[i].date,INTEREST_ptr->date);
        interest[i].three_month = INTEREST_ptr->three_month;
        interest[i].six_month = INTEREST_ptr->six_month;
        interest[i].nine_month = INTEREST_ptr->nine_month;          60
        i++;
    }
}
OS_END_TXN(t2);

OS_BEGIN_TXN(t3,0,transaction_update)
{
    database_root * INSTRUMENT_root;

    struct PRICE * PRICE;                                           70
    struct INSTRUMENT * INSTRUMENT;

    struct PRICE *PRICE_ptr;
    struct INSTRUMENT *INSTRUMENT_ptr;

    INSTRUMENT_root = database_root_find ("INSTRUMENT_entry",db1);

    INSTRUMENT_ptr = (struct INSTRUMENT*) INSTRUMENT_root;

    INSTRUMENT = (struct INSTRUMENT                                 80
                  *)database_root_get_value(INSTRUMENT_root,0);

    i = 0;

    MALLOCN(instrument, Instrument, no_of_instruments, "Malloc 2");
```

```
for (INSTRUMENT_ptr = INSTRUMENT; INSTRUMENT_ptr <
      INSTRUMENT+no_of_instruments; INSTRUMENT_ptr++)
{
    strcpy(instrument[i].name,INSTRUMENT_ptr->name);                    90
    strcpy(instrument[i].underlying,INSTRUMENT_ptr->underlying);
    strcpy(instrument[i].cusip,INSTRUMENT_ptr->cusip);
    strcpy(instrument[i].symbol,INSTRUMENT_ptr->symbol);
    strcpy(instrument[i].type,INSTRUMENT_ptr->type);
    strcpy(instrument[i].expiration,INSTRUMENT_ptr->expiration);

    instrument[i].conversion = INSTRUMENT_ptr->conversion;
    instrument[i].strike = INSTRUMENT_ptr->strike;
    instrument[i].shares = INSTRUMENT_ptr->shares;
    instrument[i].ndata = INSTRUMENT_ptr->ndata;                        100
    instrument[i].nparam = INSTRUMENT_ptr->nparam;
    instrument[i].dividend = INSTRUMENT_ptr->dividend;
    instrument[i].dilution = INSTRUMENT_ptr->dilution;
    instrument[i].ptime = INSTRUMENT_ptr->ptime;
    instrument[i].pint = INSTRUMENT_ptr->pint;
    instrument[i].ptrend = INSTRUMENT_ptr->ptrend;
    instrument[i].pdiv = INSTRUMENT_ptr->pdiv;
    instrument[i].pdil = INSTRUMENT_ptr->pdil;
    instrument[i].p0 = INSTRUMENT_ptr->p0;
                                                                        110
    PRICE = INSTRUMENT_ptr->prices;

    j = 0;

    MALLOCN(instrument[i].prices, Prices,
            instrument[i].ndata, "Malloc 3");

    for (PRICE_ptr = PRICE; PRICE_ptr <
          PRICE+instrument[i].ndata; PRICE_ptr++)
    {                                                                   120
        strcpy(instrument[i].prices[j].date,PRICE_ptr->date);
        instrument[i].prices[j].price = PRICE_ptr->close;
        instrument[i].prices[j].vol = PRICE_ptr->volume;
        j++;
```

```
            }
            i++;
        }
    }
    OS_END_TXN(t3);
    database_close(db1);                                          130
}


void write_data(dbase)

    char dbase[20];
{
    int i;

    database *db1;                                                140
    start_objectstore ();

    db1 = database_lookup_open(dbase, 0, 0664);

    OS_BEGIN_TXN(t5,0,transaction_update)
    {
        database_root * INSTRUMENT_root;
        struct INSTRUMENT * INSTRUMENT;
        struct INSTRUMENT *INSTRUMENT_ptr;
                                                                  150
        INSTRUMENT_root = database_root_find ("INSTRUMENT_entry",db1);
        INSTRUMENT_ptr = (struct INSTRUMENT*) INSTRUMENT_root;
        INSTRUMENT = (struct INSTRUMENT
                    *)database_root_get_value(INSTRUMENT_root,0);
        i = 0;

        for (INSTRUMENT_ptr = INSTRUMENT; INSTRUMENT_ptr <
             INSTRUMENT+no_of_instruments; INSTRUMENT_ptr++)
        {
            INSTRUMENT_ptr->ptime = instrument[i].ptime;          160
            INSTRUMENT_ptr->pint = instrument[i].pint;
            INSTRUMENT_ptr->ptrend = instrument[i].ptrend;
```

```
            INSTRUMENT_ptr->pdiv = instrument[i].pdiv;
            INSTRUMENT_ptr->pdil = instrument[i].pdil;
            INSTRUMENT_ptr->p0 = instrument[i].p0;

            i++;
        }
    }
    OS_END_TXN(t5);                                              170
    database_close(db1);
}


void clear_parameters(dbase)

    char dbase[20];
{
    int i;
                                                                 180
    database *db1;
    start_objectstore ();

    db1 = database_lookup_open(dbase, 0, 0664);

    OS_BEGIN_TXN(t4,0,transaction_update)
    {
        database_root * INSTRUMENT_root;
        struct INSTRUMENT * INSTRUMENT;
        struct INSTRUMENT *INSTRUMENT_ptr;                       190

        INSTRUMENT_root = database_root_find ("INSTRUMENT_entry",db1);
        INSTRUMENT_ptr = (struct INSTRUMENT*) INSTRUMENT_root;
        INSTRUMENT = (struct INSTRUMENT
                    *)database_root_get_value(INSTRUMENT_root,0);
        i = 0;

        for (INSTRUMENT_ptr = INSTRUMENT; INSTRUMENT_ptr <
             INSTRUMENT+no_of_instruments; INSTRUMENT_ptr++)
        {                                                        200
```

```
            INSTRUMENT_ptr->ptime = 0;
            INSTRUMENT_ptr->pint = 0;
            INSTRUMENT_ptr->ptrend = 0;
            INSTRUMENT_ptr->pdiv = 0;
            INSTRUMENT_ptr->pdil = 0;
            INSTRUMENT_ptr->p0 = 0;

            instrument[i].ptime = 0.0;
            instrument[i].pint = 0.0;
            instrument[i].ptrend = 0.0;                                    210
            instrument[i].pdiv = 0.0;
            instrument[i].pdil = 0.0;
            instrument[i].p0 = 0.0;

            i++;
        }
    }
    OS_END_TXN(t4);
    database_close(db1);
}                                                                          220
```

C.7 Modified numerical recipes routines

C.7.1 mrqmin.c

```
/*************************************************************************
This is a modified version of the numerical recipes routine mrqmin.
The routine has been modified in two important ways: First, we have
added the while construct, which will continue to adjust the fit
parameters until convergence of chi-squared (within 1E-10 on
successive guesses) is achieved; second, we have modified the routine
to do a multivariate multi-parameter non-linear least-squares fit,
rather than the univariate fit which it was originally designed to do.
*************************************************************************/
                                                                           10

void mrqmin(s, strike, x, y, sig, ndata, a, ma, lista, mfit, covar,
```

```
        alpha, chisq, funcs, alamda, index)

    float **x, y[], sig[], a[], **covar, **alpha, *chisq, *alamda;
    float s[], strike;
    int ndata, ma, lista[], mfit, index;
    void (*funcs)();

{                                                                    20
    int k, kk, j, ihit;
    static float *da, *atry, **oneda, *beta, ochisq;
    float *vector(), **matrix();
    void mrqcof(), gaussj(), covsrt(), nrerror(), free_matrix(), free_vector();

    oneda=matrix(1, mfit, 1, 1);
    atry=vector(1, ma);
    da=vector(1, ma);
    beta=vector(1, ma);
                                                                     30
    kk=mfit+1;
    *alamda=0.001;

    for (j=1;j<=ma;j++)
    {
        ihit=0;

        for (k=1;k<=mfit;k++)
            if (lista[k] == j) ihit++;
                                                                     40
        if (ihit == 0)
            lista[kk++]=j;

        else if (ihit > 1)
            nrerror("Bad LISTA permutation in MRQMIN-1");
    }

    if (kk != ma+1)
        nrerror("Bad LISTA permutation in MRQMIN-2");
                                                                     50
```

```
mrqcof(s, strike, x, y, sig, ndata, a, ma, lista, mfit,
        alpha, beta, chisq, funcs, index);

ochisq=(*chisq);

do {

for (j=1;j<=mfit;j++)
{
   for (k=1;k<=mfit;k++)                                          60
      covar[j][k]=alpha[j][k];

   covar[j][j]=alpha[j][j]*(1.0+(*alamda));
   oneda[j][1]=beta[j];
}

gaussj(covar, mfit, oneda, 1);

for (j=1;j<=mfit;j++)
   da[j]=oneda[j][1];                                             70

for (j=1;j<=ma;j++)
   atry[j]=a[j];

for (j=1;j<=mfit;j++)
   atry[lista[j]] = a[lista[j]]+da[j];

mrqcof(s, strike, x, y, sig, ndata, atry, ma, lista, mfit,
        covar, da, chisq, funcs, index);
                                                                  80
if (*chisq < ochisq)
{
   *alamda *= 0.1;
   ochisq=(*chisq);

   for (j=1;j<=mfit;j++)
   {
      for (k=1;k<=mfit;k++)
```

```
            alpha[j][k]=covar[j][k];
                                                                    90
        beta[j]=da[j];
        a[lista[j]]=atry[lista[j]];
    }
}

else
{
    *alamda *= 10.0;
    *chisq=ochisq;
}                                                                   100

} while (sqrt((*chisq - ochisq) * (*chisq - ochisq)) > 1E-10);

    covsrt(covar, ma, lista, mfit);
    free_vector(beta, 1, ma);
    free_vector(da, 1, ma);
    free_vector(atry, 1, ma);
    free_matrix(oneda, 1, mfit, 1, 1);
    return;
}                                                                   110
```

C.7.2 mrqcof.c

```
/*********************************************************************
This is a modified version of the numerical recipes routine mrqcof.
The routine has been modified to do a multivariate multi-parameter
non-linear least-squares fit, rather than the univariate fit which it
was originally designed to do.
*********************************************************************/

void mrqcof(s, strike, x, y, sig, ndata, a, ma, lista,
            mfit, alpha, beta, chisq, funcs, index)
                                                                    10
    float **x, y[], sig[], a[], **alpha, beta[], *chisq;
    float s[], strike;
```

```
    int ndata, ma, lista[], mfit, index;
    void (*funcs)();
{
    int k, j, i;
    float ymod, wt, sig2i, dy, *dyda, *vector();
    void free_vector();

    dyda=vector(1, ma);                                          20

    for (j=1;j<=mfit;j++)
    {
        for (k=1;k<=j;k++)
            alpha[j][k]=0.0;
        beta[j]=0.0;
    }

    *chisq=0.0;
                                                                 30
    for (i=1;i<=ndata;i++)
    {
        (*funcs)(s[i], strike, x[i], a, &ymod, dyda, ma, index);
        sig2i=1.0/(sig[i]*sig[i]);
        dy=y[i]-ymod;

        for (j=1;j<=mfit;j++)
        {
            wt=dyda[lista[j]]*sig2i;
                                                                 40
            for (k=1;k<=j;k++)
                alpha[j][k] += wt*dyda[lista[k]];
            beta[j] += dy*wt;
        }

        (*chisq) += dy*dy*sig2i;
    }

    for (j=2;j<=mfit;j++)
        for (k=1;k<=j-1;k++)                                     50
```

```
            alpha[k][j]=alpha[j][k];

    free_vector(dyda, 1, ma);
}
```

Appendix D

Database Creation Utilities for Benchmark Testing

D.1 Code to generate random integer tables

D.1.1 Makefile

```
NAME     =    random

C_FILES  =    random.c

O_FILES  =    random.o

SRC_FILES =     $(INC_FILES) $(C_FILES)

CC       =    cc
CFLAGS   =    -g                                                    10
ALL_LIBS =      -lm -lc -lq
LIBS_DIR =


all:          $(NAME)

$(NAME):      $(O_FILES)
              $(CC) $(CFLAGS) $(CPPFLAGS) -o $(NAME) $(O_FILES) \
```

```
        $(ALL_LIBS) $(LIBS_DIR)

clean:  ; /bin/rm -f \#*\# *~ $(NAME) $(O_FILES) core
purge:  ; /bin/rm -f \#*\# *~ core
print:  ; enscript -2r -Pps1 $(INC_FILES) $(C_FILES)
```

D.1.2 Program code (`random.c`)

```
/*****************************************************************
This program produces tables of random integers on the interval
[0,99]. The -r and -c options specify the number of rows and columns,
while the -i options specifies the initial seed. The default vaues
are ten by 8, with a random seed of one. The result goes to the
standard output by default.
*****************************************************************/

#include <stdio.h>
#include<string.h>

#ifdef __TURBOC__
#include <stdlib.h>
#endif

int
    main(argc , argv)

int argc;
char ** argv;

{
    char s[20];
    int i,j,n,init_seed,max_row,max_col;
    float x;
```

```
    init_seed = 1;
    max_row = 10;
    max_col = 8;

    while (--argc > 0 && (*++argv)[0] == '-')
    {
        strcpy(s,++*argv);

        if (s[0] == 'i')
            init_seed = atoi(&s[1]);

        if (s[0] == 'r')
            max_row = atoi(&s[1]);

        if (s[0] == 'c')
            max_col = atoi(&s[1]);
    }

    for (i = 1; i <= max_row; ++i)
    {
        for (j = 1 + i*max_col; j <= max_col + i*max_col; ++j)
        {
            srand(j+init_seed);
            x = rand();
            n = x/21474835.2;

            printf("%d\t",n);
        }
        printf("\n");
    }
}
```

D.2 UNIX script to make flat-file versions of the test databases

```
random -r1000 -c8 > big_table
random -r10 -c7 -i1 > top
random -r4 -c7 -i71 > middle_1
random -r4 -c7 -i75 > middle_2
random -r4 -c7 -i79 > middle_3
random -r4 -c7 -i83 > middle_4
random -r4 -c7 -i87 > middle_5
random -r4 -c7 -i91 > middle_6
random -r4 -c7 -i95 > middle_7
random -r4 -c7 -i99 > middle_8
random -r4 -c7 -i103 > middle_9
random -r4 -c7 -i107 > middle_10
random -r1000 -c8 -i8000 > bottom_1_1
random -r1000 -c8 -i16000 > bottom_1_2
random -r1000 -c8 -i24000 > bottom_1_3
random -r1000 -c8 -i32000 > bottom_1_4
random -r1000 -c8 -i8000 > bottom_2_1
random -r1000 -c8 -i16000 > bottom_2_2
random -r1000 -c8 -i24000 > bottom_2_3
random -r1000 -c8 -i32000 > bottom_2_4
random -r1000 -c8 -i80000 > bottom_3_1
random -r1000 -c8 -i160000 > bottom_3_2
random -r1000 -c8 -i240000 > bottom_3_3
random -r1000 -c8 -i320000 > bottom_3_4
random -r1000 -c8 -i800000 > bottom_4_11
random -r1000 -c8 -i1600000 > bottom_4_2
random -r1000 -c8 -i2400000 > bottom_4_3
random -r1000 -c8 -i3200000 > bottom_4_4
random -r1000 -c8 -i8000000 > bottom_5_1
random -r1000 -c8 -i16000000 > bottom_5_2
random -r1000 -c8 -i24000000 > bottom_5_3
random -r1000 -c8 -i32000000 > bottom_5_4
random -r1000 -c8 -i80000000 > bottom_6_1
random -r1000 -c8 -i160000000 > bottom_6_2
```

```
random -r1000 -c8 -i240000000 > bottom_6_3
random -r1000 -c8 -i320000000 > bottom_6_4
random -r1000 -c8 -i800000000 > bottom_7_1
random -r1000 -c8 -i1600000000 > bottom_7_2
random -r1000 -c8 -i2400000000 > bottom_7_3
random -r1000 -c8 -i3200000000 > bottom_7_4
random -r1000 -c8 -i8000000000 > bottom_8_1
random -r1000 -c8 -i16000000000 > bottom_8_2
random -r1000 -c8 -i24000000000 > bottom_8_3
random -r1000 -c8 -i32000000000 > bottom_8_4
random -r1000 -c8 -i80000000000 > bottom_9_1
random -r1000 -c8 -i160000000000 > bottom_9_2
random -r1000 -c8 -i240000000000 > bottom_9_3
random -r1000 -c8 -i320000000000 > bottom_9_4
random -r1000 -c8 -i800000000000 > bottom_10_1
random -r1000 -c8 -i1600000000000 > bottom_10_2
random -r1000 -c8 -i2400000000000 > bottom_10_3
random -r1000 -c8 -i3200000000000 > bottom_10_4
```

D.3 Script to create INGRES version of test database two

```
create top(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,table=c9)\g
copy top(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,num6=c0tab,
num7=c0tab,table=c0nl) from "/u/shr/testdb/top"\g
create middle_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
create middle_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
create middle_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
create middle_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
create middle_5(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
create middle_6(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
```

```
table=c12)\g
create middle_7(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
create middle_8(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
create middle_9(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
create middle_10(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
table=c12)\g
copy middle_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_1"\g
copy middle_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_2"\g
copy middle_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_3"\g
copy middle_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_4"\g
copy middle_5(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_5"\g
copy middle_6(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_6"\g
copy middle_7(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_7"\g
copy middle_8(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_8"\g
copy middle_9(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_9"\g
copy middle_10(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,table=c0nl) from "/u/shr/testdb/middle_10"\g
create bottom_1_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_1_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_1_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_1_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_2_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
```

```
num8=i1)\g
create bottom_2_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_2_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_2_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_3_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_3_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_3_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_3_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_4_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_4_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_4_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_4_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_5_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_5_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_5_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_5_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_6_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_6_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_6_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_6_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
```

```
num8=i1)\g
create bottom_7_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_7_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_7_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_7_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_8_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_8_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_8_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_8_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_9_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_9_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_9_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_9_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_10_1(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_10_2(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_10_3(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
create bottom_10_4(num1=i1,num2=i1,num3=i1,num4=i1,num5=i1,num6=i1,num7=i1,
num8=i1)\g
copy bottom_1_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_1_1"\g
copy bottom_1_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_1_2"\g
copy bottom_1_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
```

```
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_1_3"\g
copy bottom_1_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_1_4"\g
copy bottom_2_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_2_1"\g
copy bottom_2_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_2_2"\g
copy bottom_2_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_2_3"\g
copy bottom_2_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_2_4"\g
copy bottom_3_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_3_1"\g
copy bottom_3_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_3_2"\g
copy bottom_3_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_3_3"\g
copy bottom_3_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_3_4"\g
copy bottom_4_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_4_1"\g
copy bottom_4_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_4_2"\g
copy bottom_4_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_4_3"\g
copy bottom_4_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_4_4"\g
copy bottom_5_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_5_1"\g
copy bottom_5_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_5_2"\g
copy bottom_5_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_5_3"\g
copy bottom_5_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_5_4"\g
copy bottom_6_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_6_1"\g
copy bottom_6_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
```

```
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_6_2"\g
copy bottom_6_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_6_3"\g
copy bottom_6_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_6_4"\g
copy bottom_7_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_7_1"\g
copy bottom_7_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_7_2"\g
copy bottom_7_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_7_3"\g
copy bottom_7_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_7_4"\g
copy bottom_8_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_8_1"\g
copy bottom_8_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_8_2"\g
copy bottom_8_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_8_3"\g
copy bottom_8_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_8_4"\g
copy bottom_9_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_9_1"\g
copy bottom_9_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_9_2"\g
copy bottom_9_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_9_3"\g
copy bottom_9_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_9_4"\g
copy bottom_10_1(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_10_1"\g
copy bottom_10_2(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_10_2"\g
copy bottom_10_3(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_10_3"\g
copy bottom_10_4(num1=c0tab,num2=c0tab,num3=c0tab,num4=c0tab,num5=c0tab,
num6=c0tab,num7=c0tab,num8=c0nl) from "/u/shr/testdb/bottom_10_4"\g
```

D.4 Utilities to create ObjectStore version of test database one

D.4.1 `testdb1.h` header file

```
#include <string.h>

struct BIG_TABLE
{
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    int num8;
};
```

D.4.2 `ingres_read.testdb1.q` included code

```
/*********************************************************************
This code reads the INGRES version of test database one into RAM.
*********************************************************************/

#include <stdio.h>
#include <string.h>

#define STREQ(A, B)    (*(A) == *(B)  && strcmp((A), (B)) == 0)

#define MALLOCN(P,T,N,M) \
    if (((P) = (T *) malloc((unsigned) (N) * sizeof(T))) ==  NULL) { \
        fprintf(stderr, "Malloc failure: (%d bytes) %s\n", (N) * sizeof(T), M); \
        exit(1); \
    } \
```

```
    else

#define FREE(P) free((char *) P)

int no_of_records;
int big_table[1001][9];                                                    20

void read_ingres_data(database)

    char database[20];
{

##      char    DB[21];
##      int     NUM1;
##      int     NUM2;
##      int     NUM3;                                                      30
##      int     NUM4;
##      int     NUM5;
##      int     NUM6;
##      int     NUM7;
##      int     NUM8;

    int   i, j, k;


/*************************************************************************  40
We read in the Big Table.
*************************************************************************/

    strcpy(DB,database);
    i = 0;

##      ingres DB
##      retrieve (NUM1=main.num1,
##                NUM2=main.num2,
##                NUM3=main.num3,                                          50
##                NUM4=main.num4,
##                NUM5=main.num5,
```

```
##                  NUM6=main.num6,
##                  NUM7=main.num7,
##                  NUM8=main.num8)
##      {
            big_table[i][1] = NUM1;
            big_table[i][2] = NUM2;
            big_table[i][3] = NUM3;
            big_table[i][4] = NUM4;                                    60
            big_table[i][5] = NUM5;
            big_table[i][6] = NUM6;
            big_table[i][7] = NUM7;
            big_table[i][8] = NUM8;
            i++;
##      }
##      exit
}
```

D.4.3 db1_schema.cc database schema code

```
#include <ostore/ostore.hh>
#include <ostore/manschem.hh>

#include "testdb1.h"

static void dummy ()
{
   OS_MARK_SCHEMA_TYPE(BIG_TABLE);
}
```

D.4.4 Makefile

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk

OS_COMPILATION_SCHEMA_DB_PATH= /$(LOGNAME)/testdb1.comp_schema
OS_APPLICATION_SCHEMA_DB_PATH= /$(LOGNAME)/testdb1.app_schema

LDLIBS = -los -losc -lq -lm -lc -lg
```

```
SOURCES = db1_make.c db1_schema.cc

OBJECTS = db1_make.o db1_schema.o

EXECUTABLES = db1_make

CPPFLAGS = -I$(OS_ROOTDIR)/include
CFLAGS = -g

LDFLAGS= $(OS_EXPORT)
CC = cc

all: $(EXECUTABLES)

db1_make: db1_make.o schema_standin
        $(OS_PRELINK) .os_db1_schema.cc \
            $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH) \
            db1_make.o $(LDLIBS)
        OSCC -c .os_db1_schema.cc
        $(CC) $(CFLAGS) $(LDFLAGS) -o db1_make db1_make.o .os_db1_schema.o $(LDLIBS)

db1_make.o: db1_make.c
        $(CC) $(CPPFLAGS) $(CFLAGS) -c db1_make.c

schema_standin: db1_schema.cc
        OSCC -batch_schema $(OS_COMPILATION_SCHEMA_DB_PATH) db1_schema.cc
        touch schema_standin

clean:
        osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
        rm -f $(EXECUTABLES) $(OBJECTS) schema_standin

depend:
        osmakedep .depend $(CPPFLAGS) -files $(SOURCES)

include .depend
```

D.4.5 db1_make.c database creation program

```
/*************************************************************************
This program creates the ObjectStore version of test database one from
the INGRES original.
*************************************************************************/

#include <ostore/ostore.h>
#include "testdb1.h"
#include "ingres_read_testdb1.c"

database *testdb1;                                                    10
int i, j;
int no_of_records;

main(argc , argv)
    int argc;
    char ** argv;
{
    os_typespec * BIG_TABLE_type;

    printf("reading in data from INGRES. . .\n");                     20
    read_ingres_data("testdb1");
    printf("done.\n\n");

    printf("creating ObjectStore database. . . .\n");

    start_objectstore ();

    testdb1 = database_lookup_open(argv[1], 0, 0664);

    no_of_records = 1000;                                             30

    BIG_TABLE_type = alloc_typespec ("BIG_TABLE", 0);

    OS_BEGIN_TXN(t1,0,transaction_update)
```

```
{
    database_root * BIG_TABLE_root;

    struct BIG_TABLE * BIG_TABLE;
    struct BIG_TABLE *BIG_TABLE_ptr;
                                                                    40
    BIG_TABLE_root = database_create_root (testdb1, "BIG_TABLE_entry");
    BIG_TABLE_ptr = (struct BIG_TABLE*) BIG_TABLE_root;

    BIG_TABLE = (struct BIG_TABLE
                *)objectstore_alloc(BIG_TABLE_type,no_of_records,testdb1);
    i = 0;

    for (BIG_TABLE_ptr = BIG_TABLE;
         BIG_TABLE_ptr < BIG_TABLE + no_of_records; BIG_TABLE_ptr++)
    {                                                               50
        BIG_TABLE_ptr->num1 = big_table[i][1];
        BIG_TABLE_ptr->num2 = big_table[i][2];
        BIG_TABLE_ptr->num3 = big_table[i][3];
        BIG_TABLE_ptr->num4 = big_table[i][4];
        BIG_TABLE_ptr->num5 = big_table[i][5];
        BIG_TABLE_ptr->num6 = big_table[i][6];
        BIG_TABLE_ptr->num7 = big_table[i][7];
        BIG_TABLE_ptr->num8 = big_table[i][8];

        i++;                                                        60
    }

    database_root_set_value (BIG_TABLE_root,
                             (void *)BIG_TABLE, BIG_TABLE_type);
}
OS_END_TXN(t1);

database_close(testdb1);
printf("done.\n");
exit(0);                                                            70
}
```

D.5 Utilities to create ObjectStore version of test database two

D.5.1 `testdb2.h` header file

```
#include <string.h>

struct BOTTOM
{
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    int num8;
};

struct MIDDLE
{
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    struct BOTTOM *bottom;
};

struct TOP
{
    int num1;
    int num2;
```

```
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    struct MIDDLE *middle;
};
```

D.5.2 ingres_read.testdb2.q included code

```
/*************************************************************************
This code reads the INGRES version of test database two into RAM.
*************************************************************************/

#include <stdio.h>
#include <string.h>

#define STREQ(A, B)     (*(A) == *(B)  && strcmp((A), (B)) == 0)

#define MALLOCN(P,T,N,M) \                                              10
    if (((P) = (T *) malloc((unsigned) (N) * sizeof(T))) ==  NULL) { \
        fprintf(stderr, "Malloc failure: (%d bytes) %s\n", (N) * sizeof(T), M); \
        exit(1); \
    } \
    else

#define FREE(P) free((char *) P)

typedef struct {
    int num1;                                                           20
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    int num8;
```

```
} Bottom;

typedef struct {
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    char table[11];
    Bottom *bottom;
} Middle;

typedef struct {
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    char table[9];
    Middle *middle;
} Top;

Top *top;
Middle *middle;
Bottom *bottom;

int no_of_top;
int no_of_middle;
int no_of_bottom;

void read_ingres_data(database)

    char database[20];
{
```

```
##      char    DB[21];
##      int     NUM1;
##      int     NUM2;
##      int     NUM3;                                                    70
##      int     NUM4;
##      int     NUM5;
##      int     NUM6;
##      int     NUM7;
##      int     NUM8;
##      char    MID[11];
##      char    BOT[11];
##      char    TABLE[11];

    int  i, j, k, l;                                                     80

    strcpy(DB,database);


/************************************************************************
We read in the Top Table.
************************************************************************/

    i = 0;
    no_of_top = 10;                                                      90

    MALLOCN(top, Top, no_of_top, "Malloc 1");

##      ingres DB
##      retrieve (NUM1=top.num1,
##                NUM2=top.num2,
##                NUM3=top.num3,
##                NUM4=top.num4,
##                NUM5=top.num5,
##                NUM6=top.num6,                                         100
##                NUM7=top.num7,
##                TABLE=top.table)
##      {
```

```
            strcpy(top[i].table, TABLE);
            top[i].num1 = NUM1;
            top[i].num2 = NUM2;
            top[i].num3 = NUM3;
            top[i].num4 = NUM4;
            top[i].num5 = NUM5;
            top[i].num6 = NUM6;                                          110
            top[i].num7 = NUM7;
            i++;
##      }


    no_of_middle = 4;

    for (i = 0; i < no_of_top; ++i)
    {
        MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");        120

        strcpy(MID,top[i].table);

        j = 0;

##      retrieve (NUM1=MID.num1,
##                NUM2=MID.num2,
##                NUM3=MID.num3,
##                NUM4=MID.num4,
##                NUM5=MID.num5,                                         130
##                NUM6=MID.num6,
##                NUM7=MID.num7,
##                TABLE=MID.table)
##      {
            strcpy(top[i].middle[j].table, TABLE);
            top[i].middle[j].num1 = NUM1;
            top[i].middle[j].num2 = NUM2;
            top[i].middle[j].num3 = NUM3;
            top[i].middle[j].num4 = NUM4;
            top[i].middle[j].num5 = NUM5;                                140
            top[i].middle[j].num6 = NUM6;
```

```
            top[i].middle[j].num7 = NUM7;
            j++;
##      }
    }
    no_of_bottom = 10000;

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)                                    150
        {

            MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

            strcpy(BOT,top[i].middle[j].table);

            l = 0;

##              retrieve (NUM1=BOT.num1,
##                        NUM2=BOT.num2,                                      160
##                        NUM3=BOT.num3,
##                        NUM4=BOT.num4,
##                        NUM5=BOT.num5,
##                        NUM6=BOT.num6,
##                        NUM7=BOT.num7,
##                        NUM8=BOT.num8)
##              {
                    top[i].middle[j].bottom[l].num1 = NUM1;
                    top[i].middle[j].bottom[l].num2 = NUM2;
                    top[i].middle[j].bottom[l].num3 = NUM3;                   170
                    top[i].middle[j].bottom[l].num4 = NUM4;
                    top[i].middle[j].bottom[l].num5 = NUM5;
                    top[i].middle[j].bottom[l].num6 = NUM6;
                    top[i].middle[j].bottom[l].num7 = NUM7;
                    top[i].middle[j].bottom[l].num8 = NUM8;
                    l++;
##              }
        }
    }
```

```
}
```

D.5.3 db2_schema.cc database schema code

```
#include <ostore/ostore.hh>
#include <ostore/manschem.hh>

#include "testdb2.h"

static void dummy ()
{
    OS_MARK_SCHEMA_TYPE(BOTTOM);
    OS_MARK_SCHEMA_TYPE(MIDDLE);
    OS_MARK_SCHEMA_TYPE(TOP);
}
```

D.5.4 Makefile

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk

OS_COMPILATION_SCHEMA_DB_PATH= /$(LOGNAME)/testdb2.comp_schema
OS_APPLICATION_SCHEMA_DB_PATH= /$(LOGNAME)/testdb2.app_schema

LDLIBS = -los -losc -lq -lm -lc -lg

SOURCES = db2_make.c db2_schema.cc

OBJECTS =  db2_make.o db2_schema.o

EXECUTABLES = db2_make

CPPFLAGS = -I$(OS_ROOTDIR)/include
CFLAGS = -g

LDFLAGS= $(OS_EXPORT)
CC = cc
```

```
all: $(EXECUTABLES)

db2_make: db2_make.o schema_standin
        $(OS_PRELINK) .os_db2_schema.cc \
            $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH) \
            db2_make.o $(LDLIBS)
        OSCC -c .os_db2_schema.cc
        $(CC) $(CFLAGS) $(LDFLAGS) -o db2_make db2_make.o .os_db2_schema.o $(LDLIBS)

db2_make.o: db2_make.c
        $(CC) $(CPPFLAGS) $(CFLAGS) -c db2_make.c

schema_standin: db2_schema.cc
        OSCC -batch_schema $(OS_COMPILATION_SCHEMA_DB_PATH) db2_schema.cc
        touch schema_standin

clean:
        osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
        rm -f $(EXECUTABLES) $(OBJECTS) schema_standin

depend:
        osmakedep .depend $(CPPFLAGS) -files $(SOURCES)

include .depend
```

D.5.5 db2_make.c database creation program

```
/*********************************************************************
This program creates the ObjectStore version of test database two from
the INGRES original.
*********************************************************************/

#include <ostore/ostore.h>
#include "testdb2.h"
#include "ingres_read_testdb2.c"
```

```
database *testdb2;                                                    10
int i, j, k;
int no_of_records;

main(argc , argv)
   int argc;
   char ** argv;
{
   os_typespec * TOP_type;
   os_typespec * MIDDLE_type;
   os_typespec * BOTTOM_type;                                         20

   printf("reading in data from INGRES. . .\n");
   read_ingres_data("testdb2");
   printf("done.\n\n");

   printf("creating ObjectStore database. . . .\n");

   start_objectstore ();

   testdb2 = database_lookup_open(argv[1], 0, 0664);                  30

   no_of_top = 10;
   no_of_middle = 4;
   no_of_bottom = 1000;

   BOTTOM_type = alloc_typespec ("BOTTOM", 0);
   MIDDLE_type = alloc_typespec ("MIDDLE", 0);
   TOP_type = alloc_typespec ("TOP", 0);

   OS_BEGIN_TXN(t1,0,transaction_update)                              40
   {
      database_root * TOP_root;

      struct BOTTOM * BOTTOM;
      struct MIDDLE * MIDDLE;
      struct TOP * TOP;
```

```
struct BOTTOM *BOTTOM_ptr;
struct MIDDLE *MIDDLE_ptr;
struct TOP *TOP_ptr;                                                    50

TOP_root = database_create_root (testdb2, "TOP_entry");
TOP_ptr = (struct TOP*) TOP_root;

TOP = (struct TOP
          *)objectstore_alloc(TOP_type,no_of_top,testdb2);
i = 0;

for (TOP_ptr = TOP;
     TOP_ptr < TOP + no_of_top; TOP_ptr++)                              60
{


   MIDDLE = (struct MIDDLE
            *)objectstore_alloc(MIDDLE_type,no_of_middle,testdb2);
   TOP_ptr->middle = MIDDLE;

   j = 0;

   for (MIDDLE_ptr = MIDDLE; MIDDLE_ptr <                               70
        MIDDLE + no_of_middle; MIDDLE_ptr++)
   {



      BOTTOM = (struct BOTTOM
               *)objectstore_alloc(BOTTOM_type,no_of_bottom,testdb2);

      MIDDLE_ptr->bottom = BOTTOM;
                                                                        80
      k = 0;

      for (BOTTOM_ptr = BOTTOM; BOTTOM_ptr <
           BOTTOM + no_of_bottom; BOTTOM_ptr++)
      {
```

```
            BOTTOM_ptr->num1 = top[i].middle[j].bottom[k].num1;
            BOTTOM_ptr->num2 = top[i].middle[j].bottom[k].num2;
            BOTTOM_ptr->num3 = top[i].middle[j].bottom[k].num3;
            BOTTOM_ptr->num4 = top[i].middle[j].bottom[k].num4;
            BOTTOM_ptr->num5 = top[i].middle[j].bottom[k].num5;
            BOTTOM_ptr->num6 = top[i].middle[j].bottom[k].num6;
            BOTTOM_ptr->num7 = top[i].middle[j].bottom[k].num7;
            BOTTOM_ptr->num8 = top[i].middle[j].bottom[k].num8;
            k++;
        }


        MIDDLE_ptr->num1 = top[i].middle[j].num1;
        MIDDLE_ptr->num2 = top[i].middle[j].num2;
        MIDDLE_ptr->num3 = top[i].middle[j].num3;
        MIDDLE_ptr->num4 = top[i].middle[j].num4;
        MIDDLE_ptr->num5 = top[i].middle[j].num5;
        MIDDLE_ptr->num6 = top[i].middle[j].num6;
        MIDDLE_ptr->num7 = top[i].middle[j].num7;
        j++;
    }

    TOP_ptr->num1 = top[i].num1;
    TOP_ptr->num2 = top[i].num2;
    TOP_ptr->num3 = top[i].num3;
    TOP_ptr->num4 = top[i].num4;
    TOP_ptr->num5 = top[i].num5;
    TOP_ptr->num6 = top[i].num6;
    TOP_ptr->num7 = top[i].num7;

    i++;
}

database_root_set_value (TOP_root, (void *)TOP, TOP_type);
}
OS_END_TXN(t1);

database_close(testdb2);
```

```
    printf("done.\n");
    exit(0);
}
```

Appendix E

Benchmarking Code for Test Database One (Relationally-Oriented)

E.1 General program code (`test1.c`)

```
/*********************************************************************
This program runs the benchmark tests on test database one.
*********************************************************************/

#include <stdio.h>
#include <time.h>
#include <string.h>

/*********************************************************************
Define STREQ, which compares two strings.                              10
*********************************************************************/

#define STREQ(A, B)    (*(A) == *(B)  && strcmp((A), (B)) == 0)

/*********************************************************************
Allocates N objects of a given type, using malloc().
*********************************************************************/
```

```
#define MALLOCN(P,T,N,M) \
    if (((P) = (T *) malloc((unsigned) (N) * sizeof(T))) ==  NULL) { \
        fprintf(stderr, "Malloc failure: (%d bytes) %s\n", (N) * sizeof(T), M); \
        exit(1); \
    } \
    else

#define FREE(P) free((char *) P)

int j,n;

int main()

{
    long t;
    int i;

/*************************************************************************
                            test the speed of RAM
*************************************************************************/


    n = 1000;

    t = time(0);

    for (i = 0; i < n; ++i)
        ram_test();

    t = time(0) - t;

    printf("%ld seconds elapsed in %d trials for RAM speed test.\n\n",t,n);

/*************************************************************************
                            read entire database
*************************************************************************/
```

```
    n = 1000;

    t = time(0);

    for (i = 0; i < n; ++i)
        read_data();

    t = time(0) - t;

    printf("%ld seconds elapsed in %d trials for total read.\n\n",t,n);

/*************************************************************************
                            simple query number 1
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        simple_query(1);

    t = time(0) - t;

    printf("%d hits on simple query\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                            simple query number 2
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        simple_query(25);

    t = time(0) - t;

    printf("%d hits on simple query\n",j);
```

```
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                            simple query number 3
*************************************************************************/

    t = time(0);                                                          100

    for (i = 0; i < n; ++i)
        simple_query(50);

    t = time(0) - t;

    printf("%d hits on simple query\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************  110
                            simple query number 4
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        simple_query(75);

    t = time(0) - t;

                                                                          120
    printf("%d hits on simple query\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                            simple query number 5
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)                                               130
        simple_query(99);
```

```
    t = time(0) - t;

    printf("%d hits on simple query\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                            intermediate query
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        intermediate_query(50,50);

    t = time(0) - t;

    printf("%d hits on intermediate query\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                            complex query
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        complex_query(50,50,50,50);

    t = time(0) - t;

    printf("%d hits on complex query\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                            very complex query
*************************************************************************/
```

```
    t = time(0);                                                          170

    for (i = 0; i < n; ++i)
        very_complex_query(50,50,50,50,50,50,50,50);

    t = time(0) - t;

    printf("%d hits on very complex query number 1\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************  180
                        very complex query number 2
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        very_complex_query(5,5,5,5,5,5,5,5);

    t = time(0) - t;
                                                                          190
    printf("%d hits on very complex query number 2\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                        very complex query type 2
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)                                               200
        very_complex_query2(5,95,5,95,5,95,5,95);

    t = time(0) - t;

    printf("%d hits on type 2 very complex query\n",j);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);
```

}

E.2 INGRES interface code

E.2.1 Makefile

```
NAME     =    ingres_test_db1

C_FILES  =    ingres_test_db1.c

O_FILES  =    ingres_test_db1.o

INC_FILES =   malloc.h nr.h nrutil.h
SRC_FILES =     $(INC_FILES) $(C_FILES)

CC       =    cc                                                    10
CFLAGS   =    -g
ALL_LIBS =      -lm -lc -lq
LIBS_DIR =


all:          $(NAME)

$(NAME):      $(O_FILES)
              $(CC) $(CFLAGS) $(CPPFLAGS) -o $(NAME) $(O_FILES) \
              $(ALL_LIBS) $(LIBS_DIR)                               20

clean:   ; /bin/rm -f \#*\# *~ $(NAME) $(O_FILES) core
purge:   ; /bin/rm -f \#*\# *~ core
print:   ; enscript -2r -Pps1 $(INC_FILES) $(C_FILES)
```

E.2.2 Program code (ingres_test_db1.q)

```
/*************************************************************************
This code interfaces INGRES to the benchmarking program for test
database one.
*************************************************************************/

#include "test1.c"

##      int big_table[1001][9];
##      int big_table2[1001][9];
##      int lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8,lim9;

read_data()

/*************************************************************************
Read in the entire database.
*************************************************************************/

{
   j = 0;

##      ingres "testdb1"
##      retrieve (big_table[j][1]=main.num1,
##                big_table[j][2]=main.num2,
##                big_table[j][3]=main.num3,
##                big_table[j][4]=main.num4,
##                big_table[j][5]=main.num5,
##                big_table[j][6]=main.num6,
##                big_table[j][7]=main.num7,
##                big_table[j][8]=main.num8)
##      {
            j++;
##      }
##      exit
}
```

```
simple_query(lim1)
{

   j = 0;                                                          40

##      ingres "testdb1"
##      retrieve (big_table[j][1]=main.num1,
##                      big_table[j][2]=main.num2,
##                      big_table[j][3]=main.num3,
##                      big_table[j][4]=main.num4,
##                      big_table[j][5]=main.num5,
##                      big_table[j][6]=main.num6,
##                      big_table[j][7]=main.num7,
##                      big_table[j][8]=main.num8)                 50
##      where main.num1 > lim1
##      {
            j++;
##      }
##      exit
}

intermediate_query(lim1,lim2)

{                                                                   60

   j = 0;

##      ingres "testdb1"
##      retrieve (big_table[j][1]=main.num1,
##                      big_table[j][2]=main.num2,
##                      big_table[j][3]=main.num3,
##                      big_table[j][4]=main.num4,
##                      big_table[j][5]=main.num5,
##                      big_table[j][6]=main.num6,                 70
##                      big_table[j][7]=main.num7,
##                      big_table[j][8]=main.num8)
##      where main.num1 > lim1
##      and main.num2 > lim2
```

```
##      {
            j++;
##      }
##      exit
}
```

```
complex_query(lim1,lim2,lim3,lim4)

{

   j = 0;

##      ingres "testdb1"
##      retrieve (big_table[j][1]=main.num1,
##                  big_table[j][2]=main.num2,
##                  big_table[j][3]=main.num3,
##                  big_table[j][4]=main.num4,
##                  big_table[j][5]=main.num5,
##                  big_table[j][6]=main.num6,
##                  big_table[j][7]=main.num7,
##                  big_table[j][8]=main.num8)
##      where main.num1 > lim1
##      and main.num2 > lim2
##      and main.num3 > lim3
##      and main.num4 > lim4
##      {
            j++;
##      }
##      exit
}


very_complex_query(lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8)

{

   j = 0;
```

```
##      ingres "testdb1"
##      retrieve (big_table[j][1]=main.num1,
##                big_table[j][2]=main.num2,
##                big_table[j][3]=main.num3,
##                big_table[j][4]=main.num4,
##                big_table[j][5]=main.num5,
##                big_table[j][6]=main.num6,                    120
##                big_table[j][7]=main.num7,
##                big_table[j][8]=main.num8)
##      where main.num1 > lim1
##      and main.num2 > lim2
##      and main.num3 > lim3
##      and main.num4 > lim4
##      and main.num5 > lim5
##      and main.num6 > lim6
##      and main.num7 > lim7
##      and main.num8 > lim8                                    130
##      {
            j++;
##      }
##      exit
}


very_complex_query2(lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8)

{                                                               140

   j = 0;

##      ingres "testdb1"
##      retrieve (big_table[j][1]=main.num1,
##                big_table[j][2]=main.num2,
##                big_table[j][3]=main.num3,
##                big_table[j][4]=main.num4,
##                big_table[j][5]=main.num5,
##                big_table[j][6]=main.num6,                    150
```

```
##                  big_table[j][7]=main.num7,
##                  big_table[j][8]=main.num8)
##       where main.num1 > lim1
##       and main.num1 < lim2
##       and main.num2 > lim3
##       and main.num2 < lim4
##       and main.num3 > lim5
##       and main.num3 < lim6
##       and main.num4 > lim7
##       and main.num4 < lim8                                    160
##       {
             j++;
##       }
##       exit
}


ram_test()
{
   int i;                                                        170

   for (i = 0; i < 1000; i++)
   {
      big_table2[i][1] = big_table[i][1];
      big_table2[i][2] = big_table[i][2];
      big_table2[i][3] = big_table[i][3];
      big_table2[i][4] = big_table[i][4];
      big_table2[i][5] = big_table[i][5];
      big_table2[i][6] = big_table[i][6];
      big_table2[i][7] = big_table[i][7];                        180
      big_table2[i][8] = big_table[i][8];
   }
}
```

E.3 ObjectStore interface code

E.3.1 Makefile

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk

OS_COMPILATION_SCHEMA_DB_PATH= /$(LOGNAME)/testdb1.comp_schema
OS_APPLICATION_SCHEMA_DB_PATH= /$(LOGNAME)/testdb1.app_schema

LDLIBS = -los -losc -lq -lm -lc -lg

SOURCES = ostore_test_db1.c db1_schema.cc

OBJECTS = ostore_test_db1.o db1_schema.o                                    10

EXECUTABLES = ostore_test_db1

CPPFLAGS = -I$(OS_ROOTDIR)/include
CFLAGS = -g

LDFLAGS= $(OS_EXPORT)
CC = cc

all: $(EXECUTABLES)                                                         20

db_make: ostore_test_db1.o schema_standin
        $(OS_PRELINK) .os_db1_schema.cc \
            $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH)
        ostore_test_db1.o $(LDLIBS)
        OSCC -c .os_db1_schema.cc
        $(CC) $(CFLAGS) $(LDFLAGS) -o ostore_test_db1 ostore_test_db1.o \
        .os_db1_schema.o $(LDLIBS)

ostore_test_db1.o: ostore_test_db1.c                                        30
        $(CC) $(CPPFLAGS) $(CFLAGS) -c ostore_test_db1.c

schema_standin: db1_schema.cc
        OSCC -batch_schema $(OS_COMPILATION_SCHEMA_DB_PATH) db1_schema.cc
```

```
        touch schema_standin

clean:
        osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
        rm -f $(EXECUTABLES) $(OBJECTS) schema_standin

depend:
        osmakedep .depend $(CPPFLAGS) -files $(SOURCES)

include .depend
```

E.3.2 Program code (ostore_test_db1.c)

```
/*********************************************************************
This code interfaces ObjectStore to the benchmarking program for test
database one.
*********************************************************************/

#include <ostore/ostore.h>
#include "test1.c"

int no_of_big_table;
int big_table[1001][9];
int big_table2[1001][9];
int lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8,lim9;

no_of_big_table = 1000;

struct BIG_TABLE
{
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
```

```
    int num8;
};


read_data()
                                                                        30
/*********************************************************************
We read in the entire database.
*********************************************************************/

{
    int i;

    database *db1;
    start_objectstore ();
                                                                        40
    db1 = database_lookup_open("/shr/testdb1", 1, 0664);

    OS_BEGIN_TXN(t1,0,transaction_update)
    {
        database_root * BIG_TABLE_root;
        struct BIG_TABLE * BIG_TABLE;
        struct BIG_TABLE *BIG_TABLE_ptr;

        BIG_TABLE_root = database_root_find ("BIG_TABLE_entry",db1);
        BIG_TABLE_ptr = (struct BIG_TABLE*) BIG_TABLE_root;             50

        BIG_TABLE = (struct BIG_TABLE *)database_root_get_value(BIG_TABLE_root,
                                                                0);


        i = 0;

        for (BIG_TABLE_ptr = BIG_TABLE;
              BIG_TABLE_ptr < BIG_TABLE+no_of_big_table; BIG_TABLE_ptr++)
        {
            big_table[i][1] = BIG_TABLE_ptr->num1;                      60
            big_table[i][2] = BIG_TABLE_ptr->num2;
            big_table[i][3] = BIG_TABLE_ptr->num3;
```

```
            big_table[i][4] = BIG_TABLE_ptr->num4;
            big_table[i][5] = BIG_TABLE_ptr->num5;
            big_table[i][6] = BIG_TABLE_ptr->num6;
            big_table[i][7] = BIG_TABLE_ptr->num7;
            big_table[i][8] = BIG_TABLE_ptr->num8;

            i++;
        }                                                          70
    }
    OS_END_TXN(t1);

    database_close(db1);
}


simple_query(lim1)
{
    database *db1;                                                 80
    start_objectstore ();

    db1 = database_lookup_open("/shr/testdb1", 1, 0664);

    OS_BEGIN_TXN(t1,0,transaction_update)
    {
        database_root * BIG_TABLE_root;
        struct BIG_TABLE * BIG_TABLE;
        struct BIG_TABLE *BIG_TABLE_ptr;
                                                                   90
        BIG_TABLE_root = database_root_find ("BIG_TABLE_entry",db1);
        BIG_TABLE_ptr = (struct BIG_TABLE*) BIG_TABLE_root;

        BIG_TABLE = (struct BIG_TABLE *)database_root_get_value(BIG_TABLE_root,
                                                                0);
        j = 0;

        for (BIG_TABLE_ptr = BIG_TABLE;
              BIG_TABLE_ptr < BIG_TABLE+no_of_big_table; BIG_TABLE_ptr++)
        {                                                          100
```

```
            if (BIG_TABLE_ptr->num1 > lim1)
            {
                big_table[j][1] = BIG_TABLE_ptr->num1;
                big_table[j][2] = BIG_TABLE_ptr->num2;
                big_table[j][3] = BIG_TABLE_ptr->num3;
                big_table[j][4] = BIG_TABLE_ptr->num4;
                big_table[j][5] = BIG_TABLE_ptr->num5;
                big_table[j][6] = BIG_TABLE_ptr->num6;
                big_table[j][7] = BIG_TABLE_ptr->num7;
                big_table[j][8] = BIG_TABLE_ptr->num8;                    110

                j++;
            }
        }
    }
    OS_END_TXN(t1);

    database_close(db1);
}                                                                         120


intermediate_query(lim1,lim2)
{
    database *db1;
    start_objectstore ();

    db1 = database_lookup_open("/shr/testdb1", 1, 0664);

    OS_BEGIN_TXN(t1,0,transaction_update)
    {                                                                     130
        database_root * BIG_TABLE_root;
        struct BIG_TABLE * BIG_TABLE;
        struct BIG_TABLE *BIG_TABLE_ptr;

        BIG_TABLE_root = database_root_find ("BIG_TABLE_entry",db1);
        BIG_TABLE_ptr = (struct BIG_TABLE*) BIG_TABLE_root;

        BIG_TABLE = (struct BIG_TABLE *)database_root_get_value(BIG_TABLE_root,
```

```
                                                        0);
    j = 0;                                                                  140

    for (BIG_TABLE_ptr = BIG_TABLE;
         BIG_TABLE_ptr < BIG_TABLE+no_of_big_table; BIG_TABLE_ptr++)
    {
       if (BIG_TABLE_ptr->num1 > lim1 && BIG_TABLE_ptr->num2 > lim2)
       {
          big_table[j][1] = BIG_TABLE_ptr->num1;
          big_table[j][2] = BIG_TABLE_ptr->num2;
          big_table[j][3] = BIG_TABLE_ptr->num3;
          big_table[j][4] = BIG_TABLE_ptr->num4;                            150
          big_table[j][5] = BIG_TABLE_ptr->num5;
          big_table[j][6] = BIG_TABLE_ptr->num6;
          big_table[j][7] = BIG_TABLE_ptr->num7;
          big_table[j][8] = BIG_TABLE_ptr->num8;

          j++;
       }
    }
  }
  OS_END_TXN(t1);                                                           160

  database_close(db1);
}


complex_query(lim1,lim2,lim3,lim4)
{
  database *db1;
  start_objectstore ();
                                                                            170
  db1 = database_lookup_open("/shr/testdb1", 1, 0664);

  OS_BEGIN_TXN(t1,0,transaction_update)
  {
    database_root * BIG_TABLE_root;
    struct BIG_TABLE * BIG_TABLE;
```

```
        struct BIG_TABLE *BIG_TABLE_ptr;

        BIG_TABLE_root = database_root_find ("BIG_TABLE_entry",db1);
        BIG_TABLE_ptr = (struct BIG_TABLE*) BIG_TABLE_root;                    180

        BIG_TABLE = (struct BIG_TABLE *)database_root_get_value(BIG_TABLE_root,
                                                                0);
        j = 0;

        for (BIG_TABLE_ptr = BIG_TABLE;
             BIG_TABLE_ptr < BIG_TABLE+no_of_big_table; BIG_TABLE_ptr++)
        {
           if (BIG_TABLE_ptr->num1 > lim1 && BIG_TABLE_ptr->num2 > lim2 &&
               BIG_TABLE_ptr->num3 > lim3 && BIG_TABLE_ptr->num4 > lim4)    190
           {
              big_table[j][1] = BIG_TABLE_ptr->num1;
              big_table[j][2] = BIG_TABLE_ptr->num2;
              big_table[j][3] = BIG_TABLE_ptr->num3;
              big_table[j][4] = BIG_TABLE_ptr->num4;
              big_table[j][5] = BIG_TABLE_ptr->num5;
              big_table[j][6] = BIG_TABLE_ptr->num6;
              big_table[j][7] = BIG_TABLE_ptr->num7;
              big_table[j][8] = BIG_TABLE_ptr->num8;
                                                                           200
              j++;
           }
        }
     }
     OS_END_TXN(t1);

     database_close(db1);
}

                                                                           210
very_complex_query(lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8)
{
     database *db1;
     start_objectstore ();
```

```
db1 = database_lookup_open("/shr/testdb1", 1, 0664);

OS_BEGIN_TXN(t1,0,transaction_update)
{
   database_root * BIG_TABLE_root;                                          220
   struct BIG_TABLE * BIG_TABLE;
   struct BIG_TABLE *BIG_TABLE_ptr;

   BIG_TABLE_root = database_root_find ("BIG_TABLE_entry",db1);
   BIG_TABLE_ptr = (struct BIG_TABLE*) BIG_TABLE_root;

   BIG_TABLE = (struct BIG_TABLE *)database_root_get_value(BIG_TABLE_root,
                                                           0);
   j = 0;
                                                                            230
   for (BIG_TABLE_ptr = BIG_TABLE;
        BIG_TABLE_ptr < BIG_TABLE+no_of_big_table; BIG_TABLE_ptr++)
   {
      if (BIG_TABLE_ptr->num1 > lim1 && BIG_TABLE_ptr->num2 > lim2 &&
          BIG_TABLE_ptr->num3 > lim3 && BIG_TABLE_ptr->num4 > lim4 &&
          BIG_TABLE_ptr->num5 > lim5 && BIG_TABLE_ptr->num6 > lim6 &&
          BIG_TABLE_ptr->num7 > lim7 && BIG_TABLE_ptr->num8 > lim8)
      {
         big_table[j][1] = BIG_TABLE_ptr->num1;
         big_table[j][2] = BIG_TABLE_ptr->num2;                             240
         big_table[j][3] = BIG_TABLE_ptr->num3;
         big_table[j][4] = BIG_TABLE_ptr->num4;
         big_table[j][5] = BIG_TABLE_ptr->num5;
         big_table[j][6] = BIG_TABLE_ptr->num6;
         big_table[j][7] = BIG_TABLE_ptr->num7;
         big_table[j][8] = BIG_TABLE_ptr->num8;

         j++;
      }
   }                                                                        250
}
OS_END_TXN(t1);
```

```
    database_close(db1);
}


very_complex_query2(lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8)
{
    database *db1;                                                    260
    start_objectstore ();

    db1 = database_lookup_open("/shr/testdb1", 1, 0664);

    OS_BEGIN_TXN(t1,0,transaction_update)
    {
        database_root * BIG_TABLE_root;
        struct BIG_TABLE * BIG_TABLE;
        struct BIG_TABLE *BIG_TABLE_ptr;
                                                                      270
        BIG_TABLE_root = database_root_find ("BIG_TABLE_entry",db1);
        BIG_TABLE_ptr = (struct BIG_TABLE*) BIG_TABLE_root;

        BIG_TABLE = (struct BIG_TABLE *)database_root_get_value(BIG_TABLE_root,
                                                                0);
        j = 0;

        for (BIG_TABLE_ptr = BIG_TABLE;
             BIG_TABLE_ptr < BIG_TABLE+no_of_big_table; BIG_TABLE_ptr++)
        {                                                             280
            if (BIG_TABLE_ptr->num1 > lim1 && BIG_TABLE_ptr->num1 < lim2 &&
                BIG_TABLE_ptr->num2 > lim3 && BIG_TABLE_ptr->num2 < lim4 &&
                BIG_TABLE_ptr->num3 > lim5 && BIG_TABLE_ptr->num3 < lim6 &&
                BIG_TABLE_ptr->num4 > lim7 && BIG_TABLE_ptr->num4 < lim8)
            {
                big_table[j][1] = BIG_TABLE_ptr->num1;
                big_table[j][2] = BIG_TABLE_ptr->num2;
                big_table[j][3] = BIG_TABLE_ptr->num3;
                big_table[j][4] = BIG_TABLE_ptr->num4;
                big_table[j][5] = BIG_TABLE_ptr->num5;                290
```

```
            big_table[j][6] = BIG_TABLE_ptr->num6;
            big_table[j][7] = BIG_TABLE_ptr->num7;
            big_table[j][8] = BIG_TABLE_ptr->num8;

            j++;
        }
    }
}
OS_END_TXN(t1);

database_close(db1);
}


ram_test()
{
    int i;

    for (i = 0; i < 1000; i++)
    {
        big_table2[i][1] = big_table[i][1];
        big_table2[i][2] = big_table[i][2];
        big_table2[i][3] = big_table[i][3];
        big_table2[i][4] = big_table[i][4];
        big_table2[i][5] = big_table[i][5];
        big_table2[i][6] = big_table[i][6];
        big_table2[i][7] = big_table[i][7];
        big_table2[i][8] = big_table[i][8];
    }
}
```

Appendix F

Benchmarking Code for Test Database Two (Object–Oriented)

F.1 General program code (test2.c)

```
/*****************************************************************
This program runs the benchmark tests on test database two.
*****************************************************************/

#include <stdio.h>
#include <string.h>
#include <time.h>

#define STREQ(A, B)     (*(A) == *(B)  && strcmp((A), (B)) == 0)

#define MALLOCN(P,T,N,M) \
   if (((P) = (T *) malloc((unsigned) (N) * sizeof(T))) ==  NULL) { \
      fprintf(stderr, "Malloc failure: (%d bytes) %s\n", (N) * sizeof(T), M); \
      exit(1); \
   } \
   else
```

```
#define FREE(P) free((char *) P)

typedef struct {
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    int num8;
} Bottom;

typedef struct {
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    char table[20];
    Bottom *bottom;
} Middle;

typedef struct {
    int num1;
    int num2;
    int num3;
    int num4;
    int num5;
    int num6;
    int num7;
    char table[20];
    Middle *middle;
} Top;

Top *top;
```

```
Middle *middle;
Bottom *bottom;

int no_of_top;
int no_of_middle;                                                    60
int no_of_bottom;
int j,k,l;

no_of_top = 10;
no_of_middle = 4;
no_of_bottom = 1000;

int main()

{                                                                    70
   time_t start,end;
   float c;
   long t;
   int i,n;

/*********************************************************************
                          read entire database
*********************************************************************/

   n = 1;                                                            80

   t = time(0);

   for (i = 0; i < n; ++i)
      read_data();

   t = time(0) - t;

   printf("%ld seconds elapsed in %d trials.\n\n",t,n);
                                                                     90
/*********************************************************************
                          simple query number 1
*********************************************************************/
```

```
    t = time(0);

    for (i = 0; i < n; ++i)
        simple_query(1);

    t = time(0) - t;                                                    100

    printf("%d hits on simple query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                            simple query number 2
*************************************************************************/

    t = time(0);
                                                                        110
    for (i = 0; i < n; ++i)
        simple_query(25);

    t = time(0) - t;

    printf("%d hits on simple query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                            simple query number 3                       120
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        simple_query(50);

    t = time(0) - t;

    printf("%d hits on simple query\n",l);                              130
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);
```

```
/*************************************************************************
                        simple query number 4
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        simple_query(75);                                                  140

    t = time(0) - t;

    printf("%d hits on simple query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                        simple query number 5
*************************************************************************/
    t = time(0);                                                           150

    for (i = 0; i < n; ++i)
        simple_query(99);

    t = time(0) - t;

    printf("%d hits on simple query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************  160
                        intermediate query
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        intermediate_query(50,50);

    t = time(0) - t;
```

```
                                                                    170
    printf("%d hits on intermediate(1) query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                              complex query
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)                                         180
        complex_query(50,50,50,50);

    t = time(0) - t;

    printf("%d hits on complex(1) query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                           very complex query 1
*************************************************************************/  190

    t = time(0);

    for (i = 0; i < n; ++i)
        very_complex_query(50,50,50,50,50,50,50,50);

    t = time(0) - t;

    printf("%d hits on very complex(1) query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);                200

/*************************************************************************
                           very complex query 2
*************************************************************************/

    t = time(0);
```

```
    for (i = 0; i < n; ++i)
        very_complex_query(5,5,5,5,5,5,5,5);

    t = time(0) - t;

    printf("%d hits on very complex(2) query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

/*************************************************************************
                        very complex query type 2
*************************************************************************/

    t = time(0);

    for (i = 0; i < n; ++i)
        very_complex_query2(5,95,5,95,5,95,5,95);

    t = time(0) - t;

    printf("%d hits on very complex2 query\n",l);
    printf("%ld seconds elapsed in %d trials.\n\n",t,n);

}
```

F.2 INGRES interface code

F.2.1 Makefile

```
NAME     =    ingres_test_db2

C_FILES  =    ingres_test_db2.c

O_FILES  =    ingres_test_db2.o

INC_FILES =   malloc.h nr.h nrutil.h
SRC_FILES =     $(INC_FILES) $(C_FILES)
```

```
CC      =   cc
CFLAGS  =   -g
ALL_LIBS =    -lm -lc -lq
LIBS_DIR =


all:            $(NAME)

$(NAME):        $(O_FILES)
                $(CC) $(CFLAGS) $(CPPFLAGS) -o $(NAME) $(O_FILES) \
                $(ALL_LIBS) $(LIBS_DIR)

clean:  ; /bin/rm -f \#*\# *~ $(NAME) $(O_FILES) core
purge:  ; /bin/rm -f \#*\# *~ core
print:  ; enscript -2r -Pps1 $(INC_FILES) $(C_FILES)
```

F.2.2 Program code (ingres_test_db2.q)

```
/*************************************************************************
This code interfaces INGRES to the benchmarking program for test
database two.
*************************************************************************/

#include "test2.c"

##      int     NUM1;
##      int     NUM2;
##      int     NUM3;
##      int     NUM4;
##      int     NUM5;
##      int     NUM6;
##      int     NUM7;
##      int     NUM8;
##      char    MID[21];
##      char    BOT[21];
##      char    TABLE[21];
##      int  lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8,lim9;
```

```
                                                                        20
read_data()

/*************************************************************************
We read in the entire database.
*************************************************************************/

{
   int i;

   i = 0;                                                               30

   MALLOCN(top, Top, no_of_top, "Malloc 1");

##      ingres "testdb2"
##      retrieve (NUM1=top.num1,                                        .
##                NUM2=top.num2,
##                NUM3=top.num3,                                        .
##                NUM4=top.num4,
##                NUM5=top.num5,                                        .
##                NUM6=top.num6,                                       40
##                NUM7=top.num7,
##                TABLE=top.table)
##      {

           strcpy(top[i].table,TABLE);

           top[i].num1 = NUM1;
           top[i].num2 = NUM2;
           top[i].num3 = NUM3;
           top[i].num4 = NUM4;                                          50
           top[i].num5 = NUM5;
           top[i].num6 = NUM6;
           top[i].num7 = NUM7;

           i++;
##      }
```

```
    for (i = 0; i < no_of_top; ++i)
    {
        MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");

        strcpy(MID,top[i].table);

        j = 0;

##      retrieve (NUM1=MID.num1,
##                NUM2=MID.num2,
##                NUM3=MID.num3,
##                NUM4=MID.num4,
##                NUM5=MID.num5,
##                NUM6=MID.num6,
##                NUM7=MID.num7,
##                TABLE=MID.table)
##      {
            strcpy(top[i].middle[j].table,TABLE);

            top[i].middle[j].num1 = NUM1;
            top[i].middle[j].num2 = NUM2;
            top[i].middle[j].num3 = NUM3;
            top[i].middle[j].num4 = NUM4;
            top[i].middle[j].num5 = NUM5;
            top[i].middle[j].num6 = NUM6;
            top[i].middle[j].num7 = NUM7;

            j++;
##      }
    }

        for (i = 0; i < no_of_top; ++i)
        {

            for (j = 0; j < no_of_middle; ++j)
            {

                MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");
```

```
            strcpy(BOT,top[i].middle[j].table);

            k = 0;

##              retrieve (NUM1=BOT.num1,
##                        NUM2=BOT.num2,
##                        NUM3=BOT.num3,
##                        NUM4=BOT.num4,
##                        NUM5=BOT.num5,
##                        NUM6=BOT.num6,
##                        NUM7=BOT.num7,
##                        NUM8=BOT.num8)
##              {
                    top[i].middle[j].bottom[k].num1 = NUM1;
                    top[i].middle[j].bottom[k].num2 = NUM2;
                    top[i].middle[j].bottom[k].num3 = NUM3;
                    top[i].middle[j].bottom[k].num4 = NUM4;
                    top[i].middle[j].bottom[k].num5 = NUM5;
                    top[i].middle[j].bottom[k].num6 = NUM6;
                    top[i].middle[j].bottom[k].num7 = NUM7;
                    top[i].middle[j].bottom[k].num8 = NUM8;

                    k++;
##              }
        }
    }
##      exit

  FREE(top);

  for (i = 0; i < no_of_top; ++i)
  {
    FREE(top[i].middle);
  }

  for (i = 0; i < no_of_top; ++i)
  {
```

```
        for (j = 0; j < no_of_middle; ++j)
        {
            FREE(top[i].middle[j].bottom);
        }
    }
}
                                                                    140

simple_query(lim1)
{

    int i;

    i = 0;

    MALLOCN(top, Top, no_of_top, "Malloc 1");
                                                                    150
##      ingres "testdb2"
##      retrieve (TABLE=top.table)
##      {
            strcpy(top[i].table,TABLE);
            i++;
##      }

    for (i = 0; i < no_of_top; ++i)
    {
        MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");   160

        strcpy(MID,top[i].table);

        j = 0;

##      retrieve (TABLE=MID.table)
##      {
            strcpy(top[i].middle[j].table,TABLE);
            j++;
##      }                                                           170
    }
```

```
    l = 0;

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)
        {
            MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");
                                                                              180
            strcpy(BOT,top[i].middle[j].table);

                k = 0;

##              retrieve (NUM1=BOT.num1,
##                        NUM2=BOT.num2,
##                        NUM3=BOT.num3,
##                        NUM4=BOT.num4,
##                        NUM5=BOT.num5,
##                        NUM6=BOT.num6,                                      190
##                        NUM7=BOT.num7,
##                        NUM8=BOT.num8)
##              where BOT.num1 > lim1
##              {
                    top[i].middle[j].bottom[k].num1 = NUM1;
                    top[i].middle[j].bottom[k].num2 = NUM2;
                    top[i].middle[j].bottom[k].num3 = NUM3;
                    top[i].middle[j].bottom[k].num4 = NUM4;
                    top[i].middle[j].bottom[k].num5 = NUM5;
                    top[i].middle[j].bottom[k].num6 = NUM6;                   200
                    top[i].middle[j].bottom[k].num7 = NUM7;
                    top[i].middle[j].bottom[k].num8 = NUM8;

                    k++;
                    l++;
##              }
            }
        }
##      exit
```

```
                                                                    210
    FREE(top);

    for (i = 0; i < no_of_top; ++i)
    {
        FREE(top[i].middle);
    }

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)                          220
        {
            FREE(top[i].middle[j].bottom);
        }
    }
}


intermediate_query(lim1,lim2)
{
                                                                    230
    int i;

    i = 0;

    MALLOCN(top, Top, no_of_top, "Malloc 1");

##      ingres "testdb2"
##      retrieve (TABLE=top.table)
##      {
            strcpy(top[i].table,TABLE);                             240
            i++;
##      }

    for (i = 0; i < no_of_top; ++i)
    {
        MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");
```

```
        strcpy(MID,top[i].table);

        j = 0;                                                          250

##      retrieve (TABLE=MID.table)
##      {
            strcpy(top[i].middle[j].table,TABLE);
            j++;
##      }
    }

    l = 0;
                                                                        260
    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)
        {
            MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

            strcpy(BOT,top[i].middle[j].table);

                k = 0;
                                                                        270
##              retrieve (NUM1=BOT.num1,
##                        NUM2=BOT.num2,
##                        NUM3=BOT.num3,
##                        NUM4=BOT.num4,
##                        NUM5=BOT.num5,
##                        NUM6=BOT.num6,
##                        NUM7=BOT.num7,
##                        NUM8=BOT.num8)
##              where BOT.num1 > lim1
##              and BOT.num2 > lim2                                     280
##              {
                    top[i].middle[j].bottom[k].num1 = NUM1;
                    top[i].middle[j].bottom[k].num2 = NUM2;
                    top[i].middle[j].bottom[k].num3 = NUM3;
                    top[i].middle[j].bottom[k].num4 = NUM4;
```

```
                    top[i].middle[j].bottom[k].num5 = NUM5;
                    top[i].middle[j].bottom[k].num6 = NUM6;
                    top[i].middle[j].bottom[k].num7 = NUM7;
                    top[i].middle[j].bottom[k].num8 = NUM8;
                                                                          290
                    k++;
                    l++;
##              }
        }
    }
##    exit

    FREE(top);

    for (i = 0; i < no_of_top; ++i)                                       300
    {
        FREE(top[i].middle);
    }

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)
        {
            FREE(top[i].middle[j].bottom);
        }                                                                 310
    }
}


complex_query(lim1,lim2,lim3,lim4)
{

    int i;

    i = 0;                                                                320

    MALLOCN(top, Top, no_of_top, "Malloc 1");
```

```
##      ingres "testdb2"
##      retrieve (TABLE=top.table)
##      {
            strcpy(top[i].table,TABLE);
            i++;
##      }

    for (i = 0; i < no_of_top; ++i)
    {
        MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");

        strcpy(MID,top[i].table);

        j = 0;

##      retrieve (TABLE=MID.table)
##      {
            strcpy(top[i].middle[j].table,TABLE);
            j++;
##      }
    }

    l = 0;

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)
        {
            MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

            strcpy(BOT,top[i].middle[j].table);

                k = 0;

##              retrieve (NUM1=BOT.num1,
##                        NUM2=BOT.num2,
##                        NUM3=BOT.num3,
##                        NUM4=BOT.num4,
```

```
##                          NUM5=BOT.num5,
##                          NUM6=BOT.num6,
##                          NUM7=BOT.num7,
##                          NUM8=BOT.num8)
##                  where BOT.num1 > lim1
##                  and BOT.num2 > lim2
##                  and BOT.num3 > lim3
##                  and BOT.num4 > lim4
##                  {                                                    370
                        top[i].middle[j].bottom[k].num1 = NUM1;
                        top[i].middle[j].bottom[k].num2 = NUM2;
                        top[i].middle[j].bottom[k].num3 = NUM3;
                        top[i].middle[j].bottom[k].num4 = NUM4;
                        top[i].middle[j].bottom[k].num5 = NUM5;
                        top[i].middle[j].bottom[k].num6 = NUM6;
                        top[i].middle[j].bottom[k].num7 = NUM7;
                        top[i].middle[j].bottom[k].num8 = NUM8;

                        k++;                                           380
                        l++;
##                  }
            }
        }
##      exit

    FREE(top);

    for (i = 0; i < no_of_top; ++i)
    {                                                                  390
        FREE(top[i].middle);
    }

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)
        {
            FREE(top[i].middle[j].bottom);
        }
```

```
    }                                                                    400
}

very_complex_query(lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8)
{

    int i;

    i = 0;
                                                                         410
    MALLOCN(top, Top, no_of_top, "Malloc 1");

##      ingres "testdb2"
##      retrieve (TABLE=top.table)
##      {
            strcpy(top[i].table,TABLE);
            i++;
##      }

    for (i = 0; i < no_of_top; ++i)                                      420
    {
        MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");

        strcpy(MID,top[i].table);

        j = 0;

##      retrieve (TABLE=MID.table)
##      {
            strcpy(top[i].middle[j].table,TABLE);                        430
            j++;
##      }
    }

    l = 0;

    for (i = 0; i < no_of_top; ++i)
```

```
{
    for (j = 0; j < no_of_middle; ++j)
    {                                                                    440
        MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

        strcpy(BOT,top[i].middle[j].table);

            k = 0;

##              retrieve (NUM1=BOT.num1,
##                        NUM2=BOT.num2,
##                        NUM3=BOT.num3,
##                        NUM4=BOT.num4,                                 450
##                        NUM5=BOT.num5,
##                        NUM6=BOT.num6,
##                        NUM7=BOT.num7,
##                        NUM8=BOT.num8)
##              where BOT.num1 > lim1
##              and BOT.num2 > lim2
##              and BOT.num3 > lim3
##              and BOT.num4 > lim4
##              and BOT.num5 > lim5
##              and BOT.num6 > lim6                                       460
##              and BOT.num7 > lim7
##              and BOT.num8 > lim8
##              {
                    top[i].middle[j].bottom[k].num1 = NUM1;
                    top[i].middle[j].bottom[k].num2 = NUM2;
                    top[i].middle[j].bottom[k].num3 = NUM3;
                    top[i].middle[j].bottom[k].num4 = NUM4;
                    top[i].middle[j].bottom[k].num5 = NUM5;
                    top[i].middle[j].bottom[k].num6 = NUM6;
                    top[i].middle[j].bottom[k].num7 = NUM7;               470
                    top[i].middle[j].bottom[k].num8 = NUM8;

                    k++;
                    l++;
##              }
```

```
            }
        }
##      exit

    FREE(top);

    for (i = 0; i < no_of_top; ++i)
    {
        FREE(top[i].middle);
    }

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)
        {
            FREE(top[i].middle[j].bottom);
        }
    }
}


very_complex_query2(lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8)
{

    int i;

    i = 0;

    MALLOCN(top, Top, no_of_top, "Malloc 1");

##      ingres "testdb2"
##      retrieve (TABLE=top.table)
##      {
            strcpy(top[i].table,TABLE);
            i++;
##      }

    for (i = 0; i < no_of_top; ++i)
```

```
    {
        MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");

        strcpy(MID,top[i].table);

        j = 0;                                                          520

##          retrieve (TABLE=MID.table)
##          {
                strcpy(top[i].middle[j].table,TABLE);
                j++;
##          }
    }

    l = 0;

    for (i = 0; i < no_of_top; ++i)                                     530
    {
        for (j = 0; j < no_of_middle; ++j)
        {
            MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

            strcpy(BOT,top[i].middle[j].table);

                k = 0;

##                  retrieve (NUM1=BOT.num1,                            540
##                            NUM2=BOT.num2,
##                            NUM3=BOT.num3,
##                            NUM4=BOT.num4,
##                            NUM5=BOT.num5,
##                            NUM6=BOT.num6,
##                            NUM7=BOT.num7,
##                            NUM8=BOT.num8)
##                  where BOT.num1 > lim1
##                  and BOT.num1 < lim2
##                  and BOT.num2 > lim3                                 550
##                  and BOT.num2 < lim4
```

```
##              and BOT.num3 > lim5
##              and BOT.num3 < lim6
##              and BOT.num4 > lim7
##              and BOT.num4 < lim8
##              {
                    top[i].middle[j].bottom[k].num1 = NUM1;
                    top[i].middle[j].bottom[k].num2 = NUM2;
                    top[i].middle[j].bottom[k].num3 = NUM3;
                    top[i].middle[j].bottom[k].num4 = NUM4;                    560
                    top[i].middle[j].bottom[k].num5 = NUM5;
                    top[i].middle[j].bottom[k].num6 = NUM6;
                    top[i].middle[j].bottom[k].num7 = NUM7;
                    top[i].middle[j].bottom[k].num8 = NUM8;

                    k++;
                    l++;
##              }
        }
    }                                                                          570
##      exit


  FREE(top);

  for (i = 0; i < no_of_top; ++i)
  {
    FREE(top[i].middle);
  }
                                                                               580
  for (i = 0; i < no_of_top; ++i)
  {
    for (j = 0; j < no_of_middle; ++j)
    {
      FREE(top[i].middle[j].bottom);
    }
  }
```

F.3 ObjectStore interface code

F.3.1 Makefile

```
include $(OS_ROOTDIR)/etc/ostore.lib.mk

OS_COMPILATION_SCHEMA_DB_PATH= /$(LOGNAME)/testdb2.comp_schema
OS_APPLICATION_SCHEMA_DB_PATH= /$(LOGNAME)/testdb2.app_schema

LDLIBS = -los -losc -lq -lm -lc -lg

SOURCES = ostore_test_db2.c db2_schema.cc

OBJECTS = ostore_test_db2.o db2_schema.o                                    10

EXECUTABLES = ostore_test_db2

CPPFLAGS = -I$(OS_ROOTDIR)/include
CFLAGS = -g

LDFLAGS= $(OS_EXPORT)
CC = cc

all: $(EXECUTABLES)                                                         20

db_make: ostore_test_db2.o schema_standin
        $(OS_PRELINK) .os_db2_schema.cc \
            $(OS_COMPILATION_SCHEMA_DB_PATH) $(OS_APPLICATION_SCHEMA_DB_PATH)
        ostore_test_db2.o $(LDLIBS)
        OSCC -c .os_db2_schema.cc
        $(CC) $(CFLAGS) $(LDFLAGS) -o ostore_test_db2 ostore_test_db2.o \
        .os_db2_schema.o $(LDLIBS)

ostore_test_db2.o: ostore_test_db2.c                                        30
        $(CC) $(CPPFLAGS) $(CFLAGS) -c ostore_test_db2.c

schema_standin: db2_schema.cc
        OSCC -batch_schema $(OS_COMPILATION_SCHEMA_DB_PATH) db2_schema.cc
```

```
        touch schema_standin

clean:
        osrm -f $(OS_COMPILATION_SCHEMA_DB_PATH)
        rm -f $(EXECUTABLES) $(OBJECTS) schema_standin

depend:
        osmakedep .depend $(CPPFLAGS) -files $(SOURCES)

include .depend
```

F.3.2 Program code (ostore_test_db2.c)

```
/*********************************************************************
This code interfaces ObjectStore to the benchmarking program for test
database two.
*********************************************************************/

#include <ostore/ostore.h>
#include "testdb2.h"
#include "test2.c"

int lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8,lim9;

read_data()

/*********************************************************************
We read in the entire database.
*********************************************************************/

{
   int i;

   database *db1;
   start_objectstore ();

   db1 = database_lookup_open("/shr/testdb2", 1, 0664);
```

```
OS_BEGIN_TXN(t1,0,transaction_update)
{
   database_root * TOP_root;

   struct BOTTOM * BOTTOM;                                          30
   struct MIDDLE * MIDDLE;
   struct TOP * TOP;

   struct BOTTOM *BOTTOM_ptr;
   struct MIDDLE *MIDDLE_ptr;
   struct TOP *TOP_ptr;

   TOP_root = database_root_find ("TOP_entry",db1);
   TOP_ptr = (struct TOP*) TOP_root;
                                                                    40
   TOP = (struct TOP *)database_root_get_value(TOP_root,0);

   i = 0;

   MALLOCN(top, Top, no_of_top, "Malloc 1");

   for (TOP_ptr = TOP; TOP_ptr < TOP+no_of_top; TOP_ptr++)
   {
      top[i].num1 = TOP_ptr->num1;
      top[i].num2 = TOP_ptr->num2;                                  50
      top[i].num3 = TOP_ptr->num3;
      top[i].num4 = TOP_ptr->num4;
      top[i].num5 = TOP_ptr->num5;
      top[i].num6 = TOP_ptr->num6;
      top[i].num7 = TOP_ptr->num7;

      MIDDLE = TOP_ptr->middle;

      j = 0;
                                                                    60
      MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");
```

```
        for (MIDDLE_ptr = MIDDLE; MIDDLE_ptr <
              MIDDLE+no_of_middle; MIDDLE_ptr++)
        {
            top[i].middle[j].num1 = MIDDLE_ptr->num1;
            top[i].middle[j].num2 = MIDDLE_ptr->num2;
            top[i].middle[j].num3 = MIDDLE_ptr->num3;
            top[i].middle[j].num4 = MIDDLE_ptr->num4;
            top[i].middle[j].num5 = MIDDLE_ptr->num5;                    70
            top[i].middle[j].num6 = MIDDLE_ptr->num6;
            top[i].middle[j].num7 = MIDDLE_ptr->num7;

            BOTTOM = MIDDLE_ptr->bottom;

            k = 0;

            MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

            for (BOTTOM_ptr = BOTTOM; BOTTOM_ptr <                       80
                  BOTTOM+no_of_bottom; BOTTOM_ptr++)
            {
                top[i].middle[j].bottom[k].num1 = BOTTOM_ptr->num1;
                top[i].middle[j].bottom[k].num2 = BOTTOM_ptr->num2;
                top[i].middle[j].bottom[k].num3 = BOTTOM_ptr->num3;
                top[i].middle[j].bottom[k].num4 = BOTTOM_ptr->num4;
                top[i].middle[j].bottom[k].num5 = BOTTOM_ptr->num5;
                top[i].middle[j].bottom[k].num6 = BOTTOM_ptr->num6;
                top[i].middle[j].bottom[k].num7 = BOTTOM_ptr->num7;
                                                                         90
                k++;
            }
            j++;
        }
        i++;
    }
}
OS_END_TXN(t1);
database_close(db1);
                                                                         100
```

```
    FREE(top);

    for (i = 0; i < no_of_top; ++i)
    {
        FREE(top[i].middle);
    }

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)                    110
        {
            FREE(top[i].middle[j].bottom);
        }
    }
}


simple_query(lim1)
{
    int i;                                                    120

    database *db1;
    start_objectstore ();

    db1 = database_lookup_open("/shr/testdb2", 1, 0664);

    OS_BEGIN_TXN(t1,0,transaction_update)
    {
        database_root * TOP_root;
                                                              130
        struct BOTTOM * BOTTOM;
        struct MIDDLE * MIDDLE;
        struct TOP * TOP;

        struct BOTTOM *BOTTOM_ptr;
        struct MIDDLE *MIDDLE_ptr;
        struct TOP *TOP_ptr;
```

```
TOP_root = database_root_find ("TOP_entry",db1);
TOP_ptr = (struct TOP*) TOP_root;                                    140

TOP = (struct TOP *)database_root_get_value(TOP_root,0);

i = 0;
l = 0;

MALLOCN(top, Top, no_of_top, "Malloc 1");

for (TOP_ptr = TOP; TOP_ptr < TOP+no_of_top; TOP_ptr++)
{                                                                    150
   MIDDLE = TOP_ptr->middle;

   j = 0;

   MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");

   for (MIDDLE_ptr = MIDDLE; MIDDLE_ptr <
        MIDDLE+no_of_middle; MIDDLE_ptr++)
   {
      BOTTOM = MIDDLE_ptr->bottom;                                   160

      k = 0;

      MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

      for (BOTTOM_ptr = BOTTOM; BOTTOM_ptr <
           BOTTOM+no_of_bottom; BOTTOM_ptr++)
         if (BOTTOM_ptr->num1 > lim1)
         {
            top[i].middle[j].bottom[k].num1 = BOTTOM_ptr->num1;      170
            top[i].middle[j].bottom[k].num2 = BOTTOM_ptr->num2;
            top[i].middle[j].bottom[k].num3 = BOTTOM_ptr->num3;
            top[i].middle[j].bottom[k].num4 = BOTTOM_ptr->num4;
            top[i].middle[j].bottom[k].num5 = BOTTOM_ptr->num5;
            top[i].middle[j].bottom[k].num6 = BOTTOM_ptr->num6;
            top[i].middle[j].bottom[k].num7 = BOTTOM_ptr->num7;
```

```
                k++;
                l++;
            }
        j++;
    }
    i++;
  }
}
OS_END_TXN(t1);
database_close(db1);

FREE(top);

for (i = 0; i < no_of_top; ++i)
{
    FREE(top[i].middle);
}

for (i = 0; i < no_of_top; ++i)
{
    for (j = 0; j < no_of_middle; ++j)
    {
        FREE(top[i].middle[j].bottom);
    }
}
}

intermediate_query(lim1,lim2)
{
    int i;

    database *db1;
    start_objectstore ();

    db1 = database_lookup_open("/shr/testdb2", 1, 0664);

    OS_BEGIN_TXN(t1,0,transaction_update)
```

```
{
   database_root * TOP_root;

   struct BOTTOM * BOTTOM;
   struct MIDDLE * MIDDLE;
   struct TOP * TOP;                                                    220

   struct BOTTOM *BOTTOM_ptr;
   struct MIDDLE *MIDDLE_ptr;
   struct TOP *TOP_ptr;

   TOP_root = database_root_find ("TOP_entry",db1);
   TOP_ptr = (struct TOP*) TOP_root;

   TOP = (struct TOP *)database_root_get_value(TOP_root,0);
                                                                        230
   i = 0;
   l = 0;

   MALLOCN(top, Top, no_of_top, "Malloc 1");

   for (TOP_ptr = TOP; TOP_ptr < TOP+no_of_top; TOP_ptr++)
   {
      MIDDLE = TOP_ptr->middle;

      j = 0;                                                            240

      MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");

      for (MIDDLE_ptr = MIDDLE; MIDDLE_ptr <
           MIDDLE+no_of_middle; MIDDLE_ptr++)
      {
         BOTTOM = MIDDLE_ptr->bottom;

         k = 0;
                                                                        250
         MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");
```

```
            for (BOTTOM_ptr = BOTTOM; BOTTOM_ptr <
                  BOTTOM+no_of_bottom; BOTTOM_ptr++)
               if (BOTTOM_ptr->num1 > lim1 && BOTTOM_ptr->num2 > lim2)
               {
                  top[i].middle[j].bottom[k].num1 = BOTTOM_ptr->num1;
                  top[i].middle[j].bottom[k].num2 = BOTTOM_ptr->num2;
                  top[i].middle[j].bottom[k].num3 = BOTTOM_ptr->num3;
                  top[i].middle[j].bottom[k].num4 = BOTTOM_ptr->num4;
                  top[i].middle[j].bottom[k].num5 = BOTTOM_ptr->num5;
                  top[i].middle[j].bottom[k].num6 = BOTTOM_ptr->num6;
                  top[i].middle[j].bottom[k].num7 = BOTTOM_ptr->num7;

                  k++;
                  l++;
               }
            j++;
         }
         i++;
      }
   }
   OS_END_TXN(t1);
   database_close(db1);

   FREE(top);

   for (i = 0; i < no_of_top; ++i)
   {
      FREE(top[i].middle);
   }

   for (i = 0; i < no_of_top; ++i)
   {
      for (j = 0; j < no_of_middle; ++j)
      {
         FREE(top[i].middle[j].bottom);
      }
   }
}
```

```
complex_query(lim1,lim2,lim3,lim4)
{
   int i;

   database *db1;
   start_objectstore ();

   db1 = database_lookup_open("/shr/testdb2", 1, 0664);
                                                                          300
   OS_BEGIN_TXN(t1,0,transaction_update)
   {
      database_root * TOP_root;

      struct BOTTOM * BOTTOM;
      struct MIDDLE * MIDDLE;
      struct TOP * TOP;

      struct BOTTOM *BOTTOM_ptr;
      struct MIDDLE *MIDDLE_ptr;                                          310
      struct TOP *TOP_ptr;

      TOP_root = database_root_find ("TOP_entry",db1);
      TOP_ptr = (struct TOP*) TOP_root;

      TOP = (struct TOP *)database_root_get_value(TOP_root,0);

      i = 0;
      l = 0;
                                                                          320
      MALLOCN(top, Top, no_of_top, "Malloc 1");

      for (TOP_ptr = TOP; TOP_ptr < TOP+no_of_top; TOP_ptr++)
      {
         MIDDLE = TOP_ptr->middle;

         j = 0;
```

```
        MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");
                                                                        330
        for (MIDDLE_ptr = MIDDLE; MIDDLE_ptr <
              MIDDLE+no_of_middle; MIDDLE_ptr++)
        {
           BOTTOM = MIDDLE_ptr->bottom;

           k = 0;

           MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

           for (BOTTOM_ptr = BOTTOM; BOTTOM_ptr <                       340
                 BOTTOM+no_of_bottom; BOTTOM_ptr++)
              if (BOTTOM_ptr->num1 > lim1 && BOTTOM_ptr->num2 > lim2 &&
                   BOTTOM_ptr->num3 > lim3 && BOTTOM_ptr->num4 > lim4)
              {
                 top[i].middle[j].bottom[k].num1 = BOTTOM_ptr->num1;
                 top[i].middle[j].bottom[k].num2 = BOTTOM_ptr->num2;
                 top[i].middle[j].bottom[k].num3 = BOTTOM_ptr->num3;
                 top[i].middle[j].bottom[k].num4 = BOTTOM_ptr->num4;
                 top[i].middle[j].bottom[k].num5 = BOTTOM_ptr->num5;
                 top[i].middle[j].bottom[k].num6 = BOTTOM_ptr->num6;    350
                 top[i].middle[j].bottom[k].num7 = BOTTOM_ptr->num7;

                 k++;
                 l++;
              }
           j++;
        }
        i++;
     }
  }                                                                     360
  OS_END_TXN(t1);
  database_close(db1);

  FREE(top);

  for (i = 0; i < no_of_top; ++i)
```

```
    {
        FREE(top[i].middle);
    }

    for (i = 0; i < no_of_top; ++i)
    {
        for (j = 0; j < no_of_middle; ++j)
        {
            FREE(top[i].middle[j].bottom);
        }
    }
}

very_complex_query(lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8)
{
    int i;

    database *db1;
    start_objectstore ();

    db1 = database_lookup_open("/shr/testdb2", 1, 0664);

    OS_BEGIN_TXN(t1,0,transaction_update)
    {
        database_root * TOP_root;

        struct BOTTOM * BOTTOM;
        struct MIDDLE * MIDDLE;
        struct TOP * TOP;

        struct BOTTOM *BOTTOM_ptr;
        struct MIDDLE *MIDDLE_ptr;
        struct TOP *TOP_ptr;

        TOP_root = database_root_find ("TOP_entry",db1);
        TOP_ptr = (struct TOP*) TOP_root;

        TOP = (struct TOP *)database_root_get_value(TOP_root,0);
```

```
i = 0;
l = 0;

MALLOCN(top, Top, no_of_top, "Malloc 1");
                                                                    410
for (TOP_ptr = TOP; TOP_ptr < TOP+no_of_top; TOP_ptr++)
{
   MIDDLE = TOP_ptr->middle;

   j = 0;

   MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");

   for (MIDDLE_ptr = MIDDLE; MIDDLE_ptr <
        MIDDLE+no_of_middle; MIDDLE_ptr++)                          420
   {
      BOTTOM = MIDDLE_ptr->bottom;

      k = 0;

      MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");

      for (BOTTOM_ptr = BOTTOM; BOTTOM_ptr <
           BOTTOM+no_of_bottom; BOTTOM_ptr++)
         if (BOTTOM_ptr->num1 > lim1 &&                             430
             BOTTOM_ptr->num2 > lim2 &&
             BOTTOM_ptr->num3 > lim3 &&
             BOTTOM_ptr->num4 > lim4 &&
             BOTTOM_ptr->num5 > lim5 &&
             BOTTOM_ptr->num6 > lim6 &&
             BOTTOM_ptr->num7 > lim7 &&
             BOTTOM_ptr->num8 > lim8)
         {
            top[i].middle[j].bottom[k].num1 = BOTTOM_ptr->num1;
            top[i].middle[j].bottom[k].num2 = BOTTOM_ptr->num2;     440
            top[i].middle[j].bottom[k].num3 = BOTTOM_ptr->num3;
            top[i].middle[j].bottom[k].num4 = BOTTOM_ptr->num4;
```

```
                    top[i].middle[j].bottom[k].num5 = BOTTOM_ptr->num5;
                    top[i].middle[j].bottom[k].num6 = BOTTOM_ptr->num6;
                    top[i].middle[j].bottom[k].num7 = BOTTOM_ptr->num7;

                    k++;
                    l++;
                }
            j++;                                                    450
        }
        i++;
    }
}
OS_END_TXN(t1);
database_close(db1);

FREE(top);

for (i = 0; i < no_of_top; ++i)                                     460
{
    FREE(top[i].middle);
}

for (i = 0; i < no_of_top; ++i)
{
    for (j = 0; j < no_of_middle; ++j)
    {
        FREE(top[i].middle[j].bottom);
    }                                                               470
}
}

very_complex_query2(lim1,lim2,lim3,lim4,lim5,lim6,lim7,lim8)
{
    int i;

    database *db1;
    start_objectstore ();

                                                                    480
```

```
db1 = database_lookup_open("/shr/testdb2", 1, 0664);

OS_BEGIN_TXN(t1,0,transaction_update)
{
   database_root * TOP_root;

   struct BOTTOM * BOTTOM;
   struct MIDDLE * MIDDLE;
   struct TOP * TOP;

   struct BOTTOM *BOTTOM_ptr;
   struct MIDDLE *MIDDLE_ptr;
   struct TOP *TOP_ptr;

   TOP_root = database_root_find ("TOP_entry",db1);
   TOP_ptr = (struct TOP*) TOP_root;

   TOP = (struct TOP *)database_root_get_value(TOP_root,0);

   i = 0;
   l = 0;

   MALLOCN(top, Top, no_of_top, "Malloc 1");

   for (TOP_ptr = TOP; TOP_ptr < TOP+no_of_top; TOP_ptr++)
   {
      MIDDLE = TOP_ptr->middle;

      j = 0;

      MALLOCN(top[i].middle, Middle, no_of_middle, "Malloc 2");

      for (MIDDLE_ptr = MIDDLE; MIDDLE_ptr <
            MIDDLE+no_of_middle; MIDDLE_ptr++)
      {
         BOTTOM = MIDDLE_ptr->bottom;

         k = 0;
```

```
        MALLOCN(top[i].middle[j].bottom, Bottom, no_of_bottom, "Malloc 3");          520

        for (BOTTOM_ptr = BOTTOM; BOTTOM_ptr <
              BOTTOM+no_of_bottom; BOTTOM_ptr++)
           if (BOTTOM_ptr->num1 > lim1 && BOTTOM_ptr->num1 < lim2 &&
                BOTTOM_ptr->num2 > lim3 && BOTTOM_ptr->num2 < lim4 &&
                BOTTOM_ptr->num3 > lim5 && BOTTOM_ptr->num3 < lim6 &&
                BOTTOM_ptr->num4 > lim7 && BOTTOM_ptr->num4 < lim8)
           {
              top[i].middle[j].bottom[k].num1 = BOTTOM_ptr->num1;
              top[i].middle[j].bottom[k].num2 = BOTTOM_ptr->num2;                   530
              top[i].middle[j].bottom[k].num3 = BOTTOM_ptr->num3;
              top[i].middle[j].bottom[k].num4 = BOTTOM_ptr->num4;
              top[i].middle[j].bottom[k].num5 = BOTTOM_ptr->num5;
              top[i].middle[j].bottom[k].num6 = BOTTOM_ptr->num6;
              top[i].middle[j].bottom[k].num7 = BOTTOM_ptr->num7;

              k++;
              l++;
           }
        j++;                                                                        540
     }
     i++;
  }
}
OS_END_TXN(t1);
database_close(db1);

FREE(top);

for (i = 0; i < no_of_top; ++i)                                                     550
{
   FREE(top[i].middle);
}

for (i = 0; i < no_of_top; ++i)
{
```

```
        for (j = 0; j < no_of_middle; ++j)
        {
            FREE(top[i].middle[j].bottom);
        }                                                    560
    }
}
```

Appendix G

Equipment Used

All work in this study, including database construction, program development, and the preparation of this document, was done on a SUN SPARCstation 10/30GX. In addition, all database management was performed with University INGRES, version 8.9 (6/12/88), and ObjectStore,[1] Release 1.2.4.

[1]For more information, contact Object Design, Inc., One New England Executive Park, Burlington, MA 01803.

Bibliography

[Bevington 1969] Bevington, Philip R., *Data Reduction and Error Analysis for the Physical Sciences,* McGraw-Hill, New York, 1969.

[Buslenko et al. 1966] Buslenko, N. P., D. I. Golenko, I. M. Sobol', V. G. Sragovich, and Yu. A. Shreider, *The Monte Carlo Method,* Pergamon Press, New York, 1966.

[Cattell 1991a] Cattell, R. G. G., ed., "Next Generation Database Systems," *Communications of The ACM* 34, Number 10 (October 1991), 30–120.

[Cattell 1991b] Cattell, R. G. G., *Object Data Management: Object-oriented and extended relational database systems,* Addison-Wesley, Reading, Massachusetts, 1991.

[Date 1987] Date, C. J., *A Guide to INGRES,* Addison-Wesley, Reading, Massachusetts, 1991.

[Epstein 1977] Epstein, Robert, *A Tutorial on INGRES,* revised, Memorandum No. ERL-M77–25, Electronics Research Laboratory, Colege of Engineering, University of California, Berkeley, Berkeley, 15 December 1977.

[Gastineau 1988] Gastineau, Gary L., *The Options Manual,* McGraw-Hill, New York, 1988.

[Giguère 1958] Giguère, Guynemer, "Warrants: A Mathematical Method of Evaluation," *The Analysts Journal* 14 (November 1958), 17–25.

[Gray 1991] Gray, Jim, ed., *The Benchmark Handbook for Database and Transaction Processing Systems,* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[Hammersley–Handscomb 1964] Hammersley, J. M., and D. C. Handscomb, *Monte Carlo Methods,* John Wiley & Sons, New York, 1964.

[Harvey 1981] Harvey, Andrew C., *The Econometric Analysis of Time Series,* John Wiley and Sons, New York, 1981.

[Hill 1978] Hill, Edward Jr., *A Comparative Study of Very Large Data Bases,* Springer-Verlag, Berlin, 1978.

[Kalash et al. 1992] Kalash, Joe, Lisa Rodgin, Zelaine Fong, and Jeff Anton, *INGRES Version 8 Reference Manual,* unpublished manuscript, 13 February 1992.

[Kalos 1986] Kalos, Malvin H., *Monte Carlo Methods,* John Wiley & Sons, New York, 1986.

[Kassouf 1965] Kassouf, Sheen T., *A Theory and an Econometric Model for Common Stock Purchase Warrants,* Analytical Publishers Co., New York, 1965.

[Kassouf 1969] Kassouf, Sheen T., *Evaluation of Convertible Securities, rev. ed.,* Analytical Publishers Co., New York, 1969.

[Lamb et al. 1991] Lamb, Charles, Gordon Landis, Jack Orenstein, and Dan Weintraub, "The ObjectStore Database System," in [Cattell 1991a], 50–63.

[Marquardt 1963] Marquardt, Donald W., "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *Journal of the Society of Industrial Applied Mathematics* 11, Number 2 (June 1963), 431–441.

[OstoreAdmin 1991] *ObjectStore Administration and Development Tools,* release 1.1, Object Design, Inc., Burlington, Massachusetts, March 1991.

[OstoreGuide 1991] *ObjectStore User Guide,,* release 1.1, Object Design, Inc., Burlington, Massachusetts, March 1991.

[OstoreRef 1991] *ObjectStore Reference Manual,,* release 1.1, Object Design, Inc., Burlington, Massachusetts, March 1991.

[Ozkarahan 1990] Ozkarahan, Esen, *Database Management: Concepts, Design, and Practice,* Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[Press et al. 1988] Press, William H., Brian P. Flannery, Saul A. Tuekolsky, and William T. Vetterling, *Numerical Recipes in C,* Cambridge University Press, Cambridge, 1988.

[Shelton 1967a] Shelton, John P., "The Relation of the Price of a Warrant to the Price of Its Associated Stock, Part I," *Financial Analysts Journal* 23 (May–June 1967), 143–151.

[Shelton 1967b] Shelton, John P., "The Relation of the Price of a Warrant to the Price of Its Associated Stock, Part II," *Financial Analysts Journal* 23 (July–August 1967), 84–99.

[Silberschatz et al. 1991] Silberschatz, Avi, Michael Stonebraker, and Jeff Ullman, editors, "Database Systems: Achievements and Opportunities," in [Cattell 1991a], 110–120.

[Sobol' 1974] Sobol', I. M., *The Monte Carlo Method,* University of Chicago Press, Chicago, 1974.

[Stonebraker 1986] Stonebraker, Michael, ed., *The INGRES Papers,* Addison–Wesley Publishing Company, Reading, Massachusetts, 1986.

[Ullman 1988] Ullman, Jeffrey D., *Principles of Database and Knowledge-base Systems,* Computer Science Press, Rockville, Maryland, 1988.