Animal Breeder's Tool Kit
User's Guide and Reference Manual

B. L. Golden, W. M. Snelling and C. H. Mallinckrodt[1]

[1] Assistant professor, research associate and graduate assistant, Department of Animal Sciences.

The authors wish to thank Dr. Richard Bourdon, Associate Professor, Department of Animal Sciences, Dr. James Brinks, Professor, Department of Animal Sciences, and Dr. Thomas Field, Assistant Professor, Department of Animal Sciences for their comments and assistance in the preparation of this bulletin.

These manuals describe release 1 of the abtk software.

Animal Breeder's Tool Kit

User's Guide

User's Guide Table of Contents

1 User Environment, Terminology and Definitions

The **Animal Breeder's Tool Kit** (*abtk*) is a software package that assists the technically trained animal breeder in assembling and solving equation systems without needing to know a third generation programming language such as FORTRAN or C. The *abtk* was originally developed around the linear systems common in animal breeding problems. However, it was intended to be flexible enough that it should accommodate a wide variety of equation systems, linear or otherwise. Obviously, the field of mathematics is such that not all routines to construct and solve every possible equation system are incorporated in this package. But the current release of *abtk* includes a substantial number of **tools** that will assist most all users working in the field of livestock genetics and form a foundation that can be built upon. It is hoped and intended that if routines a particular user finds helpful to fulfill his particular research needs are missing, he will develop them and share them with the rest of the animal breeding community by making them available on the anonymous ftp site at Colorado State University. The procedure for doing this is discussed later in this chapter.

We are certain that to try and accomplish for mixed models what, say, SAS (1990) PROC GLM has done for fixed models would be impossible. It may be possible to prepare some subset of useful models, but even specifying the structure and solution characteristics of the most basic mixed models is much more complex then fixed effects analysis. Thus, we have chosen the "toolkit" approach.

To some more experienced users part of the *abtk* will look and behave like the older symbolic matrix interpreters such as SMIS or SAS's PROC MATRIX. Because we are targeting this package at trained animal breeders it was important to be able to take the information from literature and easily transfer it to application. Therefore, many parts of the system look like symbolic matrix interpretation. However, as explained in detail in the Technical Reference, the underlying algorithms and data storage modes are very dissimilar from traditional matrix packages. Traditional approaches to software for matrix interpretation have been limited in size of equation systems due to algorithm performance and data storage modes. For the most part the algorithms used in the *abtk* avoid these limitations.

The *abtk* was designed to take advantage of what appear to be the most promising trends in performance computing. The advent of small (in physical size), high performance architectures with large memory and high speed I/O devices has directed the development of the methods used in the *abtk*. Because of the success in the development of technology such as RISC computing, the algorithms are based on high speed scaler processing with I/O that uses large band widths. The methods used in the *abtk* were not explicitly coded for a particular hardware configuration and have been used in environments ranging from UNIX running on an Intel 80386 processor at 25 Mh to a Cray Y/MP 864. All algorithms take advantage of dynamic memory allocation and, thus problem size is limited only by the available memory and mass storage capacity of the system.

About This Document

The documentation for the *abtk* consists of three manuals. The first manual is a "Users Guide." This guide is not meant to be a training manual in best linear unbiased prediction, but it does demonstrate how to build several mixed model linear systems using the *abtk*. It is hoped users will find this document a useful reference manual for not only the software but also common methods and notations used by animal breeders. Many users should find the appendix particularly helpful.

The discussion of the *abtk* in the guide is strictly limited to its use in linear models. Once a substantial number of routines have been developed for handling other types of linear or non-linear systems a supplementary guide will be developed. However, even though only fixed and mixed model systems common in animal breeding are discussed, *abtk* is not limited to these uses.

The second manual is the "Reference Manual." The reference manual is an alphabetized list of the commands and their descriptions. Users are strongly encouraged to keep a copy of the reference manual available when working with the *abtk*. The printed paper copy of the Reference Manual and the Users Guide are bound together.

A good UNIX reference is also necessary. The *abtk* is closely bound to the UNIX operating system and many of the data manipulations that users make are performed using the UNIX user utilities (awk, join, sort, etc.). For a discussion of the UNIX user utilities any good UNIX reference and guide will work. Many other public domain or freeware utilities exist for UNIX. Users may want to supplement the user utilities with a package such as **Perl**, an advanced data manipulation package that combines many of the features of **awk, sed join** and other UNIX utilities. **Perl** can be obtained from the usenet archives on the Internet. The discussion in this guide will be restricted to utilities that are available on all UNIX systems.

The third manual, or "Technical Reference", is a discussion of the algorithms and storage modes used by the abtk. This manual should be very useful for the application developer who needs examples and ideas for expanding the *abtk*.

The User Environment

The *abtk* is closely bound to the UNIX user utilities. Early in the development of the *abtk* we realized that any animal breeder doing even the most basic statistical analysis would benefit from knowing the UNIX user utilities. Moreover, relying on the UNIX user utilities avoids the development of redundant routines and algorithms.

The *abtk* is a series of commands that process data into symbolic representations of matrices and matrix components and solve the equation systems assembled. The commands are all given from the UNIX prompt (any shell). All commands generally follow the proposed ANSI specification for parameters and arguments, with one exception for a few simple commands. This exception involves the order dependence of arguments. For example, to multiply two matrices together and save them as a third matrix the syntax would be,

```
mult A B C
```

indicating that matrix A is to be multiplied by matrix B and store in matrix C (where A premultiplies B). The proposed standard for command arguments specifies arguments should be order independent, but this did not make sense for certain commands that involved matrix operations. Most other commands use UNIX parameter flags indicating what the argument is, and thus the arguments are order independent. For example, to iteratively solve a linear system, the following command could be used,

```
itgen -R beta.old -n coef -r rhs -m 1000 -b beta.new
```

This command would result in the iterative solution (Gauss-Siedel) using the vector **beta.old** for starting values, **coef** as the coefficient matrix, **rhs** as the right hand side vector, 1000 as the maximum number of rounds to iterate and place the solutions into **beta.new**. The arguments could be given in any order (just like a UNIX command).

In addition to using a UNIX like command syntax, the commands behave in a UNIX like fashion. If a command is issued without arguments it returns a syntax message. This is particularly useful if the exact syntax of the arguments is forgotten. For example, if you have forgotten the syntax of the generalized incidence matrix construction routine, entering,

```
gen_z
```

will result in the following syntax explaination,

```
gen_z: Syntax: gen_z -d data_vector -e effect_list -r rhs_vector -o output_matrix
```

The *abtk* relies on the UNIX system to supply components that are critical to its operation. Several of the UNIX user utilities provide important data handling capabilities that must be performed in the setup of a problem for solution. The most used commands are the **awk** pattern scanning and recognition language, the **join** relational data operator, the **sort** data sorter, and occasionally the **paste** horizontal flat file concatinator.

The most often used user utility is **awk**. Awk is a very easy to use language that gives you powerful data manipulation capability without needing to know a third generation programming language. For a complete and easily understood discussion of the awk language see "AWK Programming Language" (Aho, 1988) or any basic UNIX guide.

It is also useful for the advanced user to learn shell programming. The codes that make up the *abtk* are a mixture of C language, awk and bourne shell scripts. The *abtk* is intended to work like a macro language that is processed by the command interpreter. Users should not hesitate to program analyses they will use multiple times into a script program that they can batch process. The developers have had success taking this approach when performing even REML type analyses. Example scripts are included with the release version in the **examples** subdirectory.

Definitions and Terminology

The discussion in this manual is a blend of terms commonly used among UNIX users and among animal breeders. Since this manual cannot instruct in every aspect of computing or animal breeding, a certain level of understanding will be assumed. This documentation and the level of sophistication of the *abtk* is intended for people who have received or are currently enrolled in at least one graduate level course in mixed models methodology.

A complete discussion of how the *abtk* forms and stores its matrices appears in section 3 of the Users Guide. It is important that the user understand how the UNIX file system works and particularly how it works with the *abtk*. The *abtk* uses an object oriented approach. Each matrix and data vector must be thought of as an individual object. It is helpful if the beginning user tries to forget his understanding of data files, file systems and data analysis developed from working in environments such as SAS, MS-DOS or LSML76. In the *abtk* each data vector from a traditional flat text file will be stored in its own UNIX "file". Each matrix formed from data vectors or other matrices will also be stored in a UNIX "file".

The user should take a structured approach to solving his particular problem. For example, the first step when assembling a particular problem is to separate a traditional flat data file into separate data vectors for each column in the flat file. By storing each data vector of the flat file in a separate UNIX file, each vector can be worked with as a unique object. This is analogous to a typical fourth generation data base manager where each category is given a unique name, and the UNIX subdirectory where the flat file is stored can be thought of as the data table. Users can easily accomplish these types of manipulations using the UNIX user utilities. This guide contains many examples of these tasks.

Adding New Routines

The *abtk* can be useful for testing and developing new models or new algorithms. It is intended that as new research is performed, the *abtk* can provide a starting point, or at least an environment that will allow the researcher to develop new methods without having to generate computer code that performs the setup work. For example, if a researcher were working in solution methods for non-linear systems, *abtk* could provide many of the assembly routines for the equation systems. The researcher would then be free to concentrate on the solution algorithm.

Researchers who would like to use the *abtk* for this purpose are encouraged to study the *abtk* Technical Reference for a discussion of algorithms and storage modes. Pay particular attention to the way in which storage modes have increased not only problem size but also performance of the operation. For an example, see the discussion of how the matrix multiply is performed with the SPARSE storage mode.

Obtaining a Copy of *abtk*

Users can obtain a magnetic copy free of charge from anonymous FTP at anonymous@cgel.agsci.colostate.edu. The *abtk* is distributed in a compressed tar file so the user will need both an uncompress utility and a good tar (Unix tape archive) utility. Both are made available through the Internet at several anonymous FTP sites offering the GNU software collection.

The distribution file contains magnetic copies of the manuals in post script format. Users can also obtain copies of the manuals in WordPerfect format from the anonymous server.

Paper copies of the documentation can be obtained by contacting the Department of Animal Sciences, Colorado State University, Ft Collins, CO 80523. There will be a charge for the paper copies of the documentation to cover printing costs.

The *abtk* will also be distributed for a minimal $20 processing fee on reel-to-reel tape (6250 or 1600 bpi) in a compressed tar format.

Users with special needs are encouraged to contact the author to make arrangements.

2 Preparing Data for Analysis

Data preparation for an *abtk* analysis is performed primarily using the UNIX user utilities. It is not recommended that a user attempt to supplement the data preparation tasks with a fourth generation relational database manager such as SAS or Oracle. The experiences of our shop indicate that it is more efficient to manage data using the UNIX user utilities.

The objective is to prepare a data set to work the way animal breeders are trained to think and to work the way information about models is presented in the literature. Therefore, the data must be organized according to the object oriented model. For all traditional statistical analysis packages the data are managed as a flat file. In some of the more sophisticated commercial statistical software packages the data are managed using an object oriented approach, but are managed as binary data held in a single file that is not generally available to the researcher except through the data management package. The *abtk* uses the data stored as text files (ASCII) and any component of the files are generally available to the user through the UNIX user utilities.

The Unix User Utilities

Typically, data analyzed in animal breeding studies are organized in one or a series of two dimensional data files. The data structure in these data files is usually conceptualized as having categories represented as columns and items represented with different rows. For example, take the following data file,

A	**1**	**5**	**500**
B	**2**	**3**	**450**
C	**1**	**2**	**475**
D	**1**	**5**	**550**
E	**2**	**4**	**425**
F	**2**	**5**	**490**

Say this file represents six beef cattle weaning weight observations . The animal identification is held in the first field (column). The second field is the sex code, 1=bull and 2=heifer. The third field represents the age in years of the dam, and the last field represents the observed weaning weight of each animal in a row (item).

Traditionally, these data would be held in a single file as input to the analytical software that would perform, say, a fixed model analysis of weaning weight. The *abtk* requires that each field be held in a separate UNIX file. To the first time user this approach may seem a little awkward, but, with some experience users will realize that this is not different from traditional approaches except that now there is much more precise control of the contents of these files. They can be conceptualized as the objects represented by notations commonly seen in animal breeding literature.

File names should correspond to the field contained in the file (id, sex, aod etc.). The name of the directory containing a series of files should reflect the contents of the directory. Working from our example data set above, say the file was copied into the

HOME directory of a UNIX account with the filename of "wean.dat". The next logical step to prepare these data for analysis would be to separate its categories into separate files in a subdirectory. The user would need to create the subdirectory,

```
mkdir data
```

and then begin "awk'ing" the file into components. To create a file in the data subdirectory for the id category the user would issue,

```
awk '{print $1}' wean.dat >data/id
```

Preparing the other three categories would proceed as follows,

```
awk '{print $2}' wean.dat >data/sex
awk '{print $3}' wean.dat >data/aod
awk '{print $4}' wean.dat >data/yw
```

Now the subdirectory called data is logically equivalent to the file called wean.dat but the user has more flexibility in making manipulations on the objects represented by the directory. For example, say the user wanted to view a list of just the id and observation fields. The following command could accomplish this,

```
paste id yw
```

An additional advantage of this strategy is flexibility when drawing information from several two-dimensional data structures or manipulating data with many to one relationships. For example, many problems in animal breeding involve the manipulation of pedigree information and performance measures. A category such as sire identification number may involve a one to many relationship to offspring weaning weight observations. Managing the data using this strategy can make the necessary manipulations easier to accomplish.

Missing Values

Missing values in a data field can be designated with a "." character placed in the appropriate field. If a single field has a missing value it may be necessary (or desirable) to remove the record or to indicate missing values in the field(s) for dependent variables for the same item. For example, if animal D in the above data set had an unknown age of dam, a "." would be placed in the age of dam field instead of the "5". It may be necessary to also place a "." in the field for weaning observation because forming the incidence matrix for the other fixed effect of sex occurs independently and would consider the observation as valid.

This is done to give the user maximum flexibility when constructing a linear system. There maybe situations when the user is confident the missing values in one fixed effect are randomly and uniformly distributed across all other levels of other fixed effects. In this situation the user may not want to designate the dependent value as missing since it would add information without adding bias or error.

Additionally, for many random effects such as sire or dam, it is common to have an unknown parent without designating the observation as missing.

3 Storage Modes

Since it was intended in the design of this package to maximize performance while optimizing utility, it was necessary to store and work with data held on mass storage in multiple storage modes. It is helpful (but probably not necessary) to have a limited understanding of the most common storage modes to effectively operate the *abtk*. The storage modes are the key to the feasibility of solving problems of realistic size. The storage modes help to increase the capacity of the software to handle large problems by reducing wasted mass storage space and by allowing the implementation of special algorithms that take advantage of the features of some storage modes.

Users who want to develop additional tools are encouraged to carefully study the discussion of storage modes in the technical reference.

General Description of Storage Modes

There are ten storage modes supported in the *abtk*. Not all ten are implemented in all routines, and thus for a few "non-typical" operations, translation of storage modes may be necessary. For example, the routine **ident**, used to form an identity matrix, builds the identity matrix using only a storage mode called **DIAGONAL.**

Many routines accommodate several storage modes. The storage modes chosen to be used by a given routine were determined by how likely it was that the routine would receive as input a matrix in a given storage mode and what the effects of a given storage mode would be on the computer performance of the operation.

Users are encouraged to contact the developers and request changes in how storage modes are implemented. If a given routine does not support a given storage mode, and the user sees this as a shortcoming, please contact the developers.

Headers, Text and Compression. All storage modes of data use a **header record** except the storage mode called **DEFAULT.** The DEFAULT storage mode is a traditional storage method where a matrix or vector is stored as a complete, two-dimensional representation of the data. For example, when the example data set in section 1 of the Users Guide was separated into its component fields, each field was stored in the DEFAULT storage mode. Another example is the following two dimensional matrix,

```
10  20
20  30
```

If the user issued the UNIX **cat** command to a file that contained this matrix and it appeared as it does above, then the storage mode was DEFAULT.

All other storage modes have a **header record**. The header record is the first line of the file. If a user uses the **cat** command to view the above matrix, and it was in, for example, the **SPARSE** storage mode, it would appear as,

```
SPARSE 2 2 4
1 1 10
```

```
1 2 20
2 1 20
2 2 30
```

The first line of this five line file is the **header record**. The header record always contains four fields. The first field indicates the storage mode, the next two fields indicate the row and column dimensions of the matrix being stored respectively, and the last field indicates the number of records in the file. A record is always terminated by a newline (0x0A) character.

All storage modes use ascii character data. Even though performance would be improved and total storage may be reduced by offering storage modes that contain binary data, it is important that the user be able to easily view the contents of a file and manipulate it with the UNIX user utilities.

Compressed storage. Most of the tools accept input files (matrices) stored in compressed modes. The compression is performed using the **compress** utility installed on the users system. The user must be sure that his PATH variable is set to point to the location of the compress utilities and that this variable is globally available (see the export command if using the Bourne or Korn shells).

If you do not have a compress utility that makes ".Z" files you can either reconfigure the compile to use the **pack** utility (see the technical reference) or build the GNU **compress** utility on your system. Installing the GNU compress routines is preferable because they are very efficient.

To use a matrix file that has been compressed pre-pend the matrix file name argument with the compress flag, "-z". For example, to multiply the compressed matrix A by the uncompressed matrix B and store in C,

```
mult -zA B C
```

No routines (except for the inverse numerator relationship builder, **ainv**) currently allow for the direct writing of compressed files on output. However, the only routines that really need it are the concatination routines, **shcat** and **svcat**. These routines, and a few others, have variations in the syntax that indicate the output is to come to standard output. Users who know how to redirect **stdout** will realize they can pipe the output from these commands to **compress** and accomplish the compression. See examples in this guide and the sample scripts.

4 Basic Commands for Assembling Linear Systems

The *abtk* can be used for much more than the solution of linear systems of equations. However, statistical models and their solution make a good vehicle for learning how to use the *abtk*. This guide assumes that users are familiar with the algebra of linear models and will be able to learn the *abtk* by comparing that understanding with the syntax of commands.

An Example Model

It is important to keep a structured approach to the construction and solution of a linear system using *abtk*. The first thing the researcher should do is write out the linear model and the normal equations (using matrix notation).

To begin the discussion, a basic linear model will be used as an example. This model is commonly termed a "fixed effects" model, where the only random effect is the error term. The model is typically written,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{e}$$

where **y** is a vector of observations, $\boldsymbol{\beta}$ is a vector of fixed effects on observations in **y**, **X** is a matrix relating observations in **y** to effects in $\boldsymbol{\beta}$**,** and **e** is a vector of random errors in observations in **y**. Assume var(**e**) = $\mathbf{I}\sigma^2$.

The normal equations to be solved are,

$$\mathbf{X'X}\boldsymbol{\beta} = \mathbf{X'y}$$

This example will be applied to the data set from section 2 of the Users Guide with one missing value,

A	**1**	**5**	**500**
B	**2**	**3**	**450**
C	**1**	**2**	**475**
D	**1**	**5**	**.**
E	**2**	**4**	**425**
F	**2**	**5**	**490**

The first step is to construct the vectors of the factors in **X** and the dependent variable vector **y** using the UNIX **awk** command. It is wise to do this in a subdirectory the user creates to hold the data. If the data file of the above data were named **wean.dat**, the sequence of commands might look like,

```
mkdir data
awk '{print $2}' wean.dat >data/sex
awk '{print $3}' wean.dat >data/aod
awk '{print $4}' wean.dat >data/yw
```

All three data vectors are now in the **DEFAULT** storage mode (see section 3 of the Users Guide).

The next step is to create the **X** matrix. Since the **X** matrix will not be of "full row rank" the user must decide what set of constraints will be imposed to solve the linear system. For this example, an equation will be constructed for the mean and an equation will be constructed for each level of both fixed factors. The normal equations will be constructed and then equations for the last level of each factor will be eliminated to obtain a solution. This is the type of constraint that is imposed by PROC GLM in SAS (1990). Other constraints are possible.

Equation for the Mean

As with most things performed on a computer, building an equation for the mean can be accomplished in several ways. For most applications the best way will be to use the **mu** tool. **mu** creates a vector of 0's or 1's that have a one to one correspondence with the elements in the dependent variable vector. **mu** sends it's output to standard out and so the output must be piped into a file. To build the incidence matrix for the mean from our example,

```
mu data/yw >x_mu
```

If the matrix file **x_mu** were listed using the **cat** command, it would look like,

```
1
1
1
0
1
1
```

(Notice x_mu is in DEFAULT storage mode).

Incidence of the Fixed Effects

The *abtk* contains three tools for building the components of **X** for fixed factors. If the factor is a covariate, the tool **cov** should be used to construct the equation. This routine will look at the values of the covariate and the dependent variable to construct a vector of covariate values that are corrected for missing values in the dependent variable.

To construct components of **X** for factors that are "class" variables such as in the example, the easiest tool is **fef**. A more generalized incidence matrix builder called **gen_z** is available. **Fef** is preferable for this example because it will construct a list of unique levels of the effect. This list of unique levels of the effect corresponds to the columns in **X** that represent the fixed factor. This list must be constructed and supplied to **gen_z**.

To construct the component of **X** that represents sex in the example,

```
fef -d data/sex -r data/yw -e sex.list >x_sex
```

The -d parameter indicates the next argument is the data vector containing the effect values with a row wise correspondence to the observations in the vector of values of the dependent variable (data/yw). The -r parameter indicates the next argument is the vector of the dependent variable. The -e parameter indicates the next argument is the output file name of the list of levels of the fixed effect. The output matrix of interest is sent to standard output and therefore, should be piped into a file.

In this example the file **sex.list** will contain (when **cat** is used to list its contents),

```
1
2
```

indicating two levels of the sex factor were identified in the data and the incedence matrix **x_sex** will have two columns, the first one corresponding to individuals which have a sex level of 1 and the second corresponding to a sex level of 2. **Fef** will work with character data in the data vector, e. g. "M" and "F" could have been in the data vector, **data/sex**, instead of "1" and "2".

The output matrix file **x_sex** will look like,

```
SPARSE 6 2 5
1 1 1
2 2 1
3 1 1
5 2 1
6 2 1
```

(NOTE: This matrix can be viewed in **DEFAULT** storage mode by typing **mprint x_sex.**)

Likewise, the incidence matrix for the age of dam factor can be constructed using,

```
fef -d data/aod -r data/yw -e aod.list >x_aod
```

The file **aod.list** would look like,

```
5
3
2
4
```

Notice that it is not in any special sort order. **fef** constructs the list of levels in the order it receives them from the data file. If the user wants a special order such as a sort order, then **gen_z** can be used instead of **fef**.

The output matrix file **x_aod** would look like,

```
SPARSE 6 4 5
1 1 1
2 2 1
3 3 1
5 4 1
6 1 1
```

Concatenating the Blocks

The user has two options for proceeding with the construction of the coefficient matrix. The incidence matrices can be concatinated and then the transpose of X can be multiplied by X or each submatrix of the coefficient matrix can be built and then concatenated. That is, the user could concatinate the incidence matrices for the sex and age of dam factors together then build the coefficient matrix or the user could build the sex equations then the age of dam equations and concatinate them together. The first method will be illustrated since it is probably preferable for this simple example.

Two tools are available for horizontally concatinating, **hcat** and **shcat**. For example, to concatinate the **x_mu** and **x_sex** matrices together,

hcat x_mu x_sex x_mu_sex

This will create a third matrix, **x_mu_sex**, a six by three in **SPARSE** storage mode that is the combination of the first two matrices.

The other routine that will perform a horizontal concatination is **shcat**. **Shcat** differs from **hcat** in that it accepts input matrices stored only in **SPARSE** mode. However, it is preferable for concatination when the input matrices are all in **SPARSE** mode because it will process several input matrices in one command. It also requires much less electronic memory since it will not hold entire input matrices in electronic memory.

In our example **x_mu** was in DEFAULT storage mode **hcat** had to be used for the first concatination. The user can now create the final incidence matrix,

shcat x_mu_sex x_aod X

The last argument to **shcat** is the output matrix.

Creating the Normal Equations.

The linear system to be solved is,

$$\mathbf{X'X\beta = X'y}$$

The **mult** tool can be used to create **X'X** without requiring a separate tool to build the **X'**,

mult -tX X xtx

The "-t" parameter flag indicates transposition of **X** before performing the multiplication. The right hand side of the linear system is also constructed with the **mult** tool,

```
mult -tX yw xtyw
```

Now that the normal equations have been constructed, all that is left to do is to impose constraints on the equations and solve the linear system. Constraint setting and solution methods for linear systems are discussed in section eight of the Users Guide.

5 Assembling Single Trait Animal Models

Several tools are available for building equation systems with both fixed and random factors with differing covariance structures between the levels of the random factors. This section demonstrates the basic methods and tools for assembling a particular type of random factor linear system that is commonly called the "single trait animal model". These techniques will be extrapolated to more complex random factor analyses in the succeeding sections of this guide.

For the discussion in this section additional information must be added to the example data file used in the previous sections. A field for sire and dam of the animal who is represented by each item is included,

A	**.**	**.**	**1**	**5**	**500**
B	**.**	**.**	**2**	**3**	**450**
C	**A**	**B**	**1**	**2**	**475**
D	**.**	**B**	**1**	**5**	**.**
E	**C**	**.**	**2**	**4**	**425**
F	**A**	**E**	**2**	**5**	**490**

The linear model now becomes,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u} + \mathbf{e}$$

where **u** is a vector of random effects on **y** and **Z** relates those effects to observations in **y**. The values in **u** are typically called breeding values and represent the effects due to additive gene action (breeding values). The variance structure of **u** is,

$$\text{var}[\mathbf{u}] = \mathbf{A}\sigma^2_a$$

where **A** is the matrix of Wright's numerator relationships and σ^2_a is the variance of the additive genetic effects.

The normal equations are,

$$\begin{bmatrix} X'X & X'Z \\ Z'X & Z'Z + A^{-1}\sigma^2/\sigma_a^2 \end{bmatrix} \begin{bmatrix} \beta \\ u \end{bmatrix} = \begin{bmatrix} X'y \\ Z'y \end{bmatrix}$$

Constructing the Inverse Numerator Relationship Matrix

The construction of Wright's inverse numerator relationship matrix is accomplished using the **ainv** tool. The input to **ainv** is a pedigree file containing three fields. The records in the pedigree file must be in the same order as the levels of the random factor. In the example above the pedigree file will contain,

A	**.**	**.**
B	**.**	**.**
C	**A**	**B**
D	**.**	**B**
E	**C**	**.**
F	**A**	**E**

The three fields are the identification code of the individual and the individual's parents (the order of sire or dam field does not effect the result). The pedigree file must be ordered so that records for parents occur before records for offspring.

The values for individual identification code, sire identification and dam identification must be character sortable so that a sire or dam identification will always occur before an offspring's. This is because the routines that compute the inverse numerator relationship matrix need to perform several sorts to process the pedigree. For example, if birth years are available it is a good technique to prepend the identification code with the birth year.

If birth year data are not available for all animals or if the birth year information is not reliable, another way to insure sortability is to perform the initial ordering using the **stack_ped** tool and then recode the pedigree identification values using the **ped_recode** tool. **stack_ped** will construct a pedigree file ordered ascending by age for any input pedigree file regardless of its data types and input sort order. **ped_recode** will recode the animal, sire and dam identifications in an ascending order pedigree file beginning with the oldest animal as "1".

No further preparation of the example data is required since the pedigree has the correct properties for the **ainv** tool. The pedigree file called **ped** can be constructed by issuing,

```
awk '{print $1, $2, $3}' wean.dat >data/ped
```

The inverse numerator relationship matrix can be constructed with,

```
ainv ped
```

The **ainv** tool is included in but was not originally constructed for the *abtk*. The **ainv** package can be obtained separately from anomymous@cgel.agsci.colostate.edu. A companion article about the algorithm used in **ainv** and other features of the package is published in the Journal of Animal Sciences (69:3564-3573).

The output from this tool is substantially more verbose than for any other tool. Two output files are produced; a list of inbreeding coefficients in a file called **inbred.lst** and the inverse numerator relationship matrix in **SPARSE** storage mode. The inverse numerator relationship matrix will be stored in a file called **v** in a subdirectory called **a_inverse** which is created from the current directory when **ainv** was executed.

Assembling the Normal Equations

There are two tools that can be used to assemble the **Z** matrix for a model with animal effects. The generalized incidence matrix builder tool, **gen_z**, can be used but a special tool has been programmed for animal model incedence matrix construction, **anim_z**. The **anim_z** tool is preferable because it is substantially faster and does not consume as much electronic memory to process.

To construct the random effect incidence matrix for the example,

```
anim_z data/yw >z_ww
```

The resulting matrix, **z_ww** is in **DIAGONAL** storage mode. If the user were to **cat** the file it would look like,

```
DIAGONAL 6 6 6
1
1
1
0
1
1
```

This matrix can then be premultiplied by its transpose,

```
mult -tz_ww z_ww zwtzw
```

The multiplication of the variance components ratio by $\mathbf{A}^{-1}$ can be performed at the same time as the addition to **zwtzw** using the **sadd** tool. The **sadd** tool is a high performance algorithm for addition of matrices. For the example, say the variance components ratio is equal to 2.0. The resulting matrix, **zw.A**, will be in **SPARSE** storage mode,

```
sadd zwtzw a_inverse/v*2.0 zw.A
```

and will have 22 non-zero values.

For the example given here, users will note that **z_ww** and **zwtzw** are identical, except for the different storage modes (**Z** is idempotent), and thus we could have avoided the multiply step. As always, there is more than one way to skin a cat or assemble a linear system. The method presented is intended to illustrate a generalized approach. Users are encouraged to determine more efficient methods of assembling the particular linear systems of interest when the savings in CPU cycles is important. However, novice users may avoid problems by using standard methods and limiting the short cuts taken.

The off diagonal blocks of the coefficient matrix can be assembled,

```
mult -tX z_ww xtzw
```

The transpose of **xtzw** can be taken using the **trans** tool. However, because of the high speed performance of the mult algorithm, **mult** is more efficient for problems with typical sizes,

```
mult -tz_ww X zwtx
```

The **mult** tool will produce matrices that are in **SPARSE** storage mode and the resulting matrices can be concatinated using **shcat** and **svcat**,

```
shcat xtx xtzw x.EQ
shcat zwtx z.A z.EQ
```

and

```
svcat x.EQ z.EQ coef
```

The right hand side can also be assembled in a similar manner,

```
mult -tz_ww yw zwtyw
svcat xtyw zwtyw RHS
```

The completed coefficient matrix is **coef** and the right hand side vector is **RHS**. The solution of this linear system will be discussed in section G-8.

Maternal Effects Model

Adding maternal effects to the example is a relatively trivial task using the *abtk*. The model now becomes,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Zu} + \mathbf{Z_m u_m} + \mathbf{Z_p u_p} + \mathbf{e}$$

where $\mathbf{u_m}$ is a vector of random effects on observations in **y** due to additive genetic influences on the dam of the individual who was observed, $\mathbf{Z_m}$ is an incidence matrix relating observations in **y** to the dams additive effect and $\mathbf{Z_p}$ is an incidence matrix relating observations in **y** to nonadditive and permanent environmental effects due to the dam, $\mathbf{u_p}$. The variance structure of random effects is now,

$$var\begin{bmatrix} u \\ u_m \\ u_p \\ e \end{bmatrix} = \begin{bmatrix} G_a & G_{am} & 0 & 0 \\ G_{am} & G_m & 0 & 0 \\ 0 & 0 & V_p & 0 \\ 0 & 0 & 0 & R \end{bmatrix} = \begin{bmatrix} A\sigma_a^2 & A\sigma_{am} & 0 & 0 \\ A\sigma_{am} & A\sigma_m^2 & 0 & 0 \\ 0 & 0 & I\sigma_p & 0 \\ 0 & 0 & 0 & I\sigma_e^2 \end{bmatrix}$$

The **gen_z** tool can be used to construct the incidence matrix for maternal effects,

```
awk '{print $3}' wean.dat >data/dam
```

```
gen_z -e data/id -d data/dam -r data/yw -o zm_ww
```

The -e parameter indicates the name of the vector that contains the animal identification corresponding to the random effects of animal. The -d indicates the data vector that corresponds to the dam identification. The -r indicates the name of the observation vector and has a one-to-one correspondence to the vector indicated by -d. The -o parameter indicates the name of the output matrix file.

The **gen_z** tool can also be used to construct $\mathbf{Z_p}$. The columns of $\mathbf{Z_p}$ correspond to only those females who had offspring, so the -e parameter of **gen_z** should point to a data vector containing a list of dams. This data vector can be created with the UNIX **sort** utility,

```
sort -u data/dam | awk '$1!="."{print $1}' >data/pef
```

The -u parameter indicates suppression of all but one line in each set of identical lines. The output of the sort is passed to the **awk** utility so it can sift out the **UNKNOWN** code. The **gen_z** tool then creates $\mathbf{Z_p}$,

```
gen_z -e data/pef -d data/dam -r data/yw -o zp_ww
```

Maternal Effects Normal Equations

One set of normal equations for the maternal effects model is,

$$\begin{bmatrix} X'X & X'Z & X'Z_M & X'Z_P \\ Z'X & Z'Z+\sigma^2 g^1 A^{-1} & Z'Z_m+\sigma^2 g^{12} A^{-1} & Z'Z_p \\ Z_m'X & Z_m'Z+\sigma^2 g^{12} A^{-1} & Z_m'Z_m+\sigma^2 g^2 A^{-1} & Z_m'Z_p \\ Z_p'X & Z_p'Z & Z_p'Z_m & Z_p'Z_p+\sigma^2/\sigma^2_p I \end{bmatrix} = \begin{bmatrix} \beta \\ u \\ u_m \\ u_p \end{bmatrix} = \begin{bmatrix} X'y \\ Z'y \\ Z_m'y \\ Z_p'y \end{bmatrix}$$

where,

$$\begin{bmatrix} \sigma^2_d & \sigma_{dm} \\ \sigma_{dm} & \sigma^2_m \end{bmatrix}^{-1} = \begin{bmatrix} g^1 & g^{12} \\ g^{12} & g^2 \end{bmatrix}$$

Assembling equations using a ***script*** *file.* The components can be assembled using the **mult, sadd, shcat** and **svcat** tools. These commands can be placed into a file using a UNIX editor such as **vi** and executed as a batch file in background. Figure 1 shows an example script file for building the normal equations for the model with maternal effects. If this file were named "mkcoef", it could be run in background by issuing,

sh mkcoef &

the "&" indicates background processing of the script file. The "sh" is the UNIX bourne shell command processor.

Figure 1. Batch file to assemble normal equations for maternal effects model.

```
# Build a single trait MME with maternal effect

# Fixed effect equations...
mult -tX X xtx
mult -tX z_ww xtzw
mult -tX zm_ww xtzm
mult -tX zp_ww xtzp

# Random direct effect equations...
mult -tz_ww X zwtx
mult -tz_ww z_ww zwtzw
sadd zwtzw a_inverse/v*2.0 zw.A
mult -tz_ww zm_ww zwtzm
sadd zwtzm a_inverse/v*4.0 zwzm.A
mult -tz_ww zp_ww zwtzp

# Random maternal effect equations...
mult -tzm_ww X zmtx
mult -tzm_ww zm_ww zmtzm
mult -tzm_ww zp_ww zmtzp
sadd zmtzm a_inverse/v*3.0 zm.A

# Random permanent environment equations...
mult -tzp_ww X zptx
mult -tzp_ww z_ww zptzw
mult -tzp_ww zm_ww zptzm
mult -tzp_ww zp_ww zptzp
ident -mX >Ip
sadd zptzp Ip*5.0 zp.I

# Concatinate it all together...
scat xtx xtzw xtzm xtzp -n zwtx zw.A zwzm.A zwtzp -n \
    zmtx zwzm.A zm.A zmtzp -n zptx zptzw zptzm zptzp - | compress >coef.Z
```

6 Assembling Reduced Animal Models

The normal equations for assembling a reduced animal model can be expressed in two ways. Many researchers represent the incidence matrix for non-parent random effects using a separate incidence matrix, usually **P.** So now **Z** becomes the incidence matrix relating the random additive effects of only parent animals to their observations. However, it is easier to conceptualize the model and construct it with the *abtk* if the **Z** matrix represents both parent and nonparent observations and **P** is not considered. This is the method used by Quaas and Pollak (1980, Journ. Anim. Sci 51:1277) to express the linear system for RAM. It has the additional advantage of using the same notational method of expressing the normal equations as does the model with animal effects. For example, after manipulating the algebra the animal model normal equations can be expressed as,

$$\begin{bmatrix} X'R^{-1}X & X'R^{-1}Z \\ Z'R^{-1}X & Z'R^{-1}Z + A^{-1}\sigma_a^{-2} \end{bmatrix} \begin{bmatrix} \beta \\ u \end{bmatrix} = \begin{bmatrix} X'R^{-1}y \\ Z'R^{-1}y \end{bmatrix}$$

Z, **A**$^{-1}$, **u** and **R**$^{-1}$ are redefined for the reduced animal model.

The **Z** matrix for random additive effects in the example would be,

$$Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ .5 & 0 & 0 & .5 \end{bmatrix}$$

The columns of **Z** correspond to the four animals who became parents, A, B, C and E. This incidence matrix can be constructed using the **ram_z** tool,

```
ram_z -i data/id -s data/sires -d data/dams -e parents -r data/yw -o zr_ww
```

where data/id is the vector of animal identifications from the data, data/sires is the vector of sires from the data and data/dams is the vector of dam identifications from the data. These three vectors can be constructed using **awk**. They have a one-to-one correspondence to the observation vector **data/yw**. The **parents** vector is a listing of the parent identifications that correspond to the columns of the incidence matrix. It is an input vector that can be constructed by passing a list of all parents to the sort utility and sorting it with a -u parameter (sifting out the **UNKNOWN** code, see section 5 of the Users Guide). This vector should have a one to one correspondence with the $\mathbf{A}^{-1}$ matrix.

The $\mathbf{R}^{-1}$ has a different structure then the $\mathbf{R}^{-1}$ used for a full animal model set of normal equations. The residual variance of non-parents includes the sampling variance. A special tool, **form_rr**, will construct $\mathbf{R}^{-1}$ for a reduced animal model. Form_rr can construct an inverse residual matrix for a single trait model or it will construct the three blocks of the inverse required for a two trait model (see section G7). For the single trait example discussed in this section $\mathbf{R}^{-1}$ can be formed by,

```
form_rr -r 50.0 -g 25.0 -i data/id -p parents -1 data/yw Ri
```

where the argument to the -r parameter is the residual variance (50.0), the argument to the -g parameter is the additive genetic variance (25.0). The -i indicates the name of the animal identification vector from the data, -p indicates the name of the vector that contains the parent identifications that corresponds to the columns of the incidence matrix, -1 indicates the name of the observation vector. The last argument is the name of the file where the inverse residual matrix will be stored.

It is important to point out that **form_rr** assumes a relationship coefficient of .5 between offspring and parent when constructing the inverse residual variance matrix. This may influence an individual researchers decision to construct a full animal model instead a reduced animal model.

The $\mathbf{A}^{-1}$ becomes the relationship matrix for only animals who became parents. Therefore, a pedigree file must be constructed for the parents vector. This can be easily accomplished using the UNIX **join** utility. For example, the script in figure 2 will construct a pedigree file for parents (**par_ped**) only from the pedigree file called **ped**.

```
# Form parents only pedigree file.

awk '{printf "%s\n%s\n", $2, $3}' ped | sort -u | awk '$1!= "."{print $1}' >tmp1
sort ped | join -a1 -e "." -o 1.1 2.2 2.3 tmp1 - >tmp2
stack_ped tmp2 >par_ped
rm tmp1 tmp2
```

Figure 2. Script to form a pedigree file for parents only from a complete pedigree file.

Forming the Normal Equations

Assembling the normal equations proceeds as discussed in section G5 with one exception. The assembly of the blocks of the normal equations to include $\mathbf{R}^{-1}$ can occur in one command instead of requiring two separate **mult** commands. For example, to assemble,

$$\mathbf{Z'R^{-1}X}$$

a single **mult** can be issued,

mult -tzr_ww*Ri X zrtx

The **mult** tool uses a high performance algorithm that takes advantage of the **DIAGONAL** storage mode of **Ri** and the **SPARSE** storage modes of **zr_ww** and **X**. The remainder of the assembly proceeds as discussed before.

7 Assembling Multiple Trait Models

The discussion in this section deals with models that simultaneously consider two or more dependent variables. For example, a multi-variate model with two dependent variables, each with one random factor for the additive genetic effect would be represented as,

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} X_1 & 0 \\ 0 & X_2 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} Z_1 & 0 \\ 0 & Z_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$$

where,

$$var\begin{bmatrix} u_1 \\ u_2 \\ e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} G_1 & G_{12} & 0 & 0 \\ G_{12} & G_2 & 0 & 0 \\ 0 & 0 & R_1 & R_{12} \\ 0 & 0 & R_{12} & R_2 \end{bmatrix} = \begin{bmatrix} A\sigma^2{}_{a_1} & A\sigma_{a_{12}} & 0 & 0 \\ A\sigma_{a_{12}} & A\sigma^2{}_{a_2} & 0 & 0 \\ 0 & 0 & I\sigma^2{}_1 & I\sigma_{12} \\ 0 & 0 & I\sigma_{12} & I\sigma^2{}_2 \end{bmatrix}$$

Multiple Trait Normal Equations

The normal equations for this model are,

$$\left|\begin{array}{cccc} X_1'R^1X_1 & X_1'R^{12}X_2 & X_1'R^1Z_1 & X_1'R^{12}Z_2 \\ X_2'R^{12}X_1 & X_2'R^2X_2 & X_2'R^{12}Z_1 & X_2'R^2Z_2 \\ Z_1'R^1X_1 & Z_1'R^{12}X_2 & Z_1'R^1Z_1+g^1A^{-1} & Z_1'R^{12}Z_2+g^{12}A^{-1} \\ Z_2R^{12}X_1 & Z_2R^2X_2 & Z_2'R^{12}Z_1 & Z_2'R^2Z_2+g^2A^{-1} \end{array}\right| \left|\begin{array}{c} \beta_1 \\ \beta_2 \\ u_1 \\ u_2 \end{array}\right| = \left|\begin{array}{c} X_1'(R^1y_1+R^{12}Y_2) \\ X_2'(R^{12}y_1+R^2y_2) \\ Z_1'(R^1y_1+R^{12}y_2) \\ Z_2'(R^{12}y_1+R^2y_2) \end{array}\right|$$

Inverse Residual Matrix Blocks

The only tool required to assemble a multiple trait full animal model that has not already been discussed is **form_r**. The **form_r** tool works like the **form_rr** tool for the reduced animal model only it assembles the block matrices for a two trait inverse residual variance matrix for a full animal model. Users will realize that **form_r**, as with several of the tools is generalizable. It can be used to construct residual variance matrices for models that have other random factors besides animal. Such models include herd effects, sire effects or sire and dam effects.
22

RAM Multiple Trait Models

As discussed in section six of the Users Guide, reduced animal models can be represented using the same matrix algebra symbols as full animal models. The only things that need to change are the definitions of the random factor incidence matrices, the inverse residual variance matrix and the inverse numerator relationship matrix.

When assembling the multiple trait RAM the random factor incidence matrices would each be assembled the same way as for a single trait RAM. The only matrix that changes substantially is the inverse residual matrix. The reader is encouraged to see Quaas and Pollak (1980 JAS 51:1277) for a complete treatment of the subject. The **form_rr** tool was used to assemble **R**$^{-1}$ for the single trait RAM in section six of the Users Guide. The **form_rr** tool can also be used to construct the three blocks of the multiple-trait RAM,

$$R^{-1} = \left|\begin{array}{cc} R_1 & R_{12} \\ R_{12} & R_2 \end{array}\right|^{-1} = \left|\begin{array}{cc} R^1 & R^{12} \\ R^{12} & R^2 \end{array}\right|$$

The user should refer to the reference manual page to see this syntax. The user is also encourage to see the example scripts that come with the *abtk* and are given in the appendix of this guide.

8 Solving The Linear System

The current release of the *abtk* comes with a direct solver and an iterative solver routine. Both methods of solution are best suited for linear systems. It is possible to construct non-linear systems or optimization systems using the *abtk*. Users are highly encouraged to develop source codes to work with these types of systems and distribute these codes to their colleagues through the anonymous FTP at cgel.agsci.colostate.edu.

Direct Solution Using an Inverse

Inverting the coefficient matrix of a linear system can be a very computationally intensive procedure, even for relatively small systems of equations. The **invert** tool, like all other *abtk* tools, uses dynamic memory allocation. Therefore problem size is limited by available memory. However, practically the problem size is also limited by processing time and precision of arithmetic used by the C language compiler that built the toolkit. Unfortunately, the inverse is required for the prediction error variance of the solutions. Current approximation methods for prediction error variance are not adequate for many mixed models, particularly when variance components are being estimated using procedures such as REML.

Not many realistic mixed model linear systems can be solved directly using even the largest compute architectures currently available. However, the rapid pace of change in computing technology will continue to allow the user to increase problem sizes suited for direct solution. Users are encouraged to inspect the "CHANGES" file at the anonymous FTP at cgel.agsci.colostate.edu regularly to be informed about updates to the *abtk*.

The **invert** tool can take a direct inverse or a generalized inverse of a matrix.

Iterative Solution

An iterative solution can be obtained to the linear system using the **itgen** tool. **itgen** uses a Gauss-Siedel iteration strategy. Users are encouraged to see the reference manual page to see the optional variations on the iterative method that may speed convergence for their linear system.

Imposing constraints for an iterative solution. Van Vleck (personal communication, 1990) indicates that it is not necessary to impose constraints on linear systems that have dependencies when computing an iterative solution. The values obtained for fixed effects may look well outside the parameter space, but the differences between the values still have the same expectations. However, it is advantageous to impose constraints on fixed effects because it reduces the total number of non-zero elements processed. Additionally, if the equations eliminated are chosen carefully the diagonal dominance of the linear system can be substantially increased, and thus, the total number of rounds required to obtain a converged solution can be decreased.

A very practical constraint to impose is the elimination of the equation for the mean. This equation is the most dense and usually slows the rate of convergence. Another reason to eliminate the equation for the mean is that the easiest way to eliminate it would be to never build it. This way the user can avoid processing all non-zero values through the entire assembly phase.

However, the example given here from the linear system constructed in sections four and five of the Users Guide does not eliminate the equation for the mean. Instead the constraints imposed are to eliminate the equations for the last level (class) of the two fixed effects, sex and age of dam. This is the type of constraint typically imposed by PROC GLM of the SAS procedures (1990). The solutions for the other levels of each effect are then the deviations from the last level. The last levels of the sex and age of dam factors are in equations three and seven, respectively. The **genrliz** tool will impose the constraints

```
genrlz -c coef -n coef_g -r RHS -R RHS_g 3 7
```

Solving the linear system. To compute five hundred rounds of Gauss-Siedel iteration for the example problem assembled in section five of the Users Guide and constrained above, the invocation of **itgen** would be,

```
itgen -m 500 -r RHS_g -b beta -n coef_g
```

The -m parameter indicates the number of rounds to iterate, the -r parameter indicates the name of the right hand side vector, -n indicates the name of the coefficient matrix and -b indicates the name of the solution vector, in this example "beta". Additionally, there is an optional restart parameter, -R, who's argument indicates the name of a vector to use as starting values.

The current implementation of **itgen** does not have a stopping criteria other than the -m parameter. It was determined that convergence criteria were too costly to compute and yielded too little information about the behavior of the linear system. Users can compute convergence values such as the residual variance or genetic trend, in between sets of iteration and restart **itgen** as needed. There is a weak convergence criteria printed to standard error that is useful in indicating if the solutions are diverging. This criteria is the average change in all solutions.

The order of the values in the solution vector file will be the same order that the linear system was assembled. For the example, the first value in beta will be the solution to the equation for the mean. The second value will be the solution corresponding to sex code 1. Its expectation will be the effect of sex 1 minus the effect of sex 2. The third value will be a zero because equation three was eliminated. Values in the fourth through the sixth rows will be solutions for age of dam categories 5, 3 and 2 respectively. This is the order that the **fef** tool constructed the incidence matrix and corresponds to the order of values in the a.list file (see section four of the Users Guide). The expectations of these solutions are the deviation of the effects from the last effect. The value in row seven is zero since its equation was eliminated to break the dependencies. The remaining values are "breeding values" of the animals in the order they appear in the pedigree file.