A Computing Environment for Spatial Data Analysis in the Microelectronics Industry

Mark H. Hansen and David A. James
Statistics and Information Analysis
Bell Laboratories, Murray Hill

December 6, 1996

Abstract

This paper describes a computing environment called S–wafers tailored for the analysis of spatial data collected during semiconductor manufacturing processes. At the core of S–wafers lies a new statistical methodology that systematically exploits the basic spatial nature of these data. S–wafers builds on the experience of Lucent Technologies Microelectronics engineers and Bell Laboratories researchers who have noticed that patterns in mapped wafer data can provide "signatures" that can be used to help identify and correct process problems. The S–wafers environment provides the means to formalize these observations into implementable strategies by providing extensive tools for data visualization, the identification of groups of similarly patterned wafers, the spatial analysis of designed experiments, the assessment of the impact of particle contamination, and much more. In addition to supporting statistical research into problems of process improvement in this area, S–wafers has been used for over two years by various groups in Lucent Technologies Microelectronics Division. Extensions to both the software environment and the underlying statistical methodology continue at a fast pace.

1 A Rich Computing Environment for a Complex Process

Manufacturing processes in many high-technology industries are becoming increasingly complex. One manifestation of this complexity is the need to collect and analyze vast amounts of data for monitoring and control. Typically, however, effective strategies for extracting useful information from these data require sophisticated computing tools. In the context of semiconductor manufacturing, for example, a successful program of process improvement must routinely digest large quantities of intrinsically *spatial* data. In this paper, we will focus mainly on the fabrication of integrated circuits (IC's) and other microelectronic devices, but our approach to software development could easily be applied the manufacture of optical fiber, flat-panel displays, hard disk platens, and circuit board assembly.

The fabrication of IC's (or chips) is a costly process involving hundreds of separate steps and lasting up to two months. Several hundred chips are fabricated simultaneously on a *wafer*, and the wafers themselves are processed in groups called *lots*. Each week, thousands of wafers enter production in Lucent Technologies' Microelectronics manufacturing facilities. As these wafers progress through the line, large amounts of data are collected on each wafer, including shop flow information and the results of various in-line measurements. After processing is completed, each chip on every wafer is subjected to a number of functionality tests and is declared either good or defective. This last type of data are called *final probe* data.

Very few of the manufacturing processes leading to these final probe tests have uniform effects on all parts of the wafer. Failure mode analysis (FMA) engineers within Lucent have known for some time that the spatial patterns in final probe data contain valuable information for identifying likely causes of chip loss. In addition, by understanding "normal" patterns in properly tuned processes engineers can better anticipate and correct problems early. Driven in part by the limitations of available software and the overwhelming production volume, however, traditional data analysis in this area has been based on simple one-dimensional summaries like *yield*, the proportion of good chips on a wafer. While process problems ultimately manifest themselves in yield loss, analyzing the spatial configurations of failing chips in all wafers that pass through the line is an efficient and cost-effective technique for monitoring and control. In general, we have found that by respecting the spatial character of the measurements taken during fabrication we can begin to untangle the complex effects in the manufacturing line.

Working closely with product, process and FMA engineers, we have used this idea to create a software environment S–wafers for computing with the spatial data collected during the manufacture of microelectronic devices. Written in the S language [1, 2, or the Appendix], S–wafers provides tools that range in complexity from simple wafer visualization to specialized statistical computations that extract spatial patterns from mapped wafer data and identify likely root causes. Three ingredients make possible the development of this environment: a powerful graphical and analytic engine provided by the underlying S language, the encapsulation of spatial information into objects in the language, and the definition of methods for manipulating these new classes of objects. Although the range of specialized analyses that are possible with S–wafers is fairly extensive, an object-oriented approach allows users to invoke familiar S commands that extend automatically to use the spatial information. The functionality of the S–wafers system includes:

- visualization of spatial wafer data;

- assessment of complete spatial randomness of defects and other data;
- estimation of area defects and background defect densities from final probe data (these quantities are denoted by Y_0 and D_0 in the semiconductor manufacturing literature);
- classification of wafers according to defect patterns;
- correlation of groups of patterned wafers to other in-line and process data;
- visualization of bitmap defects collected from memory arrays, patterns of failed zone testers, and other point defect data (e.g., particles scans);
- spatial analysis of data from designed experiments, e.g., spatial analysis of variance and regression models;
- specialized non-parametric modeling including classification and regression trees as well other multivariate techniques, such as principle components, clustering, and spatial smoothing; and
- bivariate spline-based estimation of defect densities.

In addition to a language-based system, Lorraine Denby, a colleague in the Statistics and Information Analysis Department, has prototyped a Visual Basic [9] front-end to the S–wafers software that runs on personal computers and implements a wide range of standard analyses[4].

In the next section, we will introduce two applications that illustrate the process by which engineers familiar with traditional, non-spatial data analysis in the S environment used the functionality in S–wafers to more fully exploit the spatial information contained in the data collected at final probe. In these examples, we track the progression from exploratory data analysis based on the visualization tools in S–wafers, to the development and ultimate incorporation of new statistical techniques that more effectively address the underlying engineering questions. After this motivation, Section 3 takes us behind the scenes of S–wafers to reveal its underlying software design and implementation. In Section 4 we state our conclusions, and in Section 5 we list our collaborators both from Bell Labs and Lucent Technologies' Microelectronics Division.

2 Data Analysis in the S–wafers Environment

We begin this section with a brief description of some of the basic tools for visualizing and manipulating mapped wafer data. After this introduction, we present two examples in which the application of such simple tools highlights deficiencies in traditional analysis techniques. These observed shortcomings have in turn motivated new statistical methodologies and ultimately S–wafers functionality. Our goal in discussing this material is to demonstrate how the S–wafers software environment has been used not only to support data analysis and statistical research in quality and process improvement, but also to transfer these new ideas to the factory. In each example, we will focus our attention on post-production data where either yield or a transformation of yield is the metric of interest.

2.1 S–wafers Basics

Final probe test results for a single wafer can be displayed as a *wafer map*. In the top row of Figure 1, we present an actual wafer and two software equivalents: In each map, good chips are represented by white squares, while the colors of the remaining squares encode the index of the test failed by the corresponding chip. The black and white map on the right in this figure is a *binary wafer map*, and records only the distinction between good and defective chips. In the bottom row of Figure 1, we present several other types of data that can be analyzed in the S–wafers environment. The wafer on the left is a chip-level summary of an optical, in-line particle scan, shaded according the the number of defects found on each chip. In the center map, we display the results of a series of tests applied to the memory array on each chip. The various black marks represent the locations of failed single bits, addresses and data rows. Finally, the rightmost wafer in the second row of this figure is a color contour plot representing *parametric* test results that in this case were taken just prior to final probe. Values lower than the median measurement are colored in deepening shades of red, while test results that are higher than the wafer median area colored in deepening shades of blue.

These examples demonstrate the variety of mapped wafer data that is collected during the fabrication process. As we will see, the S–wafers environment allows the user to manipulate these different data types in a consistent and natural way. Computationally, wafer objects consist of measurements taken on physical wafers that are associated with a set of spatial coordinates. These measurements may be numerical, categorical (e.g., failure modes “A”, “B”) or logical (sites where a condition is satisfied). Often, the spatial coordinates are simply an index referring to one or more of the available chip sites. Slightly more complicated geometries arise when considering post-production, parametric test data where measurements are recorded on special structures between the chips. Bitmap data is typically categorical, indicating for example whether a bit failed high or low, while the associated spatial position could easily index both chip and location within the memory array. In the case of particle data, the optical scanning instrument records not only the absolute position of the particle on the wafer but also some estimate of its size. To compare particle data with bitmap or final probe results, for example, it is often necessary to translate these spatial coordinate systems to tally particle counts per chip or to identify bitmap failures that can be attributable to particles seen in-line. Providing a convenient software environment that allows

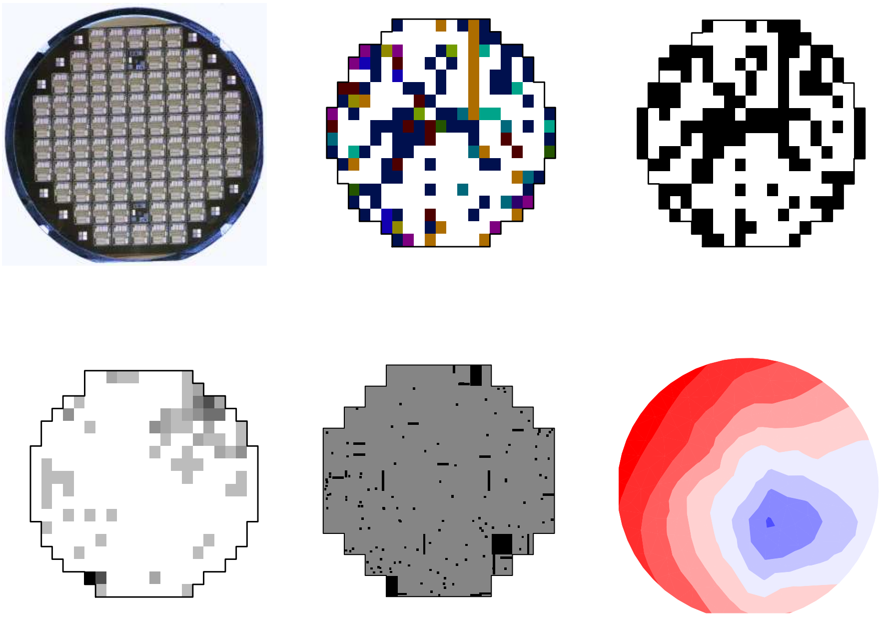


Figure 1: *Instances of wafer objects. The image in the upper left hand corner is an actual wafer map, while the other five panels display various types of mapped wafer data that are collected routinely during the manufacturing process. The two remaining maps in the top row are two representation of final probe data: a binned probe map in which colors are used to encode the various failure modes and the associated binary probe map obtained from the binned map by coloring any defective device black. In the bottom row, moving from left to right, we present a particle map in which the grey level encodes the square root of the numbers of particles found on each device; a bitmap of each chip's memory in which dark areas denote failing single bits, addresses and data rows; and finally a contour plot displaying one set of parametric test results.*

engineers to both formulate and answer questions about the various types of data collected during the fabrication process is the goal of the S–wafers project.

To this end, for each class of wafer objects we have defined methods for printing, plotting, computing numerical summaries, and various "selection" operators to extract either individual wafers from a collection of wafers, or a subset of chip measurements according to the wafer geometry. The plots in Figure 1, for example, were each generated with a call to the function plot() and were as easy to generate as a scatter plot in S. Another effective device for exploratory data analysis is the points() function which can be used to overlay mapped wafer data onto an existing scatter plot. In Figure 2, we present a standard plot of wafer yield as a function of cassette number and position within the cassette, which describes the order in which wafers were processed at a particular step. Engineers might examine a graph of this type to identify one or more machines that might be responsible for an observed yield loss. To this graph we have added wafer maps,

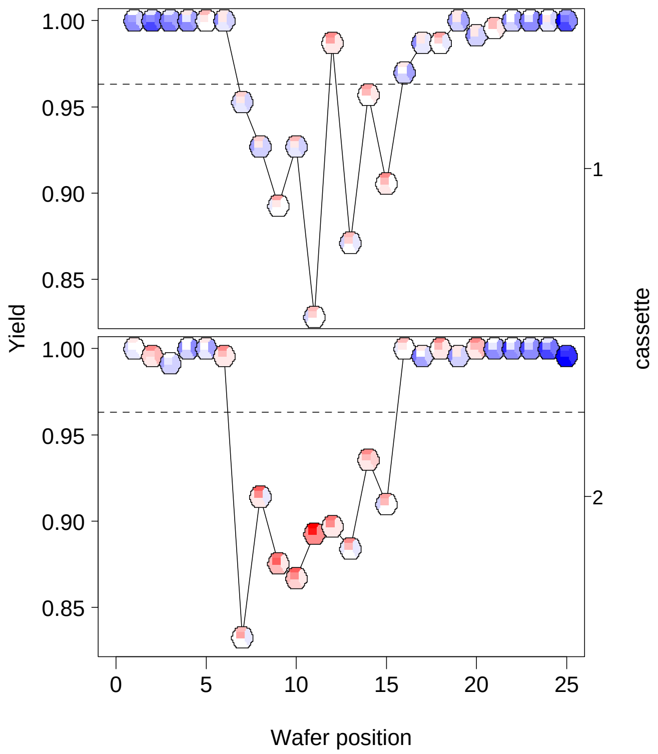


Figure 2: *The top panel shows wafer yield as a function of wafer position in processing enclosure (cassette) 1 during one step; the plotted color wafer maps depict the value of an electrical test with a color scale that goes from deep red to indicate low values though white that indicates target value, to dark blue to indicate large values. The bottom panel similarly shows data for a second cassette. From this plot we see a significant drop in yield in adjacent wafers located in positions 6 though 15 in both cassettes (this locations correspond, roughly, to the middle of the cassette).*

which in this case correspond to a post-production, parametric test that was taken at five sites on each wafer. In the next section, we will encounter several examples in which this technique of superimposing wafer maps onto standard plots of simple wafer summaries like yield is used to incorporate spatial information into an existing analysis. Augmenting a plot in this way allows us to assess the appropriateness of a given technique, and often suggests a variety of more effective alternatives.

2.2 Two Examples

To simplify our discussion, we focus for the moment on binary wafer map data where each chip is classified as either good or defective. In general, the pattern of defective chips on a given wafer map is caused by a superposition of two types of destructive mechanisms. Acting independently, the first produces clusters or spatial patterns of defects, while the second generates defects randomly across the wafer. This model is depicted graphically in Figure 3. In terms of traditional statistical process control terminology, the first mechanism can be viewed as having a special or assignable cause while the second is said to have a random or common cause. The first component impacting wafer yield can be particle or process related, with particle related clusters being assignable to individual machines, and process related clusters being attributable to one or more process steps not meeting specification requirements. Spatially random defects, on the other hand, are often the product of the cleanroom environment: Random defect densities tend to rise and fall with the overall cleanliness of the cleanroom, and can only be reduced through a program of long term, gradual improvement or expensive equipment overhaul.

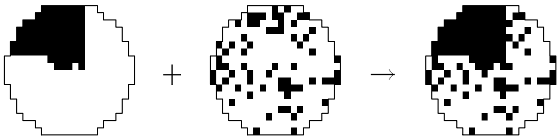

(1)

Figure 3: *The superposition of two destructive forces acting to produce a final binary wafer map. The leftmost wafer represents a patterned or clustered defect, while the map on the right depicts defects that occur in a spatially random fashion.*

This decomposition into clustered and random defects is the basis of our approach to process monitoring and quality improvement. It is unique in that we take advantage of the spatial information in binary probe data, the shape, location, and size of clustered defects, to identify manufacturing problems and their root-causes. The S–wafers environment provides the means to apply this powerful idea to production data on a routine basis. In the next two examples, we illustrate our approach and demonstrate the flexibility of the S–wafers environment to support both effective data visualization and statistical research, as well as the transfer of new methodologies to the factory. We begin each example by conducting an analysis based on one-dimensional summaries of binary wafer map data. For ease of exposition, we will focus on wafer yield, but other metrics like the *good-to-functional ratio* or the *percent-of-planned-yield* could be treated similarly. We then illustrate how spatial information can easily be incorporated by making use of the powerful S–wafers tools for manipulating and visualizing mapped wafer data. As we will see, this type of exploratory data analysis is extremely efficient and can provide a much deeper understanding of how summaries like yield vary in time or as changes in the process are introduced. Because this type of investigation inevitably leads to questions that require more formal statistical procedures, we conclude each example by demonstrating how insights gained from simple visualization techniques led to new procedures. The resulting specialized computations were developed within the S–wafers environment and serve to enrich the toolset available to our users. It is important to recognize that each of the three stages of analysis discussed in these examples is carried out easily and naturally within the S–wafers package.

2.2.1 Monitoring Probe Data

Typical process monitoring efforts will track some function of wafer yield summarized by lot. In Figure 4 we have generated one such plot, tracking the average daily wafer yield of a particular product for a two week period. Because yield data is proprietary, we have left off the axis labels, but have made sure that the vertical scales in each of the next three plots are identical. The grey bars highlight three days for which the probed lots had lower than expected yields. The function yield(w) computes the proportion of good chips in each of the wafers stored in the object w. As numerical objects in S, binary wafer maps are created by assigning defective chips the value one and good chips zero. Therefore, yield() returns the proportion of zeros contained in the corresponding binary wafer maps, and the plot in Figure 4 is generated with the call plot(yield(w)).

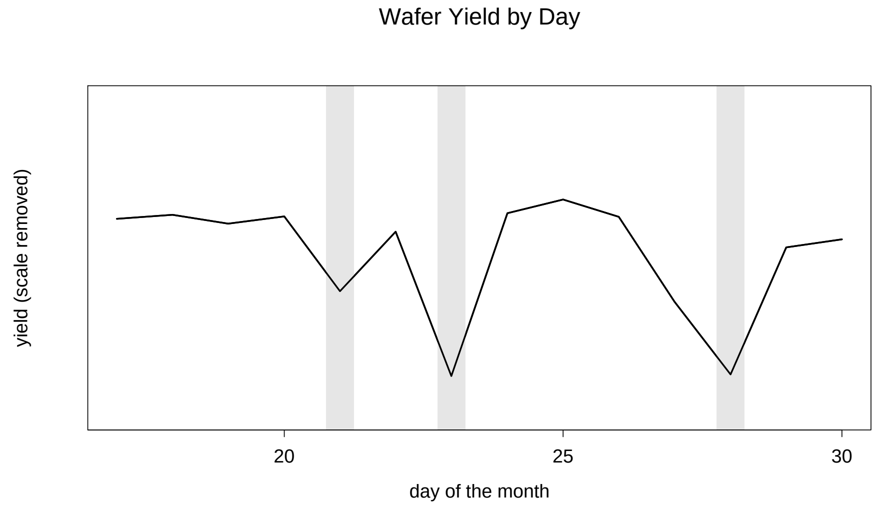
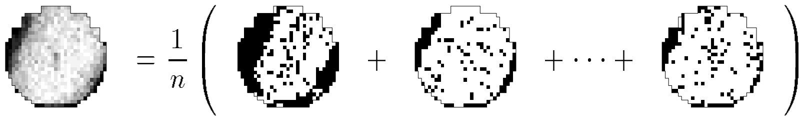


Figure 4: *A simple plot of lot yield averaged by day for the last two weeks of April.*

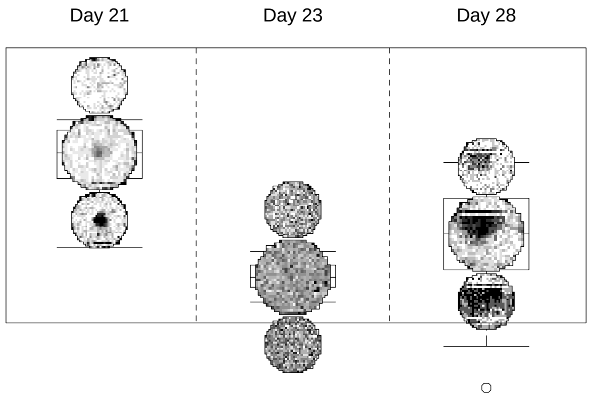
When working with a collection of maps, calculating the averages site-wise and displaying the results in wafer map form is a popular technique for identifying spatial patterns of defective chips common to the group. The resulting *composite* map is shaded according to the proportion of defectives found at each site, with black representing a chip that never yielded and white denoting a site at which all the chips in the group were good.



Like plot(), we have written a wafer-specific method for the S function mean(), so that when applied to wafer objects, this function returns a composite map. To illustrate how effective this simple tool can be, we present boxplots, a standard display in S to depict univariate distributions, of wafer yields for lots probed on the three highlighted days in Figure 4. Over each set of boxes we have placed a triplet of composite wafer maps, representing the site-wise averages calculated for wafers grouped by yield: the uppermost wafer map in each triplet is a composite of the wafers in the top 25% of the yield distribution for a given day, the central maps hold the site-wise average of wafers yielding in the middle 50% of the yield distribution for that day, while wafers with yields below the daily lower quartile are averaged and displayed at the bottom of each triplet.

Below Figure 5, we list the few lines of S code that were used to generate each boxplot. The function wapply() on line four simply applies the function mean() repeatedly for each the three groups of wafers. From these boxplots and composites, we see that chips lost on the first and third days, for example, tend to come from the center of the wafer, although have a much different spatial pattern. By contrast, on the second day, the yield distribution is much tighter and there does not seem to be any obvious patterning to the arrangement of failing chips. Having seen the spatial differences between the wafers probed on the three low yielding days, one is naturally led to ask questions about how to devise more intelligent monitoring procedures that can distinguish between random and spatially clustered yield loss, possibly producing spatial *signatures* of clustered problems.

To understand the *extent* of yield lost to patterned versus random defect mechanisms, S–wafers provides specialized routines for classifying failed chips on a wafer as being either part of a spatial cluster or part of the random yield loss, essentially reversing the arrow in Figure 1. This division of yield is plotted in Figure 6. When applied routinely to daily probe test results, this procedure can be used to flag exceptional lots for FMA engineers to inspect visually. The progression from Figure 4 through Figure 6 represents the three stages of analysis that are possible in the S–wafers environment starting with plots that can be made easily using standard S, enhancing the analysis with the visualization tools in S–wafers, and finally designing specialized functions to implement new statistical procedures. As mentioned above, from the standpoint of process monitoring, the




```
yld <- yield ( w )                                   # compute yield for each wafer in w
p <- quantile ( yld, p = c ( 0, .25, .75, 1 ) )      # compute 0, 25, 75, 100% percentiles
bins <- cut ( yld, breaks = p )                      # split yield into 3 bins
avg <- wapply ( w, bins, mean )                      # compute mean wafers for each bin
x <- boxplot ( yld )                                 # plot boxplots
y <- quantile ( yld, p = c ( .25, .50, .75 ) )       # compute points for wafers
points ( avg, at = list(x=x, y=y) )                  # add average wafers
```

Figure 5: *Adding simple composites to a standard* S *boxplot. The few lines of code given above illustrate how simple it is using* points() *to overlay wafer maps on an existing display. This technique can be extremely effective for exploratory data analysis, prior to more formal statistical modelling.*

ultimate goal of these investigations is to be able to track patterns forming over time. Libraries of patterns are being built to aid process, product and FMA engineers in efficiently uncovering the root cause of process problem. For more details, the interested reader is referred to [7]. This type of specialized routine has been developed for binary probe maps, and the underlying methodology is currently being extended to classify in-line particle scans.

As this example demonstrates, it is through exploratory data analysis (augmented boxplots) that the need for more elaborate statistical procedures (spatial process monitoring) emerges and the statistical problems themselves can be better formulated to improve underlying manufacturing processes. S–wafers provides a framework to express the computations, ranging from simple visualization to the development of new statistical methodology.

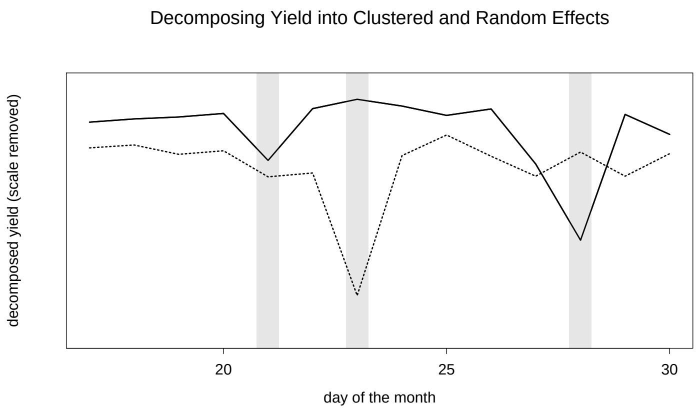


Figure 6: *Tracking spatial and non-spatial components impacting wafer yield. For each day wafer yield is decomposed into contributions from spatially random and spatially clustered defects. The dashed line follows the random component while the solid line tracks the clustered component. This specialized computation can be carried out routinely to identify lots that should be examined visually using the* S–wafers *tools.*

2.2.2 Analyzing Data Collected from a Designed Experiment

Engineers frequently use designed experiments to evaluate the impact of proposed process changes on yield. In this example, we consider an experiment run to uncover optimal levels of factors governing the thickness of a given coating (thin or thick) and the duration of a particular diffusion process (short, medium or long). In Figure 7, we present a graphical summary of the yields for wafers processed in each of the six possible factor settings. In the figure on the right, we plot the average yield of wafers in each cell, connecting the means of common levels of the coating factor. From this analysis, we see immediately that the short diffusion time combined with a thick coating produced the greatest number of chips. To quantify the magnitude of the change in yield attributable to each of these two factors, it is typical to fit a linear model to the data. In this case, we will start by relating yield to coating thickness and diffusion time via the expression:

$$\begin{aligned} \text{yield} \;&=\; \beta_0 + \beta_1 \text{coating} + \beta_2 \text{diffusion} + \beta_3 \text{diffusion}^2 \\ &\quad + \beta_4 \text{coating} \times \text{diffusion} + \beta_5 \text{coating} \times \text{diffusion}^2 \end{aligned}$$

where we code the two ordered factors under study

$$\text{coating} \;=\; \begin{cases} -1 & \text{thin} \\ 1 & \text{thick} \end{cases} \qquad \text{and} \qquad \text{diffusion} \;=\; \begin{cases} -1 & \text{short} \\ 0 & \text{medium} \\ 1 & \text{long} \end{cases} .$$

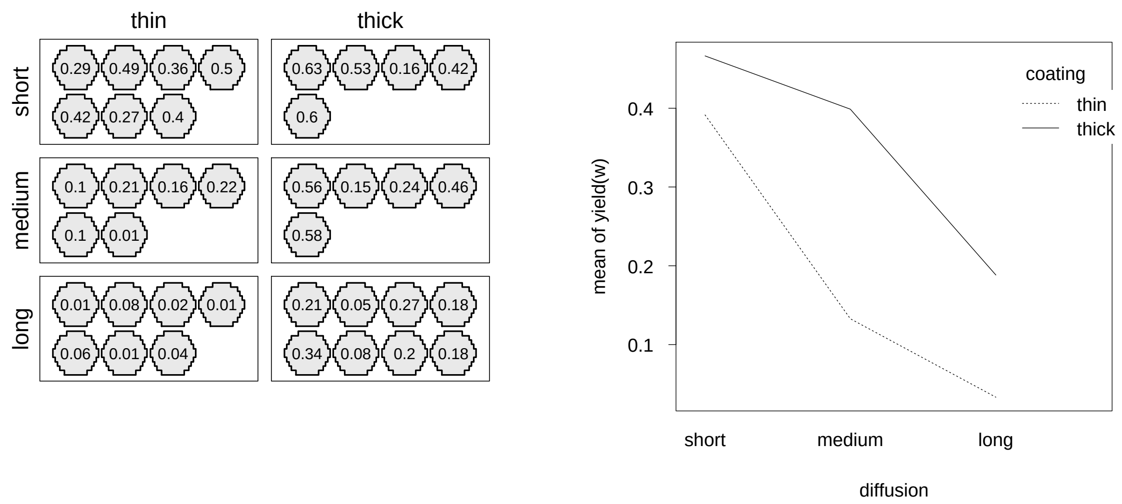


Figure 7: *The left panel shows yields for all wafers in a factorial experiment. From the right panel we see that increasing diffusion and decreasing coating adversely affects wafer yield.*

In the S formula language [2], this relationship can be expressed as yield(w) ~ diffusion * coating, and this linear model can be fitted by passing the formula to the function lm().

```
> fit <- lm( yield ( w ) ~ diffusion * coating )
> summary ( fit )
```

Coefficient	Value	Std.Err.	T-val
(Intercept)	0.269	0.020	13.76
coating	0.116	0.028	4.23
diffusion.L	-0.225	0.032	-6.87
diffusion.Q	0.003	0.034	0.09
coating diffusion.L	0.040	0.046	0.86
coating diffusion.Q	-0.087	0.050	-1.76

Standard summaries like the one generated above do not suggest that the dependence of yield on coating thickness and diffusion time is more complicated than a quadratic function. By examining the coefficients in this table, we can quantify the impact on yield that can be expected as we vary these two factors over the range of values covered in the experiment; from a practical standpoint we have achieved our goals for performing the experiment.

However, as we have seen in the previous section, the S–wafers tools for visualizing and manipulating probe data can help us understand how the effects of these factors vary across the surface

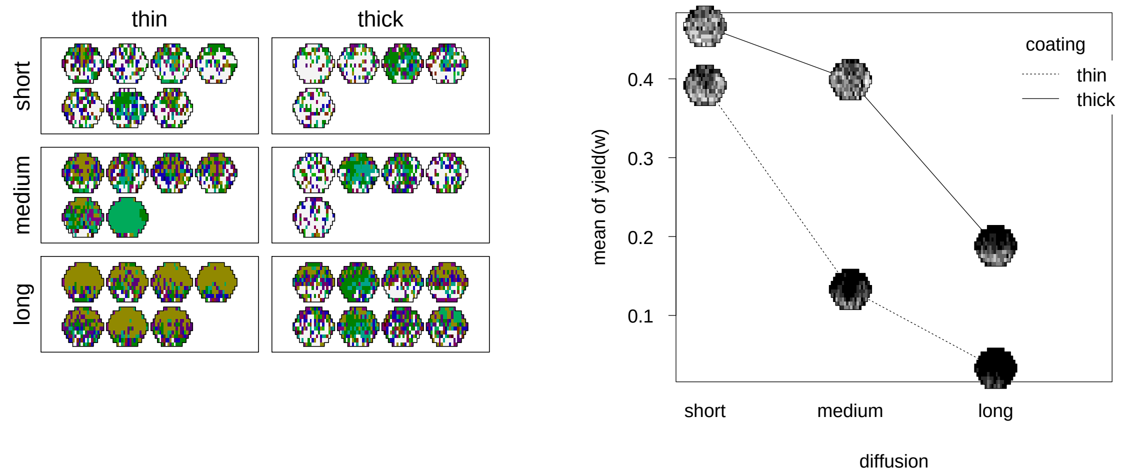
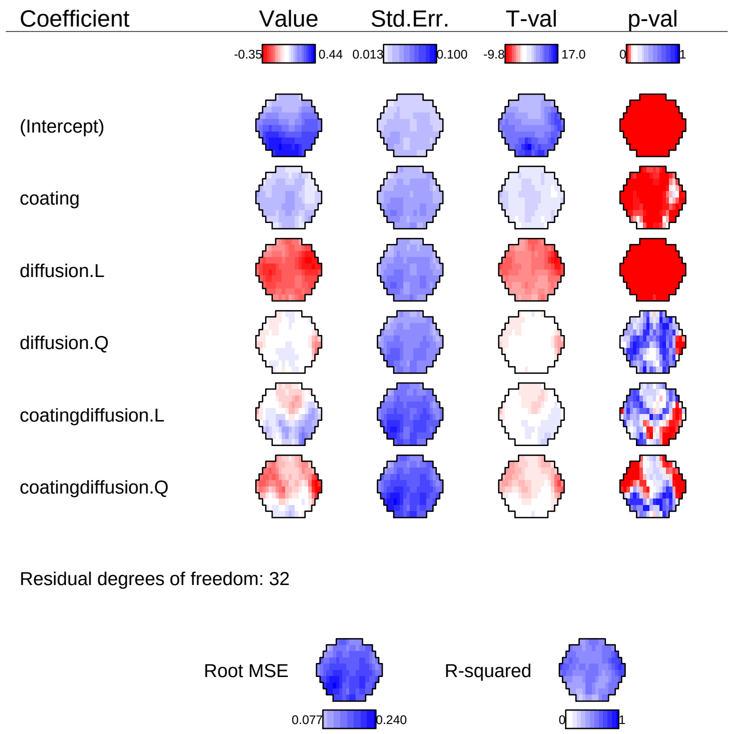


Figure 8: *Augmenting simple tabular summaries of the experimental data with spatial information. In the left panel we see the full probe maps for each experimental condition. In the right panel we have superimposed composite wafers on the average yield for each experimental condition; we now see that as diffusion increases and coating decreases, the yield loss increases in the top and center of the wafers.*

of the wafers. Consider, for example, augmenting Figure 7 with composite maps calculated from wafers receiving the same combination of factor levels. In Figure 8 we see immediately not only that the yield is lower for the thin coating thickness and the long diffusion time, but exactly where on the wafer the yield loss occurs. To untangle the spatial effects from the experimental factors and understand the spatial patterns observed in the table on the right in Figure 8, we again propose a model for wafer yield, this time incorporating spatial location:

$$\begin{aligned} \text{yield}_i \;&=\; \beta_{0i} + \beta_{1i}\text{coating} + \beta_{2i}\text{diffusion} + \beta_{3i}\text{diffusion}^2 \\ &\quad + \beta_{4i}\text{coating} \times \text{diffusion} + \beta_{5i}\text{coating} \times \text{diffusion}^2 \end{aligned} \qquad (2)$$

where now i indexes chip locations on the wafer and yield_i denotes the proportion of good chips in a neighborhood around chip i. The actual fitting procedure is discussed in detail in the next section because it highlights how standard S routines can be extended and applied to these specialized wafer-level computations. Since the coefficients associated with each factor are now also indexed by chip, the effects in the expression above can be displayed as wafer maps. In Figure 9 we present maps corresponding both to the coefficients in the model, as well as *waferized* diagnostics appearing in the standard S summary. Legends corresponding to each column of wafers appear at the top of the table. As we observed from the previous analysis, increasing diffusion time lowers yield




```
fit <- lm.wafers ( smooth ( w ) ~coating * diffusion )
out <- summary ( fit )
plot( out )
```

Figure 9: *A spatial summary of the new regression fit that directly incorporates spatial information about failing chips. Columns in this table correspond directly to those in a standard* S *summary.*

while applying a thicker coating improves yield. In addition, these plots suggest that the increased coating boosts yield more in the center of the wafer than on the edges. To verify this rigorously, we can consider a model that directly incorporates this radial dependence.

As this second example shows, S–wafers allows users to easily and naturally extend their analyse from computations with simple summaries (like fitting equations to yield) to richer spatial analyses that provide profound insights into the underlying processes. Once a particular effect has been identified, more refined modeling can be done to further understand the patterns of yield loss.

3 How it works

The examples in the previous section illustrate the advantages of spatial data analysis over traditional approaches; however to extract meaningful information from spatial data one needs to express fairly complex computations that make use of that extra information, both in standard and new statistical analyses. S provides a rich language for computations with data, but we still need to write software to specialize the computations for our application. To organize the computations efficiently we define a set of objects that encapsulate the spatial information, and then cast the spatial computations as methods that extend the capabilities of existing S functions, and in some cases, methods that implement new statistical methodology.

We now describe the software implementation behind S–wafers, including the software definition of wafer objects, and some of the specialized computations or methods for spatial data analysis. We start our description with the basic building blocks, such as methods for printing, subsetting, computing summaries, and performing arithmetic on wafer objects; then we move to more interesting methods that extend existing techniques by adding spatial information; we close this section with methods used for wafer data exclusively.

3.1 Object-based Computations in the S Language

To fully appreciate the advantages of the S–wafers implementation, it is helpful to have an idea of how the S language carries out object-based computations. S functions, like those in other computing languages, take a number of arguments and perform various operations with them, like computing averages and medians or fitting a linear model. In most cases, functions return some kind of S object with components that store the output of the computations, although some functions, like those for plotting and printing, are used primarily for their side effects. Arguments supplied to S functions may be constants or S objects that belong to "classes", such as time-series, ordered factors, matrices, or wafers. Generic functions provide a mechanism for S to invoke a special function, called a *method*, depending on the class of its arguments. For instance, the generic function plot() is used to display a graphical representation of almost any object, and the expression plot(x) applies the method appropriate for the class of its x argument. Therefore, plot(yield(w)) generates a simple scatter plot of yield calculated for each wafer stored in the object w as in Figure 4, while plot(w) displays in graphical form the complete wafer data in w, creating plots similar to those in Figure 1. The generic function plot() signals S to invoke the function plot.*class*() where *class* refers to the actual class of the argument to plot(). In the case of wafers, this is simply plot.wafers(). As we showed earlier, mean() is also a generic function that can be specialized to compute composite maps from binary probe data by writing a method mean.wafers().

Classes help us organize data, while methods help us organize the computations defined on those classes of objects. The class inheritance mechanism allows us to build on pre-existing methods as we write methods for derived classes. In this section, we describe how we exploit the inheritance mechanism to fit certain statistical models to spatial wafer data with minimal effort. The class() function is used to assign one or more classes to an object, for instance, the expression class(x) <–"foo" assigns the "foo" class to the object x. The preferred method for creating objects from special classes is to use *generator* functions that take as input some kind of raw, unstructured data and produce objects of the appropriate class.

3.2 Wafer Objects

As we saw in Section 2, there are various wafer classes reflecting the type of data collected through the various steps of manufacturing; the classes include probe maps, particle scans maps, memory array bitmaps, electrical tests measurements, point defect maps. Conceptually all these wafer classes are similar, and we use this fact to localize common operations into methods of the parent class wafers. But by differentiating among classes through inheritance, we are better able to share common computations among them and still apply specialized statistical methodology tailored to specific types of data. The S–wafers environment relies on external manufacturing data bases for its inputs. The inputs typically consist of text files in tabular form that are read by various generator functions to create S objects of the appropriate class. Once these objects have been created, we construct a core set of simple methods to carry out very basic operations, such as plotting, printing, and selecting subsets of wafers.

We have already seen the power of the plot() and points() functions as defined for wafers. Equally useful are methods for printing or more descriptively, "printing to the screen," which allow S to generate textual displays of objects. Typically, when printing a wafer w, users are only interested in summary statistics that describe the type of data contained in the object, rather than the thousands of values that have been assembled into w. Therefore, by default the printing method for wafers simply displays a brief description of the object (note that S automatically invokes print() when the user types the name of an object):

```
> w2
TestWafers:
    10 wafers/6 parameters on 60 test sites (0 NAs)
wafers:
    00001 00002 00003 00004 00005 00006 00007 00008 00009 00010
parameters:
    gate.vlt gate.stitch ngate.lkg pgate.lkg nplus.vlt pplus.vlt
Map:
  Coords: 60 points in (0, 49.68, 0, 48.69)
  Center: (24.68, 23.69)
  Radius: 25
  Flats: (4, 8)
```

A reasonably formatted text version of the actual data can be generated by setting the flag verbose in print(), print(w2, verbose=TRUE).

Wafer objects store data on multiple wafers and multiple sites, and frequently we need methods for extracting and replacing data by wafer and by test site. The double-bracket operator extracts subsets of complete wafers from a wafers object. For instance w[[1:5]] returns the first five wafers in w. The same rich set of subsetting expressions for S vectors can be used for wafers, i.e., the expression inside the double brackets can be either integers (positive or negative), logical vectors, or character vectors that name the elements to extract. In addition, the single-bracket operator extracts data according to the wafer geometry; in the case of probe wafers w[1:10, 25:35] extracts the data on chips that are in the first 10 rows and columns 25 through 35 for each wafer in w. Stacking operators also makes sense, so that

```
w[ [ yield(w) < 0.25 ] ] [ 1:20, ]
```

extracts the top 20 rows of chips of those wafers whose yield is worse than 25%. Replacement methods allow the user to modify portions of wafer objects, which is particularly useful for imputing missing data (a situation that occurs frequently). For example, the expression

```
w[is.na(w)] <-- median(w, na.rm = T)
```

replaces all missing values with the median of the available data.

Other core functions defined for wafer objects include extensions to the family of "apply" functions, also called iterators in other languages (FIT, for instance [10]). wapply invokes a function repeatedly for each group of wafers as specified by another variable. This is a very handy operation used, for instance, in Figures 4, and 8 to compute group averages according to yield and experimental factors.

3.3 Computing with Wafer Objects

We now show how the inheritance mechanism in S together with its rich collection of numerical algorithms allows us to specialize the computations that we covered in the examples in Sections 2.2.1 and 2.2.2.

We define probe wafer objects as matrices with extra slots to store the spatial information; individual wafers index the rows of the matrix, while chip sites index the columns. The fact that these objects are specialized matrices allows us to re-use all matrix computations already in S. For instance, all mathematical functions applied to probe maps return probe maps, like log(), exp(), sqrt(), sin(), round(). Arithmetic on probe wafers is thus simply carried out as arithmetic on matrices (notice that we simply pre-multiply the matrix w by a row vector of 1's to compute column sums):

```
> sum.wafers
function(w)
{
     n <-- nrow(w)                      # number of wafers (rows) in w
     ones <-- rep(1, n)                 # row vector of 1's
     ones %*% w                        # matrix multiplication
}
```

sum(w) returns a probe wafer with site totals of the various wafers in w. Once we have arithmetic defined on wafer objects computing "composite" or mean wafers (site-wise averages) is trivial

```
> mean.wafers
function(w)
{
     sum(w)/nrow(w)
}
```

The more useful implication of having probe wafers as matrices is in terms of statistical methodology; many important techniques are based on linear algebra, and these techniques are an integral part of the S environment.

As we saw in Section 2.2.2, when planning new manufacturing processes or modifying existing ones engineers are faced with yield issues: will changes to the current process affect the yield? If so, how? Data from designed experiments allows us to answer these questions. One way to analyze the data spatially is to extend the lm() function for fitting multivariate linear models to accommodate probe wafers. The lm() function can fit models of the type $\mathbf{Y} = \mathbf{X}\,\mathbf{B} + \mathbf{E}$ where all terms are matrices. In the case of probe wafers, $\mathbf{Y}$ refers to possibly smoothed binary data or yield with wafers across rows and chip sites across columns; thus, lm() relates each site's yield $(y)_{ij}$ in the $\mathbf{Y}$ matrix to the variables in the x_i row of the $\mathbf{X}$ design matrix of experimental variables (see Equation 2) by a linear combination whose coefficients are the elements of the column β_j in $\mathbf{B}$. The elements $(e)_{ij}$ in the $\mathbf{E}$ matrix represents the inherent stochastic variation for the $j'th$ chip in the $i'th$ wafer.

In addition to the model coefficients in $\mathbf{B}$, the function lm() computes ancillary information necessary for assessing how well the model describes the data; for instance, how significant each coefficient is (compared to the noise in the data), how much variation in yield can be accounted for by the changes in the experimental variables. The main task in writing the lm.wafers() method is to take all the components returned by the existing lm() method and re-arrange them in an efficient manner for easy display by the available wafer plotting methods. This is done by using the probe wafers generator function which takes as input a matrix and a map describing the wafer geometry and returns a wafer object; we then just replace the various matrix components returned by lm() with their corresponding wafer versions. In the example below the "$" operator extracts a component from a list, so that z$x extracts a component named x from the list z.

```
> lm.wafers
function ( formula, data, ..., model = T )
{
    z <-- NextMethod ( "lm" )                        # invoke parent method lm
    w <-- extract.model ( z$model, "response" )     # extract wafer from formula,
    map <-- Map ( w )                                # and spatial information in w
    #
    # coerce appropriate components into wafers
    #
    z$coefficients <-- wafers ( z$coefficients, map )
    z$fitted.values <-- wafers ( z$fitted.values, map )
    z$residuals <-- wafers ( z$residuals, map )
    z$effects <-- wafers ( z$effects, map )
    z$sigma <-- wafers ( z$sigma, map )
    z$r.squared <-- wafers ( z$r.squared, map )
    class ( z ) <-- c ( "wlm", class ( z ) )
    return ( z )
}
```

The newly created class wlm for wafers linear models inherits from lm, the class of the value returned by the lm() function. Meaningful methods for wlm objects are few, namely summary.wlm(), print.wlm() and plot.wlm(). The summary.wlm() method simply coerces to wafers the t-statistics and p-values computed by summary.lm(). For this wlm class of objects, we decided to make the printing and plotting methods identical, since the resulting objects can be fairly large (about 4, 500 numbers in our example), and the graphical representation better conveys the spatial effects than a large table. Writing the plotting method for wlm objects is simple, if a bit tedious, since it only requires to plot as a table the various components like z$coefficients and z$t-values, produced by the summary.wlm() method.

3.4 Wafer-specific Computing

Exploiting the existing computing algorithms in the S environment through inheritance is a wise approach to tailoring computing to particular applications – which is the main thrust behind inheritance after all. However there are instances when no existing code can help solve a certain specific problem. Our collaborations with Lucent Technologies' Microelectronics engineers has fueled research into new statistical methodology, the software implementation of which had to be written from the ground up. These techniques include the assessment of complete and conditional spatial randomness for wafer data [6] and the model-free decomposition of D_0 and Y_0, [8]. To incorporate these computationally intensive techniques into the S–wafers environment, we needed to write Fortran code for efficiency, and through the S/Fortran interface pass wafer data back and forth. The overall design, however, closely follows the paradigm described in Section 3.1 and we expect that we will be able to re-use this code in future problems.

4 Conclusions

In this paper, we have described S–wafers, a statistical software environment written in the S language for computing with the spatial data collected during the fabrication of integrated circuits and other microelectronic devices. Our main design goal was to enable engineers to conduct effective analysis on wafer data as a means for quality improvement and process monitoring. Through examples, we have illustrated how spatial information can be easily incorporated into most analyses: from simple box- and scatter plots to statistical process control and analysis of variance. We have also demonstrated how these straightforward, data analytic techniques can lead engineers to ask deeper questions, often requiring the development of new statistical methodologies.

The S–wafers environment exploits the object-oriented paradigm of the S language to express graphics, data manipulations and data analytic operations easily and uniformly. Key elements of the environment include powerful graphical methods, sound underlying statistical methodology, and most importantly extensibility. The real strength of S–wafers is that it can support many levels of users, from engineers in the factories performing data analysis to research statisticians developing new methodologies for extracting more information from mapped wafer data.

Throughout its development, S–wafers has been influenced by on-going collaborations between engineers in Lucent's manufacturing divisions and statisticians in Bell Laboratories. Currently, S–wafers is the engine behind a number of data analysis efforts within Lucent, and is used to generate a series of daily reports available on the intranet to engineers in each of the plants. New sources of data are continually being incorporated into the system, and our interaction with the factories is increasing. Current information is available from our external Web page http://cm.bell-labs.com/stat/project/icmanuf/.

5 Acknowledgement

This collaboration would not have been possible without the involvement of David J. Friedman (Integral, Inc.), Vijay Nair (University of Michigan) and Daryl Pregibon (AT&T Research). In addition, from Bell Labs we would like to thank John M. Chambers, Lorraine Denby, and Bill Joyce. From Lucent Technologies Microelectronics Division, James R. Hoyer, Bill Russel, Fred Weck, Steve Whitlock, Dan Wroge, and Tim Willingham (Orlando); Randi Anderson, Russ Burock, Joe Cassano, Zbigniew A. Kozlowski, Norm Lawrence, Tom Long, Ed Majka (Allentown); Benito Campos, Alberto Colina, Allan Dunbar, Marga Espino, Estaban Navarro Lopez-Hermosa, and Maria Jesus Peman (Madrid); Chung-Ming Hsieh, Sam Jabar and Mark Martin (Reading); Erin Byrne, Pat Carroll, Janet Lentz and Alex Robertson (Optoelectronics group).

A The S Language

S is a language and programming environment, designed at Bell Labs, for data analysis, graphics, and other computations([1], [2]). S is currently available in the S-Plus extension from the MathSoft corporation, which has an exclusive license from Lucent Technologies to distribute S-based software ([12]). Research on S continues at Bell Labs: a major new version of S is under beta test, providing extensions to the features described below, as well as many other new programming techniques (see the paper [3] or look at the web site http://cm.bell-labs.com/stat/project/S/.

S is a *functional*, *object-oriented* interactive language. In syntax it resembles C, Java, and other similar languages, with a few differences (e.g., the assignments operator is an arrow, <−). It differs much more in how it is used: typically, users type S expressions and S responds by evaluating the expression and displaying the results by printing or plotting the value of the expression.

S expressions are nearly all made up of function calls (including all operators):

```
plot(w)
residuals <-- yield - mean(yield)
oak <-- tree(yield(w) ~ furnace + stepper + epi + code + lot)
```

In S, everything is an object, from large datasets to numbers or character strings. All S objects are created dynamically, are self-describing, and are kept in permanent databases by S when assigned in the user's expression. Users never have to declare types for objects, the type or *class* of an object is stored with the object.

S uses the class of objects to choose methods for evaluating function calls; new classes and new methods can be easily programmed by users. The methods, for the programmer, are just ordinary S functions. After writing the function the programmer tells S that this is a method for some function, say plot() for some class of objects.

Objects of one class can extend or inherit from another class; for instance, chronological objects extend the class of numeric vectors by adding an origin and allowing calendar-based computations (days between dates, weekends, months, weeks, etc.).

To programmers, S also provides a set of well-defined, easy to use interfaces to other languages and systems, namely, to C, Fortran, Java, and the host operating system; the interfaces can be used to further extend the S environment.

To users, the S environment provides the S language together with a rich collection of S functions that allows them to conduct advanced computations and graphics on data. Graphics can range from simple scatter-plots and bar-plots to modern dynamic visualization techniques; data analysis can range from simple numerical summaries to more advanced techniques, such as fitting linear and nonlinear regression models by least-squares, time-series and fourier analysis, classification and regression trees, multivariate techniques for cluster and discriminant analysis, designed experiments, and more. Other computations include matrix algebra and eigenvalue decompositions, neural nets, etc. See [13] and [5] for an introduction to data analysis in S; [11] is a good introduction to S programming.

References

[1] R. A. Becker, J. M. Chambers, and A. R Wilks. *The New S Language.* Wadsworth, Pacific Grove, California, 1988.

[2] J. M. Chambers and T. Hastie, editors. *Statistical Models in S.* Wadsworth, Pacific Grove, California, 1992.

[3] John M. Chambers. Evolution of the S language. In *Computing Science and Statistics*, volume to appear. Interface Foundation of North America, 1996.

[4] Lorraine Denby and David A. James. A graphical user interface for spatial data analysis in integrated circuit manufacturing. In Michael Meyer and James Rosenberg, editors, *Computing Science and Statistics*, volume 27. Interface Foundation of North America, 1995.

[5] Brian S. Everitt. *A Handbook of Statistical Analyses using S-PLUS.* Chapman & Hall, London, 1994.

[6] D. Friedman, M. Hansen, V. Nair, and D. James. Model-free estimation of d_0 and y_0 from wafer map data. *Submitted to IEEE Trans. on Semiconductor Manufacturing*, 1995.

[7] M. Hansen, V. N. Nair, and D. Friedman. Monitoring wafer map data from integrated circuit fabrication processes for spatially clustered defects. *Submitted to Technometrics*, 1996.

[8] Mark H. Hansen and Vijayan N. Nair. Process and productivity improvement in semiconductor manufacturing. In Michael Mayer, editor, *Computing Science and Statistics*, volume 27. Interface Foundation of North America, June 1995. to appear.

[9] Microsoft Corporation, Redmond, WA. *Visual Basic Language*, 1994. Version 3.0.

[10] Lloyd H. Nakatani and Laurence W. Ruedisueli. *The FIT Programming Language.* AT&T Bell Laboratories, Murray Hill, NJ, 1991. Reference Manual.

[11] Phil Spector. *An Introduction to S and S-plus.* Duxbury, Belmont, CA, 1994.

[12] StatSci Division of MathSoft Inc., Seattle, WA. *S-PLUS Software Documentation*, 1995. Version 3.3.

[13] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S-Plus.* Springer-Verlag, New York, 1994.