1991 Pakistan Objective Yield Research

Thomas R. Birkett
July 1991

1991 Wheat Validation Study

In this study there was a random sample of approximately 400 objective yield samples and a random subsample (of the objective yield samples) of 46 validation samples. The following model was used to analyze this data.

$$E(V) = E(Y) + E(V - Y)$$

V is the farmer utilization

Y is the objective yield

These parameters are estimated by

$$\hat{V} = \bar{Y} + \bar{D}$$

$\bar{Y}$ is the average yield for the objective yield samples

$\bar{D}$ is the average difference of validation yield minus objective yield

$$D_i = V_i - Y_i$$

The variance of this estimator is

$$var(\hat{V}) = var(\bar{Y}) + var(\bar{D}) + 2cov(\bar{Y},\bar{D})$$

$$= \frac{\sigma^2{}_Y}{n} + \frac{\sigma^2{}_D}{n_D} + \frac{2}{n}\sigma_{YD}$$

n is the number of objective yield samples
and n_D is the subset of validation samples

From the data we estimate the values of these parameters.

$$n = 400$$
$$n_D = 46$$
$$\sigma^2{}_Y = 1689401$$
$$\sigma^2{}_D = 922412$$
$$\sigma_{YD} = -906095$$

$$\bar{Y} = 2600$$
$$\bar{D} = -691$$
$$\hat{V} = 1909$$
$$\sqrt{var(\hat{V})} = 139$$
$$CV(\hat{V}) = 7.3$$

411

This model can be contrasted to observing V directly, i.e.

$$E(V) = E(\bar{V})$$

where $\bar{V}$ is the mean for the farmer utilization sample.

In this case,

$$\bar{V} = 2292$$

$$\sigma^2_V = 654483$$

$$\sqrt{var(V)} = 119$$

$$CV(\bar{V}) = 5.2$$

So we see that the estimator of farmer utilization with the lowest variance in this sample is the mean of the observed farmer yields from the validation subsample, excluding the rest of the survey data.

The reason for first proposing the model as objective yield plus the difference between objective yield and validation yield is that it is possible that validation surveys will not be done each year. In that case the first model could be used, whereby the observed objective yield mean is added to the average difference. The unobserved average difference would be predicted by the average difference observed when the validation survey was done.

II. Village Master Sample

In 1991 objective yield size plots were also laid out in each village master sample. Now we propose a model that uses the data from these village master objective yield samples, the regular village master sample, and the farmer utilization subsample of the regular objective yield sample. The model is

$$E(V) = E(M) + E(Y - M) + E(V - Y)$$

V is the farmer utilization

Y is the objective yield

M is the village master yield

These parameters are estimated by

$$\hat{V} = \bar{M} + \bar{D}_{YM} + \bar{D}_{V}$$

$\bar{M}$ is the average yield for the village master samples

$\bar{D}_{YM}$ is the average difference of objective yield minus village master yield

$$D_{YMi} = Y_i - M_i$$

$\bar{D}_{VY}$ is the average difference of farmer utilization yield minus objective yield

$$D_{VYi} = V_i - Y_i$$

The variance of this estimator is

$$var(\hat{V}) = var(\bar{M}) + var(\bar{D}_{YM}) + var(\bar{D}_{VY}) + 2cov(\bar{M}, \bar{D}_{YM})$$

$$= \frac{\sigma^2_M}{n} + \frac{\sigma^2_{D_{YM}}}{n} + \frac{\sigma^2_{D_{VY}}}{n_D} + \frac{2}{n}\sigma_{12}$$

n is the number of village master samples

n_D is the number of farmer utilization samples

$$\sigma_{13} = \sigma_{23} = 0$$

The mean village master yield and mean difference between village master and objective yield is estimated from the village master sample. The average difference between objective yield and validation yield is estimated from the validation subsample. The covariance is estimated from the village master sample. The two covariances that would normally appear in the expression are zero because the village master sample and the regular objective yield sample are independent. This model assumes that both the village master sample and the objective yield sample are random samples from the same distribution.

From the data we estimate

$$n = 400$$

$$n_D = 46$$

$$\bar{M} =$$

$$\bar{D}_{YM} =$$

$$\bar{D}_{VY} = -691$$

$$\hat{V} =$$

$$\sigma^2_M =$$

$$\sigma^2_{D_{YM}} =$$

$$\sigma^2_{D_{VY}} = 932741$$

$$\sigma_{12} =$$

$$var(\hat{V})$$

II. Two plots in each objective yield sample

For the first time in 1991 two plots were independently located in each sample. Let

Y_1 be the yield from the first plot
Y_2 be the yield from the second plot

$$Y = \frac{1}{2}(Y_1+Y_2)$$

be the combined yield

From the data, we estimate

$$\sigma_Y = 1509$$
$$\sigma_{Y_1} = 1660$$
$$\sigma_{Y_2} = 1551$$
$$corr(Y_1, Y_2) = .77$$

We can lower the standard deviation from 1660 to 1509 by doing the extra plot. This is a decrease of 9.1 percent. While the yields from the two plots have positive correlation of .77, it is not so high that we don't increase our accuracy by doing the extra plot.

CORRELATION ANALYSIS

3 'VAR' Variables: WHTYLD STDYLD

Variances and Covariances
COV('W','V') / VAR('W') / VAR('V') / DF('W','V')

'W'\'V'	WHTYLD	STDYLD	D
WHTYLD	765625.771	763347.320	2278.450
	765625.771	765625.771	765625.771
	765625.771	1638994.217	877925.347
	46	46	46
STDYLD	763347.320	2205966.449	-875646.897
	1638994.217	2205966.449	1638994.217
	765625.771	2205966.449	877925.347
	46	393	46
D	2278.450	-875646.897	877925.347
	877925.347	877925.347	877925.347
	765625.771	1638994.217	877925.347
	46	46	46

CORRELATION ANALYSIS

Sinple Statistics

Variable	N	Mean	Std Dev	Sum
WHTYLD	47	2336	875.00044	109813
STDYLD	394	2700	1485	1063653
D	47	-650.41315	936.97671	-30569

Sinple Statistics

Variable	Minimum	Maximum
WHTYLD	635.45709	3750
STDYLD	0	7994
D	-2610	1123

$V = T + E = 2205966.449 - 875646.897 = 1330320$

$V(\bar{Y}) = \frac{S_T^2}{n} + \frac{S_E^2}{n_D} + \frac{2}{n} C_{YD}$

$= \frac{2205966.449}{394} + \frac{877925.347}{47} + \frac{2}{394}(-875646.897)$

$= 5598.398 + 18677.016 - 4444.9$

$V(\bar{Y}) = 19,830.56$

$S_{\bar{Y}} = 140.83$

$CV(\bar{Y}) = 6.8$

1991 Pakistan Objective Yield Research

Thomas R. Birkett
July 1991

I. 1991 Wheat Validation Study

In this study there was a random sample of approximately 400 objective yield samples and a random subsample (of the objective yield samples) of 46 validation samples. The following model was used to analyze this data.

$$E(V) = E(Y) + E(V - Y)$$

V is the farmer utilization

Y is the objective yield

These parameters are estimated by

$$\hat{V} = \bar{Y} + \bar{D}$$

$\bar{Y}$ is the average yield for the objective yield samples

$\bar{D}$ is the average difference of validation yield minus objective yield

$$D_i = V_i - Y_i$$

The variance of this estimator is

$$var(\hat{V}) = var(\bar{Y}) + var(\bar{D}) + 2cov(\bar{Y},\bar{D})$$

$$= \frac{\sigma^2{}_Y}{n} + \frac{\sigma^2{}_D}{n_D} + \frac{2}{n}\sigma_{YD}$$

n is the number of objective yield samples

and n_D is the subset of validation samples

From the data we estimate the values of these parameters.

$$n = 400$$
$$n_D = 46$$
$$\sigma^2{}_Y = 1689401$$
$$\sigma^2{}_D = 922412$$
$$\sigma_{YD} = -906095$$

$$\bar{Y} = 2600$$

$$\bar{D} = -691$$

$$\hat{V} = 1909$$

$$\sqrt{var(\hat{V})} = 139$$

$$CV(\hat{V}) = 7.3$$

This model can be contrasted to observing V directly, i.e.

$$E(V) = E(\bar{V})$$

where $\bar{V}$ is the mean for the farmer utilization sample.

In this case,

$$\bar{V} = 2292$$

$$\sigma^2_V = 654483$$

$$\sqrt{var(V)} = 119$$

$$CV(\bar{V}) = 5.2$$

So we see that the estimator of farmer utilization with the lowest variance in this sample is the mean of the observed farmer yields from the validation subsample, excluding the rest of the survey data.

The reason for first proposing the model as objective yield plus the difference between objective yield and validation yield is that it is possible that validation surveys will not be done each year. In that case the first model could be used, whereby the observed objective yield mean is added to the average difference. The unobserved average difference would be predicted by the average difference observed when the validation survey was done.

II. Village Master Sample

In 1991 objective yield size plots were also laid out in each village master sample. Now we propose a model that uses the data from these village master objective yield samples, the regular village master sample, and the farmer utilization subsample of the regular objective yield sample. The model is

$$E(V) = E(M) + E(Y - M) + E(V - Y)$$

V is the farmer utilization

Y is the objective yield

M is the village master yield

These parameters are estimated by

$$\hat{V} = \bar{M} + \bar{D}_{YM} + \bar{D}_{VY}$$

$\bar{M}$ is the average yield for the village master samples

$\bar{D}_{YM}$ is the average difference of objective yield minus village master yield

$$D_{YMi} = Y_i - M_i$$

$\bar{D}_{VY}$ is the average difference of farmer utilization yield minus objective yield

$$D_{VYi} = V_i - Y_i$$

The variance of this estimator is

$$var(\hat{V}) = var(\bar{M}) + var(\bar{D}_{YM}) + var(\bar{D}_{VY}) + 2cov(\bar{M}, \bar{D}_{YM})$$

$$= \frac{\sigma^2_M}{n} + \frac{\sigma^2_{D_{YM}}}{n} + \frac{\sigma^2_{D_{VY}}}{n_D} + \frac{2}{n}\sigma_{12}$$

n is the number of village master samples

n_D is the number of farmer utilization samples

$$\sigma_{13} = \sigma_{23} = 0$$

The mean village master yield and mean difference between village master and objective yield is estimated from the village master sample. The average difference between objective yield and validation yield is estimated from the validation subsample. The covariance is estimated from the village master sample. The two covariances that would normally appear in the expression are zero because the village master sample and the regular objective yield sample are independent. This model assumes that both the village master sample and the objective yield sample are random samples from the same distribution.

From the data we estimate

$$n = 400$$

$$n_D = 46$$

$$\bar{M} =$$

$$\bar{D}_{YM} =$$

$$\bar{D}_{VY} = -691$$

$$\hat{V} =$$

$$\sigma^2_M =$$

$$\sigma^2_{D_{YM}} =$$

$$\sigma^2_{D_{VY}} = 932741$$

$$\sigma_{12} =$$

$$var(\hat{V}) =$$

II. Two plots in each objective yield sample

For the first time in 1991 two plots were independently located in each sample. Let

Y_1 be the yield from the first plot
Y_2 be the yield from the second plot

$$Y = \frac{1}{2}(Y_1+Y_2)$$

be the combined yield

From the data we estimate

$$\sigma_Y = 1509$$
$$\sigma_{Y_1} = 1660$$
$$\sigma_{Y_2} = 1551$$
$$corr(Y_1, Y_2) = .77$$

We can lower the standard deviation from 1660 to 1509 by doing the extra plot. This is a decrease of 9.1 percent. While the yields from the two plots have positive correlation of .77, it is not so high that we don't increase our accuracy by doing the extra plot.

Chapter 4 Working with Matrices

Introduction 31

Entering Data as Matrix Literals 32
Scalars 32
Matrices with Multiple Elements 32

Using Assignment Statements 33
Simple Assignment Statements 34
Matrix-generating Functions 35
The BLOCK Function 36
The J Function 36
The I Function 37
The DESIGNF Function 37
The SHAPE Function 38
Index Vectors 38

Using Matrix Expressions 39
Operators 39
Compound Expressions 40
Elementwise Binary Operators 41
Subscripts 43
Selecting a Single Element 43
Selecting a Row or Column 44
Submatrices 45
Subscripted Assignment 45
Subscript Reduction Operators 47

Printing Matrices with Row and Column Headings 48
Using the AUTONAME Option 48
Using the ROWNAME= and COLNAME= Options 49
Using the MATTRIB Statement 49

More on Missing Values 50

Introduction

SAS/IML software provides many ways to create matrices. You can create matrices by doing any of the following:

- entering data yourself as a matrix literal
- using assignment statements
- using matrix-generating functions
- creating submatrices from existing matrices with subscripts
- using SAS data sets (see Chapter 6, "Working with SAS Data Sets," for more information).

Once you have defined matrices, you have access to many operators and functions for working on them in matrix expressions. These operators and functions facilitate programming and make referring to submatrices efficient and simple.

Finally, you have several means available for tailoring your printed output.

Entering Data as Matrix Literals

The most basic way to create a matrix is to define a matrix literal, either numeric or character, by entering the matrix elements. A matrix literal can be a single element (called a *scalar*), a single row of data (called a *row vector*), a single column of data (called a *column vector*), or a rectangular array of data (called a *matrix*). The *dimension* of a matrix is given by its number of rows and columns. An $n \times m$ matrix has n rows and m columns.

Scalars

Scalars are matrices that have only one element. You define a scalar with the matrix name on the left-hand side of an assignment statement and its value on the right-hand side. You can use the statements below to create and print several examples of scalar literals. First, you must invoke PROC IML.

```
> proc iml;

  IML Ready

> x=12;
> y=12.34;
> z=.;
> a='Hello';
> b="Hi there";
> print x y z a b;

              X          Y          Z  A      B
             12      12.34             Hello  Hi there
```

Notice that when defining a character literal, you need to use either single quotes (') or double quotes ("). Using quotes preserves uppercase and lowercase distinctions and embedded blanks. It is also always correct to enclose the data values inside of braces ({ }).

Matrices with Multiple Elements

To enter a matrix having multiple elements, use braces ({ }) to enclose the data values and, if needed, commas to separate rows. Inside of the braces, all elements must be either numeric or character. You cannot have a mixture of data types within a matrix. Each row must have the same number of elements.

For example, suppose that you have one week of data on daily coffee consumption (cups per day) for your office of four people. Create a matrix

COFFEE with each person's consumption as a row of the matrix and each day represented by a column. First, submit the RESET PRINT statement so that results are printed as you submit statements.

```
> reset print;
> coffee={4 2 2 3 2,
>         3 3 1 2 1,
>         2 1 0 2 1,
>         5 4 4 3 4};

COFFEE        4 rows      5 cols     (numeric)

              4           2          2          3          2
              3           3          1          2          1
              2           1          0          2          1
              5           4          4          3          4
```

Now create a character matrix called **NAMES** with rows containing the names the people in your office. Note that when you do not use quotes, characters are converted to uppercase.

```
> names={Jenny, Linda, Jim, Samuel};

NAMES         4 rows      1 col      (character, size 6)

                         JENNY
                         LINDA
                         JIM
                         SAMUEL
```

Notice that the output with the RESET PRINT statement includes the dimension, type, and when type is character, the element size of the matrix. The element size represents the length of each string and is determined from the length of the longest string.

Now, print **COFFEE** using **NAMES** as row labels with the ROWNAME= option to the PRINT statement.

```
> print coffee [rowname=names];

COFFEE
JENNY        4          2          2          3          2
LINDA        3          3          1          2          1
JIM          2          1          0          2          1
SAMUEL       5          4          4          3          4
```

Using Assignment Statements

Assignment statements create matrices by evaluating expressions and assigning the results to a matrix. The expressions can be composed of operators (for

example, the matrix addition operator (+)), functions (for example, the INV function), and subscripts. Assignment statements have the general form

result=expression;

where *result* is the name of the new matrix and *expression* is an expression that is evaluated. The resulting matrix automatically acquires the appropriate dimension, type, and value. Details on writing expressions are described in "Using Matrix Expressions" later in this chapter.

Simple Assignment Statements

Simple assignment statements involve an equation having the matrix name on the left-hand side and either an expression involving other matrices or a matrix-generating function on the right-hand side.

Suppose you want to generate some statistics for the weekly coffee data. If a cup of coffee costs 30 cents, then you can create a matrix with the daily expenses, **DAYCOST**, by multiplying the per-cup cost with the matrix **COFFEE** using the elementwise multiplication operator (#). Turn off the automatic printing so that you can tailor the output with the ROWNAME= and FORMAT= options in the PRINT statement.

```
> reset noprint;
> daycost=0.30#coffee;
> print "Daily totals", daycost[rowname=names format=8.2];

      Daily totals

      DAYCOST
      JENNY      1.20     0.60     0.60     0.90     0.60
      LINDA      0.90     0.90     0.30     0.60     0.30
      JIM        0.60     0.30     0.00     0.60     0.30
      SAM        1.50     1.20     1.20     0.90     1.20
```

You can calculate the weekly total cost for each person using the matrix multiplication operator (*). First create a 5×1 vector of 1's. This vector will sum the daily costs for each person when multiplied with **COFFEE**. (You will see later that there is a more efficient way to do this using subscript reduction operators.)

```
> ones=[1,1,1,1,1];
> weektot=daycost*ones;
> print "Week total", weektot[rowname=names format=8.2];

        Week total

        WEEKTOT
        JENNY      3.90
        LINDA      3.00
        JIM        1.80
        SAM        6.00
```

Finally, you can calculate the average number of cups drunk per day by dividing the grand total of cups by days. To find the grand total, use the SUM function, which returns the sum of all elements of a matrix. Next, divide the grand total by 5, the number of days (which is the number of columns) using the division operator (/) and the NCOL function. These two matrices are created separately, but the entire calculation could be done in one statement.

```
> grandtot=sum(coffee);
> average=grandtot/ncol(coffee);
> print "Total number of cups", grandtot,,"Daily average",average;
```

```
Total number of cups

GRANDTOT
      49

Daily average

AVERAGE
    9.8
```

Matrix-generating Functions

SAS/IML software has many built-in functions that generate useful matrices. For example, the J function creates a matrix with a given dimension and element value when you supply the number of rows, columns, and an element value for the new matrix. This function is useful to initialize a matrix to a predetermined size. Several matrix-generating functions are listed below:

BLOCK	creates a block-diagonal matrix.
DESIGNF	creates a full-rank design matrix.
I	creates an identity matrix.
J	creates a matrix of a given dimension.
SHAPE	shapes a new matrix from the argument.

The sections that follow illustrate these matrix-generating functions. Again, they are shown with automatic printing of results, activated by invoking the RESET PRINT statement.

```
reset print;
```

The BLOCK Function

The BLOCK function has the general form

BLOCK(*matrix1*,<*matrix2*,...,*matrix15*>)

and creates a block-diagonal matrix from the argument matrices. For example, the statements

```
> a={1 1,1 1};

        A            2 rows      2 cols     (numeric)

                       1          1
                       1          1


> b={2 2, 2 2};

        B            2 rows      2 cols     (numeric)

                       2          2
                       2          2

> c=block(a,b);
```

result in the matrix

```
        C            4 rows      4 cols     (numeric)

          1          1          0          0
          1          1          0          0
          0          0          2          2
          0          0          2          2
```

The J Function

The J function has the general form

J(*nrow*<,*ncol*<,*value*>>)

and creates a matrix having *nrow* rows, *ncol* columns, and all element values equal to *value*. The *ncol* and *value* arguments are optional, but you will usually want to specify them. In many statistical applications, it is helpful to be able to create a row (or column) vector of 1's (you did so to calculate coffee totals in the last section). You can do this with the J function. For example, the statement below creates a 1×5 row vector of 1's:

```
> one=j(1,5,1);

        ONE          1 row       5 cols     (numeric)
          1          1          1          1          1
```

The I Function

The I function creates an identity matrix of a given size. It has the general form

I(*dimension*)

where *dimension* gives the number of rows. For example, the following statement creates a 3×3 identity matrix:

```
> I3=I(3);

I3          3 rows      3 cols     (numeric)

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

The DESIGNF Function

The DESIGNF function generates a full-rank design matrix, useful in calculating *ANOVA* tables. It has the general form

DESIGNF(*column-vector*)

For example, the following statement creates a full-rank design matrix for a one-way *ANOVA*, where the treatment factor has three levels and there are $n_1=3$, $n_2=2$, and $n_3=2$ observations at the factor levels:

```
> d=designf({1,1,1,2,2,3,3});

D           7 rows      2 cols     (numeric)

            1           0
            1           0
            1           0
            0           1
            0           1
           -1          -1
           -1          -1
```

The SHAPE Function

The SHAPE function shapes a new matrix from an argument matrix. It has the general form

SHAPE(*matrix*<,*nrow*<,*ncol*<,*pad-value*>>>)

Although the *nrow*, *ncol*, and *pad-value* arguments are optional, you will usually want to specify them. The example below uses the SHAPE function to create a 3×3 matrix containing the values 99 and 33. The function cycles back and repeats values to fill in when no *pad-value* is given.

```
> aa=shape({99 33,99 33},3,3);

        AA          3 rows      3 cols    (numeric)

                   99          33          99
                   33          99          33
                   99          33          99
```

In the next example, a *pad-value* is specified for filling in the matrix:

```
> aa=shape({99 33,99 33},3,3,0);

        AA          3 rows      3 cols    (numeric)

                   99          33          99
                   33           0           0
                    0           0           0
```

The SHAPE function cycles through the argument matrix elements in row-major order and then fills in with 0's after the first cycle through the argument matrix.

Index Vectors

You can create a vector by using the index operator (:). Several examples of statements involving index vectors are shown below:

```
> r=1:5;

        R           1 row       5 cols    (numeric)
                 1       2       3       4       5

> s=10:6;

        S           1 row       5 cols    (numeric)
                10       9       8       7       6

> t='abc1':'abc5';

        T           1 row       5 cols    (character, size 4)
                 abc1 abc2 abc3 abc4 abc5
```

If you want an increment other than 1, use the DO function. For example, if you want a vector ranging from minus 1 to 1 by .5, use the following statement:

```
> r=do(-1,1,.5);

                R            1 row       5 cols     (numeric)
                      -1     -0.5          0         0.5          1
```

Using Matrix Expressions

Matrix expressions are a sequence of names, literals, operators, and functions that perform some calculation, evaluate some condition, or manipulate values. These expressions can appear on either side of an assignment statement.

Operators

Operators used in matrix expressions fall into three general categories:

prefix operators — are placed in front of operands. For example, −A uses the sign reverse prefix operator (−) in front of the operand A to reverse the sign of each element of A.

infix operators — are placed between operands. For example, A+B uses the addition infix operator (+) between operands A and B to add corresponding elements of the matrices.

postfix operators — are placed after an operand. For example, A` uses the transpose postfix operator (`) after the operand A to transpose A.

Matrix operators are listed in Appendix 1, "SAS/IML Quick Reference," and described in detail in Chapter 15, "SAS/IML Language Reference." Table 4.1 shows the precedence of matrix operators in IML.

Table 4.1 Operator Precedence

Priority Group	Operators
I (highest)	^ ` subscripts −(prefix) ## **
II	* # <> >< / @
III	+ −
IV	|| // :
V	< <= > >= = ^=
VI	&
VII (lowest)	|

Compound Expressions

With SAS/IML software, you can write compound expressions involving several matrix operators and operands. For example, the following statements are valid matrix assignment statements:

```
a=x+y+z;
a=x+y*z';
a=(-x)#(y-z);
```

The rules for evaluating compound expressions are as follows:

Evaluation follows the order of operator precedence, as shown in Table 4.1. Group I has the highest priority; that is, Group I operators are evaluated first. Group II operators are evaluated after Group I operators, and so forth. For example, the statement

```
a=x+y*z;
```

first multiplies matrices **Y** and **Z** since the * operator (Group II) has higher precedence than the + operator (Group III). It then adds the result of this multiplication to the matrix **X**, and assigns the new matrix to **A**.

□ If neighboring operators in an expression have equal precedence, the expression is evaluated from left to right, except for the highest priority operators. For example, the statement

```
a=x/y/z;
```

first divides each element of matrix **X** by the corresponding element of matrix **Y**. Then, using the result of this division, it divides each element of the resulting matrix by the corresponding element of matrix **Z**. The operators in Group 1 in Table 4.1 are evaluated from right to left. For example, the expression

```
-x**2
```

is evaluated as

```
-(x**2)
```

When multiple prefix or postfix operators are juxtaposed, precedence is determined by their order from inside to outside.

For example, the expression

```
^-a
```

is evaluated as ^(−A), and the expression

```
a`[i,j]
```

is evaluated as (A`)[i,j].

□ All expressions enclosed in parentheses are evaluated first, using the two rules above. Thus, the IML statement

```
a=x/(y/z);
```

is evaluated by first dividing elements of Y by the elements of Z, then dividing this result into X.

Elementwise Binary Operators

Elementwise binary operators produce a result matrix from element-by-element operations on two argument matrices. Table 4.2 lists the elementwise binary operators.

Table 4.2 Elementwise Binary Operators

Operator	Action
+	addition, concatenation
−	subtraction
#	elementwise multiplication
##	elementwise power
/	division
<>	element maximum
><	element minimum
||	logical OR
&	logical AND
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
^=	not equal to
=	equal to
MOD(*m*,*n*)	modulo (remainder)

For example, consider the two matrices A and B given below.

$$\text{Let } A = \begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix} \text{ and } B = \begin{bmatrix} 4 & 5 \\ 1 & 0 \end{bmatrix}$$

The addition operator (+) adds corresponding matrix elements:

$$A+B \text{ yields } \begin{bmatrix} 6 & 7 \\ 4 & 4 \end{bmatrix}.$$

The elementwise multiplication operator (#) multiplies corresponding elements:

$$A\#B \text{ yields } \begin{bmatrix} 8 & 10 \\ 3 & 0 \end{bmatrix}$$

The elementwise power operator (##) raises elements to powers:

$$A\#\#2 \text{ yields } \begin{bmatrix} 4 & 4 \\ 9 & 16 \end{bmatrix}.$$

The element maximum operator (<>) compares corresponding elements and chooses the larger:

$$A<>B \text{ yields } \begin{bmatrix} 4 & 5 \\ 3 & 4 \end{bmatrix}$$

The less than or equal to operator (<=)returns a 1 if an element of A is less than or equal to the corresponding element of B, and returns a 0 otherwise:

$$A<=B \text{ yields } \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

The modulo operator returns the remainder of each element div led by the argument.

$$\text{MOD(A,3) yields } \begin{bmatrix} 2 & 2 \\ 0 & 1 \end{bmatrix}$$

All operators can also work in a one-to-many or many-to-one manner, as well as in an element-to-element manner; that is, they allow you to perform tasks such as adding a scalar to a matrix or dividing a matrix by a scalar. For example, the statement

```
x=x|(x>0);
```

replaces each negative element of the matrix X with 0. The expression (X>0) is a many-to-one operation that compares each element of X to 0 and creates a temporary matrix of results; an element in the result matrix is 1 when the expression is true and 0 when it is false. When the expression is true (the element is positive), the element is multiplied by 1. When the expression is false (the element is negative or 0), the element is multiplied by 0. To fully understand the intermediate calculations, you can use the RESET PRINTALL command to have the temporary result matrices printed.

Subscripts

Subscripts are special postfix operators placed in square brackets ([ ]) after a matrix operand. Subscript operations have the general form

operand[*row*,*column*]

where

operand	is usually a matrix name, but it can also be an expression or literal.
row	refers to an expression, either scalar or vector, for selecting one or more rows from the operand.
column	refers to an expression, either scalar or vector, for selecting one or more columns from the operand.

You can use subscripts to

- refer to a single element of a matrix
- refer to an entire row or column of a matrix
- refer to any submatrix contained within a matrix
- perform a reduction across rows or columns of a matrix.

In expressions, subscripts have the same (high) precedence as the transpose postfix operator (`). Note that when both *row* and *column* subscripts are used, they are separated by a comma.

Selecting a Single Element

You can select a single element of a matrix in two ways. You can use two subscripts (*row*, *column*) to refer to its location, or you can use one subscript to look for the element down the rows. For example, referring to the coffee example used earlier, to find the element corresponding to the number of cups that Linda drank on Monday, you can use either of two statements.

First, you can refer to the element by row and column location. In this case, you want the second row and first column. You can call this matrix **c21**.

```
> print  coffee[rowname=names];

        COFFEE
        JENNY      4      2      2      3      2
        LINDA      3      3      1      2      1
        JIM        2      1      0      2      1
        SAM        5      4      4      3      4

> c21=coffee[2,1];
> print c21;

                        C21
                          3
```

You can also look for the element down the rows. In this case, you refer to this element as the sixth element of **COFFEE** in row-major order.

```
> c6=coffee[6];
> print c6;

                        C6
                         3
```

Selecting a Row or Column

To refer to an entire row or column of a matrix, write the subscript with the row or column number, omitting the other subscript but not the comma. For example, to refer to the row of **COFFEE** that corresponds to Jim, you want the submatrix consisting of the third row and all columns:

```
> jim=coffee[3,];
> print jim;

        JIM
          2      1      0      2      1
```

If you want the data for Friday, you know that the fifth column corresponds to Friday, so you want the submatrix consisting of the fifth column and all rows:

```
> friday=coffee[,5];
> print friday;

                      FRIDAY
                           2
                           1
                           1
                           4
```

Submatrices

You refer to a submatrix by the specific rows and columns you want. Include within the brackets the rows you want, a comma, and the columns you want. For example, to create the submatrix of COFFEE consisting of the first and third rows and second, third, and fifth columns, submit the following statements:

```
> submat1=coffee[{1 3},{2 3 5}];
> print submat1;
```

```
SUBMAT1
      2        2        2
      1        0        1
```

The first vector, {1 3}, selects the rows, and the second vector, {2 3 5}, selects the columns. Alternately, you can create the vectors beforehand and supply their names as arguments.

```
> rows={1 3};
> cols={2 3 5};
> submat1=coffee[rows,cols];
```

You can use index vectors generated by the index creation operator (:) in subscripts to refer to successive rows or columns. For example, to select the first three rows and last three columns of COFFEE, use the following statements:

```
> submat2=coffee[1:3,3:5];
> print submat2;
```

```
SUBMAT2
      2        3        2
      1        2        1
      0        2        1
```

Note that, in each example, the number in the first subscript defines the number of rows in the new matrix; the number in the second subscript defines the number of columns.

Subscripted Assignment

You can assign values into a matrix using subscripts to refer to the element or submatrix. In this type of assignment, the subscripts appear on the left-hand side of the equal sign. For example, to change the value in the first row, second column of COFFEE from 2 to 4, use subscripts to refer to the appropriate element in an assignment statement:

```
> coffee[1,2]=4;
> print coffee;
```

```
COFFEE
      4        4        2        3        2
      3        3        1        2        1
      2        1        0        2        1
      5        4        4        3        4
```

To change the values in the last column of **COFFEE** to 0s use the following statement:

```
> coffee[,5]=[0,0,0,0];
> print coffee;

COFFEE
    4     4     2     3     0
    3     3     1     2     0
    2     1     0     2     0
    5     4     4     3     0
```

In the next example, you first locate the positions of negative elements of a matrix and then set these elements equal to 0. This can be useful in situations where negative elements may indicate errors or be impossible values. The LOC function is useful for creating an index vector for a matrix that satisfies some condition. In the example below, the LOC function is used to find the positions of the negative elements of the matrix T and then to set these elements equal to 0 using subscripted assignment:

```
> t=[ 3  2 -1,
>     6 -4  3,
>     2  2  2 ];
> print t;

T
3     2    -1
6    -4     3
2     2     2

> i=loc(t<0);
> print i;

I
3     5

> t[i]=0;
> print t;

T
3     2     0
6     0     3
2     2     2
```

Subscripts can also contain expressions whose results are either row or column vectors. These statements can also be written

```
> t[loc(t<0)]=0;
```

If you use a noninteger value as a subscript, only the integer portion is used. Using a subscript value less than one or greater than the dimension of the matrix results in an error.

Subscript Reduction Operators

You can use reduction operators, which return a matrix of reduced dimension, in place of values for subscripts to get reductions across all rows and columns. Table 4.3 lists the eight operators for subscript reduction in IML.

Table 4.3 Subscript Reduction Operators

Operator	Action
+	addition
#	multiplication
<>	maximum
><	minimum
<:>	index of maximum
>:<	index of minimum
:	mean (different from the MATRIX procedure)
##	sum of squares

For example, to get column sums of the matrix X (sum across the rows, which reduces the row dimension to 1), specify X[+,]. The first subscript (+) specifies summation reduction take place across the rows. Omitting the second subscript, corresponding to columns, leaves the column dimension unchanged. The elements in each column are added, and the new matrix consists of one row containing the column sums.

You can use these operators to reduce either rows or columns or both. When both rows and columns are reduced, row reduction is done first.

For example, the expression A[+,<>] results in the maximum (<>) of the column sums (+).

You can repeat reduction operators. To get the sum of the row maxima, use the expression A[,<>] [+,] .

A subscript such as A[{2 3},+] first selects the second and third rows of A and then finds the row sums of that matrix.

The following examples demonstrate how to use the operators for subscript reduction.

$$\text{Let } \mathbf{A} = \begin{bmatrix} 0 & 1 & 2 \\ 5 & 4 & 3 \\ 7 & 6 & 8 \end{bmatrix}$$

The following statements are true:

A[{2 3},+] yields $\begin{bmatrix} 12 \\ 21 \end{bmatrix}$ (row sums for rows 2 and 3)

A[+,<>] yields $\begin{bmatrix} 13 \end{bmatrix}$ (maximum of column sums)

A[<>,+] yields $\begin{bmatrix} 21 \end{bmatrix}$ (sum of column maxima)

A[,><] [+,] yields $\begin{bmatrix} 9 \end{bmatrix}$ (sum of row minima)

A[,<:>] yields $\begin{bmatrix} 3 \\ 1 \\ 3 \end{bmatrix}$ (indices of row maxima)

A[>:<] yields $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ (indices of column minima)

A[:] yields $\begin{bmatrix} 4 \end{bmatrix}$ (mean of all elements)

Printing Matrices with Row and Column Headings

You can tailor the way your matrices are printed in several ways with the AUTONAME option, the ROWNAME= and COLNAME= options, or the MATTRIB statement.

Using the AUTONAME Option

You can use the RESET statement with the AUTONAME option to automatically print row and column headings. If your matrix has *n* rows and *m* columns, the row headings are ROW1 to ROW*n* and the column headings are COL1 to COL*m*. For example, the following statements produce the result shown below:

```
> reset autoname;
> print coffee;
```

COFFEE	COL1	COL2	COL3	COL4	COL5
ROW1	4	2	2	3	2
ROW2	3	3	1	2	1
ROW3	2	1	0	2	1
ROW4	5	4	4	3	4

Using the ROWNAME= and COLNAME= Options

You can specify your own row and column headings. The easiest way is to create vectors containing the headings and then print the matrix with the ROWNAME= and COLNAME= options. For example, the following statements produce the result shown below:

```
> names={jenny linda jim samuel};
> days={mon tue wed thu fri};
> print coffee{rowname=names colname=days};
```

COFFEE	MON	TUE	WED	THU	FRI
JENNY	4	2	2	3	2
LINDA	3	3	1	2	1
JIM	2	1	0	2	1
SAMUEL	5	4	4	3	4

Using the MATTRIB Statement

The MATTRIB statement associates printing characteristics with matrices. You can use the MATTRIB statement to print COFFEE with row and column headings. In addition, you can format the printed numeric output and assign a label to the matrix name. The following example shows how to tailor your printed output:

```
> mattrib coffee rowname={{jenny linda jim samuel}}
>         colname={{mon tue wed thu fri}}
>         label='Weekly Coffee'
>         format=2.0;
> print coffee;
```

Weekly Coffee	MON	TUE	WED	THU	FRI
JENNY	4	2	2	3	2
LINDA	3	3	1	2	1
JIM	2	1	0	2	1
SAMUEL	5	4	4	3	4

More on Missing Values

Missing values in matrices were discussed in Chapter 2, "Understanding the SAS/IML Language." You should read that chapter and Chapter 14, "Further Notes," carefully so that you are aware of the way IML treats missing values. Below are several examples that show how IML handles missing values in a matrix.

$$\text{Let } \mathbf{X} = \begin{bmatrix} 1 & 2 & . \\ . & 5 & 6 \\ 7 & . & 9 \end{bmatrix} \text{ and } \mathbf{Y} = \begin{bmatrix} 4 & . & 2 \\ 2 & 1 & 3 \\ 6 & . & 5 \end{bmatrix}$$

The following statements are true:

$$\mathbf{X}+\mathbf{Y} \text{ yields } \begin{bmatrix} 5 & . & . \\ . & 6 & 9 \\ 13 & . & 14 \end{bmatrix} \text{(matrix addition)}$$

$$\mathbf{X}\#\mathbf{Y} \text{ yields } \begin{bmatrix} 4 & . & . \\ . & 5 & 18 \\ 42 & . & 45 \end{bmatrix} \text{(element multiplication)}$$

$$\mathbf{X}[+,] \text{ yields } \begin{bmatrix} 8 & 7 & 15 \end{bmatrix} \text{(column sums)}$$

Chapter 6 Working with SAS Data Sets

Introduction 71

Opening a SAS Data Set 73

Making a SAS Data Set Current 73

Displaying SAS Data Set Information 74

Referring to a SAS Data Set 74

Listing Observations 75
Specifying a Range of Observations 75
Selecting a Set of Variables 78
Selecting Observations 79

Reading Observations from a SAS Data Set 81
Using the READ Statement with the VAR Clause 82
Using the READ Statement with the VAR and INTO Clauses 82
Using the READ Statement with the WHERE Clause 83

Editing a SAS Data Set 84
Updating Observations 85
Deleting Observations 86

Creating a SAS Data Set from a Matrix 87
Using the CREATE Statement with the FROM Option 88
Using the CREATE Statement with the VAR Clause 89

Understanding the End-of-File Condition 89

Producing Summary Statistics 90

Sorting a SAS Data Set 90

Indexing a SAS Data Set 91

Data Set Maintenance Functions 92

Overview of Commands 93

Similarities and Differences with the SAS DATA Step 94

In Conclusion 94

Introduction

SAS/IML software has many statements for passing data from SAS data sets to matrices and from matrices to SAS data sets. You can create matrices from the variables and observations of a SAS data set in several ways. You can create a column vector for each data set variable, or you can create a matrix where columns correspond to data set variables. You can use all the observations in a data set or use a subset of them.

You can also create a SAS data set from a matrix. The columns correspond to data set variables and the rows correspond to observations. Data management

commands enable you to edit, append, rename, or delete SAS data sets from within the SAS/IML environment.

When reading a SAS data set, you can read any number of observations into a matrix either sequentially, directly by record number, or conditionally according to conditions in a WHERE clause. You can also index a SAS data set. The indexing capability facilitates retrievals by the indexed variable.

Operations on SAS data sets are performed with straightforward, consistent, and powerful statements. For example, the LIST statement can perform the following tasks:

- list the next record
- list a specified record
- list any number of specified records
- list the whole file
- list records satisfying one or more conditions
- list specified variables or all variables.

If you want to read values into a matrix, use the READ statement instead of the LIST statement with the same operands and features as LIST. You can specify operands that control which records and variables are used indirectly, as matrices, so that you can dynamically program the records, variables, and conditional values you want.

In this chapter, you will use the SAS data set CLASS, which contains the variables NAME, SEX, AGE, HEIGHT, and WEIGHT, to learn about

- opening a SAS data set
- examining the contents of a SAS data set
- printing data values with the LIST statement
- reading observations from a SAS data set into matrices
- editing a SAS data set
- creating a SAS data set from a matrix
- printing matrices with row and column headings
- producing summary statistics
- sorting a SAS data set
- indexing a SAS data set
- similarities and differences with the SAS DATA step.

Throughout this chapter, the right angle brackets (>) indicate statements you submit; responses from IML follow below.

First, invoke the IML procedure:

```
> proc iml;

IML Ready
```

Opening a SAS Data Set

Before you can access a SAS data set, you must first submit a command to open it. There are three ways to open a SAS data set:

- To simply read from an existing data set, submit a USE statement to open it for read access. The general form of the USE statement is

  USE *SAS-data-set* <VAR *operand*> <WHERE(*expression*)>;

  With read access you can use the FIND, INDEX, LIST, and READ statements on the data set.

- To read and write to an existing data set, use the EDIT statement. The general form of the EDIT statement is

  EDIT *SAS-data-set* <VAR *operand*> <WHERE(*expression*)>;

  This statement enables you to use both the reading statements (LIST, READ, INDEX, and FIND) and the writing statements (REPLACE, APPEND, DELETE, and PURGE).

  To create a new data set, use the CREATE statement to open a new data set for both output and input. The general form of the CREATE statement is

  CREATE *SAS-data-set* <VAR *operand*>;
  CREATE *SAS-data-set* FROM *from-name*
  <[COLNAME=*column-name* ROWNAME=*row-name*]>;

  Use the APPEND statement to place the matrix data into the newly created data set. If you don't use the APPEND statement, the new data set has no observations.

If you want to list observations and create matrices from the data in the SAS data set CLASS, you must first submit a statement to open CLASS. Because CLASS already exists, the USE statement is the one you want.

Making a SAS Data Set Current

IML data processing commands work on the current data set. This feature makes it unnecessary for you to specify the data set as an operand each time. There are two current data sets, one for input and one for output. IML makes a data set the current one as it is opened. You can also make a data set current by using two setting statements, SETIN and SETOUT:

- The USE and SETIN statements make a data set current for input.
- The SETOUT statement makes a data set current for output.
- The CREATE and EDIT statements make a data set current for both input and output.

If you issue a USE, EDIT, or CREATE statement for a data set that is already open, the data set is made the current data set. To find out which data sets are open and which are current input and current output data sets, use the SHOW DATASETS statement.

The current observation is set by the last operation that performed input/output (I/O). If you want to set the current observation without doing an I/O, use the SETIN (or SETOUT) statement with the POINT option. After a data set is opened, the current observation is set to 0. If you attempt to list or read the current observation, the current observation is converted to 1. You can make the data set CLASS current for input and position the pointer at the tenth observation with the statement

```
> setin class point 10;
```

Displaying SAS Data Set Information

You can use SHOW statements to display information about your SAS data sets. The SHOW DATASETS statement lists all open SAS data sets and their status. The SHOW CONTENTS statement displays the variable names and types, the size, and the number of observations in the current input data set. For example, to get data set information for CLASS, issue the following statements:

```
> use class;
> show datasets;

LIBNAME MEMNAME   OPEN MODE   STATUS
------- -------   ---------   ------
WORK    .CLASS    Input       Current Input

> show contents;

VAR NAME   TYPE   SIZE
NAME       CHAR    8
SEX        CHAR    8
AGE        NUM     8
HEIGHT     NUM     8
WEIGHT     NUM     8
Number of Variables:     5
Number of Observations: 19
```

As you can see, CLASS is the only data set open. The USE statement opened it for input, and it is the current input data set. The full name for CLASS is WORK.CLASS. The libref is the default, WORK. The next section tells you how to change the libref to another name.

Referring to a SAS Data Set

The USE, EDIT, and CREATE statements take as their first operand the data set name. This name can have either one or two levels. If it is a two-level name, the first level refers to the name of the SAS data library; the second name is the data set name. If the libref is WORK, the data set is put into a directory for temporary data sets; these are automatically deleted at the end of the session. Other librefs are associated with SAS data libraries using the LIBNAME statement.

If you specify only a single name, then IML supplies a default libref. At the beginning of an IML session, the default libref is SASUSER, if SASUSER is defined as a libref, or WORK otherwise. You can reset the default libref by using the RESET DEFLIB statement. If you want to create a permanent SAS data set, you must specify a two-level name using the RESET DEFLIB statement (see Chapter 6, "SAS Files," in *SAS Language: Reference, Version 6, First Edition* for more information about permanent SAS data sets).

```
> reset deflib=name;
```

Listing Observations

You can list variables and observations in a SAS data set with the LIST statement. The general form of the LIST statement is

LIST <*range*> <VAR *operand*> <WHERE(*expression*)>;

where

range	specifies a range of observations.
operand	selects a set of variables.
expression	is an expression that is evaluated for being true or false.

The next three sections discuss how to use each of these clauses with the CLASS data set.

Specifying a Range of Observations

You can specify a range of observations with a keyword or by record number using the POINT option. You can use the *range* operand with the data management statements DELETE, FIND, LIST, READ, and REPLACE.

You can specify *range* using any of the following keywords:

ALL	all observations
CURRENT	the current observation
NEXT <*number*>	the next observation or next *number* of observations

AFTER	all observations after the current one
POINT *operand*	observations by number, where *operand* can be one of the following:

Operand	Example
a single record number	`point 5`
a literal giving several record numbers	`point (2 5 10)`
the name of a matrix containing record numbers	`point p`
an expression in parentheses	`point (p+1)`

If you want to list all observations in the CLASS data set, use the keyword ALL to indicate that the range is all observations:

```
> list all;
```

OBS	NAME	SEX	AGE	HEIGHT	WEIGHT
1	JOYCE	F	11.0000	51.3000	50.5000
2	THOMAS	M	11.0000	57.5000	85.0000
3	JAMES	M	12.0000	57.3000	83.0000
4	JANE	F	12.0000	59.8000	84.5000
5	JOHN	M	12.0000	59.0000	99.5000
6	LOUISE	F	12.0000	56.3000	77.0000
7	ROBERT	M	12.0000	64.8000	128.0000
8	ALICE	F	13.0000	56.5000	84.0000
9	BARBARA	F	13.0000	65.3000	98.0000
10	JEFFREY	M	13.0000	62.5000	84.0000
11	CAROL	F	14.0000	62.8000	102.5000
12	HENRY	M	14.0000	63.5000	102.5000
13	ALFRED	M	14.0000	69.0000	112.5000
14	JUDY	F	14.0000	64.3000	90.0000
15	JANET	F	15.0000	62.5000	112.5000
16	MARY	F	15.0000	66.5000	112.0000
17	RONALD	M	15.0000	67.0000	133.0000
18	WILLIAM	M	15.0000	66.5000	112.0000
19	PHILIP	M	16.0000	72.0000	150.0000

Without a *range* specification, the LIST statement lists only the current observation, which in this example is now the last observation because of the previous LIST statement:

```
list;

   OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
    19 PHILIP   M          16.0000   72.0000  150.0000
```

Use the POINT keyword with record numbers to list specific observations. You can follow the keyword POINT with a single record number or literal giving several record numbers.

```
> list point 5;

   OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
     5 JOHN     M          12.0000   59.0000   99.5000

> list point {2 4 9};

   OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
     2 THOMAS   M          11.0000   57.5000   85.0000
     4 JANE     F          12.0000   59.8000   84.5000
     9 BARBARA  F          13.0000   65.3000   98.0000
```

You can also indicate the range indirectly by creating a matrix containing the records you want listed:

```
> p={2 4 9};
> list point p;

   OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
     2 THOMAS   M          11.0000   57.5000   85.0000
     4 JANE     F          12.0000   59.8000   84.5000
     9 BARBARA  F          13.0000   65.3000   98.0000
```

The *range* operand is usually listed first when you are using the access statements DELETE, FIND, LIST, READ, and REPLACE. Listed below are access statements and their default ranges.

Statement	Default Range
LIST	current
READ	current
FIND	all
REPLACE	current
APPEND	always at end
DELETE	current

Selecting a Set of Variables

You can use the VAR clause to select a set of variables. The general form of the VAR clause is

VAR *operand*

where *operand* can be specified using one of the following:

- a literal containing variable names
- the name of a matrix containing variable names
- an expression in parentheses yielding variable names
- one of the keywords listed below:

ALL	for all variables
CHAR	for all character variables
NUM	for all numeric variables.

Below are examples showing each possible way you can use the VAR clause:

```
var {time1 time5 time9};      /* a literal giving the variables */
var time;                      /* a matrix containing the names */
var('time1':'time9');                        /* an expression   */
var _all_;                                         /* a keyword */
```

For example, to list students' names from the CLASS data set, use the VAR clause with a literal:

```
> list point p var{name};

      OBS NAME
   ------ --------
        2 THOMAS
        4 JANE
        9 BARBARA
```

To list AGE, HEIGHT, and WEIGHT, you can use the VAR clause with a matrix giving the variables:

```
> v={age height weight};
> list point p var v;

      OBS       AGE    HEIGHT    WEIGHT
   ------ --------- --------- ---------
        2   11.0000   57.5000   85.0000
        4   12.0000   59.8000   84.5000
        9   13.0000   65.3000   98.0000
```

The VAR clause can be used with the following statements for the tasks described in the table below:

Statement	VAR Clause Function
APPEND	specifies which IML variables contain data to append to the data set
CREATE	specifies the variables to go in the data set
EDIT	limits which variables are accessed
LIST	specifies which variables to list
READ	specifies which variables to read
REPLACE	specifies which data set variable's data values to replace with corresponding IML variable data values
USE	limits which variables are accessed

Selecting Observations

The WHERE clause conditionally selects observations, within the *range* specification, according to conditions given in the *expression*. The general form of the WHERE clause is

WHERE(*variable comparison-op operand*)

where

variable is a variable in the SAS data set.

comparison-op is one of the following comparison operators:

<	less than
<=	less than or equal to
=	equal to
>	greater than
>=	greater than or equal to
^=	not equal to
?	contains a given string
^?	does not contain a given string
=:	begins with a given string
=*	sounds like or is spelled similar to a given string.

operand is a literal value, a matrix name, or an expression in parentheses.

WHERE comparison arguments can be matrices. For the following operators the WHERE clause succeeds if *all* the elements in the matrix satisfy the condition:

^= ^ ? < <= > >=

For the following operators, the WHERE clause succeeds if *any* of the elements in the matrix satisfy the condition:

= ? =: =*

Logical expressions can be specified within the WHERE clause using the AND (&) and OR (|) operators. The general form is

clause&*clause* (for an AND clause)
clause | *clause* (for an OR clause)

where *clause* can be a comparison, a parenthesized clause, or a logical expression clause that is evaluated using operator precedence.

For example, to list the names of all males in CLASS, use the following statement:

```
> list all var(name) where(sex='M');

                    OBS                NAME
                ------ ----------------------
                     2                THOMAS
                     3                JAMES
                     5                JOHN
                     7                ROBERT
                    10                JEFFREY
                    12                HENRY
                    13                ALFRED
                    17                RONALD
                    18                WILLIAM
                    19                PHILIP
```

The WHERE comparison arguments can be matrices. In the cases using the =* operator shown below, the comparison is made on each name to find a ring that sounds like or is spelled similar to the given string or strings:

```
> n=(name sex age);
> list all var n where(name=*("ALFRED","CAROL","JUDY"));

          OBS         NAME     SEX           AGE
        ----- --------------- -------- ---------
           11          CAROL    F          14.0000
           13          ALFRED   M          14.0000
           14          JUDY     F          14.0000

> list all var n where(name=*("JON","JAN"));

                OBS NAME     SEX           AGE
              ------ -------- -------- ---------
                   4 JANE     F          12.0000
                   5 JOHN     M          12.0000
```

To list AGE, HEIGHT, and WEIGHT for all students in their teens, use the following statement:

```
> list all var v where(age>12);
```

OBS	AGE	HEIGHT	WEIGHT
8	13.0000	56.5000	84.0000
9	13.0000	65.3000	98.0000
10	13.0000	62.5000	84.0000
11	14.0000	62.8000	102.5000
12	14.0000	63.5000	102.5000
13	14.0000	69.0000	112.5000
14	14.0000	64.3000	90.0000
15	15.0000	62.5000	112.5000
16	15.0000	66.5000	112.0000
17	15.0000	67.0000	133.0000
18	15.0000	66.5000	112.0000
19	16.0000	72.0000	150.0000

Note: In the WHERE clause, the expression on the left-hand side refers to values of the data set variables, and the expression on the right-hand side refers to matrix values. You cannot use comparisons involving more than one data set variable in a single comparison; for example, you cannot use either of the expressions

```
list all where(height>weight);
list all where(weight-height>0);
```

You could use the first statement if WEIGHT were a matrix name already defined rather than a variable in the SAS data set.

Reading Observations from a SAS Data Set

Transferring data from a SAS data set to a matrix is done using the READ statement. The SAS data set you want to read data from must already be open. You can open a SAS data set with either the USE or the EDIT statement. If you already have several data sets open, you can point to the one you want with the SETIN statement, making it the current input data set. The general form of the READ statement is

READ <*range*> <VAR *operand*> <WHERE(*expression*)> <INTO *name*>;

where

range	specifies a range of observations.
operand	selects a set of variables.
expression	is an expression that is evaluated for being a true or false.
name	names a target matrix for the data.

Using the READ Statement with the VAR Clause

Use the READ statement with the VAR clause to read variables from the current SAS data set into column vectors of the VAR clause. Each variable in the VAR clause becomes a column vector with the same name as the variable in the SAS data set. The number of rows is equal to the number of observations processed, depending on the range specification and the WHERE clause. For example, to read the numeric variables AGE, HEIGHT, and WEIGHT for all observations in CLASS, use the statements

```
> read all var {age height weight};
```

Now submit the SHOW NAMES statement to display all the matrices you have created so far in this chapter:

```
> show names;

        AGE          19 rows      1 col  num      8
        HEIGHT       19 rows      1 col  num      8
        N             1 row       3 cols char     4
        P             1 row       3 cols num      8
        V             1 row       3 cols char     6
        WEIGHT       19 rows      1 col  num      8
        Number of symbols = 8  (includes those without values)
```

You see that with the READ statement, you have created the three numeric vectors AGE, HEIGHT, and WEIGHT. (Notice that the matrices you created earlier, N, P, and V, are also listed.) You can select the variables you want to access with a VAR clause in the USE statement. The two previous statements can also be written

```
use class var{age height weight};
read all;
```

Using the READ Statement with the VAR and INTO Clauses

Sometimes you want to have all of the numeric variables in the same matrix so you can determine correlations. Use the READ statement with the INTO clause and the VAR clause to read the variables listed in the VAR clause into the single matrix named in the INTO clause. Each variable in the VAR clause becomes a column of the target matrix. If there are p variables in the VAR clause and n observations processed, the target matrix in the INTO clause is an $n \times p$ matrix.

The following statement creates a matrix X containing the numeric variables of the CLASS data set. Notice the use of the keyword _NUM_ in the VAR clause to specify all numeric variables be read.

```
> read all var _num_ into x;
> print x;
```

```
X
11      51.3      50.5
11      57.5        85
12      57.3        83
12      59.8      84.5
12        59      99.5
12      56.3        77
12      64.8       128
13      56.5        84
13      65.3        98
13      62.5        84
14      62.8     102.5
14      63.5     102.5
14        69     112.5
14      64.3        90
15      62.5     112.5
15      66.5       112
15        67       133
15      66.5       112
16        72       150
```

Using the READ Statement with the WHERE Clause

Use the WHERE clause as you did with the LIST statement, to conditionally select observations from within the specified range. If you want to create a matrix FEMALE containing the variables AGE, HEIGHT, and WEIGHT for females only, use the following statements:

```
> read all var _num_ into female where(sex="F");
> print female;
```

```
FEMALE
11      51.3      50.5
12      59.8      84.5
12      56.3        77
13      56.5        84
13      65.3        98
14      62.8     102.5
14      64.3        90
15      62.5     112.5
15      66.5       112
```

Now try some special features of the WHERE clause to find values that begin with certain characters (the =: operator) or contain certain strings (the ? operator). To create a matrix J containing the students whose names begin with J, use the following statements:

```
> read all var{name} into j where(name=:"J");
> print j;
```

```
JOYCE
JAMES
JANE
JOHN
JEFFREY
JUDY
JANET
```

Creating a matrix AL of children with names containing the string AL, use the statement

```
> read all var{name} into al where(name?"AL");
> print al;
```

```
AL
ALICE
ALFRED
RONALD
```

Editing a SAS Data Set

You can edit a SAS data set using the EDIT statement. You can update values of variables, mark observations for deletion, delete the marked observations, and save the changes you make. The general form of the EDIT statement is

EDIT *SAS-data-set* <VAR *operand*> <WHERE(*expression*)>;

where

SAS-data-set	names an existing SAS data set.
operand	selects a set of variables.
expression	is an expression that is evaluated for being true or false.

Updating Observations

Suppose you have updated data and want to change some values in the CLASS data set. For instance, suppose that the student named HENRY has had a birthday since the data were added to CLASS. You can

1. make the data set CLASS current for input and output
2. read the data
3. change the appropriate data value
4. replace the changed data in the data set.

First, submit an EDIT statement to make CLASS current for input and output. Then use the FIND statement, which finds observation numbers and stores them in a matrix, to find the observation number of the data for HENRY and store it in the matrix d.

```
> edit class;
> find all where(name=('HENRY')) into d;
> print d;

                              D
                             12
```

List the observation containing the data for HENRY.

```
> list point d;

   OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
    12 HENRY    M           14.0000   63.5000  102.5000
```

As you see, the observation number is 12. Now read the value for AGE into a matrix and update its value. Finally, replace the value in the CLASS data set and list the observation containing the data for HENRY again.

```
> age=15;
> replace;

                        1 observations replaced.

> list point 12;

   OBS NAME     SEX            AGE    HEIGHT    WEIGHT
------ -------- -------- --------- --------- ---------
    12 HENRY    M           15.0000   63.5000  102.5000
```

Deleting Observations

Use the DELETE statement to mark an observation to be deleted. The general form of the DELETE statement is

DELETE <*range*> <WHERE(*expression*)>;

where

range specifies a range of observations.

expression is an expression that is evaluated for being true or false.

Below are examples of valid uses of the DELETE statement.

Code	Action
`delete;`	deletes the current observation
`delete point 12;`	deletes observation 12
`delete all where (age>12);`	deletes all observations where AGE is greater than 12

If a file accumulates a number of observations marked as deleted, you can clean out these observations and renumber the remaining observations by using the PURGE statement.

Suppose that the student named John has moved and you want to update the CLASS data set. You can remove the observation using the EDIT and DELETE statements. First, find the observation number of the data for JOHN and store it in **d** using the FIND statement. Then submit a DELETE statement to mark the record for deletion. A deleted observation is still physically in the file and still has an observation number, but it is excluded from processing. The deleted observations appear as gaps when you list the file by observation number:

```
> find all where(name=('JOHN')) into d;
> print d;

                    D
                    5

> delete point d;

                observation deleted.
```

```
> list all;
```

OBS	NAME	SEX	AGE	HEIGHT	WEIGHT
1	JOYCE	F	11.0000	51.3000	50.5000
2	THOMAS	M	11.0000	57.5000	85.0000
3	JAMES	M	12.0000	57.3000	83.0000
4	JANE	F	12.0000	59.8000	84.5000
6	LOUISE	F	12.0000	56.3000	77.0000
7	ROBERT	M	12.0000	64.8000	128.0000
8	ALICE	F	13.0000	56.5000	84.0000
9	BARBARA	F	13.0000	65.3000	98.0000
10	JEFFREY	M	13.0000	62.5000	84.0000
11	CAROL	F	14.0000	62.8000	102.5000
12	HENRY	M	15.0000	63.5000	102.5000
13	ALFRED	M	14.0000	69.0000	112.5000
14	JUDY	F	14.0000	64.3000	90.0000
15	JANET	F	15.0000	62.5000	112.5000
16	MARY	F	15.0000	66.5000	112.0000
17	RONALD	M	15.0000	67.0000	133.0000
18	WILLIAM	M	15.0000	66.5000	112.0000
19	PHILIP	M	16.0000	72.0000	150.0000

Notice that there is a gap in the data where the deleted observation was (observation 5). To renumber the observations and close the gaps, submit the PURGE statement. Note that the PURGE statement deletes any indexes associated with a data set.

```
> purge;
```

Creating a SAS Data Set from a Matrix

SAS/IML software provides the ability to create a new SAS data set from a matrix. Use the CREATE and APPEND statements to create a SAS data set from a matrix, where the columns of the matrix become the data set variables and the rows of the matrix become the observations. Thus, an $n \times m$ matrix creates a SAS data set with m variables and n observations. The CREATE statement opens the new SAS data set for both input and output, and the APPEND statement writes to (outputs to) the data set.

Using the CREATE Statement with the FROM Option

You can create a SAS data set from a matrix using the CREATE statement with the FROM option. This form of the CREATE statement is

CREATE *SAS-data-set* FROM *matrix*
<[COLNAME=*column-name* ROWNAME=*row-name*]>;

where

SAS-data-set	names the new data set.
matrix	names the matrix containing the data.
column-name	names the variables in the data set.
row-name	adds a variable containing row titles to the data set.

Suppose you want to create a SAS data set named RATIO containing a variable with the height-to-weight ratios for each student. You first create a matrix containing the ratios from the matrices **HEIGHT** and **WEIGHT** that you have already defined. Next, use the CREATE and APPEND statements to open a new SAS data set called RATIO and append the observations, naming the data set variable HTWT instead of COL1.

```
htwt=height/weight;
create ratio from htwt[colname='htwt'];
append;
```

Now submit SHOW DATASETS and SHOW CONTENTS statements.

```
> show datasets;

  LIBNAME MEMNAME   OPEN MODE   STATUS
  ------- -------   ---------   ------
  WORK    .CLASS    Update
  WORK    .RATIO    Update      Current Input    Current Output

> show contents;

  VAR NAME   TYPE   SIZE
  HTWT       NUM      8
  Number of Variables:
  Number of Observations:

> close ratio;
```

As you can see, the new SAS data set RATIO has been created. It has 18 observations and 1 variable (recall that you deleted 1 observation earlier).

Using the CREATE Statement with the VAR Clause

You can use a VAR clause with the CREATE statement to select the variables you want to include in the new data set. In the previous example, the new data set RATIO had one variable. If you want to create a similar data set but include the second variable NAME, you use the VAR clause. You could not do this using the FROM option because the variable HTWT is numeric and NAME is character. The statements below create a new data set RATIO2 having the variables NAME and HTWT.

```
> create ratio2 var{name htwt};
> append;
> show contents;

            VAR NAME   TYPE   SIZE
            NAME       CHAR     8
            HTWT       NUM      8
            Number of Variables:    2
            Number of Observations: 18

> close ratio2;
```

Notice that now the variable NAME is in the data set.

Understanding the End-of-File Condition

If you try to read past the end of a data set or point to an observation greater than the number of observations in the data set, you create an end-of-file condition. If an end of file occurs while inside a DO DATA iteration group, IML transfers control to the next statement outside the current DO DATA group.

The example below uses a DO DATA loop while reading the CLASS data set. It reads the variable WEIGHT in one observation at a time and accumulates the weights of the students in the IML matrix SUM. When the data are read, the total class weight is stored in SUM.

```
setin class point 0;
sum=0;
do data;
   read next var{weight};
   sum=sum+weight;
end;
print sum;
```

Producing Summary Statistics

Summary statistics on the numeric variables of a SAS data set can be obtained with the SUMMARY statement. These statistics can be based on subgroups of the data by using the CLASS clause in the SUMMARY statement. The SAVE option in the OPT clause enables you to save the computed statistics in matrices for later perusal. For example, consider the statement below:

```
> summary var (height weight) class (sex) stat(mean std) opt(save);
```

SEX	Nobs	Variable	MEAN	STD
F	9	HEIGHT	60.58889	5.01833
		WEIGHT	90.11111	19.38391
M	9	HEIGHT	64.45556	4.90742
		WEIGHT	110.00000	23.84717
All	18	HEIGHT	62.52222	5.20978
		WEIGHT	100.05556	23.43382

This summary statement gives the mean and standard deviation of the variables HEIGHT and WEIGHT for the two subgroups (male and female) of the data set CLASS. Since the SAVE option is set, the statistics of the variables are stored in matrices under the name of the corresponding variables, with each column corresponding to a statistic requested and each row corresponding to a subgroup. Two other vectors, SEX and _NOBS_, are created. The vector SEX contains the two distinct values of the class variable SEX used in forming the two subgroups. The vector _NOBS_ has the number of observations in each subgroup defined by the above.

Note that the combined means and standard deviations of the two subgroups are displayed but are not saved.

More than one class variable can be used, in which case a subgroup is defined by the combination of the values of the class variables.

Sorting a SAS Data Set

The observations in a SAS data set can be ordered (sorted) by specific key variables. To sort a SAS data set, close the data set if it is currently open, and issue a SORT statement for the variables by which you want the observations to be ordered. Specify an output data set name if you want to keep the original data set. For example, the statement

```
> sort class out=sorted by name;
```

creates a new SAS data set named SORTED. The new data set has the observations from the data set CLASS, ordered by the variable NAME.

The statement

```
> sort class by name;
```

sorts in place the data set CLASS by the variable NAME. However, at the completion of the SORT statement, the original data set is replaced by the sorted data set.

You can specify as many key variables as needed, and, optionally, each variable can be preceded by the keyword DESCENDING, which denotes that the variable that follows is to be sorted in descending order.

Indexing a SAS Data Set

Searching through a large data set for information about one or more specific observations may take a long time because the procedure must read each record. You can reduce this search time by first indexing the data set by a variable. The INDEX statement builds a special companion file containing the values and record numbers of the indexed variables. Once the index is built, IML may use the index for queries with WHERE clauses if it decides that indexed retrieval is more efficient. Any number of variables can be indexed, but only one index is in use at a given time. Note that purging a data set with the PURGE statement results in the loss of all associated indexes.

Once you have indexed a data set, IML can use this index whenever a search is conducted with respect to the indexed variables. The indexes are updated automatically whenever you change values in indexed variables. When an index is in use, observations cannot be randomly accessed by their physical location numbers. This means that the POINT range cannot be used when an index is in effect. However, if you purge the observations marked for deletion, or sort the data set in place, the indexes become invalid and IML automatically deletes them.

For example, if you want a list of all female students in the CLASS data set, you can first index CLASS by the variable SEX. Then use the LIST statement with a WHERE clause. Of course, the CLASS data set is small and indexing does little if anything to speed queries with the WHERE clause. If the data set had thousands of students, though, indexing could save search time.

To index the data set by the variable SEX, submit the statement

```
> index sex;

  NOTE: Variable SEX indexed.
  NOTE: Retrieval by SEX.
```

Now list all students. Notice the ordering of the special file built by indexing by the variable SEX. Retrievals by SEX will be quick.

```
> list all;
```

OBS	NAME	SEX	AGE	HEIGHT	WEIGHT
1	JOYCE	F	11.0000	51.3000	50.5000
4	JANE	F	12.0000	59.8000	84.5000
6	LOUISE	F	12.0000	56.3000	77.0000
8	ALICE	F	13.0000	56.5000	84.0000
9	BARBARA	F	13.0000	65.3000	98.0000
11	CAROL	F	14.0000	62.8000	102.5000
14	JUDY	F	14.0000	64.3000	90.0000
15	JANET	F	15.0000	62.5000	112.5000
16	MARY	F	15.0000	66.5000	112.0000
2	THOMAS	M	11.0000	57.5000	85.0000
3	JAMES	M	12.0000	57.3000	83.0000
7	ROBERT	M	12.0000	64.8000	128.0000
10	JEFFREY	M	13.0000	62.5000	84.0000
12	HENRY	M	15.0000	63.5000	102.5000
13	ALFRED	M	14.0000	69.0000	112.5000
17	RONALD	M	15.0000	67.0000	133.0000
18	WILLIAM	M	15.0000	66.5000	112.0000
19	PHILIP	M	16.0000	72.0000	150.0000

Data Set Maintenance Functions

Two functions and two subroutines are provided to perform data set maintenance:

DATASETS function
: obtains members in a data library. This function returns a character matrix containing the names of the SAS data sets in a library.

CONTENTS function
: obtains variables in a member. This function returns a character matrix containing the variable names for the SAS data set specified by *libname* and *memname*. The variable list is returned in alphabetic order.

RENAME subroutine
: renames a SAS data set member in a specified library.

DELETE subroutine
: deletes a SAS data set member in a specified library.

See Chapter 15 for details and examples of these functions and routines.

Overview of Commands

You have seen that IML has an extensive set of commands that operate on SAS data sets. The table below summarizes the data management commands you can use to perform management tasks for which you might normally use the SAS DATA step.

Table 6.1
Data Management Commands

Command	Action
APPEND	adds observations to the end of a SAS data set
CLOSE	closes a SAS data set
CREATE	creates and opens a new SAS data set for input and output
DELETE	marks observations for deletion in a SAS data set
EDIT	opens an existing SAS data set for input and output
FIND	finds observations
INDEX	indexes variables in a SAS data set
LIST	lists observations
PURGE	purges all deleted observations from a SAS data set
READ	reads observations into IML variables
REPLACE	writes observations back into a SAS data set
RESET DEFLIB	names default libname
SAVE	saves changes and reopens a SAS data set
SETIN	selects an open SAS data set for input
SETOUT	selects an open SAS data set for output
SHOW CONTENTS	shows contents of the current input SAS data set
SHOW DATASETS	shows SAS data sets currently open
SORT	sorts a SAS data set
SUMMARY	produces summary statistics for numeric variables
USE	opens an existing SAS data set for input

Similarities and Differences with the SAS DATA Step

If you want to remain in the IML environment and mimic DATA step processing, you need to learn the basic differences between IML and the DATA step:

- With SAS/IML software, you start with a CREATE statement instead of a DATA statement. You must explicitly set up all your variables with the right attributes before you create a data set. This means you must define character variables having the desired string length beforehand. Numeric variables are the default, so that any variable not defined as character is assumed to be numeric. In the DATA step, the variable attributes are determined from context across the whole step.
- With SAS/IML software, you must use an APPEND statement to output an observation; in the DATA step you either use an OUTPUT statement or let the DATA step output it automatically.
- With SAS/IML software, you iterate with a DO DATA loop. In the DATA step, the iterations are implied.
- With SAS/IML software, you have to close the data set with a CLOSE statement unless you plan to leave the IML environment with a QUIT statement. The DATA step closes the data set automatically at the end of the step.
- The DATA step usually executes faster than IML.

In short, the DATA step treats the problem with greater simplicity, allowing shorter programs. However, IML has more flexibility because it is both interactive and has a powerful matrix-handling capability.

In Conclusion

In this chapter, you have learned many ways to interact with SAS data sets from within the IML environment. You learned how to open and close a SAS data set, how to make it current for input and output, how to list observations by specifying a range of observations to process, a set of variables to use, and a condition for subsetting observations. You also learned summary statistics. You also know how to read observations and variables from a SAS data set into matrices as well as create a SAS data set from a matrix of values.

The results are shown below:

```
Solving the system: X1+X2-X1*X2+2=0, X1*EXP(-X2)-1=0

                    X          F
            0.0977731 5.3523E-9
            -2.325106 6.1501E-8
```

Example 3: Regression

This example shows a regression module that calculates statistics not calculated by the two previous examples:

```
/*          Regression Routine                  */
/* Given X, and Y, this fits Y = X B + E        */
/* by least squares.                            */

start reg;
   n=nrow(x);                          /* number of observations */
   k=ncol(x);                          /* number of variables */
   xpx=x`*x;                           /* cross-products */
   xpy=x`*y;
   xpxi=inv(xpx);                      /* inverse crossproducts */
   b=xpxi*xpy;                         /* parameter estimates */
   yhat=x*b;                           /* predicted values */
   resid=y-yhat;                       /* residuals */
   sse=resid`*resid;                   /* sum of squared errors */
   dfe=n-k;                            /* degrees of freedom error */
   mse=sse/dfe;                        /* mean squared error */
   rmse=sqrt(mse);                     /* root mean squared error */
   covb=xpxi#mse;                      /* covariance of estimates */
   stdb=sqrt(vecdiag(covb));           /* standard errors */
   t=b/stdb;                           /* ttest for estimates=0 */
   probt=1-probf(t#t,1,dfe);           /* significance probability */
   print name b stdb t probt;
   s=diag(1/stdb);
   corrb=s*covb*s;                     /* correlation of estimates */
   print ,"Covariance of Estimates", covb[r=name c=name] ,
         "Correlation of Estimates",corrb[r=name c=name] ;
```

```
   if nrow(tval)=0 then return;          /* is a t-value specified? */
   projx=x*xpxi*x`;                                     /* hat matrix */
   vresid=(I(n)-projx)*mse;              /* covariance of residuals */
   vpred=projx#mse;              /* covariance of predicted values  */
   h=vecdiag(projx);                          /* hat leverage values */
   lowerm=yhat-tval#sqrt(h*mse); /* lower confidence limit for mean */
   upperm=yhat+tval#sqrt(h*mse);            /* upper limit for mean */
   lower=yhat-tval#sqrt(h*mse+mse);       /* lower limit for indiv  */
   upper=yhat+tval#sqrt(h*mse+mse);       /* upper limit for indiv  */
   print ,,"Predicted Values, Residuals, and Limits"
   y yhat resid h lowerm upperm lower upper;
finish reg;

   /* Routine to test a linear combination of the estimates  */
   /* given L, this routine tests hypothesis that LB = 0.    */

start test;
   dfn=nrow(L);
   Lb=L*b;
   vLb=L*xpxi*L`;
   q=Lb`*inv(vLb)*Lb /dfn;
   f=q/mse;
   prob=1-probf(f,dfn,dfe);
   print ,f dfn dfe prob;
finish test;

   /* Run it on population of U.S. for decades beginning 1790 */

x= { 1 1 1,
     1 2 4,
     1 3 9,
     1 4 16,
     1 5 25,
     1 6 36,
     1 7 49,
     1 8 64 };

y= {3.929,5.308,7.239,9.638,12.866,17.069,23.191,31.443};
name={"Intercept", "Decade", "Decade**2" };
tval=2.57; /* for 5 df at .025 level to get 95% confidence interval */
reset fw=7;
run reg;
do;
   print ,"TEST Coef for Linear";
   L={0 1 0 };
   run test;
   print ,"TEST Coef for Linear,Quad";
   L={0 1 0,0 0 1};
   run test;
   print ,"TEST Linear+Quad = 0";
   L={0 1 1 };
   run test;
end;
```

The results are shown below:

NAME	B	STDB	T	PROBT
Intercept	5.06934	0.96559	5.24997	0.00333
Decade	-1.1099	0.4923	-2.2546	0.07385
Decade**2	0.53964	0.0534	10.106	0.00016

Covariance of Estimates

COVB	Intercept	Decade	Decade**2
Intercept	0.93237	-0.4362	0.04277
Decade	-0.4362	0.24236	-0.0257
Decade**2	0.04277	-0.0257	0.00285

Correlation of Estimates

CORRB	Intercept	Decade	Decade**2
Intercept	1	-0.9177	0.8295
Decade	-0.9177	1	-0.9762
Decade**2	0.8295	-0.9762	

Predicted Values, Residuals, and Limits

Y	YHAT	RESID	H	LOWERM	UPPERM	LOWER	UPPER
3.929	4.49904	-0.57	0.70833	3.00202	5.99606	2.17419	6.82389
5.308	5.00802	0.29998	0.27976	4.06721	5.94883	2.99581	7.02023
7.239	6.59627	0.64273	0.23214	5.73926	7.45328	4.62185	8.57069
9.638	9.26379	0.37421	0.27976	8.32298	10.2046	7.25158	11.276
12.866	13.0106	-0.1446	0.27976	12.0698	13.9514	10.9984	15.0228
17.069	17.8367	-0.7677	0.23214	16.9797	18.6937	15.8622	19.8111
23.191	23.742	-0.551	0.27976	22.8012	24.6828	21.7298	25.7542
31.443	30.7266	0.71637	0.70833	29.2296	32.2236	28.4018	33.0515

```
TEST Coef for Linear

      F      DFN      DFE     PROB
5.08317        1        5  0.07385

TEST Coef for Linear,Quad

      F      DFN      DFE     PROB
666.511        2        5  8.54E-7

TEST Linear+Quad = 0

      F      DFN      DFE     PROB
1.67746        1        5  0.25184
```

Example 4: Alpha Factor Analysis

This example shows how an algorithm for computing alpha factor patterns (Kaiser and Caffrey 1965) is transcribed into IML code.

For later reference you could store the ALPHA subroutine created below in an IML catalog and load it when needed.

```
/*                    Alpha Factor Analysis                          */
/*  Ref: Kaiser et al., 1965 Psychometrika, pp. 12-13                */
/*  r correlation matrix (n.s.) already set up                       */
/*  p number of variables                                            */
/*  q number of factors                                              */
/*  h communalities                                                  */
/*  m eigenvalues                                                    */
/*  e eigenvectors                                                   */
/*  f factor pattern                                                 */
/*  (...,H2,HI,G,HH) temporary use. freed up                         */
/*
```

ENUMERATOR INSTRUCTIONS

for

PAKISTAN PILOT WHEAT VALIDATION SURVEY

by

THOMAS R. BIRKETT

ROBERT L. ADDISON

ENUMERATOR INSTRUCTIONS

for

PAKISTAN PILOT WHEAT VALIDATION SURVEY

THOMAS R. BIRKETT

ROBERT L. ADDISON

I. Objective of the validation survey

The objective of this pilot "Wheat Validation Survey" is to determine the <u>average farmer realized per hectare wheat yield for Pakistan.</u>

The wheat yield indicated by the Area Sampling Frame Objective Yield (ASF OY) survey is essentially a biological yield and does not take into account normal losses incurred by the farmer during the harvesting, drying, and threshing of his crop. This ASF yield is naturally higher than the yield realized by the average farmer. A measure of the average difference between yield from the ASF survey and the farmer realized yield is needed so a statistically sound adjustment factor can be determined.

The "Wheat Validation Survey" will make possible the computation of this adjustment factor.

II. What is a wheat validation survey?

A wheat validation survey is a survey in which all wheat is harvested from a probability sample of wheat field, threshed, and weighed so that exact farmer realized yield can be computed.

III. Your primary responsibilities as an enumerator

Each farmer you visit during this survey may have several wheat fields. The sample field that has been selected for this validation survey will contain one or more OY plots. In addition to doing your regular OY work in this field, you will also determine the exact amount of wheat that is harvested by the farmer from this validation sample field.

You have three primary responsibilities which follow:

(1) Be sure the farmer does not get the wheat from the validation field mixed up with wheat from his other fields during the harvesting and threshing operations. To do this, you will need to be at the validation field during the <u>entire</u> time the field is being harvested and threshed.

(2) You must ensure that the wheat field is harvested and threshed exactly the way the farmer normally does it (mechanical or manual).

Example: The farmer normally threshes his wheat manually but does not cover the ground at the threshing site with a cloth. However since you are doing a special survey, he offers to cover the ground with a large cloth so no grain will be lost. You should reject this offer.

If the farmer does not follow his usual harvesting and threshing procedures, the yield computed from the validation field will not represent the farmer's normal yield.

(3) You must personally weigh all wheat harvested from the validation field if the field produces 25 or fewer bags. If it produces more than 25 bags, you will weigh the first 25 and only every 50th bag thereafter.

Do not allow the farmer to weigh his own grain.

IV. Materials

You will be provided the following materials:

1. A questionnaire for each validation sample field which contains an appropriate label.

2. Large cloth bags to use when bagging the grain from the validation field for weighing.

3. String for tying the bag after they have been filled with grain.

4. A scale for weighing the grain from the validation field.

5. Small cloth bags for taking grain samples which will be mailed to the Islamabad lab for moisture tests.

6. Strings for tying the small cloth bags.

Follow these steps to complete the validation survey

Farmer Visit

1. When laying out the OY sample in fields that will also be used in the validation survey, obtain the farmer's cooperation for the validation survey, explain the purpose of the survey, and ask the farmer when harvest

will take place. Enter this date on your questionnaire.

2. Ask the farmer if you can observe the harvesting and threshing operations related to the validation sample field--and explain why you need to do this.

3. Make clear to the farmer that you would like to personally weigh the grain coming from the sample field and that all grain from this field must be kept separate from grain in other fields through the harvesting, threshing, and weighing operations.

Harvest

4. When harvest of the field starts, you must first do your OY work.

5. Observe harvesting and threshing of the validation field to be sure all wheat in that field is kept separate from wheat in all other fields.

Note: If the wheat is manually threshed, it may take several days for threshing of the validation field to be completed. In this case, you should ask the farmer to bag the grain at the end of each day and hold all bags

from that validation field until you (the enumerator) can return to weigh them and take samples for moisture determination. You may have to leave some large cloth bags for his use.

6. As a wheat from the validation field is threshed and bagged, weigh each of the first 25 bags and record the weight on the form provided. Then weigh only every 50th bag thereafter, and also record these weights on the form.

   Example: Weigh the first 25 bags, then weigh the 75th, 125th bags, etc.

   Note: If the farmer does not have appropriate bags for weighing grain, put the threshed grain in the large cloth bags that have been provided to you so they can properly weighed.

7. Take a small sample of wheat (about 1/2 tea cup) from the 25th and every 50th sack of grain thereafter (Example: Take samples from bag 25, 75, 125, etc.) and place it in the small cloth bags provided to you. These bags should be properly labeled and mailed to the Islamabad laboratory for determining moisture content of the

wheat. If there are less than 25 bags of grain in the field, no sample will be taken for moisture determination.

<u>Note:</u> Each 1/2 tea cup should be put in a separate bag.

8. If you need additional pages for recording weight of the bags, use the supplemental sheets provided for that purpose.

9. After all of the grain has been harvested, threshed, and weighed, you have completed your validation survey task.

   Thank the farmer for his cooperation.

10. Return to the validation field (within 3 days) and lay out and harvest the OY post harvest plots.

    <u>Note:</u> If possible, this should be done immediately after weighing of the grain from the validation field to (a) save time, and (b) reduce expenses. That way, another trip to the field will not be necessary.

12 July 1990

1990 Pakistan Wheat Plot Size Study: An Investigation into How Yields Vary Across Pakistan Wheat Fields

by

Thomas R. Birkett
Robert L. Addison, Jr.
Agricultural Data Collection Project

(See p. 10 for conclusions of wheat optimal plot size study)

12 July 1990

1990 Pakistan Wheat Plot Size Study: An Investigation into How Yields Vary Across Pakistan Wheat Fields

by

Thomas R. Birkett
Robert L. Addison, Jr
Agricultural Data Collection Project

I. Purpose of the study

The current system of collecting Pakistan's agricultural statistics is based on a Village Master Sample (VMS) and is criticized on grounds that it does not fully meet the requirement of planners and policy makers as the crop estimates are not available on a timely basis, and in addition, their reliability is not known. The US aided project, named Agricultural Data Collection (ADC), was, therefore, started in 1985 with the main objective of improving Pakistan's agricultural statistics.

The surveys under the ADC project, which are based on Area Sampling Frame (ASF) tehnology , were started in 1989 in seven districts: Shiekhupura, Faisalabad, Jhang, Multan, Nawabshah, Hyderabad and Larkana. The objective yield per hectare indications, based on ASF survey for the year 1988-89, were higher than the VMS survey for the same period.

One of the reason for this difference was thought to be the small size of the plot (1.8' X 1.8') used for the wheat objective yield survey under the ADC project as compared to the VMS plot of 15 X 20 feet being used for the same purpose. To investigate whether plot size was the cause of the difference, a research project was designed to study the statistical efficiency of plot sizes 1.8' X 1.8' and 15' X 20' and several sizes in between these two.

The factor that determines optimal plot size is how yields vary across the Pakistan wheat fields. As the variability increases, the optimal plot size must necessarily increase. If wheat yields are more uniform, then the plot size can decrease.

II. Design of the study

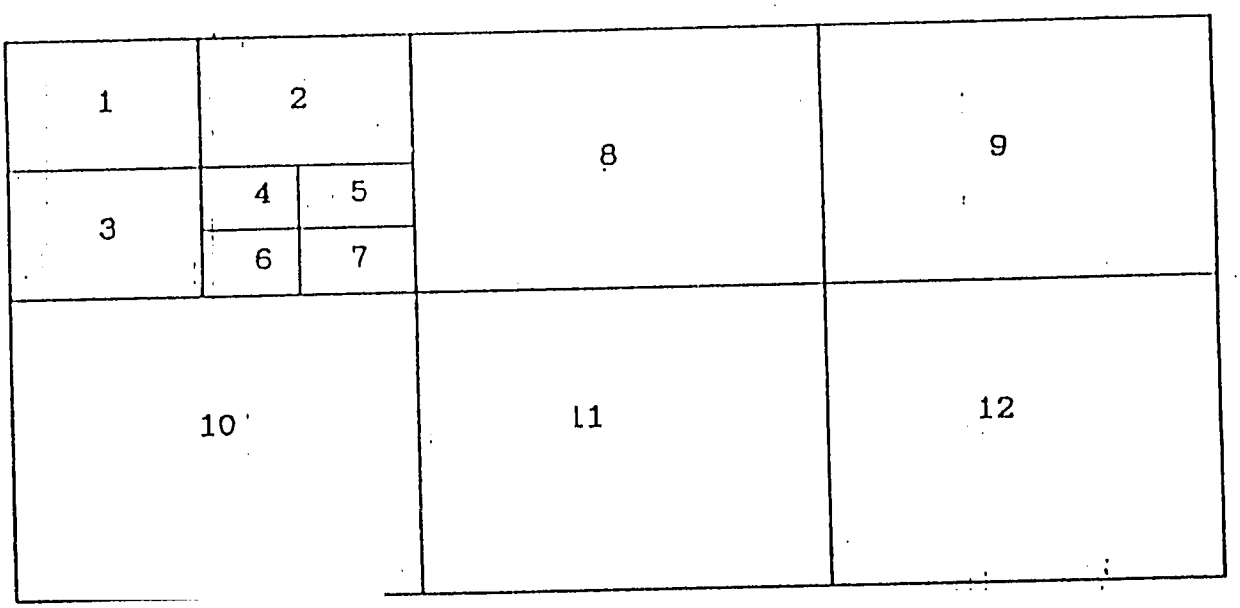
A 14.4 x 21.6 foot research plot was partitioned into 12 units as in Figure 1. Partitions 4-7 are each the size of the objective yield survey plot, 1.8 x 1.8 ft. (A 14.4 x 21.6 foot plot was used rather than the 15 x 20 foot plot mentioned above so units 4 to 7 would be exactly 1.8 feet square). Partitions 1-3 are 3.6 ft sq, and partitions 8-12 are 7.2 ft. sq. The sum of all the partitions slightly exceeds the area of the VMS plot. By harvesting and threshing each of the partitions separately, the researchers studied the variability in the ASF objective yield plot size and the VMS plot size and several sizes in between.

After the wheat from each ASF partition was harvested, it was run through plot threshing machines twice to be sure that all grains were removed from the heads. All chaff was removed from the grain and it was weighed using electronic scales. The grain from each partition was also tested for moisture using electronic moisture meters. All data from the ASF research project shown in this paper was adjusted to a 10 percent moisture basis. Wheat from the VMS surveys is normally allowed to sun dry before threshing and weighing but there is considerable variation in the procedures used. Since there is no data on the moisture content of the grain, this paper assumes that it approximates 10 percent.

A random sample of size 140 was drawn for the research project. The distribution of completed samples by district and province is given in Table 1. The sample was designed to produce statistically defensible estimates at each of the province levels. All district and province data were weighted by district hectors to ensure unbiased results. Inside of each district, the sample was random.

FIGURE 1.

14.4' X 21.6' Partition



TABLE

istrict	1988-89 Hectors of Wheat	Percent	Sample Distribution	Samples Completed
salabad	255,453	21.57	15	15
ng	349,666	29.52	21	20
tan	325,270	27.47	19	19
ikhupura	23,976	21.44	15	15
	---	---	---	---
unjab Province	1,184,365	100.00	70	69
abshah	193,253	53.04	25	25
kana	79,257	21.75	21	20
lerabad	91,858	25.21	24	19
	---	---	---	---
indh Provinvce	364,368	100.00	70	64
tal, ALL	1,548,733		140	133

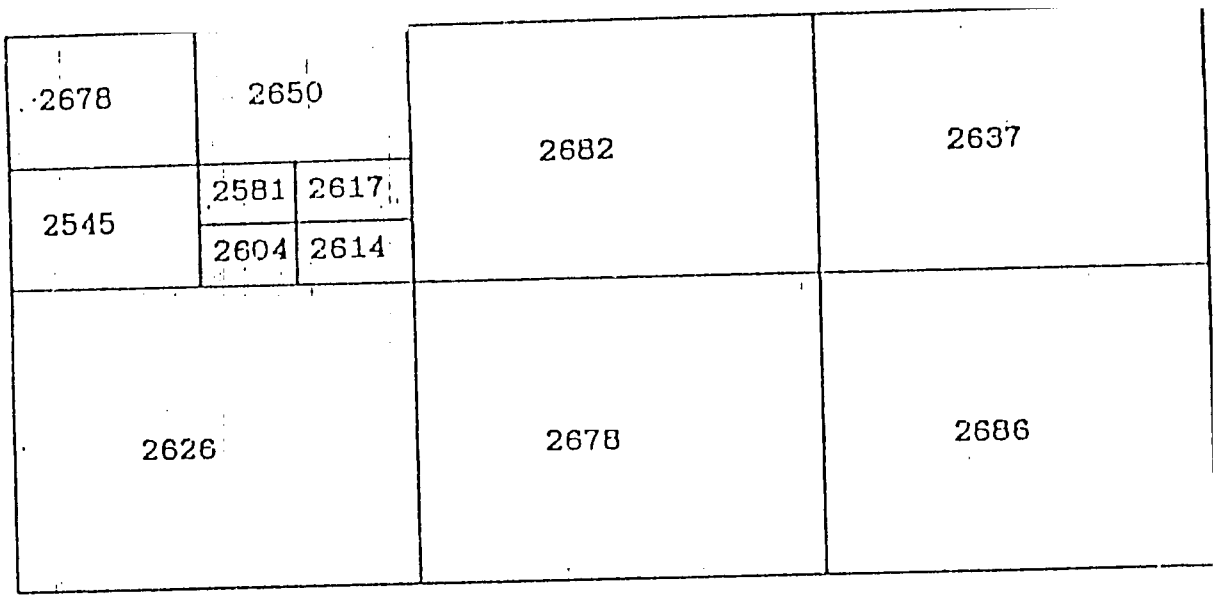
Mean Yields for each Partition and Selected Combinations of Interest

The mean yield for each partition is presented in Figure 2 for the two provinces combined (National) and for each province separately. That is followed by the mean yield for combinations of partitions that are of interest in Table 2. Since each partition and combination of partitions constitutes a random sample in its own right, from the same population, each mean has the same expected value. This is why we see that the means are generally quite close to each other. At the National level, the means range from 2545 for partition 3 to 2686 for partition 12 (different by less than 6 percent).

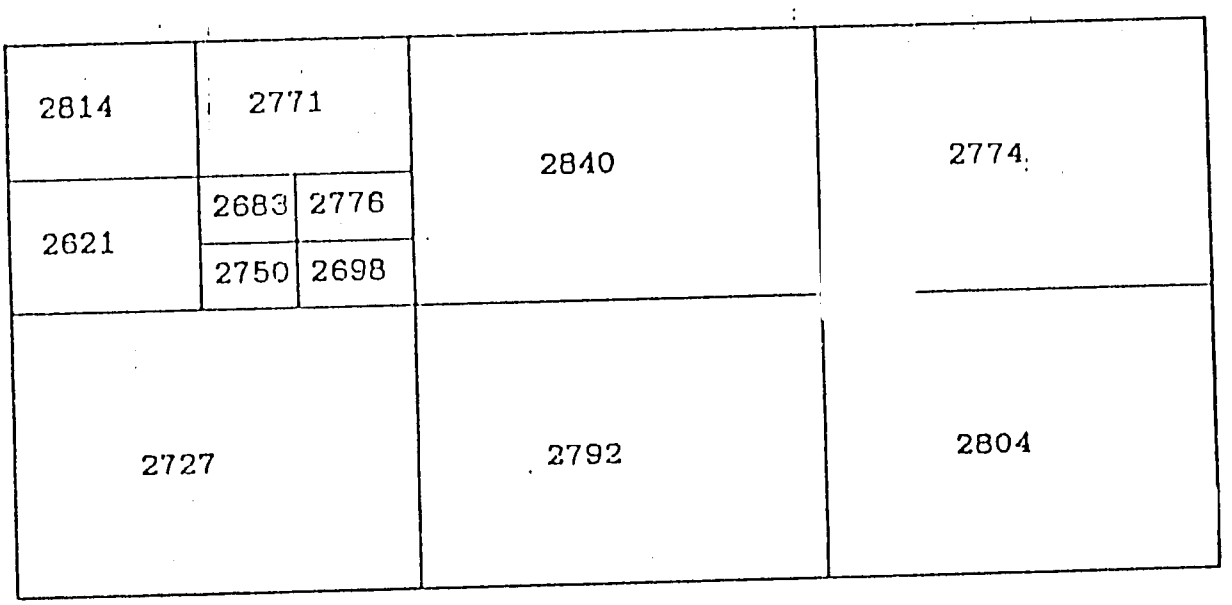
Also included in Table 2 are the results from the VMS survey in Punjab Province. The mean yield from all combinations (2778.54) is 32% larger than the corresponding VMS mean yield. The VMS mean is based on 1074 15x20 foot plots. VMS data for the Sindh Province was not available when this paper was written.

FIGURE 2:

NATIONAL



PUNJAB



SINDH

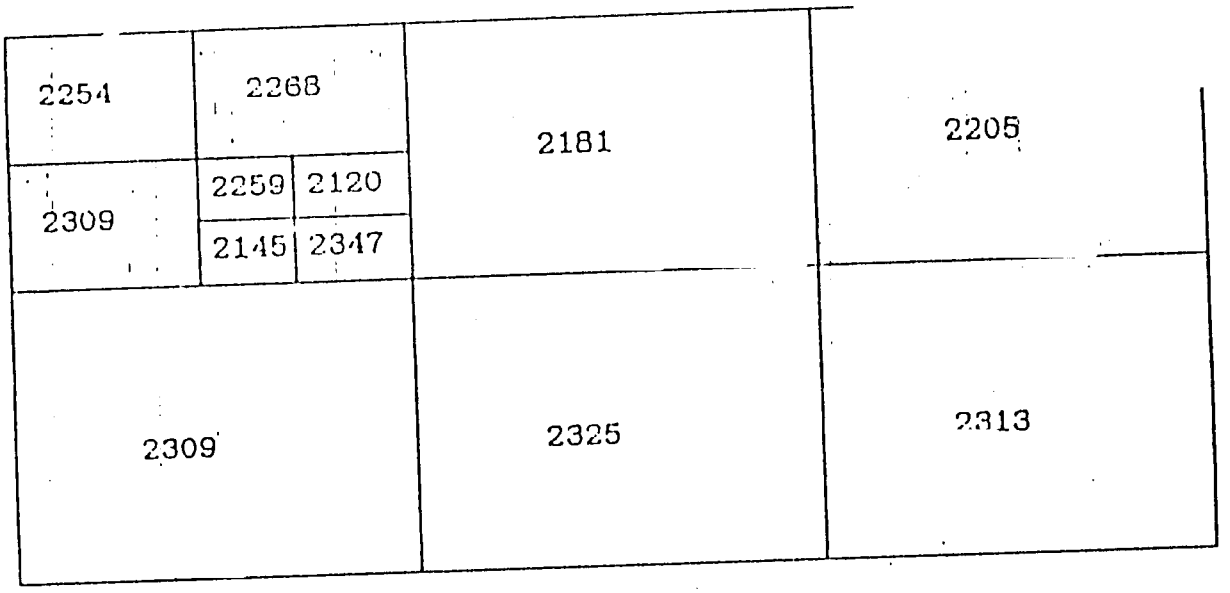


TABLE 2.

Combinations	Mean Yield All	Punjab	Sindh
4	2580.36	2682.62	2259.31
4-5	2598.82	2729.29	2189.24
4-6	2600.41	2736.09	2174.52
4-7	2603.77	2726.77	2217.60
4-7,2	2626.83	2749.10	2243.08
4-7,2-3	2599.81	2706.41	2265.16
4-7,1-3	2619.47	2733.24	2262.40
4-7,1-3,8	2650.42	2786.30	2221.71
4-7,1-3,8-9	2645.97	2782.89	2216.22
4-7,1-3,8-10	2640.89	2768.80	2239.42
4-7,1-3,8-11	2648.52	2773.38	2256.61
4-7,1-3,8-12	2654.72	2778.54	2266.04
VMS		2096.33	

Estimated Variances of Selected Combinations of Partitions of Interest

The estimated variance of each combination is a linear combination of the variances and covariances of the fields for each partition (see Appendix A for the formulas for the variances of the linear combinations). In particular, the variances will not decrease significantly as more partitions are added if there is high correlation between the partitions in the research

plot. The correlation for the National and each province is presented in Table 3. In general, the correlations are high (positive), with many greater than 0.85. The correlations decrease as the partitions become farther apart, while adjacent partitions have the highest correlations. Y1 and Y2, and Y11 and Y12 have a 0.9: correlation, while Y8 and Y11, and Y10 and Y11 have correlations of 0.90. The lowest correlations are between Y3, Y4 and Y12, because they are farthest apar physically, and also Y3 and Y4 are smaller plots and mor variable.

GURE 3

CORRELATION MATRICES

NATIONAL

titions	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11
Y1	1.00										
Y2	0.91	1.00									
Y3	0.87	0.84	1.00								
Y4	0.80	0.81	0.84	1.00							
Y5	0.74	0.83	0.77	0.76	1.00						
Y6	0.79	0.79	0.77	0.74	0.75	1.00					
Y7	0.80	0.82	0.80	0.72	0.73	0.79	1.00				
Y8	0.83	0.87	0.78	0.78	0.81	0.80	0.78	1.00			
Y9	0.76	0.79	0.71	0.70	0.72	0.74	0.73	0.90	1.00		
Y10	0.83	0.82	0.81	0.77	0.74	0.80	0.80	0.85	0.84	1.00	
Y11	0.76	0.79	0.75	0.71	0.76	0.75	0.76	0.90	0.87	0.90	1.00
Y12	0.70	0.74	0.68	0.67	0.70	0.71	0.73	0.84	0.89	0.85	0.91

PUNJAB

artitions	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11
Y1	1.00										
Y2	0.91	1.00									
Y3	0.88	0.84	1.00								
Y4	0.81	0.80	0.85	1.00							
Y5	0.73	0.83	0.82	0.76	1.00						
Y6	0.84	0.81	0.85	0.77	0.78	1.00					
Y7	0.80	0.82	0.81	0.72	0.75	0.82	1.00				
Y8	0.88	0.90	0.83	0.81	0.83	0.85	0.79	1.00			
Y9	0.79	0.82	0.77	0.73	0.75	0.79	0.74	0.92	1.00		
Y1	0.87	0.82	0.89	0.80	0.74	0.84	0.79	0.86	0.84	1.00	
Y1	0.80	0.82	0.83	0.74	0.79	0.79	0.77	0.91	0.90	0.90	1.00
Y1	0.76	0.79	0.76	0.72	0.73	0.77	0.74	0.85	0.91	0.86	0.90

SINDH

titions	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11
Y1	1:00										
Y2	0.90	1.00									
Y3	0.85	0.82	1.00								
Y4	0.76	0.80	0.81	1.00							
Y5	0.71	0.81	0.70	0.73	1.00						
Y6	0.67	0.73	0.65	0.67	0.64	1.00					
Y7	0.78	0.80	0.78	0.71	0.71	0.75	1.00				
Y8	0.75	0.81	0.71	0.71	0.76	0.71	0.75	1.00			
Y9	0.69	0.72	0.62	0.64	0.63	0.65	0.71	0.85	1.00		
Y10	0.76	0.81	0.72	0.70	0.71	0.72	0.80	0.81	0.82	1.00	
Y11	0.67	0.73	0.65	0.63	0.69	0.65	0.73	0.88	0.83	0.89	1.00
Y12	0.61	0.67	0.59	0.59	0.64	0.61	0.71	0.81	0.86	0.83	0.91

Using the variances and covariances, as in Appendix A, the estimated standard error of the means (based on the sample sizes in Table 1), of the plot combinations of interest, were calculated and are presented in Table 4.

TABLE 4.

Plot Combinations	Standard Error of Punjab	Sindh	NATIONAL
4	149.135	138.046	103.567
4-5	144.961	119.158	97.417
4-6	144.708	116.399	96.811
4-7	140.875	120.621	95.960
4-7,2	142.833	126.436	98.458
4-7,2-3	137.545	125.154	95.275
4-7,1-3	138.530	127.197	96.556
4-7,1-3,8	134.685	121.169	94.419
4-7,1-3,8-9	130.144	121.743	92.914
4-7,1-3,8-10	127.928	121.493	91.732
4-7,1-3,8-11	127.753	120.597	91.324
4-7,1-3,8-12	126.493	122.449	91.413
VMS	27.740		

Plot Combinations	Faisalabad	Jhang	Multan	Shiekhupura	PUNJAB
4	327.199	246.551	273.534	310.042	149.135
4-5	310.224	250.777	225.660	336.331	144.961
4-6	328.347	249.723	217.931	318.782	144.708
4-7	327.819	236.391	207.072	316.668	140.875
4-7,2	328.924	208.388	219.559	327.284	142.833
4-7,2-3	317.712	204.872	205.841	311.368	137.545
4-7,1-3	323.575	199.101	212.390	307.446	138.530
4-7,1-3,8	302.361	203.388	206.353	304.905	134.685
4-7,1-3,8-9	293.047	196.394	191.116	295.149	130.144
4-7,1-3,8-10	285.864	192.223	185.655	292.592	127.928
4-7,1-3,8-11	288.316	193.577	179.062	299.803	127.753
4-7,1-3,8-12	286.637	194.842	178.663	292.884	126.493

Plot Combinations	Nawabshah	Larkana	Hyderabad	SINDH
4	213.287	168.675	298.938	138.046
4-5	193.834	152.720	241.677	119.158
4-6	205.527	147.838	217.317	116.399
4-7	213.664	133.480	234.890	120.621
4-7,2	225.532	155.051	237.802	126.436
4-7,2-3	224.363	147.199	243.960	125.154
4-7,1-3	228.004	150.425	249.426	127.197
4-7,1-3,8	235.149	131.867	201.429	121.169
4-7,1-3,8-9	231.767	142.666	207.390	121.743
4-7,1-3,8-10	228.650	132.981	209.370	121.493
4-7,1-3,8-11	228.463	131.237	198.087	120.597
4-7,1-3,8-12	226.207	131.923	208.635	122.449

Referring back to Figure 2, if we take the mean yield from plot 4 (2581) plus or minus two standard errors (2 x 103.57), we get an interval that includes the means from all partitions (2374, 2788). None are significantly different, as we would expect.

The decrease in the standard error as partitions are added to the estimator is very insignificant when compared to the extra amount of work required. Adding partition 5 to partition 4, a doubling of the plot size and work, lowers the standard error by only 6% (103.570 to 97.42). Inclusion of the entire plot, which increases

the work load by a factor of 95, lowers the standard error by 12% (103.57 to 91.41). Clearly, it is inefficient to sample heavily in any one particular location in the field.

The VMS mean has a smaller standard error than the research data, as the result of a much larger sample size. (The variance of the VMS was calculated treating all plots as a simple random sample. There were 1074 plots.) The standard deviation for the VMS data under this model was 909, compared to 1050 for the research data in the Punjab Province. With the means (2,096.33 and 2,778.54) 5 to 6 standard errors apart and the t-test for the difference in the two means highly significant at a t value of 5.272, we see that the research data is quite different from the VMS data. It appears that the two surveys are estimating two distinctly different things.

The CV of partition 4, based on 133 samples, is 4.0% (see the mean of 2581, in Figure 2). The smallest plot size, at this sample size at the National level, provides this desirable level of accuracy. The Punjab CV for this same plot size is 5.6 based on 69 samples, while the Sindh CV is 6.1 based on 64 samples.

If we always chose the estimator with the smallest variance, with no regard for the work required, we would have a plot the size of the entire wheat field. Therefore, some measure of relative efficiency is necessary in order to give proper consideration to the work required as the size of the plot increases. A formal way of comparing the different plot sizes, considering the work required for the enumerator to make the field counts in the plot, is to look at relative efficiency per unit. To do this, multiply the variance of the estimator by the number of work units in it, with the smallest partition being defined as one unit. To get relative efficiency compared to the smallest unit, we then divide the variance of the smallest unit by each of these numbers. (Example: To compare all partitions to the smallest, form the ratio $(103.567)^2 / (96)(91.413)^2$). The results are as follows.

Plot Combinations	Relative Efficiency Compared To 4
4	1.000
4-5	.565
4-6	.381
4-7	.291
4-7,2	.138
4-7,2-3	.098
4-7,1-3	.072
4-7,1-3,8	.038
4-7,1-3,8-9	.026
4-7,1-3,8-10	.020
4-7,1-3,8-11	.016
4-7,1-3,8-12	.013

Note that every time more area is added, the estimator becomes less efficient. No estimator has a relative efficiency greater than one, meaning 4 is the most efficient size. A plot the size of 4 and 5 combined gives the second most efficient estimator, but it is only about half as efficient as a plot the size of 4 alone when we consider that twice as much work is required. The VMS plot size has only 1.3% efficiency compared to the smallest plot size when we consider that 96 times as much work is required.

IV CONCLUSION

Plot sizes in any one location should not be larger than 1.8 ft sq. This research project showed that the high correlation between units did decrease as the plots got further apart. For this reason, it would probably be advisable to independently locate a second plot in the field, since travel costs to the field are fixed regardless of number of plots. At summary time, data from the two plots would be combined to repesent the yield. Based on the observed CV's, it seems relatively certain that a sample size of 100 per province, each sample composed of 2 independently located plots, will guarantee 5% CV's at the province and national level. A sample size of at least 80-90 would be required for the same results at the district level. To convert to the metric system, we would also recommend plot sizes of 0.5 meters sq, approximately equivalent to the smallest size in this design instead of the 21.6 inch square plot size previously used.

The results of this research project confirms that plot size is not the reason for the wide difference in average yield levels between the VMS and ASF surveys. Therefore, it is recommended that a study be designed by an expert from the National Agricultural Statistics Service (NASS) to explore the difference in the yield levels being produced by the two different surveys. The study should be designed early in the fall of 1990 with the field work planned for the Rabi season of 1991.

APPENDIX A.

Let the areas of the partitions be represented in hectors and the production in kg.

Let $\underline{a}$ be a 12 x 1 vector whose elements represent the area each partition in hectors. Let n be the sample size on which the mean is based. Let $\underline{i}$ be a 12 x 1 vector of zero's and one's, where the incidence of ones denotes which partitions are to be combined. (Example: To combine plots 4 and 5 let $\underline{i}'$ = [0 0 0 1 1 0 0 0 0 0 0 0]). Let S be the 12 x 12 sample covariance matrix, where the observations are in kg. Then the sample variance of the mean is

$$\left(\frac{1}{n}\right)\left(\frac{1}{\underline{i}'\underline{a}}\right)^2 \underline{i}'S\underline{i}$$

where the mean is expressed in kg/n.

A RESEARCH PROJECT TO STUDY THE VARIATIONS OF WHEAT YIELDS ACROSS FIELDS

February 1990

A RESEARCH PROJECT TO STUDY THE VARIATIONS OF WHEAT YIELDS ACROSS FIELDS

General: The first part of this paper outlines in general terms the survey design, data tables that will result from this research and conclusions that will be drawn after the research is complete and the data have been analyzed.

Appendices A, B, C, and D provide the necessary specifications for: Sample Distribution, Locating, Laying Out, Partitioning and Harvesting the Research Plots; Statistical Formulations; Sketch Sheets, Tag Data, and Harvest Data Sheets; and Material and Resources Needed, respectively. Appendix E shows the design of the steel frame used in this research project.

Sample Design: A sample of size 140 (70 in Punjab and 70 in Sindh) will be necessary to ensure stable results at the Punjab and Sindh Province levels. The samples will be allocated to the districts in each Province in proportion to the number of hectors of wheat in each district in 1988-89 as follows:

District	1988-89 Hectors of Wheat	Percent	Sample Distribution
Faisalabad	255,453	21.57	15
Jhang	349,666	29.52	21
Multan	325,270	27.47	19
Sheikhupura	253,976	21.44	15
Punjab Province	1,184,365	100.00	70
Nawabshah	193,253	53.04	~~37~~ 25
Larkana	79,257	21.75	~~15~~ 24
Hyderabad	91,858	25.21	~~18~~ 21
Sindh Province	364,368	100.00	70
Total, ALL	1,548,733		140

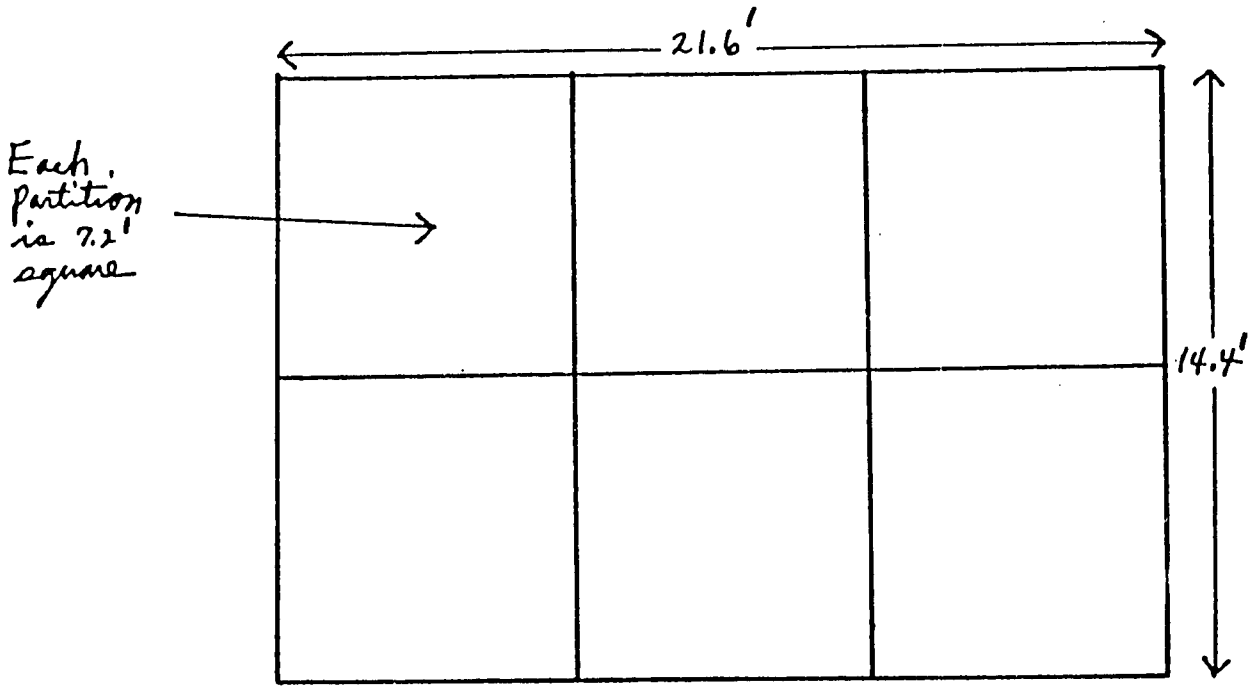
8 teams

6 teams

The samples will be placed in fields independently of the regular wheat objective yield plots in 1989-90. Placement of the research plots in the same fields with the objective yield plots won't contribute anything to the analysis since they are randomly located and the only thing that would make them different is non-sampling error.

80

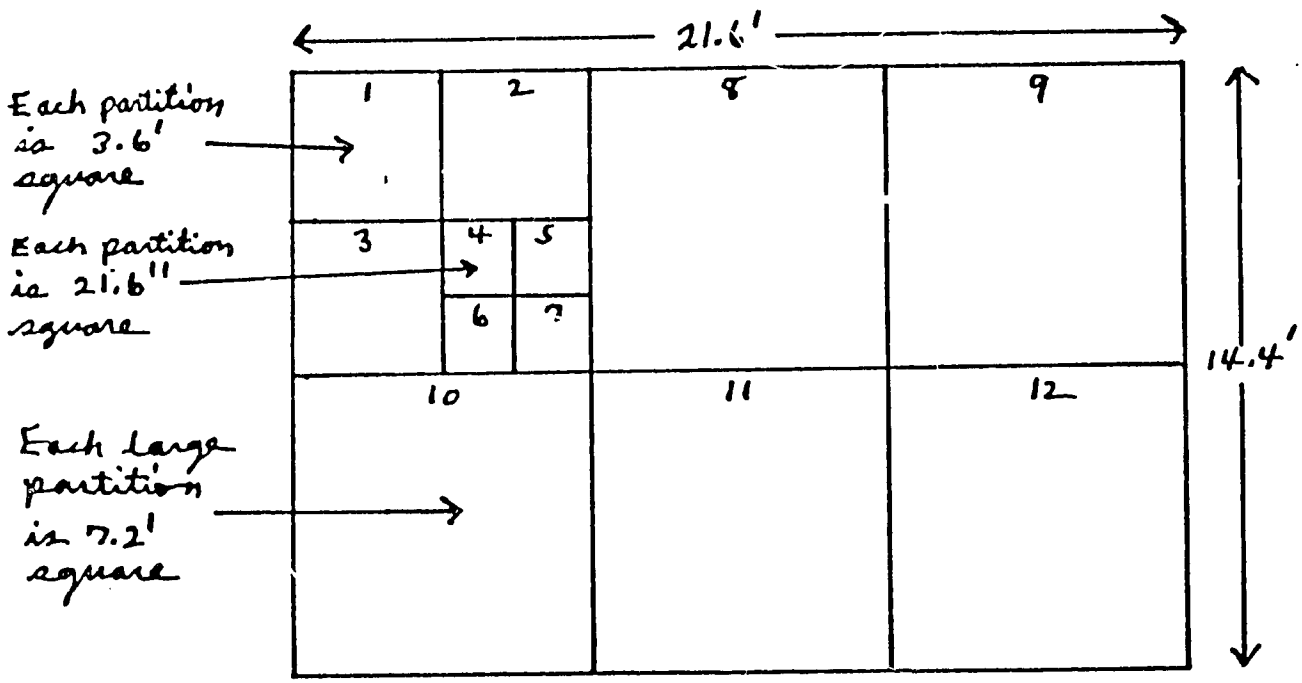
The Research Plot: The research plot would look like this:



Each of the six partitions is 7.2' square. The square footage of the entire plot is slightly larger than the 15 x 20 crop cutting plot now being used to allow for the next step which follows:

The first big partition will be sub-divided into four smaller partitions of equal size. One of these small partitions will be further sub-divided into four even smaller partitions of equal size. All partitions in the research plot will be numbered numerically from 1 through 12 for simplicity.

The research sample plot will now look like the following:



81

Each of the four smallest partitions (4 thru 7) are 21.6 inches square, exactly the size of the wheat OY plots being used in the regular program. This is the reason for making the sample research plot 21.6' x 14.4'.

Harvest of the Wheat: Wheat from each of the twelve partitions will be cut and mechanically threshed (with portable threshing machines). Portable threshing machines are necessary to control harvesting loss. Production from each of the twelve partitions will be kept separate in each of the 14.4' x 21.6' research plots.

Note: We can add up the production from partitions 1 through 7 and the combined total will be comparable to production from each of the partitions 8 through 12.

After harvest is complete and the grain is threshed and weighed for each of the twelve partitions in all 140 sample plots (12 x 140 = 1,680 total partitions), we can build the following three tables which will allow us to answer the questions concerning optimum plot size.

Table 1 -- Mean Yields Per Partition For Both Provinces Combined

Starting corner

1* $\bar{Y}_1$	8 $\bar{Y}_8$	9 $\bar{Y}_9$
10 $\bar{Y}_{10}$	11 $\bar{Y}_{11}$	12 $\bar{Y}_{12}$

* Partition 1 is the sum of small partitions 1 through 7.

If the means are about the same for each of the six big partitions over the 140 sample plots, it would indicate that a plot larger than 7.2' really serves no useful purpose. However, Table 3 at the end will answer this question more decisively.

Note: The Table, as illustrated above, can also be prepared for each of the following over the 140 sample plots:

1. Partitions 1, 2, 3, and 4 thru 7 combined.

2. Partitions 4, 5, 6, and 7.

3. All tables, the one illustrated and those suggested by 1. through 2., can also be broken down by the Punjab and Sindh Provinces.

The next table, Table 2 is constructed to obtain the values necessary for computing Table 3, which will really answer our questions about optimum plot size for wheat.

Table 2 -- Sample Covariance Matrix S (the sample covariances between individual field level raw yields)

Partitions *	Partitions *					
	Y_1	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}
Y_1	$S_{1,1}$					
Y_8	$S_{8,1}$	$S_{8,8}$				
Y_9	$S_{9,1}$	$S_{9,8}$	$S_{9,9}$			
Y_{10}	$S_{10,1}$	$S_{10,8}$	$S_{10,9}$	$S_{10,10}$		
Y_{11}	$S_{11,1}$	$S_{11,8}$	$S_{11,9}$	$S_{11,10}$	$S_{11,11}$	
Y_{12}	$S_{12,1}$	$S_{12,8}$	$S_{12,9}$	$S_{12,10}$	$S_{12,11}$	$S_{12,12}$

* Partition 1 is the sum of the small partitions 1 through 7.

The above table will also be repeated for 1., 2., and 3. on the bottom of page 3.

Once the covariance matrix is computed, the values will be used in the computation of Table 3 which follows:

Table 3 -- Standard Error Of Estimator If Data Are Used From Various Partitions For Both Provinces Combined.

Partitions		Standard Error of Estimator
1		
1 & 8	Combined	
1, 8, & 9	Combined	
1, 8, 9, & 10	Combined	
1, 8, 9, 10, & 11	Combined	
All	Combined	

Other Possibilities

1 & 12 combined
(the partitions furthest apart
in the researching plot)

8 & 11 combined
(all partitions having
one outside border)

1, 9, 10, & 12 combined
(all partitions having
two outside borders)

Etc.

---

* Partition 1 is the sum of small partitions 1 through 7.

The above table will also be repeated for 1., 2., and 3. on the bottom of page 3.

Table 3 should show that the standard errors progressively decrease as additional partitions are added to increase the plot size.

For Example: If data from the partitions are highly correlated, the standard errors will gradually decrease as partitions are added (plot size increases) and then level off. If the standard error for partitions 1, 8, and 9 combined is only 1 percent lower than the standard error for partition 1, we have increased the workload and costs by 200 percent for a 1 percent gain in precision and this couldn't be justified.

Another Example: The standard error may decrease rapidly until four partitions are added and then decrease gradually as additional partitions are added. If this is the case, perhaps we need a sample plot four partitions in size.

CONCLUSIONS FROM THIS STUDY: Table 3 will allow as to draw the following conclusions if the project is accurately executed in the field.

1) Examine the plot size (in relation to precision gained) from the current objective yield plot size (21.6") through the crop cutting plot size (14.4' x 21.6') and every size in between.

   Note: This will tell us if the current objective yield plot size is too small and it will also tell us if the current crop cutting plot is too large.

2) Determine whether the objective yield and crop cutting plots in this experiment are estimating the same universe.

   Note: Does the objective yield plots, contained within the 14.4' x 21.6' research plots, give the same yield over all samples that is produced by the 14.4' x 21.6' plots over all samples.

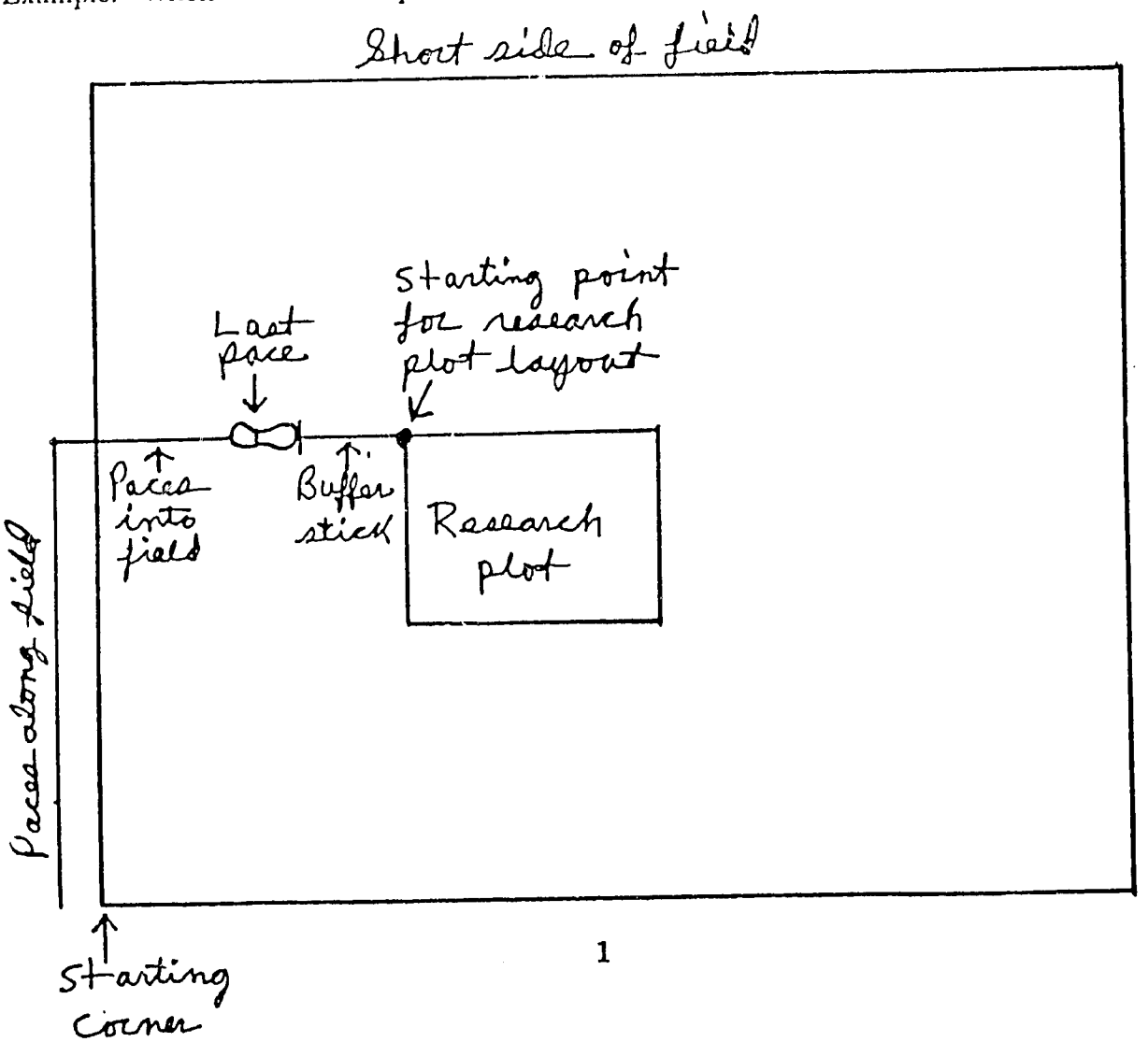
84

Sample Distribution, Locating, Laying Out, Partitioning, and Harvesting the Research Plots

Sample Distribution: The research plots will not be put in the fields until harvest time to prevent damage to the plots and extra trips to the fields. The research plots shown earlier by district will be allocated to each district with probability proportional to the wheat acreage in each field in that district.

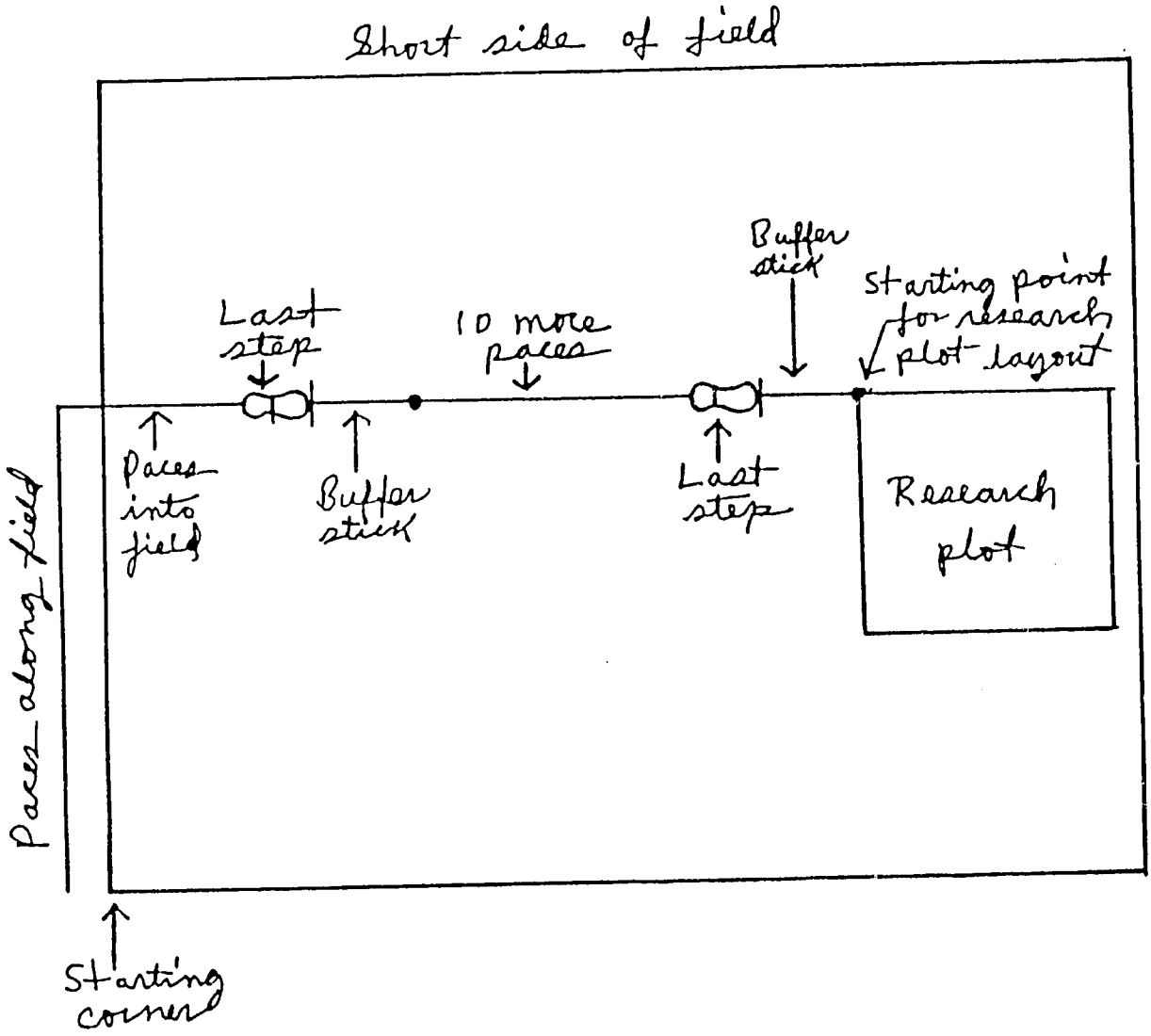
Location of the Research Plots: Enumerators will be given random numbers for paces along the long side of the field and paces into the short side of the field, using the same methods that are currently being used for locating the OY wheat plots. If a research plot falls in a field that already contains an OY plot, an independent set of random numbers will be provided to the enumerator for locating the random starting point for laying out the research plot. If by change, the research plot still falls on top of the OY plot, the enumerator will continue another 10 paces in the same direction of travel and establish another starting point for laying out the research plot.

Example: When the research plot does not fall on top of OY plot



85

Example: When research plot does fall on top of OY plot



Laying Out the Research Plot: Once the starting point for laying out the research plot has been established, the plot will be layed out as follows:

NOTE: All work must be done outside of the plot to prevent damage to the plot.

1. Place a 4' bamboo pole in the ground at the starting point. This becomes the first corner of the research plot.

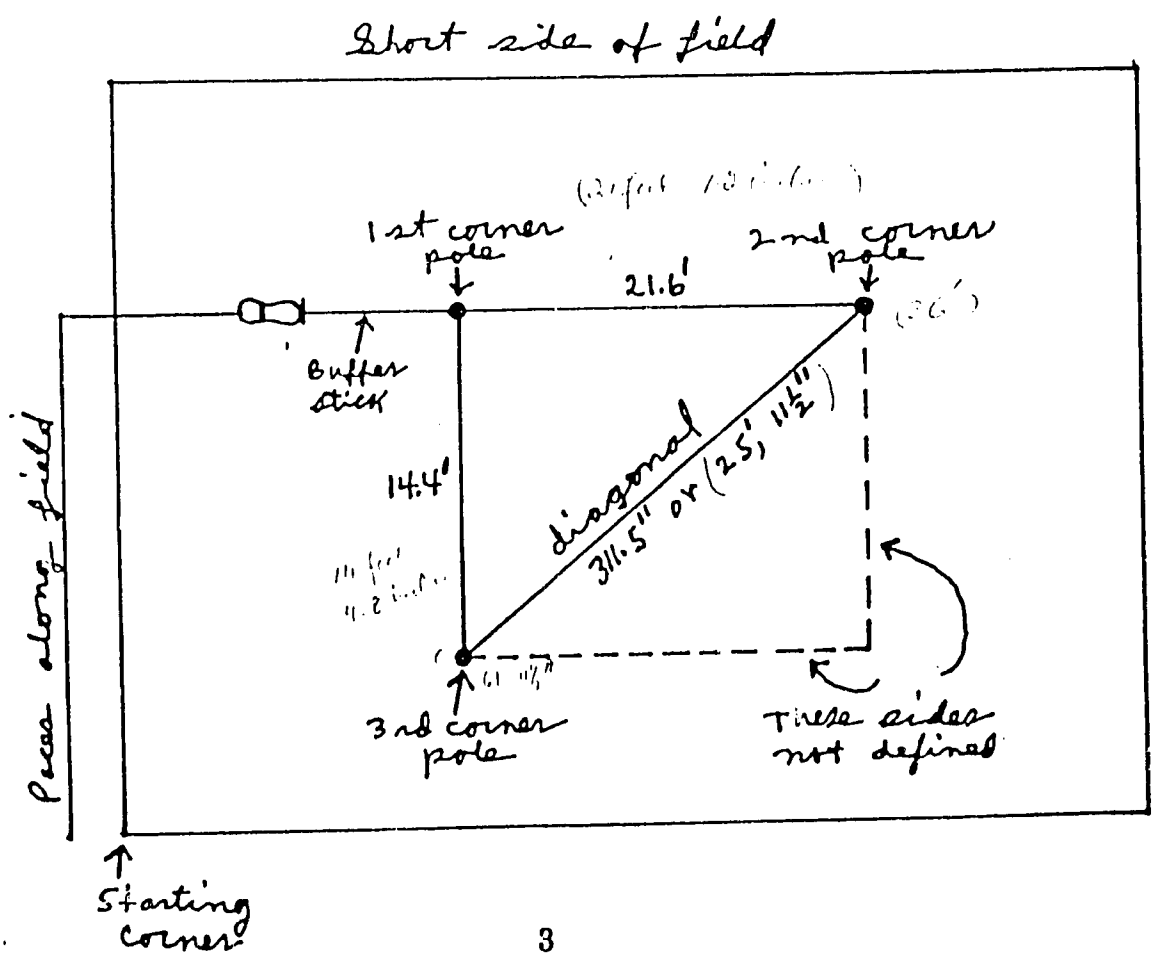
2. Attach a tape to this pole and continue in a straight line, in the same direction as before for exactly 21.6'. Place another 4' bamboo pole in the ground at this point, establishing the second corner of the research plot.

3. Leave the tape attached to the bamboo pole at the starting corner and return to the starting corner. Then measure away from the starting corner pole (to the right of the original line of travel) exactly 14.4'. Place a bamboo pole at this point making the third corner. This pole should form a perfect right angle with the poles earlier placed at the starting corner and second corner of the research plot.

4. To be sure that the third corner pole forms an exact right angle with the first and second poles, the diagonal from pole 2 to pole 3 should be measured. If the diagonal is not exactly 311.5" or 25' 11 1/2", the third corner pole should be adjusted until the diagonal is correct, being careful to keep the distance from pole 1 exactly 14.4'.

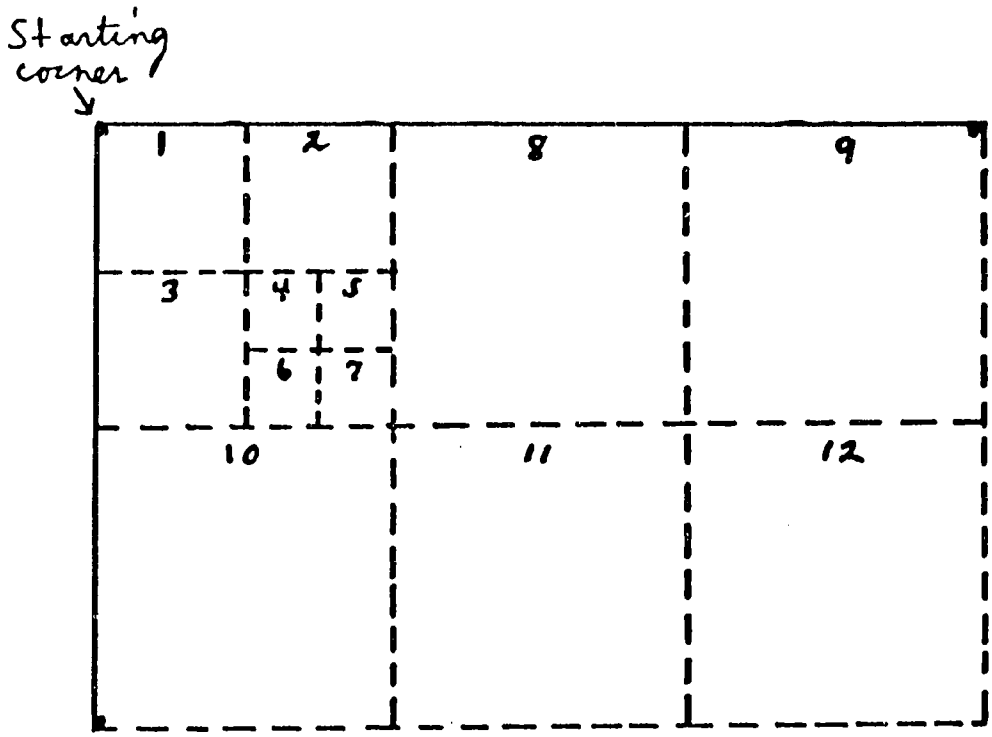
5. For purposes of this project, there is no need to establish the fourth corner and other two side of the research plot (indicated below by the broken lines).

After the plot is properly layed out, it will look like this:



Partitioning and Harvesting the Research Plot: The research plot will be divided into twelve partitions by use of a steel frame (see Appendix E for a diagram of this frame).

After the plot is divided, the twelve partitions will be numbered numerically. Each of the 140 sample plots will be partitioned and numbered exactly as shown by the diagram below:

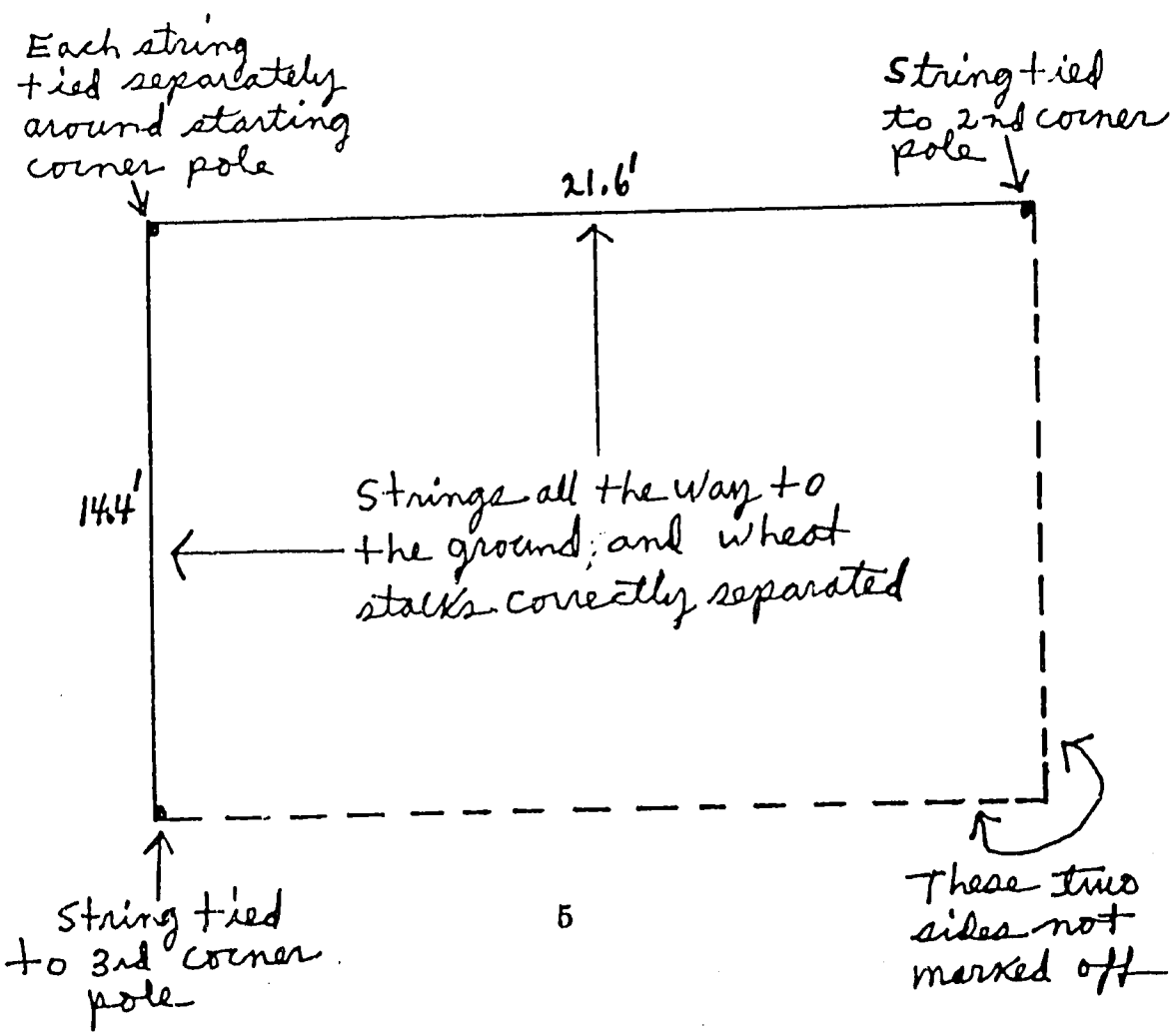


It will be necessary to lay the frame down once to form partition 1, once to form partition 2 and once to form partition 3. The four smallest partitions, 4 through 7, will be formed by laying the frame down only once. However, it will be necessary to lay the frame down four times to form each of the other partitions, 8 through 12. Each time the frame is layed down, the wheat inside the frame will be harvested. This way it will not be necessary to ever walk inside the unharvested plot, preventing damage to the wheat and loss of kernels. The plot will be partitioned and harvested as follows:

1. At a level above the top of the wheat, tie a string to the starting corner. Run this string to the top of corner pole 2. Tie a separate string to the starting corner pole and run this string to the top of corner pole 3. The string on both sides of the plot should be just above the top of the wheat.

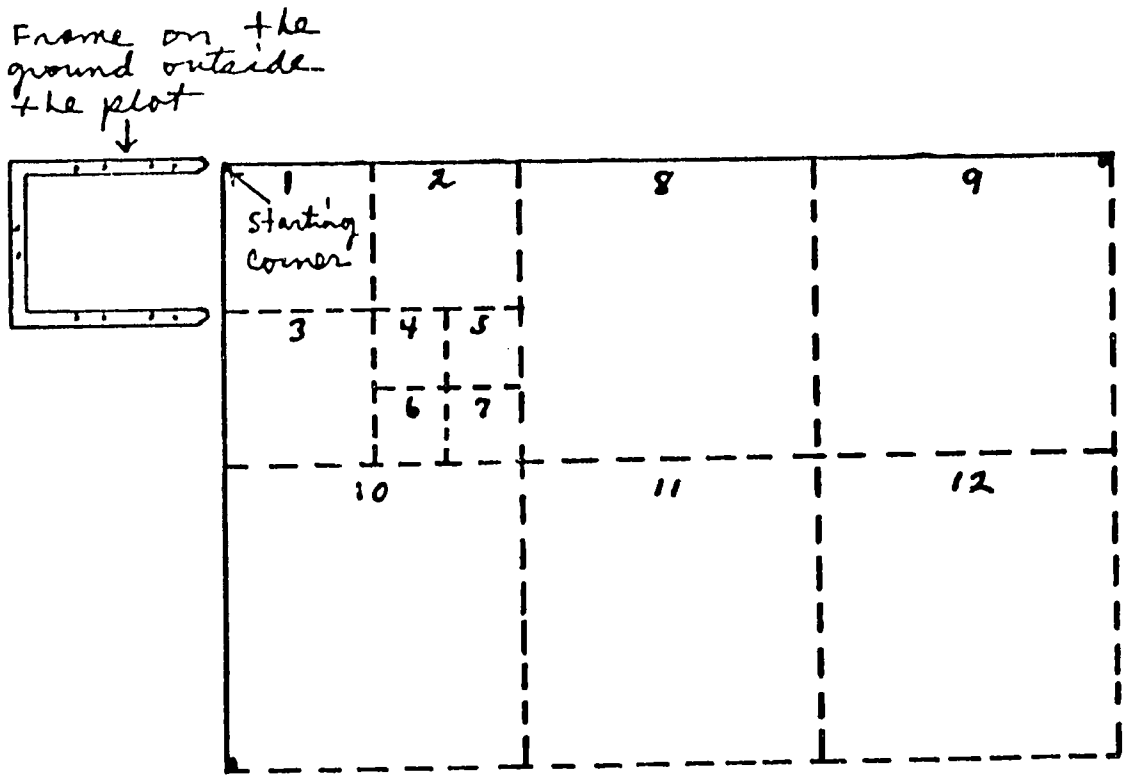
2. Next, it will be necessary to establish a line at ground level on the two sides of the plot (between the three poles) to guide our steel frame as we begin partitioning and harvesting. To do this, slowly push each string to the ground one at a time. During this process, the strings must be kept straight and tight and plants that fall directly under each string must be separated by hand to ensure that stalks inside the plot remain inside and stalks outside the plot remain outside. The strings should be pushed as close to the ground as possible.

The research plot will now look like this:



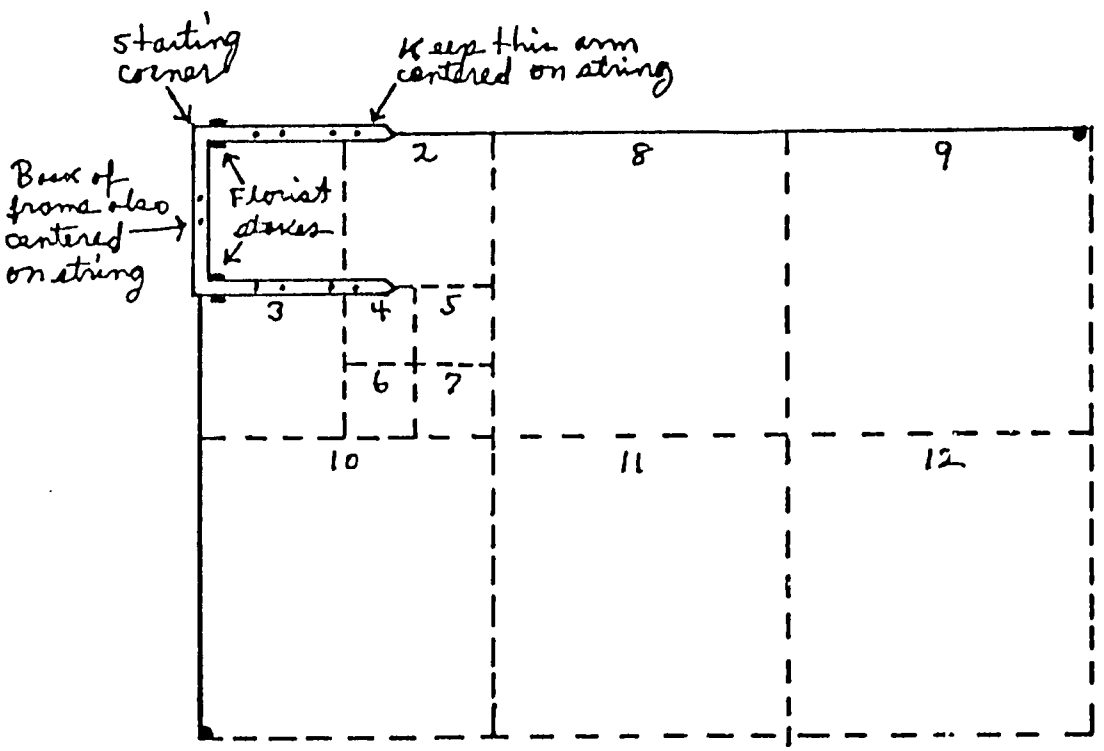
3. You are now ready to begin dividing the plot into the twelve partitions using the steel frame.

Put the frame on the ground outside of the plot at the starting corner as illustrated below (it may be necessary to temporarily remove the bamboo pole at the starting corner).



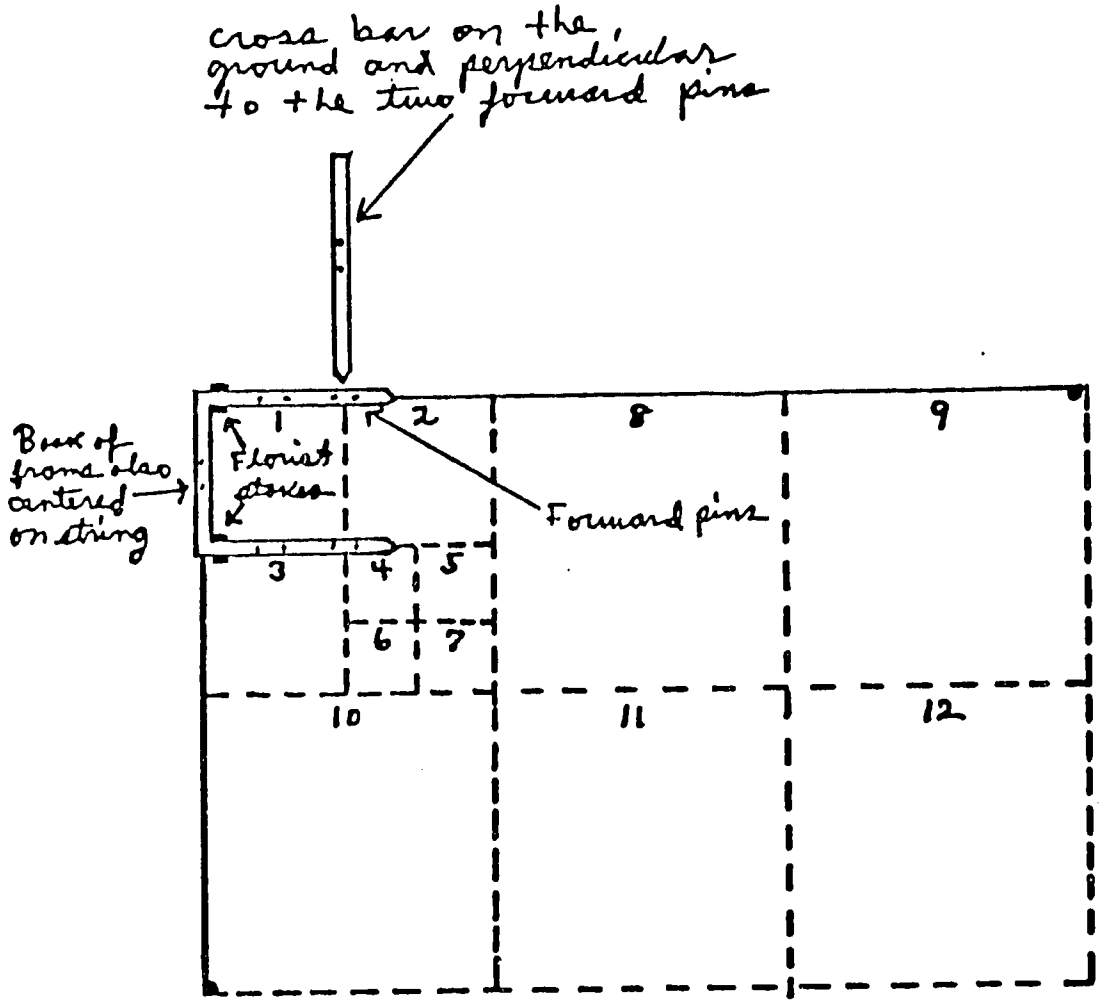
After the frame is on the ground, and in line with the string on the long side of the plot, you are ready to gradually slide it into the plot separating the stalks of wheat as you go. Make sure the outside arm of the frame remains centered on the string. Push the frame into the plot until the back of the frame is also centered on the string on the short side of the plot. When both the outside arm and back of the frame are centered on the strings, the frame should be properly positioned. We are now ready to put down four florist stakes at the back of the frame to hold it in place.

The research plot will now look like this:



91

Next, from the outside of the research plot, position the cross bar on ground perpendicular to the two forward pins on the outside arm of the frame. This is illustrated below:



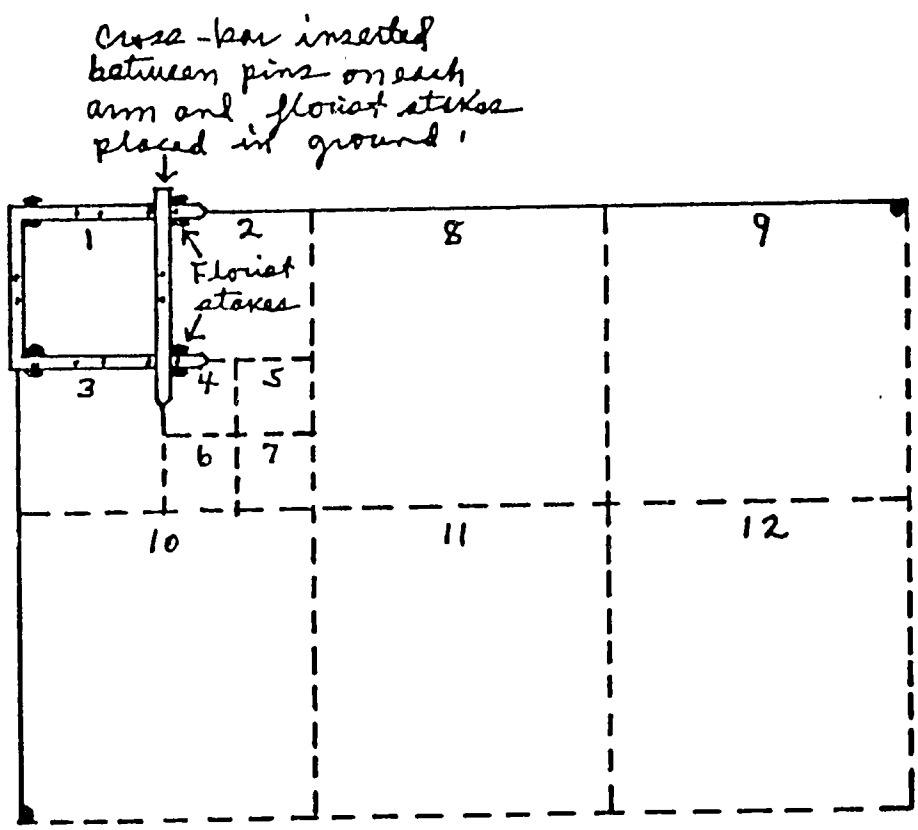
92

Now slowly push the cross bar through the wheat, separating the stalks as you go, until the end of the cross bar is between the two forward pins on the opposite side of the frame.

NOTE: It is important to properly separate the stalks of wheat as you push the cross bar across the frame, otherwise too much or too little wheat will be inside the frame.

After the forward cross bar is in place, florists stakes should be placed on both sides of each arm, in front of the cross bar.

The plot will now look like this:



93

The previous diagram shows that we have properly layed off partition 1. We are now ready to harvest all wheat inside the steel frame (partition 1).

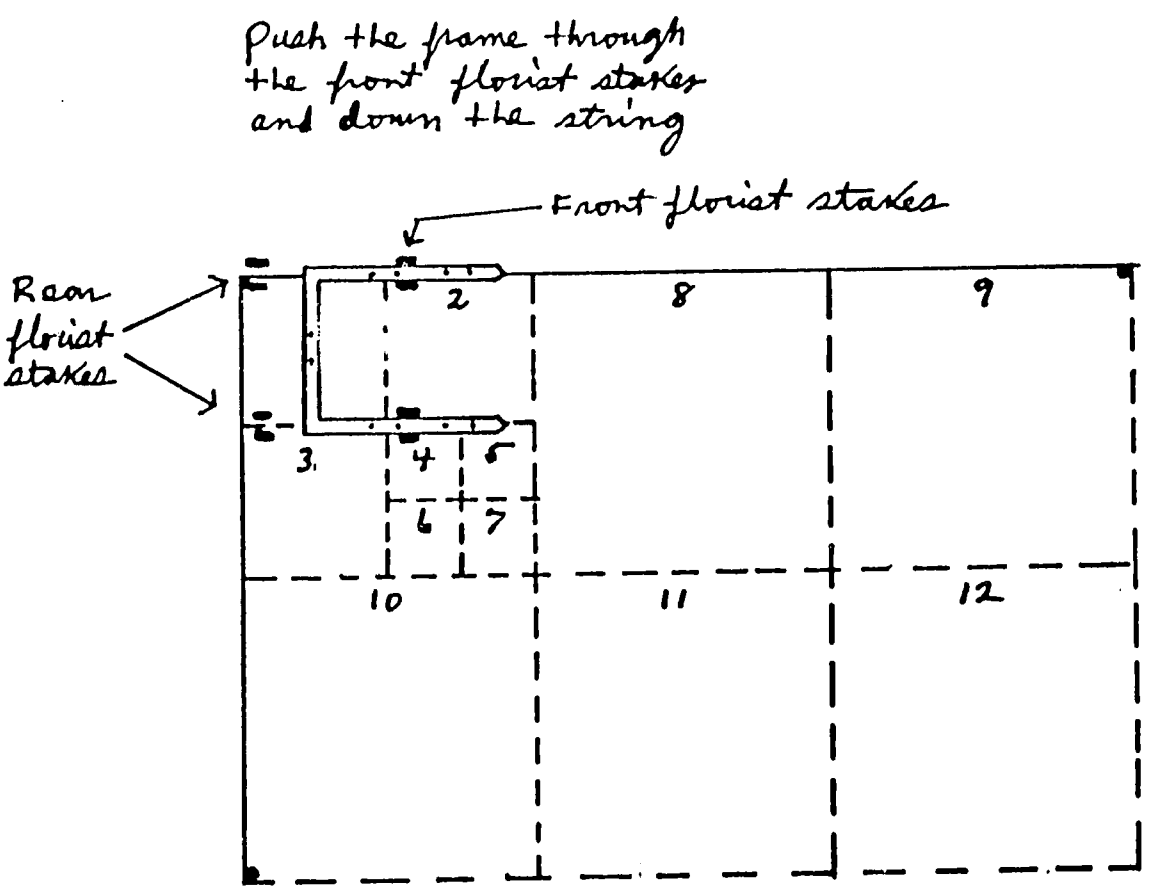
The wheat should be cut at ground level and placed in a cloth bag. A tag containing the necessary identifying information, should be securely attached to the cloth bag (see page 3 of Appendix C). Partition 1 on the Sketch Sheet shown on page 2 of Appendix C should also be marked out to show that this partition has been harvested.

4. To lay-off partition 2, we will use the florist stakes from the front of the cross bar of the previous frame (none of the previously placed florists stakes should be removed until the entire plot has been partitioned and harvested).

To lay-off partition 2, follow this procedure:

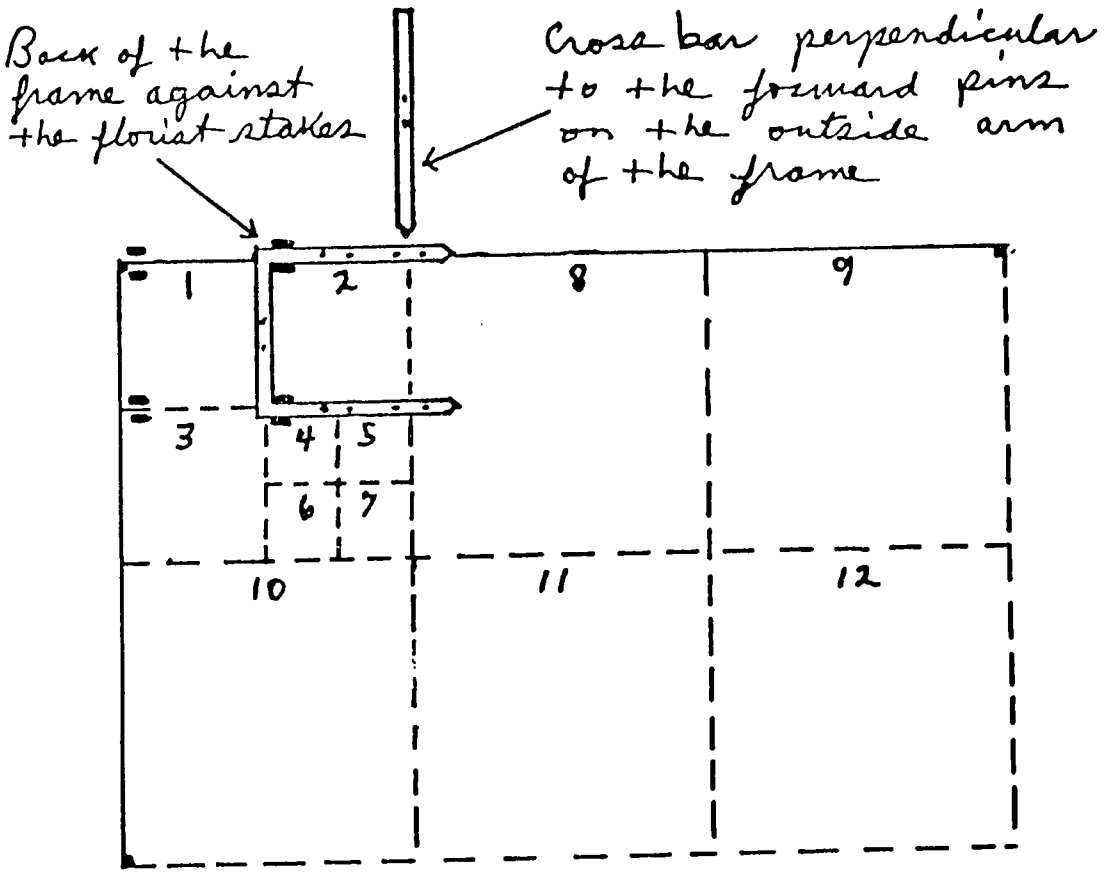
First, remove the cross bar from the frame (which is still in partition 1). Next, lift the back of the frame over the rear florist stake and push the frame along the string and through the front florist stakes.

See the diagram below:



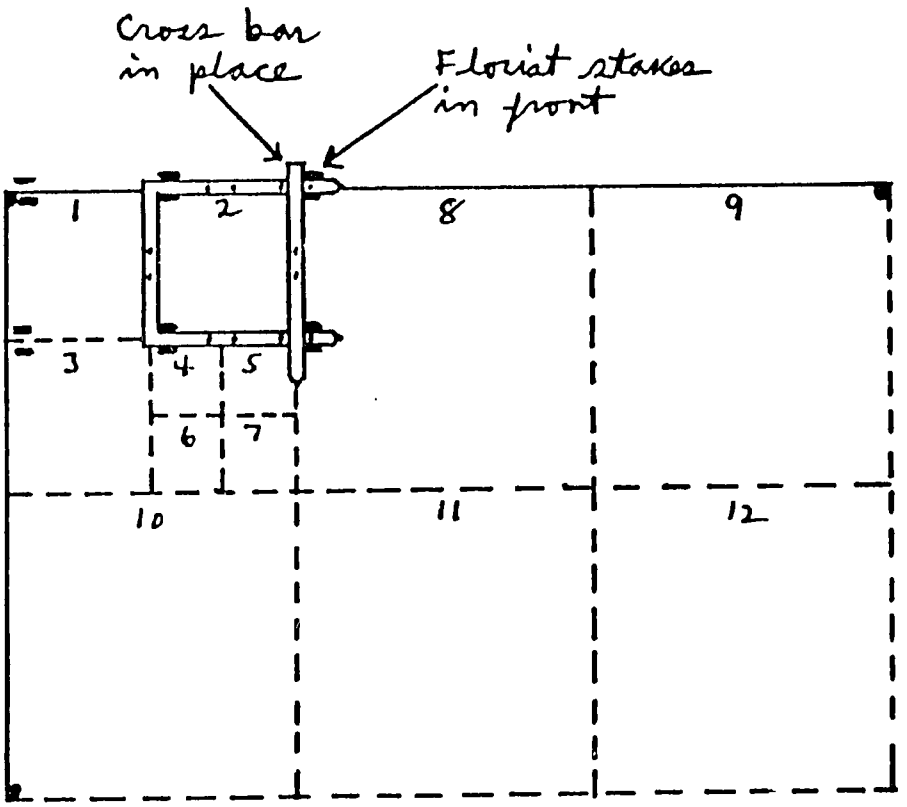
Continue to push the frame along the string and through the florist stakes correctly separating the wheat stalks with both arms of the frame. When the rear of the frame is against the florist stakes and the outside arm of the frame is directly centered over the string, the frame should be in the correct position. Lay the cross bar on the ground perpendicular to the two pins on the outside arm of the frame as before.

See the diagram below:



The cross arm should be pushed across the frame as before until it is between the forward points on both arms of the frame. Florist stakes should then be placed in front of the cross bar on both sides of each arm, as before.

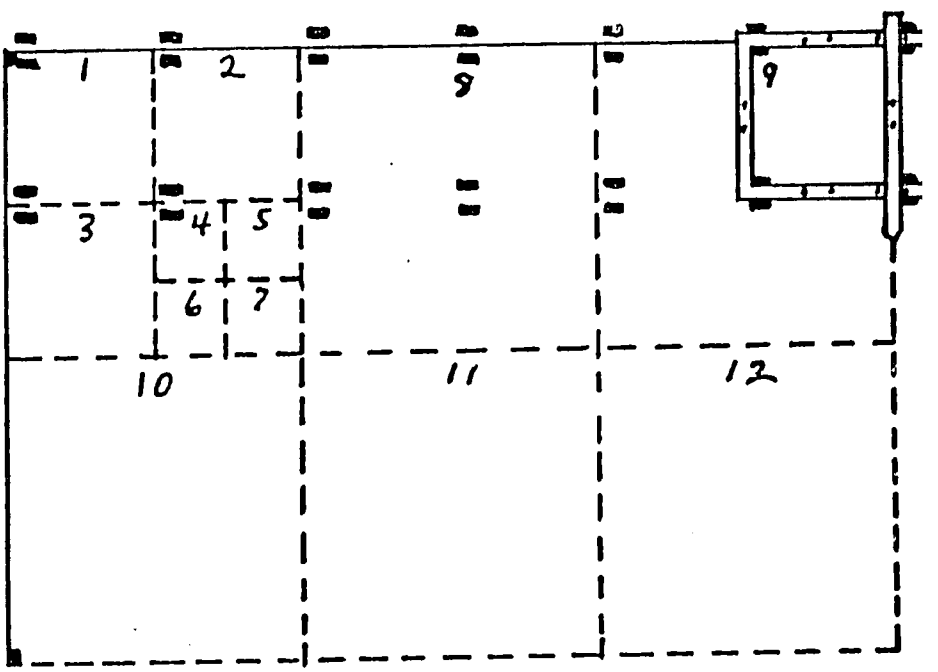
The plot will now look like this:



Partition 2 is now layed off and ready for harvesting. All wheat inside the frame should be cut, put in a cloth bag, and properly tagged (see page 3 of Appendix C). Again partition 2 on the Sketch Sheet shown on page 2 of Appendix C should be marked out to show that this partition has been harvested.

5. The procedures illustrated for laying off partitions 1 and 2 will be repeated through partitions 8 and 9 until the end of the plot is reached. The front florist stakes from the previous frame and the string will be used to guide the frame in a straight line. When the last frame is layed down, the plot will look like this:

It is necessary to lay the frame down six times on each pass through the plot



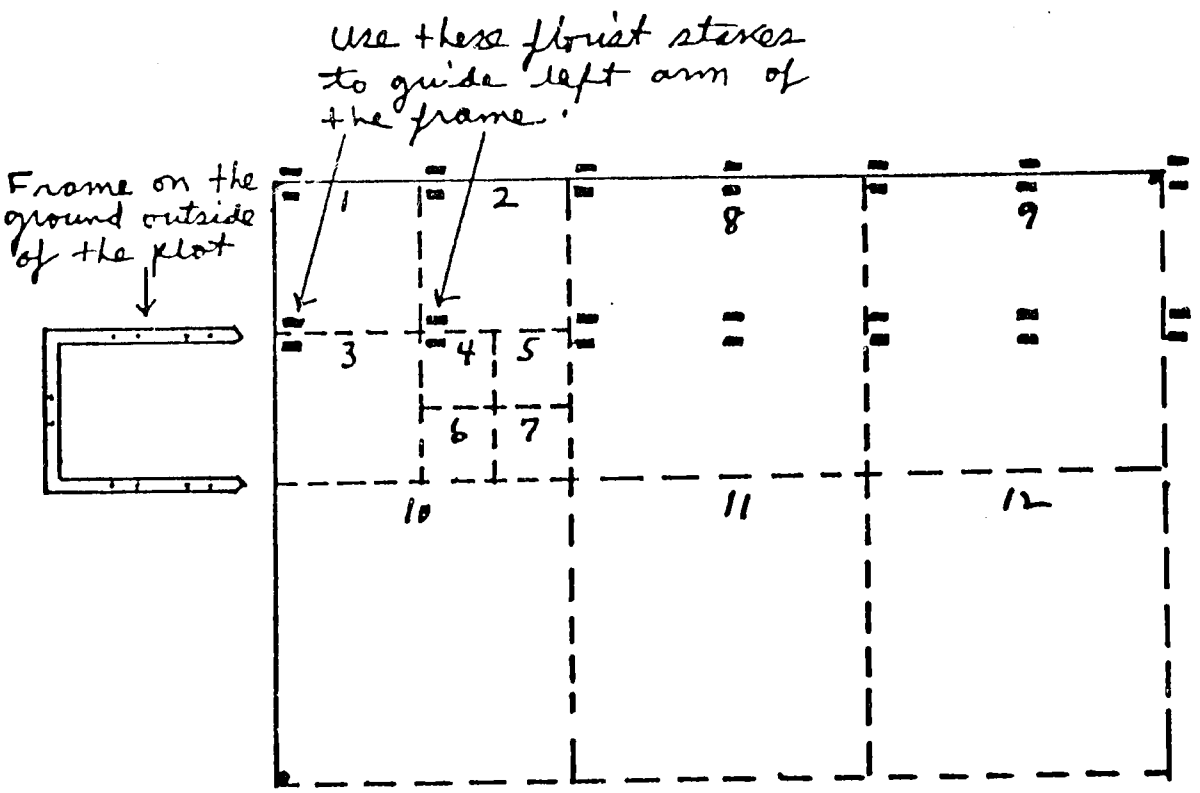
Note from the above diagram that the frame was layed down twice in partitions 8 and 9 and it will take two complete passes through the plot before all parts of partitions 8 and 9 are harvested.

NOTE: On the first pass through the plot, it is critical to follow the string in a straight line because the florist stakes placed in the ground during this pass will be used to guide the frame on the next pass through the plot. None of the florist stakes are to be removed from the plot until it is completely partitioned and harvested.

6. On the second pass of the frame through the plot, the florist stakes left in the ground from the first pass through the plot will be used to guide the frame. We will illustrate this by laying off partition 3.

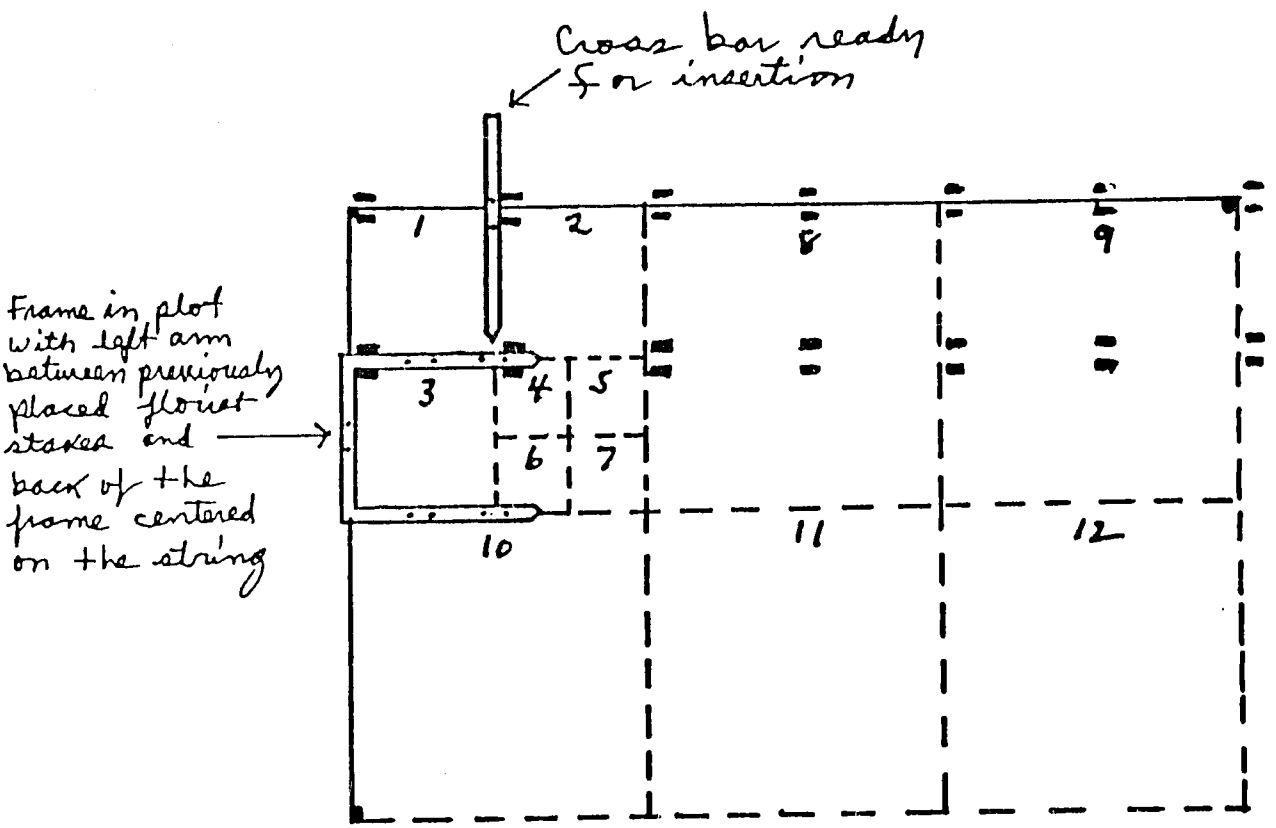
First, place the frame on the ground outside of the plot as shown by the diagram below. Slowly push the frame into the plot, using the previously placed florist stakes to guide the left arm of the frame.

See the diagram below:



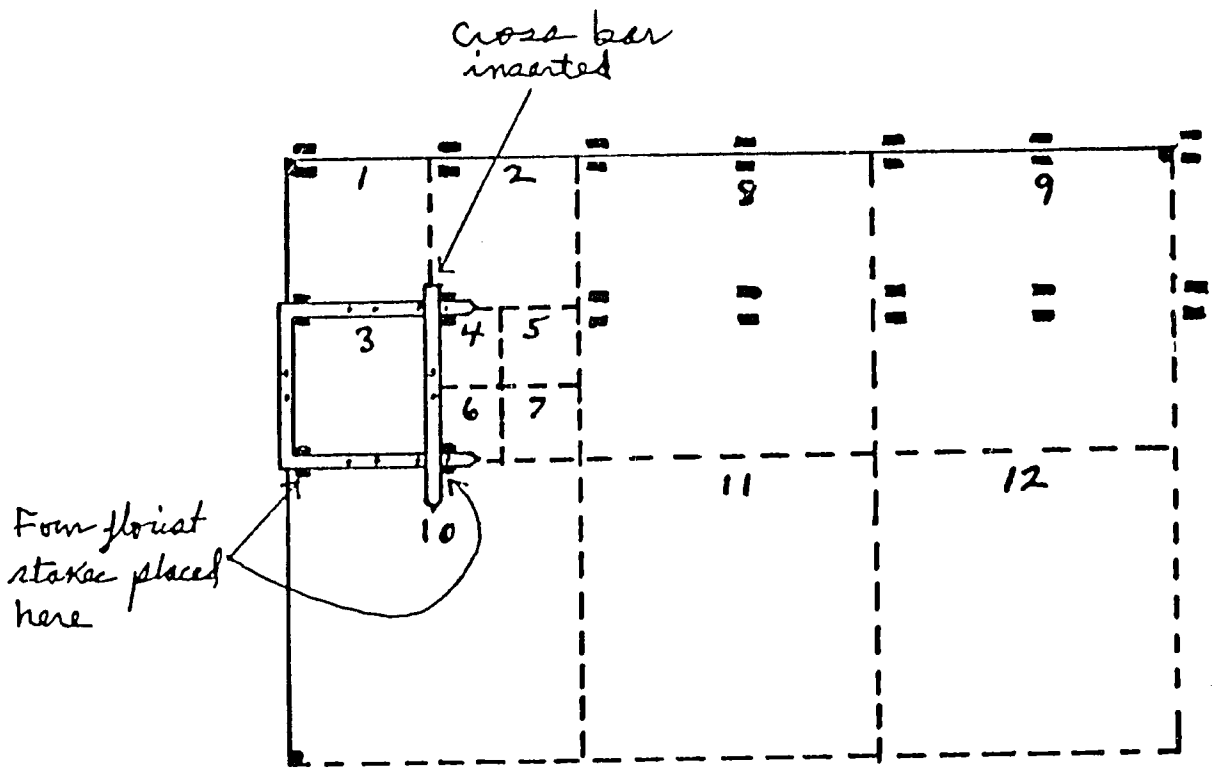
After the frame is in the plot, and the back of the frame is centered over the plot boundary string, position the front cross bar for insertion.

See the diagram below:



Now insert the cross bar and place four florist stakes on the right arm of the frame.

The plot is now ready for harvesting and will look like this:



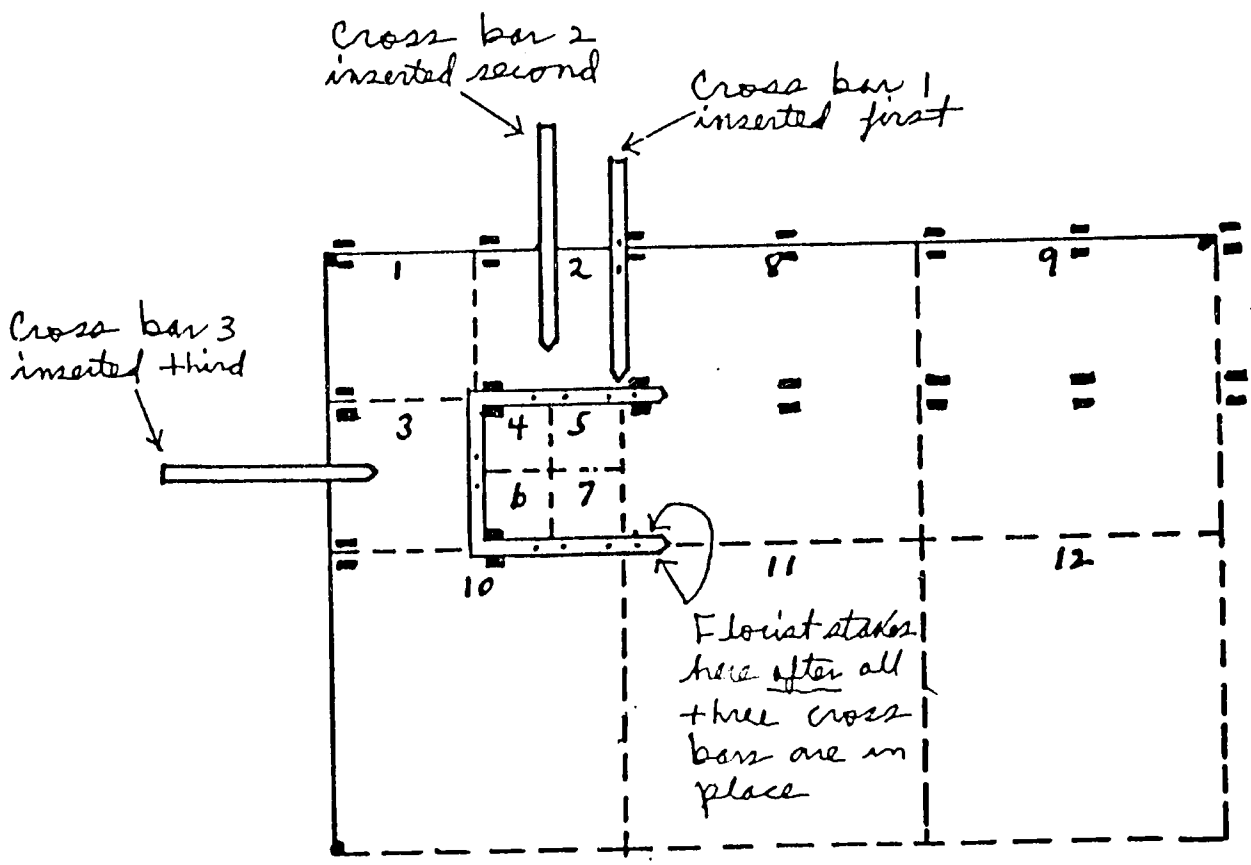
7. The laying off of partitions 4, 5, 6, and 7 will now be illustrated since the procedures differ from those previously illustrated.

Lift the back of the frame above the rear florists stakes and push the frame forward guiding the left arm of the frame between the florist stakes previously placed. When the back of the frame hits the florist stakes, the frame is correctly positioned and we are ready to insert the three cross bars.

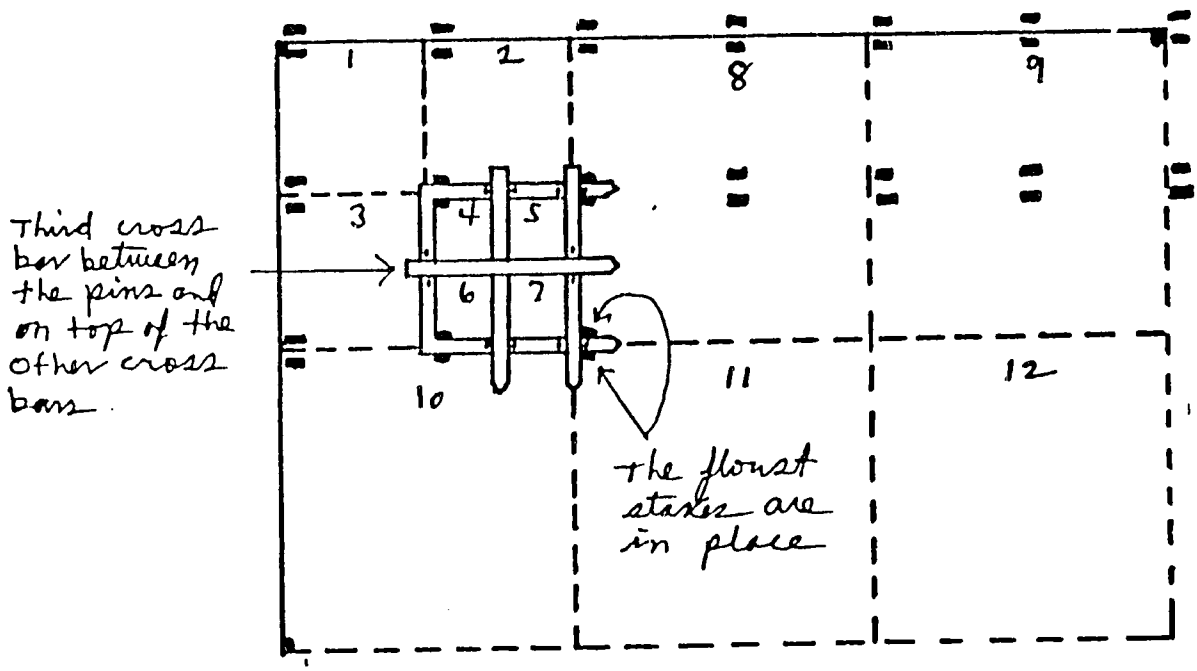
Cross bar 1 will be inserted first, cross bar 2 will be inserted next, and cross bar 3 will be inserted last. After all three cross bars are inserted, two florist stakes should be placed in front of the front cross bar on each side of the right arm of the frame.

NOTE: When pushing these three bars through the wheat, it is critical that the stalks of wheat are correctly separated by the bars. Otherwise, some of these four small partitions will contain too much wheat and some too little.

See the illustration below:



The next illustration shows all cross bars in place. Note the third cross bar is placed on top of the other two cross bars and all bars are between the pins provided. Note also that the two florist stakes were added after the three cross bars were put in place.

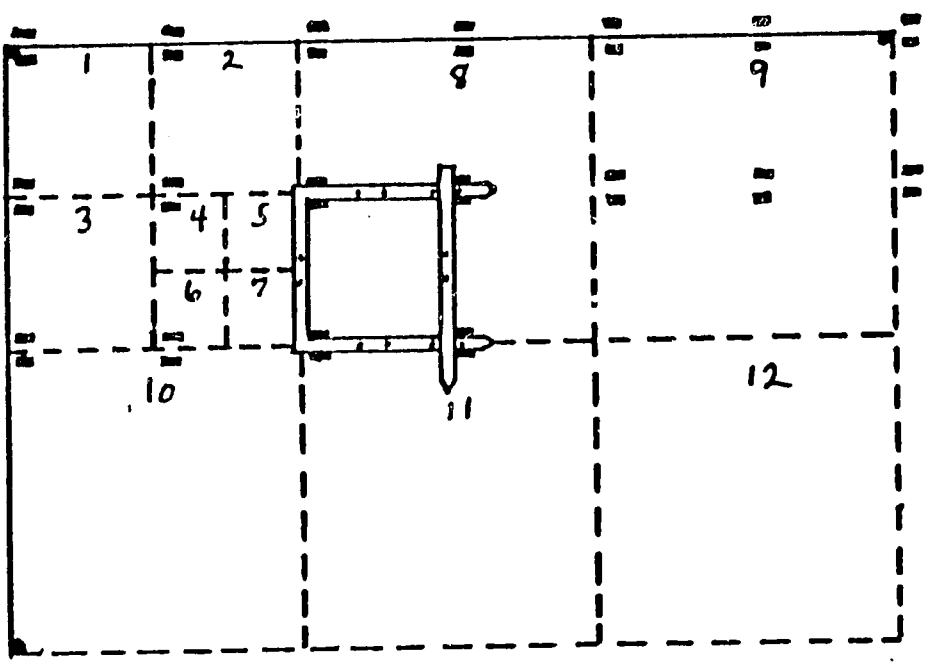


Partitions 4, 5, 6, and 7 are correctly formed and are ready for harvest. The wheat from each of these four partitions will be put in a separate bag and tagged as before.

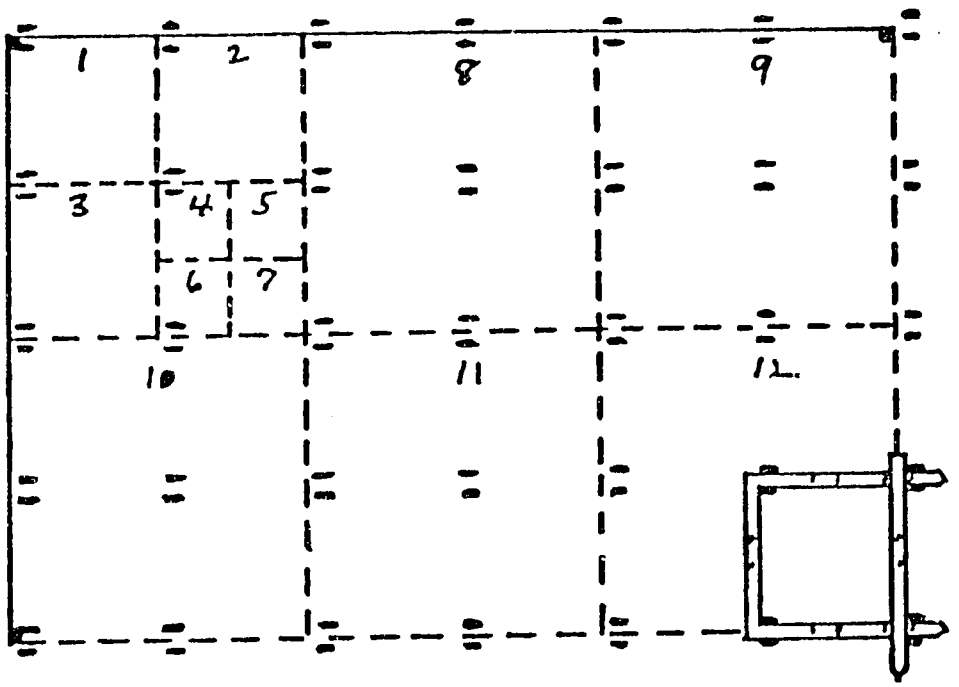
8. The frame will now be moved to partition 8 and on through the rest of the plot until the end of the plot is reached using the same procedures as before.

NOTE: It is important to remember that it is necessary to lay the frame down four times in each of the partitions 8 through 12. All of the wheat from partition 8 will go in one cloth bag, all wheat from partition 9 will go in one cloth bag, etc. If a cloth bag will not hold all of the wheat from a partition, a second bag may be used but they should be securely tied together and properly tagged.

See the two illustrations below:



All florist stakes stay in the plot until it is all partitioned and harvested



After the plot is completely partitioned and all partitions have been harvested, the florist stakes should be removed for use in the next research plot.

After all partitions have been harvested, bagged, and properly labeled, they will be taken to the mechanical thresher where the contents of each bag will be threshed. The grain will be carefully weighed and recorded on the Harvest Data Sheet provided along with the proper identification from the tag attached to the cloth bag. The weight of wheat from each partition must be recorded in grams to two decimal places. In addition, grain from partitions 1, 8, 11, and 12 will be tested for moisture and this will be recorded to one decimal place (see Appendix C for examples of these forms).

NOTE: Be absolutely sure that all grain is out of the mechanical thresher before wheat from the next partition is entered for threshing.

Statistical Formulations

Table 1

i = sample $\qquad i = 1, ----, 140$

j = partition $\qquad j = 1, 8, ---, 12$

Y_{ij} = yield from sample i and partition j

$$\bar{Y}_1 = \frac{1}{n} \sum_{i=1}^{n} Y_{i,1} \qquad n = \text{sample size} = 140$$

$$\bar{Y}_8 = \frac{1}{n} \sum_{i=1}^{n} Y_{i,8}$$

$$\vdots$$

$$\bar{Y}_{12} = \frac{1}{n} \sum_{i=1}^{n} Y_{i,12}$$

Table 2

Y_j = the yield from partition j

$Cov(Y_i, Y_j)$ = covariance between the yield from partition i and yield from partition j

$$Cov(Y_j, Y_{j'}) = \frac{1}{n} \sum_{i=1}^{n} (Y_{ij} - \bar{Y}_j)(Y_{ij'} - \bar{Y}_{j'})$$

j and j' are partitions and are fixed in this summation

n = sample size of 140

Ex: $\frac{1}{140} \sum_{i=1}^{140} (Y_{i1} - \bar{Y}_1)(Y_{i1} - \bar{Y}_1) = \sum_{i=1}^{140} (Y_i - \bar{Y}_1)^2 = S_{1,1}$

Ex: $\frac{1}{140} \sum_{i=1}^{140} (Y_{i1} - \bar{Y}_1)(Y_{i8} - \bar{Y}_8) = Cov(Y_1, Y_8) = S_{1,8}$

These covariances can be written in the form of a covariance matrix, S, as follows:

$$S_{6 \times 6} = \left[S_{i,j} \right]$$

The above covariance matrix, S, will be used in the computation of Table 3.

Table 3

For the standard error of the estimators [illegible] in this table, we first need to compute the variance of the mean of n_t of the partition means.

The variance of the mean of n_t of the means can be written as:

$$\frac{1}{N^2} \frac{1}{n_t^2} \underline{i}' S \underline{i}$$

N = sample size = 140

n_t = number of means that are being combined or averaged (which is equivalent to getting the mean yield for the combined areas).

$\underline{i}'$ = 1 x 6 row vector denoting which means are being combined

S = 6 x 6 covariance matrix from Table 2.

101

Ex: The variance of the means of partitions 1, 8, and 9 combined is computed as follows:

$$\frac{1}{140^2}\,\frac{1}{3^2}\begin{bmatrix}1 & 1 & 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix} S_{1,1} & S_{1,8} & \cdots & S_{1,12} \\ S_{8,1} & S_{8,8} & \cdots & S_{8,12} \\ \vdots & \vdots & & \vdots \\ S_{12,1} & S_{12,8} & \cdots & S_{12,12}\end{bmatrix}\begin{bmatrix}1\\1\\1\\0\\0\\0\end{bmatrix}$$

$$= \frac{1}{140^2}\,\frac{1}{3^2}\left[\mathrm{Var}(S_{1,1}) + \mathrm{Var}(S_{8,8}) + \mathrm{Var}(S_{9,9}) + 2(S_{1,8}) + 2(S_{1,9}) + 2(S_{8,9})\right]$$

The square root of the above will give the standard error for partitions 1, 8, and 9 combined in Table 3.

Ex: Variance of the mean for the entire research plot is

$$\frac{1}{140^2}\,\frac{1}{6^2}\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}\begin{bmatrix} S_{1,1} & S_{1,8} & \cdots & S_{1,12} \\ S_{8,1} & S_{8,8} & \cdots & S_{8,12} \\ \vdots & \vdots & & \vdots \\ S_{12,1} & S_{12,8} & \cdots & S_{12,12}\end{bmatrix}\begin{bmatrix}1\\1\\1\\1\\1\\1\end{bmatrix}$$

The square root of the variance from the above computation will give the standard error for partitions 1, 8, 9, 10, ---, 12 combined in Table 3.

108

Sketch Sheet, Tag Data, and Harvest Data Sheet

Two forms will be required for each research plot -- the SKETCH SHEET and the HARVEST DATA SHEET. In addition, at least 12 tags will be required for each research plot. Examples of these follows:

SKETCH SHEET

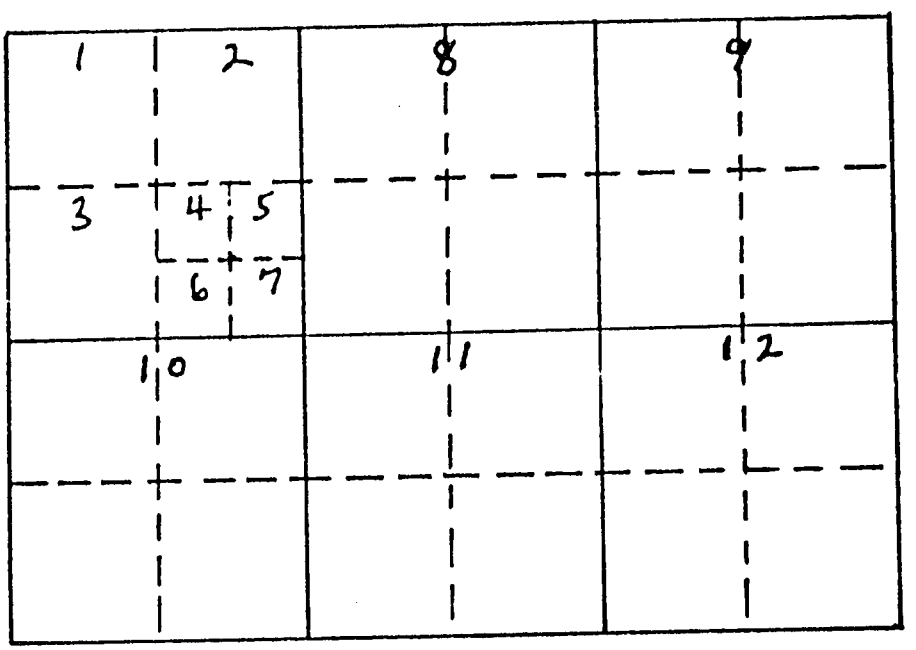
The following will be entered by the ADC

District number _____
Segment number _____
Field number _____
Paces along the field_____

Sample number _____
Grower number _____

Paces into the field _____

On the remainder of this page, the research plot has been sketched to help enumerators avoid mistakes when entering partition numbers on the tags. As each square in the diagram below is harvested and put in a cloth bag, the square should be marked out. At the same time, the enumerator should verify that the partition number being put on the tag corresponds to the partition number on the diagram.



Mark out each of the above squares as it is harvested to help avoid mistakes when entering partition numbers on the tags.

TAG DATA

The following data must be written on each tag attached to the cloth bags containing unthreshed wheat from each partition.

Sample number _____

Partition number

A minimum of 12 tags will be required for each of the 140 research plots.

This assumes samples will be numbered 1 through 140 over all 7 districts.

HARVEST DATA SHEET

The following will be entered by the ADC:

District number _____ Sample number _____
Segment number _____ Grower number _____
Field number _____
Paces along the field _____ Paces into the field _____

After the partitions have been harvested and the wheat has been threshed and weighed and tested for moisture, the following will be completed:

Partition	Weight of Threshed Grain in Grams (Show two decimal places)	Moisture Content of Threshed Grain in Percent (Show one decimal place)
1	_____.__ Grams	_____.__ Percent
2	_____.__ Grams	XXXXXXXXX.XX Percent
3	_____.__ Grams	XXXXXXXXX.XX Percent
4	_____.__ Grams	XXXXXXXXX.XX Percent
5	_____.__ Grams	XXXXXXXXX.XX Percent
6	_____.__ Grams	XXXXXXXXX.XX Percent
7	_____.__ Grams	XXXXXXXXX.XX Percent
8	_____.__ Grams	_____.__ Percent
9	_____.__ Grams	XXXXXXXXX.XX Percent
10	_____.__ Grams	XXXXXXXXX.XX Percent
11	_____.__ Grams	_____.__ Percent
12	_____.__ Grams	_____.__ Percent

APPENDIX D

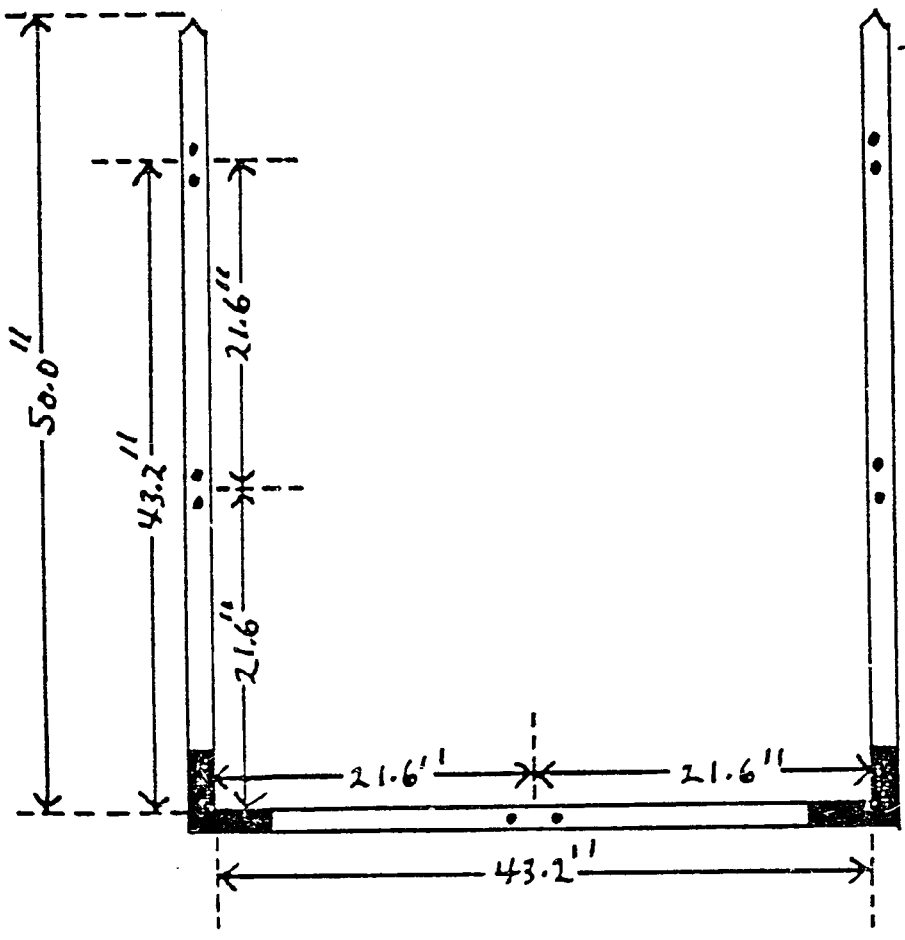
Materials and Resources Needed

The following materials will be needed in order to properly conduct this research project:

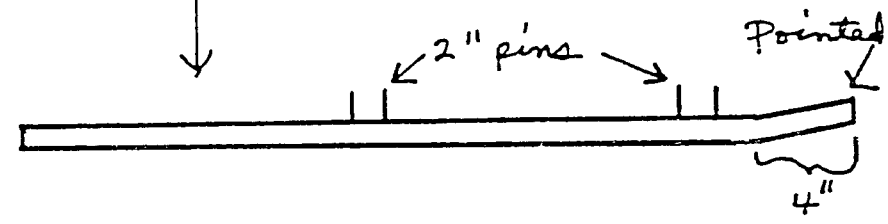
ITEM	QUANTITY	APPROXIMATE COST
14 harvesting crews	3 persons per crew	
Binder twine (or similar kind of rope or string (63 x 140) = 8,820 =	10,000 feet	
Tags to tie to cloth bags (12 x 140) = 1,680 =	2,000 tags	
Cloth bags (12 x 140) = 1,680 =	2,000 cloth bags	
Florist stakes (10 x 140) = 1,400 =	1,5000 florist stakes	
Four foot bamboo stakes per crew (12 x 3) = 36 =	50 four foot bamboo stakes	
Steel frames (12 x 1) = 12 =	15 frames	
Hammers to drive florist stakes into the ground (12 x 1) = 12 =	15 hammers	
4 machine crews	4 persons per crew	
Battery operated scales capable of weighing up to 2,000 grams to two decimal places	4 scales	
Battery operated moisture meters	4 meters	
Mechanical, portable, threshing machines	4 machines	

MAIN (Steel) FRAME

Top View



Side view of side arm -- note the last 4" (beyond the pins) are turned up and pointed

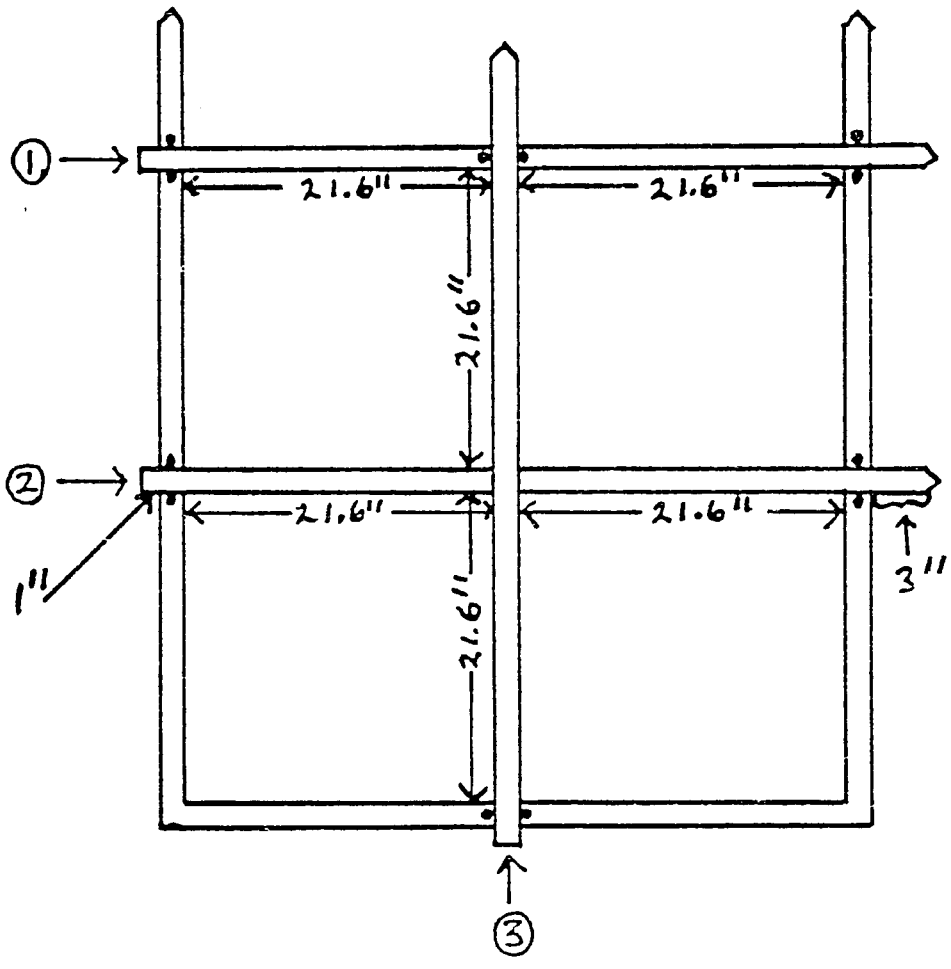


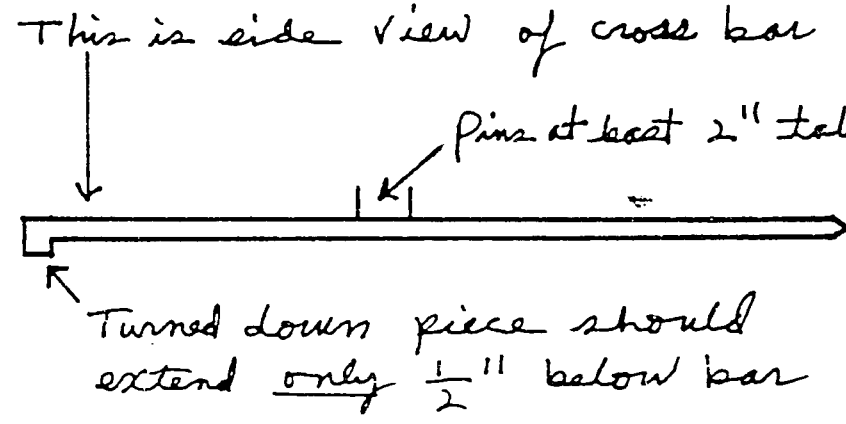
All 10 pins shown on this MAIN (steel) FRAME must be at least 2" tall, and properly positioned to form precise 21.6 square inch partitions.

Both corners need to be re-inforced with a double layer of steel for strength. However, there should not be any overhang on either the inside or outside of the frame

CROSS BARS

Top View





NOTE: All three cross bars should have pointed ends. The pins on cross bar 1 should be properly centered to form precise 21.6 square inch partitions. The ends of all three cross bars should extend beyond the frame at least 1" on one side and 3" on the other -- see cross bar 2 for illustration of this

United States
Department of
Agriculture

National
Agricultural
Statistics
Service

Estimates
Division

SMB Staff Report
Number SMB-90-02

August 1990

The New Objective Yield Models for Corn and Soybeans

Thomas R. Birkett

The New Objective Yield Models for Corn and Soybeans. Thomas R. Birkett, Estimates Division, National Agricultural Statistics Service, U. S. Department of Agriculture, Washington D. C. 20250, August 1990, Staff Report No. SMB-90-02.

Abstract

New objective yield regression models for soybeans and corn were adopted in January, 1990 for implementation in August 1990. The new models have as input the same data as the old models, but they use this data more efficiently to make a many fold increase in accuracy over the old models. The new models are state level models, rather than plot level models. In addition, a regional model sets a regional estimate, and then the state models are constrained to produce forecasts that are consistent with the regional model. This paper presents the linear model framework for this model and defines the soybean and corn objective yield variables to which it is applied.

I. CORN AND SOYBEAN OBJECTIVE YIELD VARIABLES

a. General

In 1990 new operational models will be introduced for forecasting soybean and corn net yield and will become the official operational models in 1991. The older operational models will be phased out. The new models are many times more accurate than the old models and will greatly increase the value of the objective yield survey to each State's estimating program. The new models are also less complicated and consequently easier to analyze and present. They were built using ideas from multiple sources, combined in a classical linear model framework.

b. Variables in the Regional Models

For the region made up of all the states in the corn or soybean objective yield surveys, a model for the regional yield is constructed for each month. In general each monthly model has one independent variable X. The form is

$$Y = \alpha + \beta X$$

where

Y = regional ASB yield

α, β are the unknown model parameters, to be estimated from historical data.

The independent variable X in each model varies by month. For soybeans X is a function of the following objective yield variables.

SOYBEAN VARIABLES

MONTH :	Independent variable
August	estimated number of lateral branches per 18 square foot
September	estimated number of pods with beans per 18 square foot
October-December	estimated net yield from current objective yield harvested samples and estimated number of pods with beans per 18 square foot from unharvested samples

For corn the independent variable is a function of the following variables.

CORN VARIABLES

MONTH :	Independent variable
August	stalks with ears and ears with kernels per square foot, average kernel row length per ear
September	ears with kernels per square foot, average kernel row length per ear
October-December	estimated net yield from current objective yield harvested samples, and ears with kernels per square foot and average kernel row length per ear from unharvested samples

Details on the estimation of net yield from harvested samples and the definition of forecasting categories for unharvested samples (see below) is presented in the S & E sections 8.4 and 8.6.

Soybeans

For soybeans only samples from maturity categories 2-6 (1-6 in the southern states) are used in the construction of X in August. In September only samples falling in categories 6-9 are used. From October on only category 6-10 samples are used. For the period 1980-1989 the following table shows the percentage of status 1 samples that were included in the models.

% SAMPLES USED IN MODELS (1980-1989)

	August	September
Arkansas		49
Illinois	88	96
Indiana	79	94
Iowa	93	99
Kansas		68
Louisiana		71
Minnesota	88	98
Mississippi		51
Missouri	77	75
Nebraska		98
Ohio	75	93
Region	85	88

From October on virtually all status 1 (sampled) and status 4 (harvested) samples are included.

Algebraic definitions for X

August

In August the algebraic formula for estimated number of lateral branches per 18 square feet is

$$lat_i = \frac{1}{n_i} \sum_{J_i} pl_{ij} lat_{ij}$$

where

J_i is the subset of samples classified in maturity categories 2-6 (or 1-6 in the southern states), state i.

n_i = the number of samples $\in J_i$

pl_{ij} = plants per 18 square feet for sample j, state i

lat_{ij} = lateral branches per plant, sample j, state i

lat_i = state i estimate lateral branches per 18 square feet.

The state level estimates are combined to the regional level with current ASB acres for harvest as the weight.

$$X = \frac{\sum_I a_i \, lat_i}{\sum_I a_i}$$

where a_i is the ASB acres for harvest for state i and I is the set of states in the survey.

September

The formula for estimated number of pods per 18 square feet is

$$pod_i = \frac{1}{n_i} \sum_{J_i} pl_{ij} pod_{ij}$$

where

J_i *is the subset of samples classified in maturity categories* 6-9 *, state i.*

n_i = *the number of samples* $\in J_i$

pl_{ij} = *plants per* 18 *square feet for sample j, state i*

pod_{ij} = *pods with beans per plant, sample j, state i*

pod_i = *state i estimate pods with beans per* 18 *square feet.*

The regional level estimate is

$$X = \frac{\sum_I a_i \, pod_i}{\sum_I a_i}$$

where a_i *is the ASB acres for harvest for state i and I is the set of states in the survey.*

October-December

After September X is the average of the yields from the harvested samples and the predicted yields from the unharvested samples. At this time the predicted yields are those from the current operational models. The models in categories 6-10 have pods with beans as the independent variable.

$$X_i = \frac{1}{n_{ih}+n_{ik}} \left(\sum_{J_{ih}} Y_{ij} + \sum_{J_{ik}} \hat{Y}_{ij} \right)$$

where

J_{ih} *is the subset of samples classified in maturity category* 10 *(harvested), state i.*

J_{ik} *is the subset of samples classified in maturity* 6-9 *(unharvested), state i*

n_{ih} = *the number of samples* $\in J_{ih}$

n_{ik} = *the number of samples* $\in J_{ik}$

Y_{ij} = *harvested yield for sample j, state i*

$\hat{Y}_{ij}$ = *forecasted yield, sample j, state i.*

The regional estimate is

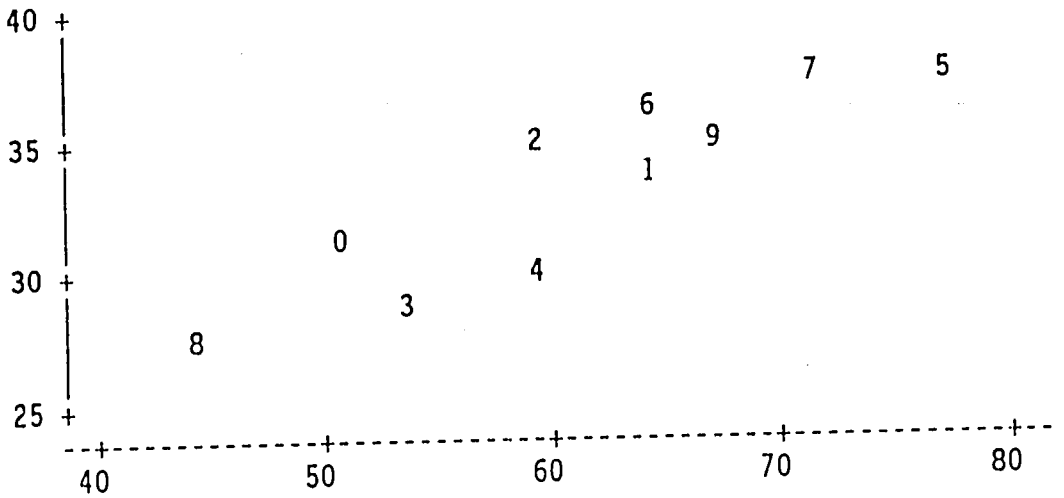
$$X = \frac{\sum_I a_i X_i}{\sum_I a_i}$$

where a_i *is the ASB acres for harvest for state i and I is the set of states in the survey.*

A plot of the ASB yields versus the values of X for 1980-1989, August-December, follows. (Note: there has been some movement of states' in and out of the program during the 10-year period depicted in the plot. The X and Y variables represent whichever states were in the program in respective years).

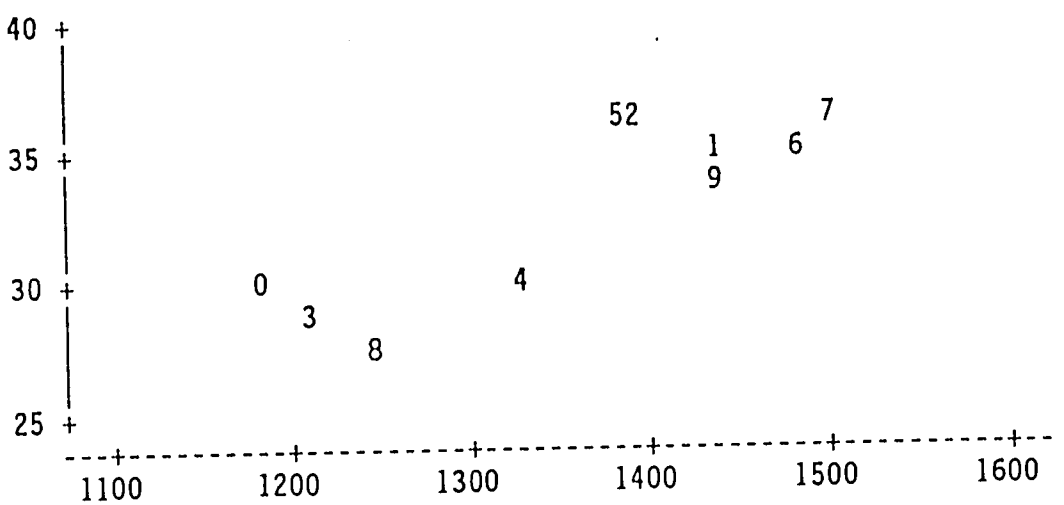
SOYBEANS

AUGUST



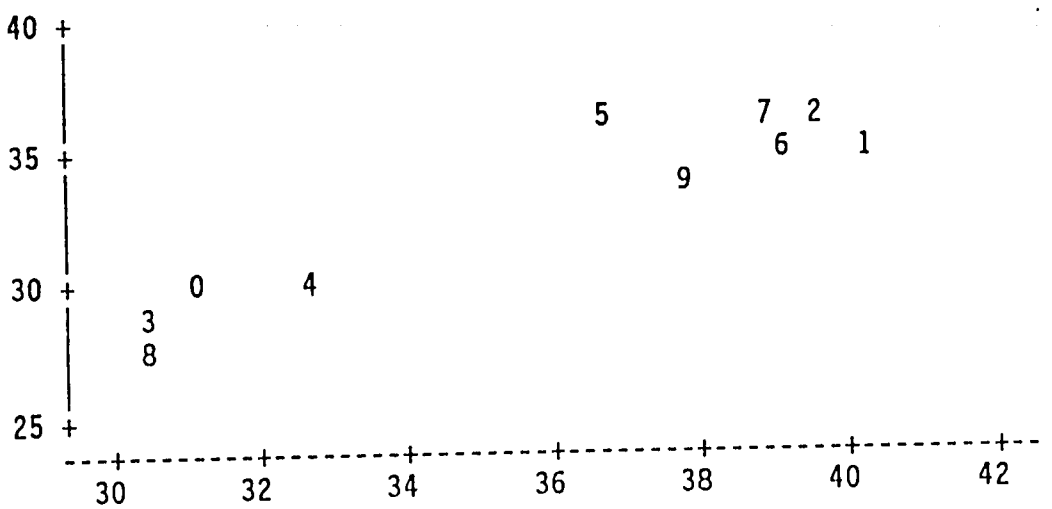
R^2 = .80 lateral branches per 18 square feet

SEPTEMBER



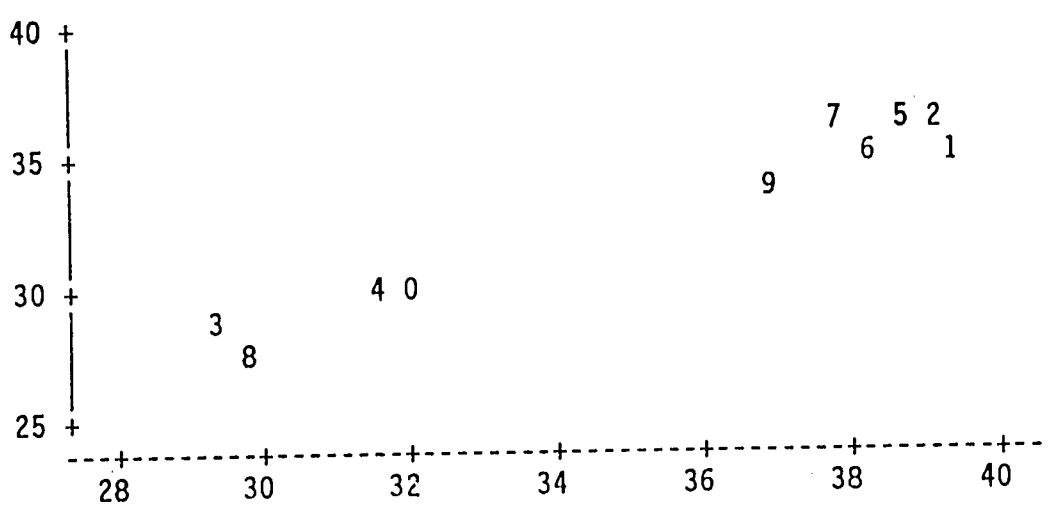
R^2 = .75 Pods with beans per 18 square feet

OCTOBER



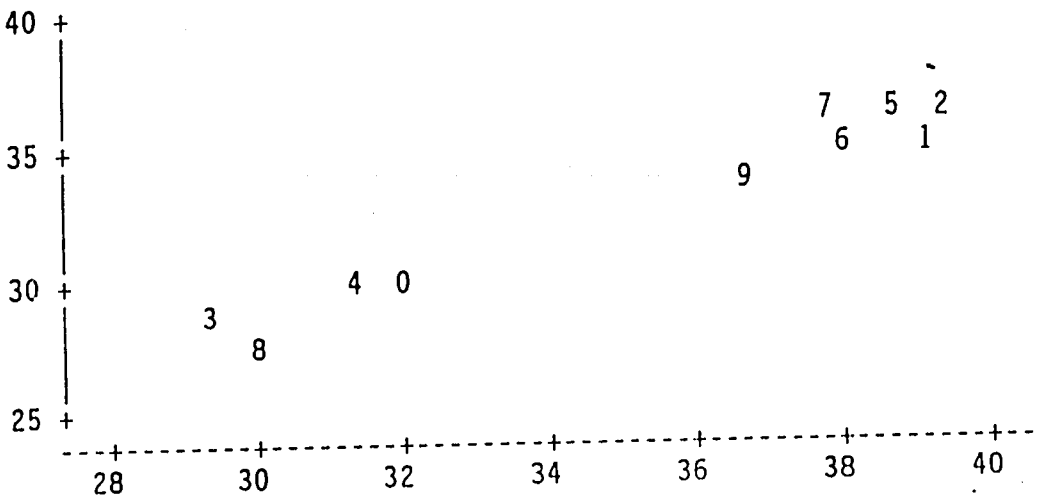
R^2 = .98 f(current net yield from objective yield harvested samples, pods with beans from unharvested samples)

NOVEMBER



R^2 = .98 Current net yield from objective yield harvested samples

DECEMBER



R^2 = .98 Current net yield from objective yield harvested samples

Corn

For corn only samples from maturity categories 3-7 are used in the construction of X. In August and September samples falling in 3-6 are used, and from October on only 3-7's are used. The following table provides percentages of the status 1's that were included in the models for the (1980-1989) period.

% SAMPLES USED IN MODELS (1980-1989)

	August	September
Illinois	28	99
Indiana	21	97
Iowa	13	99
Michigan	2	94
Minnesota	8	97
Missouri	52	98
Nebraska	12	98
Ohio	12	94
South Dakota	4	95
Wisconsin	4	94
Region	19	97

From October on virtually all status 1 (sampled) and status 4 (harvested) samples are included in the models. Because of the small percentages in Michigan, Minnesota, South Dakota and Wisconsin, samples from those states are automatically excluded from the August variable regardless of maturity.

August

In August the algebraic formula for X is algebraically more involved than it is for soybeans, because X is a function of two count variables and a size variable.

$$X = \left\{ \sum_{I} \frac{a_i}{\sum_{I} a_i} \left[\frac{1}{n_i} \sum_{J_i} \frac{(se_{ij} + ek_{ij})}{\frac{15}{4} r_{ij}} \right] \right\} \chi$$

$$\left\{ \sum_{I} \frac{a_i}{\sum_{I} a_i} \left[\frac{1}{n_i} \sum_{J_i} \frac{(se_{ij} + ek_{ij})\overline{krl}_{ij}}{\frac{15}{4} r_{ij}(se_{ij} + ek_{ij})} \right] \right\}$$

where

a_i = *current ASB acres for harvest, state i*

I = *the set of states in the survey*

J_i = *the set of samples in maturity 3-6, state i*

n_i = *the number of samples in* J_i

se_{ij} = *the number of stalks with ears, sample j, state i*

ek_{ij} = *the number of ears with kernels , sample j, state i*

r_{ij} = *four row space measurement, sample j, state i*

$\overline{krl}_{ij}$ = *average kernel row length, sample j, state i.*

September

In September it is

$$X = \left\{ \sum_{I} \frac{a_i}{\sum_{I} a_i} \left[\frac{1}{n_i} \sum_{J_i} \frac{(ek_{ij})}{\frac{15}{4} r_{ij}} \right] \right\} \chi$$

$$\left\{ \sum_{I} \frac{a_i}{\sum_{I} a_i} \left[\frac{1}{n_i} \sum_{J_i} \frac{(ek_{ij})\overline{krl}_{ij}}{\frac{15}{4} r_{ij}(ek_{ij})} \right] \right\}$$

where

a_i = *current ASB acres for harvest, state i*

I = *the set of states in the survey*

J_i = *the set of samples in maturity 3-6, state i*

n_i = *the number of samples in* J_i

ek_{ij} = *the number of ears with kernels, sample j, state i*

r_{ij} = *four row space measurement, sample j, state i*

$\overline{krl}_{ij}$ = *average kernel row length, sample j, state i.*

October-December

After September X is the average of the yields from the harvested samples and the predicted yields from the unharvested samples. At this time the predicted yields are those from the current operational models. The models in categories 3-6 have ears with kernels and average kernel row length as the independent variables.

$$X = \sum_{I} \frac{a_i}{\sum_{I} a_i} \left[\frac{1}{n_{ih} + n_{ik}} \left(\sum_{J_{ih}} Y_{ij} + \sum_{J_{ik}} \hat{Y}_{ik} \right) \right]$$

where

a_i = *ASB acres for harvest, state i,*

J_{ih} *is the subset of samples classified in maturity category 7 (harvested), state i.*

J_{ik} *is the subset of samples classified in maturity 3-6 (unharvested), state i*

n_{ih} = *the number of samples* $\in J_{ih}$

n_{ik} = *the number of samples* $\in J_{ik}$

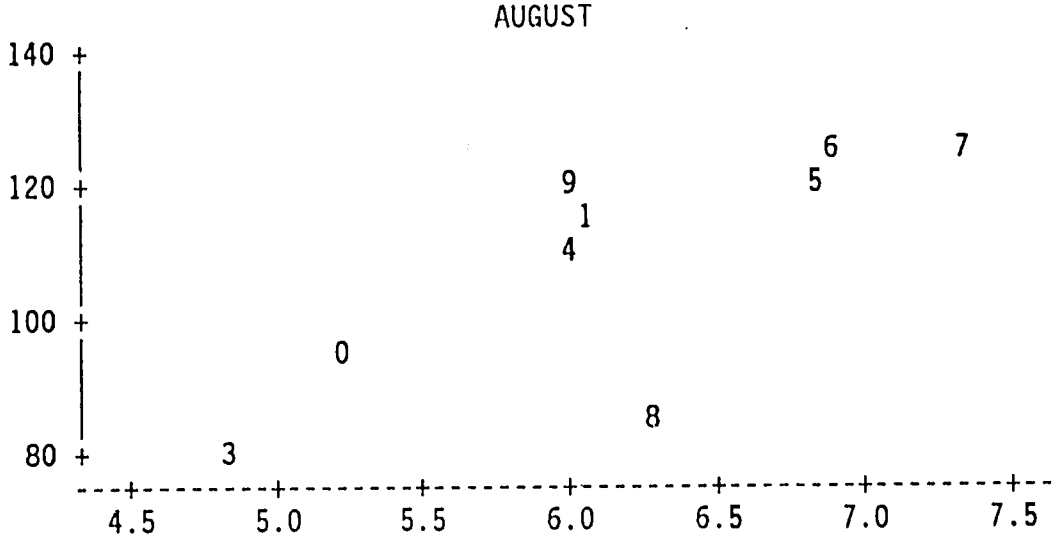
Y_{ij} = *harvested yield for sample j, state i*

$\hat{Y}_{ij}$ = *forecasted yield, sample j, state i.*

Plots of Y vs X for corn follow.

CORN

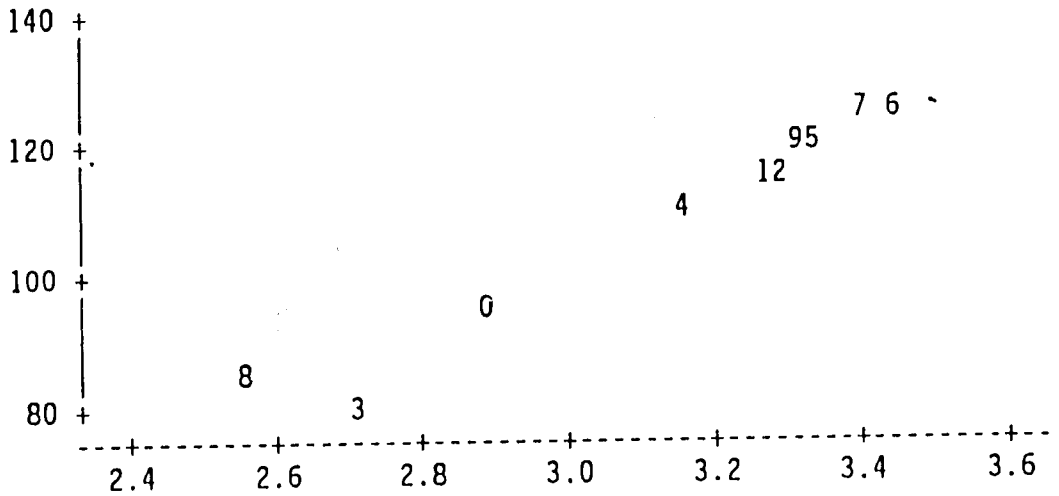
AUGUST



R^2 = .98 f(stalks with ears, ears with kernels, kernel row length)

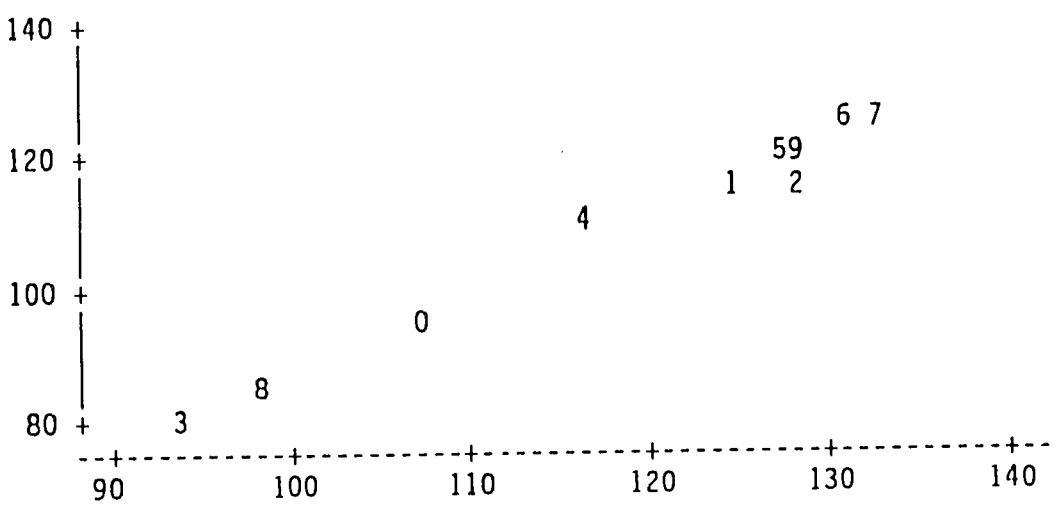
NOTE: 1 obs hidden, and 1988 is an outlier not included in the August model. This model has as independent variables X and X^2.

SEPTEMBER



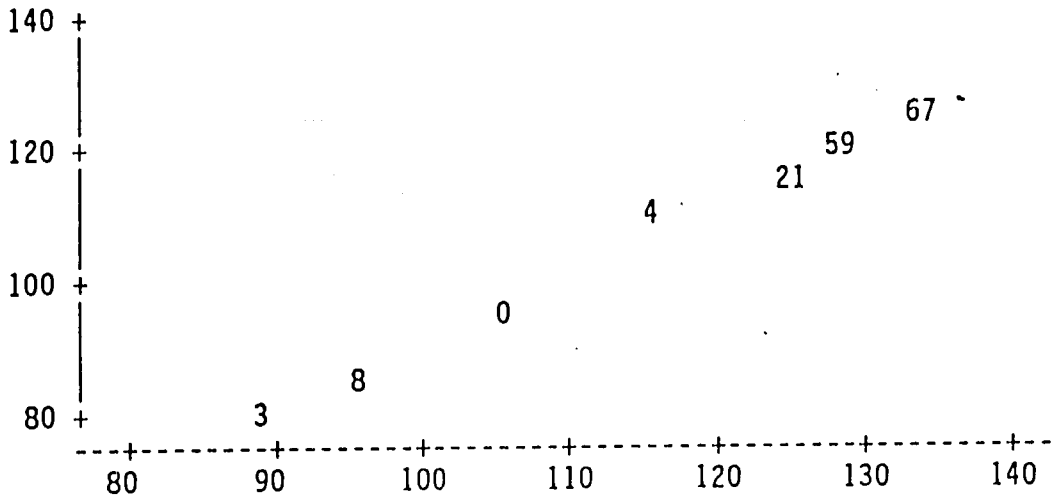
R^2 = .97 f(ears with kernels, kernel row length)

OCTOBER



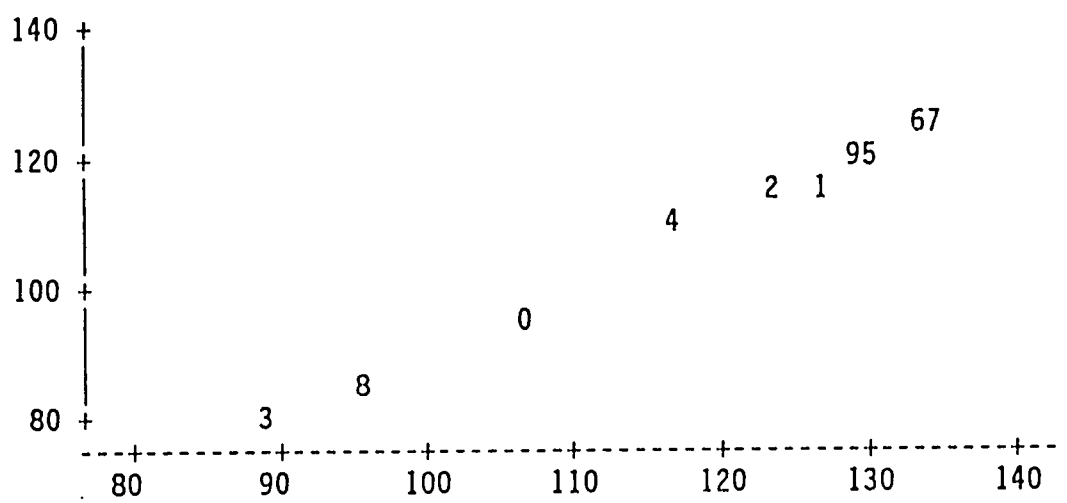
R^2 = .98 f(current net yield from objective yield harvested samples, ears with kernels and avg kernel row length from unharvested samples)

NOVEMBER



R^2 = .98 Current net yield from objective yield harvested samples

DECEMBER



R^2 = .98 Current net yield from objective yield harvested samples

II. LINEAR MODEL FRAMEWORK FOR THE NEW MODELS

a. State Models

Once the regional estimate is made, state forecasts are developed that are constrained to weight (by ASB acres) to the forecast of the regional model. To accomplish this a global model that predicts all states simultaneously is developed. The parameters of this model are estimated subject to the linear restriction that the state forecasts weight to the regional forecast.

The theory behind the state models does not require that the state models use the same independent variables as the regional models, only that they weight to the regional forecast. The key is to use independent variables that do the best job of making good state forecasts. It turns out that in August the objective yield variables are too weak at the state level to generate reliable estimates, although there is enough information to generate regional estimates. August is the only month where the state independent variables are not the same as the regional independent variables. For both corn and soybeans the August state independent variable is a function of crop condition percent as of the last week in July, obtained through the crop weather. X_i in August is

$$X_i = \frac{(vp) + 2(p) + 3(f) + 4(g) + 5(ex)}{15}$$

where

vp = percent very poor
p = percent poor
f = percent fair
g = percent good
ex = percent excellent

as of the last week in July.

Of course each crop has a separate set of conditions.

b. Details of the Constrained State Linear Models

Definitions

The individual state models can be written as a global state model using matrix notation. First we will define the various matrices that go into the model for a given month. For a specific example we will use the soybean objective yield

survey.

Let

X_i = the data matrix for the linear model for state i, i=5,17,18,19,20,22,27,28,29,31,39, the states in the survey.

X_i will be N_i x 2 where N_i is the number of years in the historic data base. In most cases X_i will have a column of 1's and one column with independent variable values, x_i.

$$X_i = [1 \; x_i]$$

This also indicates that the state parameter vector β_i is generally 2 x 1.

Recall that for soybeans $\underline{x}_i$ is crop condition percent (as of the last week in July) for August, pods with beans in September, and a combination of current net yield from harvested samples and pods with beans from unharvested samples for October-December.

In September 1990 X_{20} will be 1 x 1 and will contain the 1980 observed value for pods with beans per 18 square foot. This will force β_{20} to be 1 x 1 also. (1980 is the only history available for Kansas).

For corn the state variables ($\underline{x}_i$'s) are crop condition percent (as of the last week in July) for August, a function of ears with kernels and kernel row length in September, and a combination of current net yield from harvested samples, and ears with kernels and average

kernel row length for unharvested samples, for October-December.

For both crops the crop condition variable goes back to 1985, while the objective yield variables extend back to 1980.

For the dependent variable let

$\underline{Y}_i$ = the dependent vector of ASB net yield for state

Analogously, for the regional model let

X = the data matrix for the regional model, and

$\underline{Y}$ = the dependent vector of ASB net yield for the region.

The rows of the X's and $\underline{Y}$'s represent each of the years 80-89 (although some states are missing some years, corresponding to years when the survey was not done in those states).

The Global Model

To create the global state model vertically concatenate the $\underline{Y}_i$'s and create a block diagonal matrix of the X_i's to form

$$\begin{bmatrix} \underline{Y}_5 \\ \cdot \\ \cdot \\ \cdot \\ \underline{Y}_{39} \end{bmatrix} = \begin{bmatrix} X_5 & & & \\ & X_{17} & & \\ & & \ddots & \\ & & & X_{39} \end{bmatrix} \begin{bmatrix} \beta_5 \\ \cdot \\ \cdot \\ \cdot \\ \beta_{39} \end{bmatrix} + \begin{bmatrix} \varepsilon_5 \\ \cdot \\ \cdot \\ \cdot \\ \varepsilon_{39} \end{bmatrix}$$

$$\underline{Y}_s = X_s \beta_s + \varepsilon_s$$

To make inferences we will assume

$$\underline{\varepsilon}_s \sim N(\underline{0}, \sigma^2 I)$$

Definitions for Predicting the Current Yea.

In the current year we will observe the values of the independent variables and predict the unknown value of the future Y_{fi}. To this end let

> $\underline{x}_{fi}'$ = the observed $\underline{x}'$ for the current year for state i, before Y_{fi} is known (the f stands for future). Normally $\underline{x}_{fi}' = [1\ x_{fi}]$, where, for example, in September x_{fi} is the value of pods with beans per 18 square feet for the just completed survey for state i.

Another variable we will need is a_i, the current ASB acres for harvest for each state i.

Let

$$A = \sum_I a_i$$

the sum of the state acres, which is the current regional acreage.

The Current Year Constraint on the Parameters

The constraint that the state estimates weight to the regional estimate will take the form of a linear restriction on $\underline{\beta}_s$, specifically $\underline{k}'\underline{\beta}_s = m$.

In particular,

$$k' = \frac{\left[a_5 x'_{f5} \quad a_{17} x'_{f17} \; \cdots\cdots \; a_{39} x'_{f39}\right]}{A}$$

Parameter Estimation Under the Restricted Model

The regional model is

$$\underline{Y} = X\underline{\beta} + \underline{\varepsilon}$$

We estimate $\underline{\beta}$ from the regional model with OLS with

$$\underline{b} = (X'X)^{-1}X'\underline{Y}$$

Then the current year predicted value m for the region is

$$m = x'_f \underline{b}$$

where $\underline{x}_f'$ is the regional $\underline{x}'$ from the current survey.

This is the m that completes the specification of the linear restriction $\underline{k}'\underline{\beta}_s = m$.

For the state models the unrestricted OLS estimate of $\underline{\beta}_s$ is

$$\underline{b}_s = \left(X'_s X_s\right)^{-1} X'_s \underline{Y}_s$$

The estimate of $\underline{\beta}_s$ subject to $\underline{k}'\underline{\beta}_s = m$ is

$$b_{rs} = b_s - (X_s'X_s)^{-1} k \left(k' (X_s'X_s)^{-1} k\right)^{-1} (k' b_s - m)$$

Estimation of σ^2 under the Restricted Model

σ^2 is estimated under the restricted model as

$$\hat{\sigma}^2 = \frac{r_{rs}' r_{rs}}{(N_s - p - 1)}$$

where r_{rs} is the vector of residuals under the restricted model.

$$r_{rs} = Y_s - X_s b_{rs}$$

In addition,

N_s = the number of rows in X_s and

p = the number of columns in X_s.

X_s is always constructed to have full column rank, so that all parameters are identifiable. The additional 1 is subtracted from the denominator because of the 1 degree of freedom restriction on the parameter estimates.

The Constrained Predicted Values and Variances

The constrained predicted values for the future Y_{fi}'s then become

$$\hat{Y}_{fs} = \begin{bmatrix} X'_{f5} \\ \\ \\ X'_{f39} \end{bmatrix} b_{xs}$$

The estimated variance of the state predicted values can be found from the diagonal elements of

$$\left\{ \begin{bmatrix} X'_{f5} \\ \\ \\ X'_{f39} \end{bmatrix} \left(X'_s X_s\right)^{-1} \begin{bmatrix} X_{f5} \\ \\ \\ X_{f39} \end{bmatrix} + I \right\} \partial^2$$

c. Forecasting Accuracy

To assess the forecasting accuracy of the new models compared to the old models, the estimated standard error for the 1989 forecast of each is listed in the following table (in bushels per acre). (For the standard error of the old models see Birkett (1990)).

REGIONAL MODELS

MONTH	CORN		SOYBEANS	
	Old	New	Old	New
Aug	16.0	2.9	5.1	1.9
Sept	12.0	3.1	3.8	2.2
Oct	9.2	2.7	3.1	1.2
Nov	9.2	2.6	2.6	0.6
Dec	9.6	2.5	2.6	0.6

While the standard error of September is slightly larger than that of August for both crops, this is considered to be a statistical anomaly that will be reversed with time. Also the August and September soybean models represent different regions.

For the state forecasts the average standard errors under the restricted model for predicting 1989 are listed in the following table.

STATE MODELS

MONTH	CORN	SOYBEANS
Aug	6	3.3
Sept	6	3.5
Oct	5	2.7
Nov	4.5	2
Dec	4.3	1.7

References

Birkett, T.R. (1987), "A Production Forecasting Model for Corn", National Agricultural Statistics Service, SRB-87-02, Washington, DC

Birkett, T.R. (1990), "The General Linear Model Framework for the Objective Yield Models with Suggested Improvements", National Agricultural Statistics Service, SMB-90-01, Washington, DC

Objective Yield Survey S & E Manual, (1990), National Agricultural Statistics Service, Washington, DC, Sections 8.2 and 8.6

Searle, S.R. (1971), Linear Models, New York: John Wiley & Sons, Inc.