Chapter 6
Yield Estimation by Computing Probabilistic Hypervolumes

Chenjie Gu and Jaijeet Roychowdhury

6.1 Introduction: Parameter Variations and Yield Estimation

Parameter variations are inevitable in any IC process. Process steps such as oxidation, doping, molecular beam epitaxy, etc., are all fundamentally statistical in nature [1]. Design of functioning circuits and systems has traditionally relied heavily on the presumption that the law of large numbers [2] applies and that statistical averaging predominates over random variations – more precisely, that the statistical distributions of important process, geometrical, environmental, and electrical parameters cluster closely about their means. Unfortunately, with feature sizes having shrunk from 90 to 65 nm recently (with further scaling down to 45 and 32 nm predicted by the ITRS roadmap [3]), this assumption is no longer valid – in spite of efforts to control them [4, 5], large variations in process parameters are the norm today. This problem is most severe for circuits that try to use the minimum transistor size (e.g., memory circuits [6] for which chip area is of high priority). With transistors having become extremely small (e.g.: gates are only 10 molecules thick; minority dopants in the channel number in the 10s of atoms), small absolute variations in previous processes have become large relative ones. Lithography-related variability at nanoscales [5], which affect geometrical parameters such as effective length and width, further compound parameter variation problems.

Due to such variations in physical, process or environmental parameters, electrical characteristics of circuit devices (e.g., MOS threshold voltages) change; this leads to undesired changes in circuit performances. For example, in static RAM (SRAM) circuits, like the 6T cell [7] in Fig. 6.1, the threshold voltages of M1 and M2 (and similarly, M3 and M4) determine how well they turn on and hence determine the charge/discharge rate in read/write operations. This rate is closely

C. Gu (✉)
University of California, Berkeley, USA
e-mail: gcj@eecs.berkeley.edu

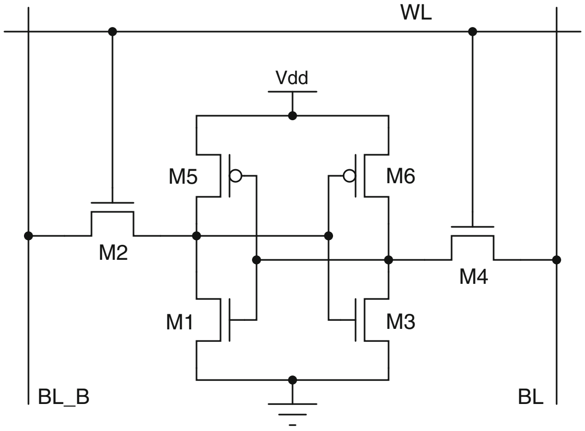


Fig. 6.1 SRAM 6T cell

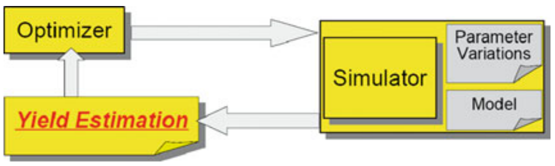


Fig. 6.2 Optimization loop in circuit design

related to SRAM read/write access time, and hence determines read/write frequency. Due to parameter variations, it is highly possible that some cells fail to finish the read/write operation within specified access time. As a result, the whole SRAM array circuit is defective, and has to be discarded. So it is crucial to predict what fraction of circuits is likely to fail because of variability, so that measures may be taken to increase the yield of correctly functioning circuits, to the extent possible via design and process changes. Reasonably accurate prediction of yield (e.g., within a percent or better) is important: over-estimation leads to fewer working components than expected, while under-estimation can lead to lowered performance as a result of design efforts to increase yield.

Due to its importance in the design process, the yield estimation problem is closely related to circuit optimization. Figure 6.2 shows a typical optimization loop for yield maximization in circuit design. In this loop, the simulator computes circuit performances of interest and their distributions, given device models and parameter distributions. Yield estimation is conducted around the simulator and the result is exported to an optimizer. The yield is one of the key quantities that guides the optimizer to achieve the "best" design point, which maximizes yield while optimizing performances.

In this chapter, we shall discuss numerical methods for yield estimation due to parameter variations. We first define the yield estimation problem in this section, and motivate the problem by SRAM circuit. In Sect. 6.2, we provide a survey of various approaches to the yield estimation problem, including statistical methods and deterministic methods. One of the deterministic methods, Yield Estimation via Nonlinear Surface Sampling (YENSS), turns out to be efficient in identifying the region (in the parameter space) that corresponds to circuits meeting all performance specifications/constraints, and in estimating the yield by computing the probabilistic hypervolume of that region. We discuss details of YENSS in Sect. 6.3. In Sect. 6.4, we show applications of YENSS on illustrative examples and an SRAM circuit.

6.1.1 Yield Estimation

We first define the yield estimation problem with a focus on its geometrical interpretations.

Briefly, the yield estimation problem is simply to compute the probability that the circuit will meet all specified performance constraints (bounds). Important inputs to a yield estimator include information about parameter variations (e.g., mean, variance, min/max bounds, and distributions), and performance specifications (e.g., min/max bounds). For example, a simple yield estimation problem with SRAM read access time as a performance metric can be stated as follows: given that the threshold voltages of MOSFETs in the SRAM cell satisfy uniform distributions on $[0.\ 1V, 0.\ 7V\ ]$, calculate the probability that the SRAM read access time is less than 5 ns.

Given the complexities of the IC manufacturing process, performance specifications are typically much easier to obtain than information about parameter variations. In fact, characterizing parameter variations is a challenging problem in itself. Sometimes only min/max bounds, or mean values of the parameters, are available. If this happens, one typically makes assumptions about parameter distributions. For example, the uniform distribution and the (truncated) Gaussian distribution are commonly used [8].

From a geometrical view point, yield estimation is equivalent to a hypervolume computation problem [9] – this geometrical interpretation leads to a better understanding of the problem, and provides insights for numerical solution.

For purposes of illustration, suppose we have two independent parameters $\mathbf{p} = [p_1, p_2]$, with nominal values $p_{1\text{nom}}, p_{2\text{nom}}$, uniformly distributed over $[p_{1\text{min}}, p_{1\text{max}}]$ and $[p_{2\text{min}}, p_{2\text{max}}]$, respectively, as shown in Fig. 6.3. We assume that the circuit performs correctly at its nominal point, i.e., the circuit performance $f_{\mathbf{p}}$ is f_{nom} when the parameters are $(p_{1\text{nom}}, p_{2\text{nom}})$. As the parameters move away from the nominal design point, the circuit performance changes away from its nominal value. Therefore, in the parameter space, there exists a region D around the nominal design point where the performance remains acceptable. We call this region the *acceptable region* later on, as opposed to the *failure region*, i.e., the region in the min/max box excluding the acceptable region. The acceptable region is depicted by the interior of the closed curve in Fig. 6.3, with the portion within the min/max bounds

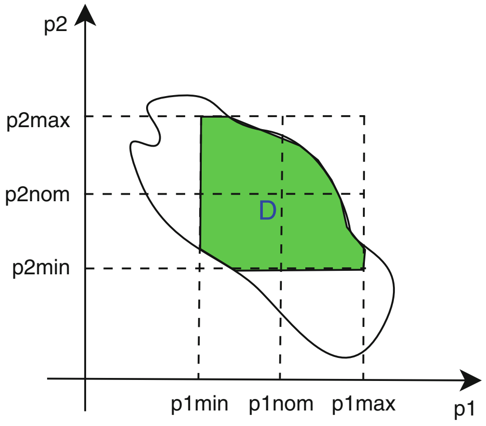


Fig. 6.3 Evaluating yield is equivalent to evaluating ratio of the area of the shaded part to the area of the box $[p_{1\min}, p_{1\max}]$ _ $[p_{2\min}, p_{2\max}]$

shown shaded. The yield is then the ratio of the area of region D to that of the box $[p_{1\min}, p_{1\max}] \times [p_{2\min}, p_{2\max}]$.

Mathematically, hypervolume computation is equivalent to calculating the integral

$$I = \int_D \frac{1}{(p_{1\max} - p_{1\min})(p_{2\max} - p_{2\min})} \mathrm{d}p_1 \mathrm{d}p_2. \tag{6.1}$$

Note that because p_1 and p_2 are uniformly distributed over $[p_{1\min}, p_{1\max}]$ and $[p_{2\min}, p_{2\max}]$, respectively, their probability density functions are $\frac{1}{p1\max - p1\min}$ and $\frac{1}{p2\max - p2\min}$, respectively. So the function to be integrated in (6.1) can be viewed as the joint probability density function of p_1 and p_2.

More generally, if the parameters $\mathbf{p}$ satisfy the joint probability density function $\pi(\mathbf{p})$, the yield, corresponding to the integral of the joint probability density function over the region D, is

$$I = \int_D \pi(\mathbf{p}) \mathrm{d}\mathbf{p}. \tag{6.2}$$

We denote the integral (6.2) to be the *probabilistic hypervolume* (or probability mass) of region D.

If we think in terms of performance and parameter spaces, yield estimation is essentially a mapping from a region in performance space to a region in parameter space, the hypervolume of which constitutes the yield, as depicted by the lower arrow in Fig. 6.4. Hence, one way to solve the yield estimation problem is first to identify the boundary of this region in parameter space, and then use it to compute the probabilistic hypervolume. In this chapter, we will look into several methods for finding this boundary numerically.

Moreover, as these methods are dealing with boundaries in parameter and performance spaces, they can be applied to other similar problems involving

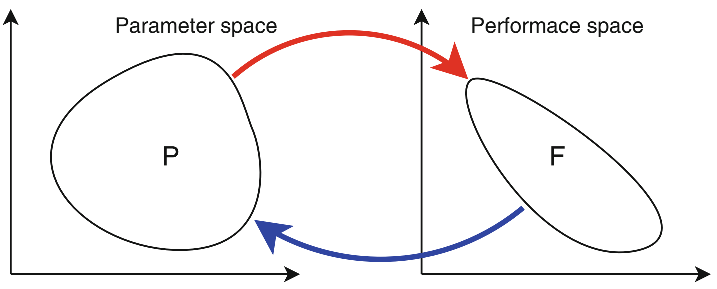


Fig. 6.4 Boundaries in parameter space and performance space

boundaries. For example, in the design centering (or yield maximization) problem [10–20], we are seeking a nominal design point[1] which maximizes the probability that circuits meet their specifications. Geometrically, we are looking for the point that maximizes the minimum distance to the boundary in parameter space, if independent and uniform parameter distributions are assumed. Being able to identify the boundary (or to compute points on the boundary) is an important subproblem for design centering.

As another example, the boundaries in performance space are used in the performance trade-off problem [21–23]. In this problem, we normally have several performance metrics to balance, say circuit speed and power consumption. For the parameters that are allowed to change, there exists a trade-off curve in the performance space – we can move on that trade-off curve to obtain the maximum performance of interest (e.g., maximize the speed within the power budget). Again, this requires that we find the boundary in the performance space, given a region in the parameter space – i.e., the upper arrow in Fig. 6.4. Similar to the design centering problem, the parameters used here are typically physical parameters.

To summarize, the yield estimation problem is equivalent to computing the probabilistic hypervolume of the region that correspond to working circuits. Technologies to identify the boundary of this region can serve as a useful first step in yield estimation, and can also be applied to other problems such as design centering and performance trade-off problems.

6.1.2 An Example: SRAM Read Access Failure

In SRAM cells, such as the 6T cell in Fig. 6.5a, several types of failures can occur due to variations of parameters such as the threshold voltages of the six MOSFETs [24]. For example: read access failure can be caused by an increase in cell read

[1]For the design centering problem, the parameters to be determined are normally physical parameters, such as transistor width W and length L; For the yield estimation problem, the parameters are "statistical" ones, such as threshold voltage V_t and channel length variation ΔL.

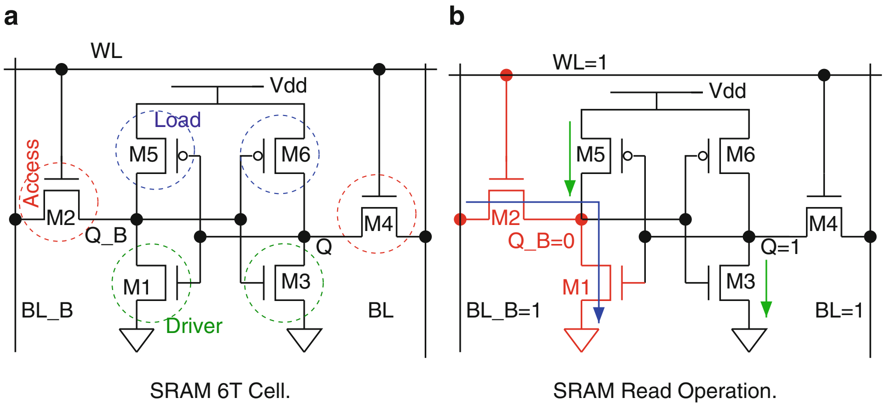


Fig. 6.5 SRAM 6T cell (**a**) and its read operation (**b**)

access time; data flipping during the read operation leads to an unstable read; write failure is the inability to write the data into a cell successfully within the write period; fluctuations of surrounding circuits/power supply can make the cell unable to hold data in standby mode, etc. Consider for example the yield loss caused by read access time failure in a 6T cell.

The SRAM 6T cell circuit is composed of a pair of cross-connected inverters (used for storing the data bit in the cell) and two pass transistors (used for accessing or writing the data from/to the SRAM cell). In Fig. 6.5a, $M2$ and $M4$ are the access transistors, and ($M1$, $M5$) and ($M6$, $M3$) form two cross-connected inverters. ($M1$, $M3$) and ($M5$, $M6$) are also commonly referred to as driver and load transistors, respectively. Q and Q_B are the nodes that actually store the data bit. BL and BL_B are the bit-lines which follow the cell data during the read operation, and provide the data to be written during the write operation. The access transistors are controlled by the wordline (WL), which enables/disables the SRAM cell to be read/written.

To analyze the read operation of this SRAM cell, we assume, without loss of generality, that 1 is stored in the SRAM cell, i.e., $Q = V_{dd}$ and $Q\_B = 0$. During read operation, both bit-lines BL and BL_B are first precharged to V_{dd}. Then, at the assertion of wordline WL, bit-line BL_B starts to discharge through $M2$ and $M1$ (as shown in Fig. 6.5b via a path through M_2 and M_1), while bit-line BL stays at V_{dd}. Besides, subthreshold leakage paths through $M3$ and $M5$ are also present and affect the discharging current. Within the read access time (denote it as t_f), however, the resulting voltage difference between BL and BL_B (denote it as ΔBL) is small, and therefore a sense amplifier is used to amplify this signal to obtain a full swing $0 - V_{dd}$.

In order for the sense amplifier to be able to detect the bit-line voltage difference ΔBL (i.e., to differentiate signal 1 from signal 0), ΔBL has to reach a minimum value $\Delta BL_{\min}$. Therefore, a read access failure is said to occur if at the specified read access time t_f, the bit-differential ΔBL is less than the minimum required value $\Delta BL_{\min}$.

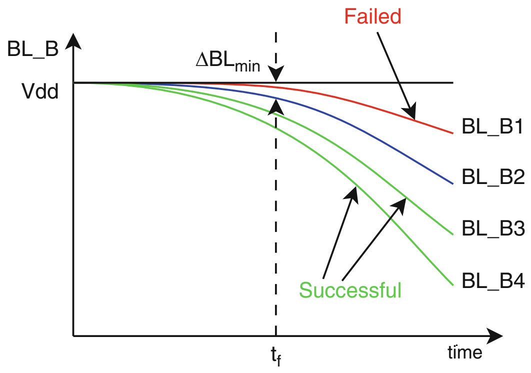


Fig. 6.6 *ΔBL* waveforms during read operation

Due to variations in the threshold voltages, the driving strengths of $M1$ and $M2$, as well as the leakage current through M_3 and M_5, change and affect the time required to generate the minimum voltage difference between BL and BL_B. For example, Fig. 6.6 shows several waveforms of BL_B in SRAM cells with different parameters, where t_f is the maximum read access time specified by the design. From Fig. 6.6, we see that if ΔBL is larger than the minimum bit-differential $\Delta BL_{\min}$, the read operation is successful (e.g., waveforms BL_B3 and BL_B4). Otherwise, the read operation is a failure (e.g., waveform BL_B2).

Therefore, the probability of SRAM failure due to read access time failure can be written as:

$$P_{\text{access-failure}} = P(\Delta BL(t_f) < \Delta BL_{\min}), \tag{6.3}$$

where $\Delta BL(t_f)$ is the bit-differential at time t_f. So the yield is computed by:

$$\text{Yield} = 1 - P_{\text{access-failure}} = P(\Delta BL(t_f) \geq \Delta BL_{\min}). \tag{6.4}$$

In the subsequent sections, we present several approaches to solve this yield estimation problem.

6.2 Approaches to Yield Estimation

Several approaches have been proposed for the yield estimation problem. They can be classified into two general categories: statistical methods and deterministic methods.

Statistical methods are mainly based on the law of large numbers and the central limit theorem []. They generate random samples in the parameter space, perform the simulation for each sample and count the number of samples that meet performance specifications. In order to obtain a high accuracy of the results, they require a large number of samples in the parameter space, and consequently require a large number of simulations. However, as the number of samples needed for a certain accuracy does not increase dramatically when the number of parameters of interest increases, these methods can be well suited to problems which involve a very large number of parameters.

On the other hand, deterministic methods, or deterministic-statistical mixed methods, can generally be decomposed into two steps:

1. In the parameter space, identifying the acceptable region, i.e., finding the boundaries of this region;
2. Calculating the probability that parameters fall in the acceptable region.

In contrast to statistical methods, deterministic methods explicitly compute boundaries in the parameter space. Therefore, these methods are more attractive for various optimization procedures that need to utilize the boundary information.

Because of this, the accuracy of deterministic methods depends strongly on how well the boundaries are approximated. Usually piecewise linear approximation to the boundaries are used: identify a set of points on the boundary, and use linear interpolation to represent the boundary. The more points on the boundary, the more accurate yield estimation is. For problems that deal with a small number of parameters, and especially for problems that result in a nearly linear boundary in the parameter space, the boundary can usually be efficiently approximated by just a few points, which is much less than the number required for statistical methods. Deterministic methods are also well suited for such situations.

However, when there are large number of parameters, the computational cost of identifying boundaries can increase exponentially, due to the curse of dimensionality. In such cases, statistical methods are generally preferred; or deterministic methods are applied to obtain a coarse approximation of the boundary, which can be utilized to speed up statistical methods. Such methods can be considered statistical-deterministic mixed methods.

6.2.1 Statistical Methods

6.2.1.1 Monte-Carlo Methods

The simplest way to estimate yield is to apply the well known Monte-Carlo method [25–27]. Monte-Carlo methods are generally composed of the following four steps:

1. Sample parameters according to their statistical distributions (random number generation);
2. Simulate the circuit for each set of sampled parameters, and compute the circuit performances of interest;
3. Count the number of sampled circuits that meet all the performance specifications;
4. Calculate the ratio of this number to the total number of samples (calculating the yield).

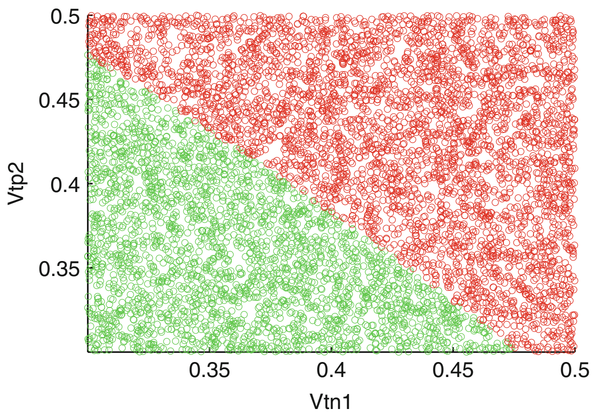


Fig. 6.7 Monte-Carlo simulation example. The *light samples* represent circuits that meet the specification, and the *dark samples* represent circuits that fail to work. The yield is calculated as the ratio of the number of *light samples* to the total number of samples

For example, Fig. 6.7 shows results from a Monte-Carlo simulation on a circuit, where the threshold voltages V_{th1}, V_{th2} of two MOSFETs are considered to be varying parameters. Random samples are first generated for V_{th1}, V_{th2}, and then simulations are performed for each (V_{th1}, V_{th2}) pair. The light samples represent circuits that meet the specification, and the dark samples represent circuits that fail to work. The yield is finally calculated as the ratio of the number of light samples to the total number of samples.

While Monte-Carlo methods have the advantages of simplicity and extremely general applicability, their main limitation is that they can require a very large number of samples as well as expensive simulations for those samples. According to the law of large numbers, if X is a random variable, μ is its mean, and σ^2 is its variance, we have

$$P(|\overline{X} - \mu| \geq \varepsilon) \leq \frac{\sigma^2}{N\varepsilon^2}, \tag{6.5}$$

where $\overline{X}$ is the sample average, N the number of samples, and ε represents how accurate the sample average approximates the real mean. Therefore, the smaller the ε is, the larger number of samples N needs to be taken.

Besides the large number of samples, the computation required for Monte-Carlo type yield estimation methods is significantly exacerbated if each individual simulation is expensive.

For example, SPICE-level transient simulations [28] are ideally run with tight tolerances (small time steps) and the best device models available, in order to extract accurate results such as timing information for digital circuits (e.g., delay of a critical path, setup/hold time of a latch/register, SRAM read/write time); RF analyses such as harmonic balance and shooting [29] are indispensable for circuits such as mixers and oscillators. These simulations are significantly more expensive than DC and AC analyses.

Whether such analyses are required depends on the circuit and performance of interest. While DC/AC analyses are usually enough to simulate linear circuits or circuits working in linear mode (such as operational amplifier circuits), accurate transient/RF analyses are essential for highly nonlinear circuits, such as SRAM cells and oscillators. For example, to find SRAM read/write times, transient simulations with very small time steps needs to be taken. To characterize the static noise margin (SNM) [7, 30] of an SRAM cell, the butterfly curve is typically identified, for which a series of DC analyses with respect to different DC inputs need to be performed.

Note that the types of simulations are not just confined to the applications just mentioned. For example, static timing analysis (STA) [31] and statistical static timing analysis (SSTA) [32] can be employed as a higher-level timing simulation instead of detailed transient analysis.

To summarize, the large number of samples that can be needed, and the expense of each simulation motivate alternatives to Monte-Carlo that can alleviate the computational cost of either or both aspects.

6.2.1.2 Improved Monte-Carlo Methods

In order to overcome the disadvantages of the classical Monte-Carlo method, various techniques have been proposed to improve its efficiency. These techniques basically try to sample data more intelligently so that the number of samples decreases without sacrificing accuracy.

The most widely used technique to reduce the number of samples is the class of variance reduction methods [26, 27]. For example, importance sampling [33, 34], stratified sampling [35, 36], Latin hypercube sampling[37], control variates [26, 27], antithetic variates [38], Rao-Blackwellization [26, 27] all fall in this category. The key idea of variance reduction methods is to reduce the variance of sample points – from (6.5), we know that for a fixed accuracy ε, the smaller the variance σ^2, the smaller number of samples are needed. Some of these techniques are detailed in other chapters.

6.2.2 Deterministic/Mixed Methods

As mentioned before, denote the acceptable region by D, the probability density functions of parameters $\mathbf{p}$ by $\pi(\mathbf{p})$; then, the yield estimation problem is to calculate the integral

$$I = \int_D \pi(\mathbf{p})\mathrm{d}\mathbf{p}. \tag{6.6}$$

The Monte-Carlo method can be viewed as a way to approximate this multi-dimensional integral. On the other hand, if we can identify region D, i.e., find the boundaries of the region D, numerical integration methods can be employed to compute (6.6) directly.

The boundaries of D can be represented as the solution of a nonlinear equation. Suppose there are N_p parameters $\mathbf{p}$, i.e., the parameter space is $\mathbb{R}^{N_p}$, then D is a N_p-dimensional closed region in $\mathbb{R}^{N_p}$, and its boundary is a $(N_p - 1)$-dimensional manifold. Also suppose there are N_f performances of interest $\mathbf{f_p}$, which depend on parameters $\mathbf{p}$. Then, the boundary for each performance constraint is defined by a *scalar* nonlinear equation[2]

$$h(\mathbf{p}, \mathbf{f_p}) = 0 \tag{6.7}$$

which states the relationship between $\mathbf{p}$ and $\mathbf{f_p}$.

We now review several methods that solve for or approximate this boundary, prior to using the boundary to perform yield estimation.

6.2.2.1 Yield Estimation by Simplicial Approximation

In Sect. 6.2.1, we have seen that Monte-Carlo methods with variance reduction techniques reduce the computational cost only by reducing the number of samples. By utilizing simplicial approximations of the boundary in parameter space, one can further speed up the procedure by also reducing the computational cost of simulating each individual sample. Later, we outline the basic structure of this algorithm[3] [39, 40].

1. Define the maximum number of samples of Monte-Carlo method to be N_{max}, and the number of working circuits to be N_{good};
2. Identify a few points on the boundary, and build up the linear constraints to represent a crude approximation to the boundary;
3. Start Monte-Carlo sampling:
   (a) If the sample lies inside the boundary (satisfies all linear constraints), $N_{\text{good}} = N_{\text{good}} + 1$;
   (b) Else, perform a full simulation to check if the sample lies inside the boundary, and apply line search to solve for another point on the boundary. If it passes the check, $N_{\text{good}} = N_{\text{good}} + 1$;
   (c) Update the boundary (by adding/deleting linear constraints that define the boundary);
4. Compute the yield: Yield $= \frac{N_{\text{good}}}{N_{\text{max}}}$.

[2]This is the case when the boundary is defined by one performance constraint. The multiple-constraint case will be discussed in Sect. 6.3.1.

[3]There are a few variations of this algorithm, but the basic structures are almost the same.

In [39, 40], the boundary in parameter space is approximated by a series of linear constraints, i.e., a simplicial approximation. The utility of the approximate linear constraints to the boundary is that testing whether a point is in the acceptable region is sped up significantly for most (but not all) Monte-Carlo samples – in order to judge whether a sample lies inside the boundary or not, one does not need to simulate the sample, but just test the linear constraints.

For those that fail this fast check, a more computationally expensive check, involving a linear search between the approximate linear constraints and the real boundary, is performed. As a by-product of this check, a new sample on the real boundary is obtained, and therefore the boundary is refined by adding/deleting linear constraints.

As we normally expect a high yield, most samples fall inside the boundary; therefore, with only a very coarse approximation of the boundary, we save the computational cost of simulating most samples. The price paid is the computation needed to identify the boundary. To identify a new point on the boundary, line search method usually takes 3–4 iterations to converge, which correspond to 3–4 simulations of the circuit.

One problem with this approach is that the number of linear constraints, or the number of points on the boundaries, increases with the number of parameters to be considered if the same accuracy of the boundary approximation is desired. Moreover, the points on the boundary are generated randomly – if a sample point fails to meet the linear constraints, a new point, on the boundary and close to this sample point, is searched for by the algorithm. That is to say, it is not guaranteed that the points that approximate the boundary are well-spaced, or that the linear constraints will approximate the acceptable region well – the shape of the approximated boundary can be quite different from that of the real boundary if these points are clustered together in a small region.

There is also an important assumption behind this approach – the boundary separating the acceptable/failure regions must be convex. It is this assumption that enables us to perform a fast check on a new sample, instead of performing a full simulation. However, this assumption is not always true in reality. Indeed, the boundary for SRAM read access failure, as we show in Fig. 6.20b, is not convex. When this assumption is not met, the algorithm might incur error in the result.

While simplicial approximation essentially uses a set of linear constraints to approximate the boundary, there is no limit for choosing the form of constraint equations. Indeed, similar to simplicial approximation, there are methods that use quadratic approximations [41] to the boundary. However, third-order or higher-order approximations are typically not employed because they are computationally much more expensive.

6.2.2.2 Yield Estimation by Worst-Case Distance Approximation

To alleviate the problem of dramatic increase of number of points with the number of parameters in simplicial approximation methods, a simpler method based on worst-case distance approximation [42] has been proposed. The parameters that have the *worst-case distance* are defined to be those that have the smallest metric distance from the nominal parameter point. These points have the highest probability of causing performance constraint to be violated. Based on this key idea, one worst-case parameter point with respect to each performance constraint is searched for by applying numerical optimization methods. Therefore, the boundary is approximated with many fewer points.

The basic flow of the algorithm is:

1. For each performance constraint, define a corresponding optimization problem that finds the worst-case parameters;
2. Solve each optimization problem, and obtain a point for each optimization problem on the boundary;
3. Use the tangent hyperplane at each point to approximate the boundary;
4. Calculate a yield estimate for each performance constraint using an analytical formula; or, calculate a yield estimate considering all performance constraints simultaneously by the Monte-Carlo method.

There are several options for the distance d between two points P_1 and P_2 in the parameter space. The most common definition is the Euclidean distance between P_1 and P_2. Equivalently, it is the L_2-norm of the vector difference between two points

$$d^2(\mathbf{p}_1, \mathbf{p}_2) = ||\mathbf{p}_1 - \mathbf{p}_2||_2^2 = (\mathbf{p}_1 - \mathbf{p}_2)^T(\mathbf{p}_1 - \mathbf{p}_2), \tag{6.8}$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are the coordinates of P_1 and P_2, respectively.

As the norms are equivalent [43], other norms such as the weighted L_2-norm

$$d^2(\mathbf{p}_1, \mathbf{p}_2) = (\mathbf{p}_1 - \mathbf{p}_2)^T C(\mathbf{p}_1 - \mathbf{p}_2), \tag{6.9}$$

the L_∞-norm

$$d(\mathbf{p}_1, \mathbf{p}_2) = \max |\mathbf{p}_1 - \mathbf{p}_2| \quad \text{(element-wise maximum)} \tag{6.10}$$

etc., can be used.

They also have intuitive geometrical interpretations, as shown in Fig. 6.8. For example, in the parameter space, the L_2-norm worst-case distance correspond to a hyper-sphere, the weighted L_2-norm corresponds to a hyper-ellipsoid, and the L_∞-norm corresponds to a hyper-rectangle (orthotope).

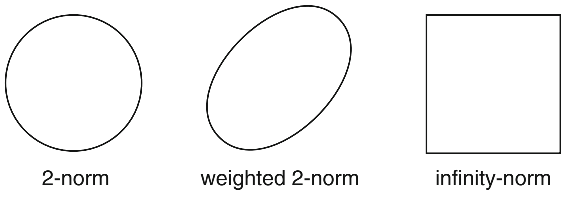


Fig. 6.8 Geometrical interpretations of norms

After the worst-case point on the boundary is returned from the nonlinear optimizer, the tangent hyperplane on this point is calculated by performing a sensitivity analysis. This tangent hyperplane separates the parameter space into two parts, one of which is assumed to be the failure region, and the other to be the acceptable region. This makes rough yield estimation an easy problem, because the multi-dimensional integration degrades to one-dimensional integration, whose computational cost is almost negligible.

When multiple constraints are considered simultaneously, the Monte-Carlo method is again employed to compute the integration, just as in simplicial approximation methods.

This method is good in situations where only rough yield estimation is desired. Because for each performance constraint, there can be infinite number of parameters that achieve the worst-case performance, choosing one point for each constraint and using the tangent hyperplane to approximate the boundary is just a coarse approximation.

6.2.2.3 Yield Estimation by Ellipsoidal Approximation

Ellipsoidal approximation methods [11, 44] are a trade-off between simplicial approximation methods and worst-case distance approximations: they are computationally cheaper than simplicial approximation methods, and are more accurate than worst-case distance approximations.

Unlike the previous two methods, which use local approximations to the boundary, ellipsoid approximation methods iteratively identify a hyper-ellipsoid to obtain a global approximation to the boundary. Variants of ellipsoid approximation differ in the way the hyper-ellipsoid is defined and calculated. For example, in [11], the ellipsoid is defined to be the largest possible one whose axial end points do not violate performance constraints. As another example, the method of [44] generates a sequence of hyper-ellipsoids of decreasing hyper-volume until a final ellipsoid satisfying a specified hyper-volume reduction ratio is obtained.

There are several advantages to using ellipsoidal approximations: firstly, the representation of a hyper-ellipsoid in a high-dimensional space is simple, i.e., it is a quadratic equation

$$(\mathbf{p}-\mathbf{p}_0)^T C^{-1}(\mathbf{p}-\mathbf{p}_0) = 1, \tag{6.11}$$

where $\mathbf{p}_0$ is the center of the ellipsoid and C is a positive definite matrix. This avoids the storage limit that is a bottleneck for simplicial approximation methods.

Secondly, the hyper-volume of the hyper-ellipsoid has an elegant formula

$$V = V_0\sqrt{|C|}, \tag{6.12}$$

where V_0 is the hyper-volume of the unit sphere. This makes it possible to use deterministic methods, rather than the Monte-Carlo method, to perform the numerical integration in the last step.

6.2.2.4 Yield Estimation by Euler–Newton Curve Tracing

Euler–Newton curve tracing [45] is another way to find the boundaries of the acceptable region D, and is best suited to the two-parameter case. Euler–Newton curve tracing is a kind of continuation method [46] to trace a curve in the parameter space. The idea is to start from a point on the boundary curve, and then to follow the curve by using the Newton–Raphson method to find adjacent points. The outline of the algorithm is as follows:

1. Use the Moore–Penrose pseudo-inverse Newton–Raphson (MPNR) method (discussed in Sect. 6.3.5) to solve for one point on the boundary;
2. Euler–Newton curve tracing:
   (a) Perform an Euler step (see later) to predict an approximate adjacent point on the boundary;
   (b) Using the Euler prediction as the initial guess for MPNR, solve for the next point on the boundary;
   (c) Repeat this prediction–correction scheme until the boundary is identified;
3. Calculate the yield by performing numerical integration of the region inside the boundary.

In this algorithm, the MPNR method is an iterative method to solve underdetermined nonlinear algebraic equations. It starts from an initial guess to the solution, and iteratively refines the solution until a required accuracy is reached. The Euler step is a prediction for a new point on the boundary. It follows the tangent at a point on the curve, and feeds the result into MPNR as the initial guess. Therefore, the Euler–Newton curve tracing method is basically a prediction–correction method [43], where the Euler step serves as the predictor and the MPNR method serves as the corrector. In the last step, there are various choices for the numerical integration method [43], such as trapezoidal rule, Simpson's rule, etc.

As an example, we consider the SRAM read access failure problem regarding variations of threshold voltages of $M1$ and $M2$. Figure 6.9a shows the parameter space of V_{th1} and V_{th2}. Suppose the curve in Fig. 6.9a is the boundary separating the acceptable region and the failure region. In the first step, we solve (6.7) to obtain one point on the boundary using MPNR – given an initial guess at point A, MPNR converges to point B on the boundary. The initial guess can be given heuristically, such as an approximate solution from designer's intuition. In MPNR, the computation of (6.7) is through a full-SPICE transient simulation, and its Jacobian matrix can be obtained by performing a sensitivity analysis (discussed in Sect. 6.3.6) or otherwise approximated.

After the starting point is obtained, we take an Euler step (denoted by the arrow $A_N \rightarrow B_E$ in Fig. 6.9b) to predict the next point on the curve. For example, starting from A_N, the Euler step predicts the next point to be B_E. The Euler prediction usually gives a very good initial guess for the next point on the curve, and therefore it makes Newton–Raphson achieve its local quadratic convergence. This prediction–correction procedure is repeated until the boundary is identified. In practice, this prediction–correction procedure turns out to be quite efficient – only

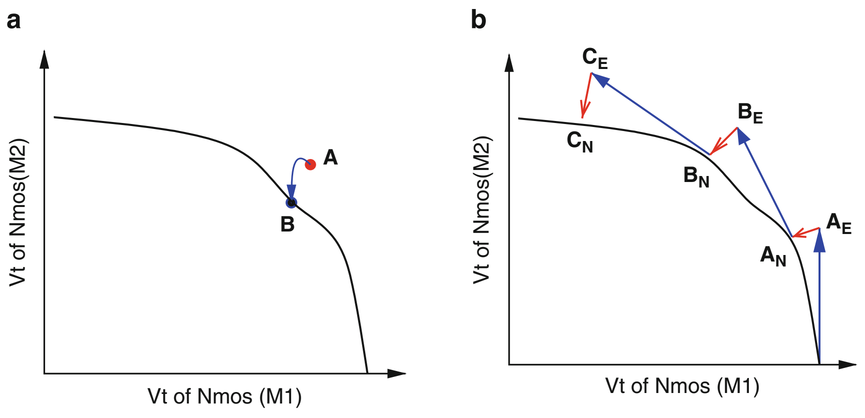


Fig. 6.9 (**a**) Convergence of a point via Newton–Raphson on the boundary. (**b**) Illustration of Euler–Newton curve tracing

up to 2–3 Newton iterations are needed to identify a new point on the boundary. Finally, a numerical integration formula, such as the trapezoidal rule [43], is employed to calculate the (probabilistically weighted) area inside the boundary.

However, the main drawback of the Euler–Newton curve tracing method is also obvious – it cannot be easily generalized to high-dimensional problems. For example, when three parameters are taken into consideration, the Euler step must be defined in two directions.

6.2.2.5 YENSS Sampling

Another method, YENSS will be discussed in detail in the next section. Like the Euler–Newton method, it also fully characterizes the boundary in the parameter space, and calculates the probabilistic hypervolume of the region inside the boundary. For the underlying computations involved in finding points on the boundary, YENSS supports two versions of the algorithm, based on Moore–Penrose Newton–Raphson and line search [47], respectively – the latter can be useful in situations where parameter derivative/sensitivity information, required by Newton–Raphson, is difficult or expensive to obtain.

Importantly, unlike the Euler–Newton curve tracing method, YENSS generalizes to high dimensions. However, if there are many parameters to be considered, fully characterizing a high-dimensional boundary manifold is also very computationally expensive. To alleviate this problem, YENSS uses an adaptive technique to calculate only as many points on the boundary as needed to provide an estimate of yield to a desired accuracy. By doing so, YENSS avoids redundant/unnecessary simulations in identifying the boundary manifold. This is especially useful when many parameters are considered simultaneously: the accuracy is controlled by the "yield volume" increment at each iteration, compared to $\frac{1}{\sqrt{N}}$ accuracy improvement in Monte-Carlo simulation, where N is the number of samples.

A crucial feature of YENSS, which is central to its efficiency for problems with many parameters, is its automatic exploitation of *linearity* of the boundary. If the boundary is perfectly linear (e.g., a plane embedded in 3-dimensional parameter space), the computational expense of YENSS increases only linearly with the number of parameters N_p. Because it adaptively estimates the probabilistic volume, the computation needed by YENSS increases gracefully to handle boundaries that are not linear. As failure/acceptable boundaries in parameter space tend to be close to linear for many practical problems, YENSS can achieve great speedups over Monte-Carlo.

6.2.2.6 Normal Boundary Intersection: The Reverse Problem

The deterministic methods we have discussed so far, including simplicial approximation, worst-case distance approximation, ellipsoidal approximation, Euler–Newton curve tracing and YENSS, are all concerned with boundaries in the parameter space. Given a region of desired (e.g., worst-case) performances of the circuit, the corresponding boundary in the parameter space is determined, as depicted by the lower arrow in Fig. 6.4. Techniques that solve the reverse problem (i.e., given a region in the parameter space such as a hypercube, find the corresponding boundary in the performance space, as depicted by the upper arrow in Fig. 6.4), also exist, and are important in performance optimization and trade-off analysis.

While this is a different class of problem from the yield estimation problem considered here, we note that the method of [21] uses a boundary approximation technique in the performance space that is similar to the methods used to solve for the boundary in YENSS and the simplicial approximation. It uses a search scheme (termed *Normal Boundary Intersection* (NBI)) to find equally spaced points on the boundary in the performance space. Instead of using a root-finding algorithm to identify the boundary, NBI formulates the problem as a sequential quadratic programming problem [47], and uses numerical optimization methods to solve for the boundary. As this method is also dealing with the boundaries, it can also be potentially adapted to solve the yield estimation problem.

Several core underlying subroutines are similar in these methods. For example, simplicial, worst-case distance, and ellipsoidal approximation methods all require a last step of Monte-Carlo simulation; simplicial approximation and nonlinear surface sampling methods both involve a line search step to identify a new point on the boundary, etc.

In the rest of the chapter, we will focus on YENSS in greater detail, in the context of which we will point out several reusable subroutines (such as principal component analysis, Newton–Raphson, line search, transient sensitivity analysis) that are used in other yield estimation methods.

6.3 Computing the Boundary and Probabilistic Hypervolumes

6.3.1 Basic Idea and Geometrical Explanation

The basic procedure of YENSS is the same as that of other deterministic yield estimation methods, and consists of two steps:

1. Identify the boundary in the parameter space (approximate the boundary manifold by hyper-triangular patches);
2. Compute the probabilistic hypervolume of the region inside the boundary manifold.

To illustrate the idea, we first consider the following simple problem, after which we deal with generalizations to more complex situations. Referring again to Fig. 6.3, suppose we have two independent parameters p_1 and p_2, with nominal values $p_{1\text{nom}}$, $p_{2\text{nom}}$, uniformly distributed over $[p_{1\min}, p_{1\max}]$ and $[p_{2\min}, p_{2\max}]$, respectively. As a result, in YENSS, the boundary curve, which is defined by a scalar nonlinear equation $h(\mathbf{p}, \mathbf{f_p}) = 0$, is first identified, and then the yield is computed to be the ratio of the area of region D to the area of the min/max box

$$\text{Yield} = \frac{\text{Area}_D}{(p_{1\max} - p_{1\min})(p_{2\max} - p_{2\min})}. \tag{6.13}$$

6.3.1.1 Extension to Multiple Parameters

The earlier notion is easily generalized to the case of multiple parameters. For the case of three parameters, the boundary is a 2-dimensional surface in the 3-dimensional parameter space, and the yield is the normalized volume of the 3-dimensional region that corresponds to acceptable performance in parameter space. For example, Fig. 6.10 shows a 3-dimensional parameter space, in which the boundary manifold is approximated by seven samples on it. The yield then is

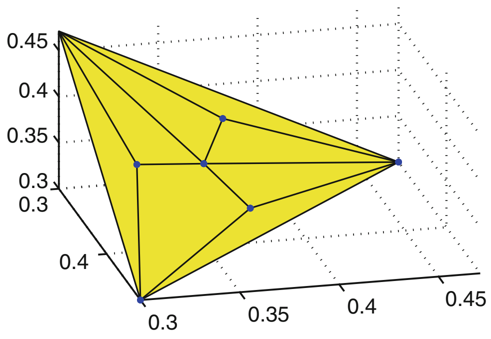


Fig. 6.10 Illustration of YENSS for three parameters

computed to be the ratio of the volume under this surface to the volume of the min/max box.

For the case of N_p parameters, the boundary is a $(N_p - 1)$-dimensional manifold that is defined by a scalar equation in the N_p-dimensional parameter space; the yield is the normalized hypervolume of the corresponding N_p-dimensional acceptable region in the parameter space.

6.3.1.2 Extension to Multiple Constraints

The earlier problem is an example where there is only one performance constraint. In this case, the boundary manifold is defined by one scalar equation:

$$h(\mathbf{p}, \mathbf{f_p}) = 0, \tag{6.14}$$

and the acceptable region (the region inside the boundary) is correspondingly defined by one scalar inequality

$$h(\mathbf{p}, \mathbf{f_p}) \leq 0. \tag{6.15}$$

A straightforward generalization to the case of multiple constraints is as follows: if there are N_c constraints, the region inside the boundary is defined by N_c inequalities

$$\begin{aligned} h_1(\mathbf{p}, \mathbf{f_p}) &\leq 0, \\ h_2(\mathbf{p}, \mathbf{f_p}) &\leq 0, \\ &\cdots, \\ h_{N_c}(\mathbf{p}, \mathbf{f_p}) &\leq 0, \end{aligned} \tag{6.16}$$

and the boundary is defined to be the common interior of the manifolds defined by $h_i(\mathbf{p}, \mathbf{f_p}) = 0, \forall i \in [1, N_c]$.

For example, Fig. 6.11 shows an example with three constraints in the 2-dimensional parameter space. Three stand-alone boundaries correspond to the boundaries defined by $h_1(\mathbf{p}, \mathbf{f_p}) = 0$, $h_2(\mathbf{p}, \mathbf{f_p}) = 0$, and $h_3(\mathbf{p}, \mathbf{f_p}) = 0$, respectively. Then the final boundary is the common interior of all the three separate boundaries, and the acceptable region is the shaded region D. The computational procedure to identify this compound boundary manifold will be discussed in Sect. 6.3.4.

6.3.1.3 Extension to Handling Probability Distributions

In practice, the parameters are not likely to all have uniform distributions. Other probability distributions, such as Gaussian and truncated Gaussian distributions

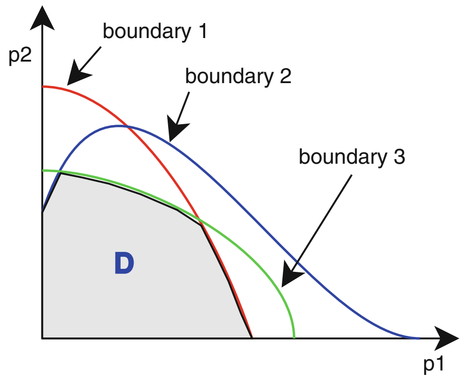


Fig. 6.11 Illustration of multiple constraints

[48, 49], are often used to model parameter variability. In this case, the hypervolume inside the boundary in the parameter space no longer represents the yield.

There are two ways in which one can still calculate yield based on the boundary. The first way is obvious from the formula Yield $= \int_D \pi(\mathbf{p})\mathrm{d}\mathbf{p}$. That is, using numerical integration of the probability density function $\pi(\mathbf{p})$ to compute the probabilistic hypervolume (or probability mass) of region D.

The second way is to transform the parameter axes $\mathbf{p}$ so that the transformed parameters $\hat{\mathbf{p}}$ are uniformly distributed. In order to do this, recall how to generate a random variable Y of probability density function $f_Y(y)$ from a random variable X which is uniformly distributed on [0, 1].

Suppose y is a monotonically increasing function of x, $y = g(x)$, i.e., we have an inverse function $x = g^{-1}(y)$.

By definition, we have

$$P(x < X < x + \mathrm{d}x) = f_X(x)\mathrm{d}x, \tag{6.17}$$

$$P(y < Y < y + \mathrm{d}y) = f_Y(y)\mathrm{d}y. \tag{6.18}$$

On the other hand, because $y = g(x)$, when x varies from x to $x + \mathrm{d}x$, y varies from $y = g(x)$ to $y + \mathrm{d}y = g(x) + g'(x)\mathrm{d}x$ (see Fig. 6.12). Therefore, the probability that $X \in [x, x + \mathrm{d}x]$ should equal the probability that $Y \in [g(x), g(x) + g'(x)\mathrm{d}x]$. So, if $\mathrm{d}y = g'(x)\mathrm{d}x$, we have

$$P(x < X < x + \mathrm{d}x) = P(y < Y < y + \mathrm{d}y). \tag{6.19}$$

Inserting (6.17) and (6.18) into (6.19), we obtain

$$f_X(x)\mathrm{d}x = f_Y(y)\mathrm{d}y. \tag{6.20}$$

Because X is uniformly distributed on [0, 1], $f_X(x) = 1$ on [0, 1]. Therefore,

$$x = \int_{-\infty}^{y} f_Y(y)\mathrm{d}y = F_Y(y), \tag{6.21}$$

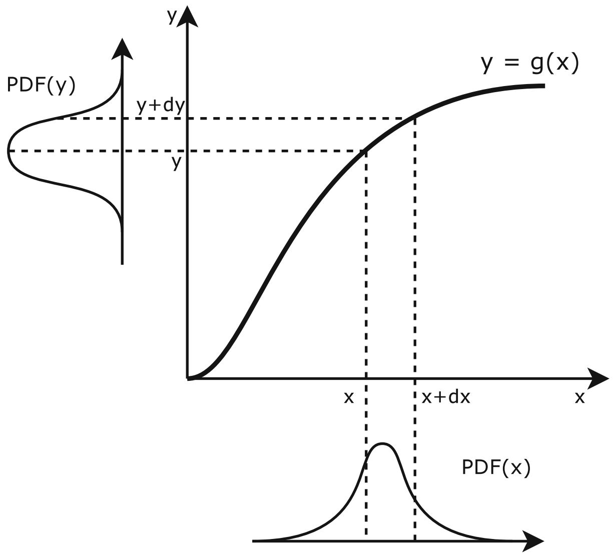


Fig. 6.12 Transformation of probability density function

or

$$y = F_Y^{-1}(x), \tag{6.22}$$

where $F_Y(y)$ is the cumulative density function of Y. Therefore,

$$g(x) = F_Y^{-1}(x). \tag{6.23}$$

As the cumulative density function $F_Y(y)$ is a monotonically increasing function, its inverse is also monotonically increasing, and our assumption regarding the monotonicity of $g(x)$ is validated.

Hence, if we have a random variable x uniformly distributed on [0, 1] then $F_Y^{-1}(x)$ is a random variable with PDF $f_Y(y)$ [2].

Similarly, here we have an inverse problem to solve, i.e., we have a random variable Y with PDF $f_Y(y)$, and we want to compute the corresponding random variable that is uniformly distributed on [0, 1]. From the previous derivation,

$$x = F_Y(y) \tag{6.24}$$

is the random variable we want.

So, given parameters $\mathbf{p} = [p_1, \ldots, p_{Np}]$, we first transform $\mathbf{p}$ to their cumulative density functions $\hat{p}_i \equiv F_i(p_i)$, $i \in [1, N_p]$, such that the scaled parameters $\hat{\mathbf{p}} = [\hat{p}_1, \ldots, \hat{p}_{N_p}]$ are uniformly distributed. For example, as shown in Fig. 6.13, the axes p_1 and p_2 in Fig. 6.3 are replaced by $\hat{p}_1 \equiv F_1(p_1)$ and $\hat{p}_2 \equiv F_2(p_2)$,

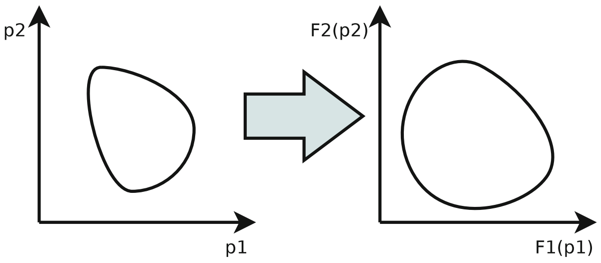


Fig. 6.13 Handling parameter probability distributions

respectively, where F_1 and F_2 are the cumulative distribution functions of the two parameters. Accordingly, the boundary in the original parameter space $\mathbf{p} \in \mathbb{R}^{N_p}$ is effectively mapped to a different boundary in the probabilistically deformed parameter space $\hat{\mathbf{p}} \in \mathbb{R}^{N_p}$. As $\hat{\mathbf{p}}$ are uniformly distributed, and if we assume that parameters $\mathbf{p}$ ($\hat{\mathbf{p}}$) are independent then the hypervolume of the region inside this new boundary is the yield.

The second method is preferred in YENSS because YENSS computes the yield by computing hypervolume of the acceptable region, which relies on the assumption of uniform distributions of parameters.

6.3.1.4 Extension to Handling Correlations of Parameters

The assumption that parameters are independent (and hence uncorrelated) is also important for the hypervolume calculated to be the yield. But this assumption is too strong to be true in reality. For example, W and L of a transistor are typically closely correlated with each other. So in the preprocessing step, we also need to de-correlate the parameters, and this can be done by principal component analysis (PCA) [50].

To see how PCA de-correlates parameters, we shall start with the definition of correlation between parameters/random variables. Suppose there are N_p parameters $\mathbf{p} = [p_1, \ldots, p_{N_p}]^T$, and $E[\mathbf{p}] = [\mu_1, \ldots, \mu_{N_p}]^T$. Here $E[\mathbf{p}]$ can be viewed as the nominal design parameters, and $\Delta\mathbf{p} = \mathbf{p} - E[\mathbf{p}]$ are zero-mean random variables that account for random parameter variations.

The covariance matrix R for $\Delta\mathbf{p}$ is defined by:

$$R = E[\Delta\mathbf{p}\Delta\mathbf{p}^T] = \begin{bmatrix} E[\Delta p_1 \Delta p_1] & E[\Delta p_1 \Delta p_2] & \cdots & E[\Delta p_1 \Delta p_{N_p}] \\ E[\Delta p_2 \Delta p_1] & E[\Delta p_2 \Delta p_2] & \cdots & E[\Delta p_2 \Delta p_{N_p}] \\ \vdots & \ddots & \ddots & \vdots \\ E[\Delta p_{N_p} \Delta p_1] & E[\Delta p_{N_p} \Delta p_2] & \cdots & E[\Delta p_{N_p} \Delta p_{N_p}] \end{bmatrix}. \quad (6.25)$$

Then, by definition, parameters $\mathbf{p}$ are uncorrelated if and only if R is diagonal.

In PCA, we try to find a linear transformation $\Delta\mathbf{q} = V\Delta\mathbf{p}$, so that the covariance matrix $R_{\mathbf{q}}$ for $\Delta\mathbf{q}$ is diagonal. As a result, we compute

$$\mathbf{q} = V\mathbf{p} = V(E(\mathbf{p}) + \Delta\mathbf{p}) = E(V\mathbf{p}) + \Delta\mathbf{q}. \tag{6.26}$$

As $\Delta\mathbf{q} = V\Delta\mathbf{p}$ are uncorrelated, and $E(V\mathbf{p})$ is constant, $\mathbf{q}$ are uncorrelated. Inserting $\Delta\mathbf{q} = V\Delta\mathbf{p}$ into the definition of $R_\mathbf{q}$, we obtain

$$R_\mathbf{q} = E[\Delta\mathbf{q}\Delta\mathbf{q}^T] = E[V\Delta\mathbf{p}\Delta\mathbf{p}^T V^T] = VE[\Delta\mathbf{p}\Delta\mathbf{p}^T]V^T = VRV^T. \tag{6.27}$$

The key step in achieving de-correlation is to eigen-decompose [51] the matrix R

$$R = P\Lambda P^{-1} = P\Lambda P^T, \tag{6.28}$$

where $P^{-1} = P^T$ because R is symmetric.

Inserting (6.28) into (6.27), we have

$$R_q = VP\Lambda P^T V^T. \tag{6.29}$$

Choosing $V = P^T$ then makes R_q a diagonal matrix. Therefore, the transformed parameters $\mathbf{q} = P^T\mathbf{p}$ are uncorrelated.

PCA successfully provides a means to remove second-order correlation between parameters. However, independence of parameters are still not guaranteed after PCA. If no higher-order correlations are present, or if higher-order correlations are negligible, the independence of parameters remains a reasonable assumption. This assumption is typically made in practice.

As an example of PCA, Fig. 6.14 shows 1,000 samples of parameters $\mathbf{p} = [p_1, p_2]^T$ which are generated from two independent Gaussian variables $\mathbf{q} = [q_1, q_2]^T$. Here we choose $q_1 \sim N(0, 1^2)$ and $q_2 \sim N(0, 3^2)$, and $p_1 = \frac{2}{\sqrt{5}}q_1 - \frac{1}{\sqrt{5}}q_2$, $p_2 = \frac{1}{\sqrt{5}}q_1 + \frac{2}{\sqrt{5}}q_2$. Therefore, we have

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} \dfrac{2}{\sqrt{5}}p_1 + \dfrac{1}{\sqrt{5}}p_2 \\ -\dfrac{1}{\sqrt{5}}p_1 + \dfrac{2}{\sqrt{5}}p_2 \end{bmatrix}. \tag{6.30}$$

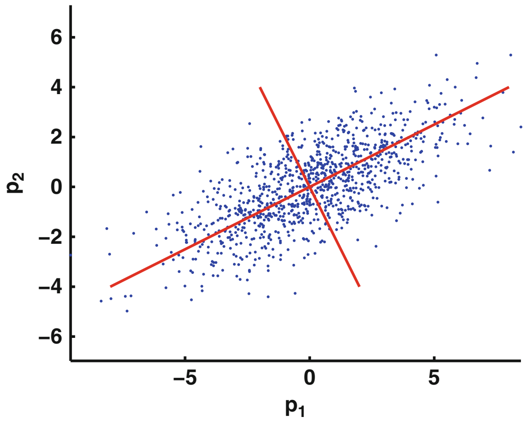


Fig. 6.14 Illustration of PCA

We assume only the covariance matrix for $\mathbf{p}$, which can be approximated from the samples, is available, and perform PCA to rediscover the uncorrelated random variables q_1 and q_2.

Starting with the covariance matrix for $\mathbf{p}$

$$R = \begin{bmatrix} \frac{13}{5} & -\frac{16}{5} \\ -\frac{16}{5} & \frac{37}{5} \end{bmatrix}, \tag{6.31}$$

we perform eigen-decomposition of R, and obtain $R = P\Lambda P^T$, where

$$P = \begin{bmatrix} \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}. \tag{6.32}$$

Therefore, the uncorrelated variables are

$$V\mathbf{p} = P^T\mathbf{p} = \begin{bmatrix} \frac{2}{\sqrt{5}}p_1 + \frac{1}{\sqrt{5}}p_2 \\ -\frac{1}{\sqrt{5}}p_1 + \frac{2}{\sqrt{5}}p_2 \end{bmatrix} \tag{6.33}$$

which coincide with q_1 and q_2. This example shows that PCA is able to automatically identify uncorrelated random variables.

Up to this point, we have shown that the idea of computing yield by probabilistic hypervolumes can be generalized to multiple parameters and multiple constraints, and that nonuniformly distributed and/or correlated parameters can, in typical situations, be mapped into uniformly distributed, uncorrelated parameters. Therefore, in the following sections, we only focus on the core problem which involves only uniformly distributed, uncorrelated parameters.

6.3.2 YENSS Algorithm Outline

The overall algorithm flow of YENSS is outlined later, and we explain and develop each step in detail in the following sections.

1. Preprocessings:
   (a) Perform a principal component analysis to de-correlate parameters (Sect. 6.3.1);
   (b) Transform parameters so that they are all uniformly distributed on [0, 1] (Sect. 6.3.1).

2. Formulate the boundary manifold as a nonlinear scalar equation (performance constraint equation) in the parameter space:

$$h(\mathbf{p}; \mathbf{f_p}) = 0. \tag{6.34}$$

   (6.34) expresses the relationship between parameters **p** and the circuit performance of interest $\mathbf{f_p}$ in a general, implicit form.
3. Find the intersection points of the boundary with each parameter axis.
   (a) Augment equation (6.34) with parametric equations (6.40) of each parameter axis.
   (b) Solve the system of (6.34) and (6.40), the solution is the intersection of the parameter axis and the boundary manifold.
4. Use an analytical volume calculation formula to calculate the initial approximation to the yield – the hypervolume of the simplex defined by those intersections.
5. Volume refinement using additional boundary samples.
   (a) Choosing and computing additional points on the boundary manifold.
      i. Construct the manifold by connecting the points already found in previous refinements, and choose the centroid of this manifold to be the initial guess P_{guess} (see Fig. 6.16 for an example) for the point to be found on the boundary.
      ii. Either use the MPNR method to solve for the intersection point;
      iii. Or, use the line search method to solve for the intersection point.
         A. Construct the parametric equations for the line that goes through point P_{guess} and is perpendicular to the manifold constructed in step 5(a)i.
         B. Apply line search to find the intersection of the boundary and this line, using either the Newton–Raphson method or the secant/bisection method [43].
         C. If the Newton–Raphson method is used, the derivatives of function $h(\cdot)$ with respect to parameters **p** need to be evaluated by performing a sensitivity analysis of the circuit.
         D. If calculating the derivatives is computationally expensive or practically difficult, bisection, the secant method, and other line search techniques [47] can be applied to find the intersection points.
   (b) Update the yield estimate using the newly found boundary points.
      i. The yield volume increment is evaluated by calculating the hypervolume of the hyper-triangle defined by the newly found point and previous boundary points, using an analytical formula.
      ii. Based on the volume increment, an error metric is estimated for the current yield approximation.
      iii. If the error estimate is small enough, finish the algorithm;
      iv. Else, repeat the volume refinement procedure step 5 until sufficiently small error is obtained.

6.3.3 Implicit Formulation for the Boundary

We start by expressing the relationship between the performance metric and the varying parameters as an implicit *scalar equation* (6.34) which encapsulates all the complex nonlinear dynamics in the circuit. Suppose $\mathbf{p} \in \mathbb{R}^{N_p}$, then this scalar equation defines a $(N_p - 1)$-dimensional manifold/hypersurface in the parameter space. Our goal is to find points on the surface, i.e., to solve this nonlinear equation, efficiently and accurately.

Normally, there is only one acceptable region that is defined by:

$$h(\mathbf{p}; \mathbf{f_p}) \equiv \mathbf{f_p}(\mathbf{p}) - \mathbf{f}_{\text{worst}} = 0. \tag{6.35}$$

In situations where there are several acceptable regions, YENSS can be applied on each acceptable region separately.

Depending on the problem or the circuit being considered, (6.34) can encapsulate different kinds of analyses. We next provide two examples to illustrate how (6.34) can be formulated: an SRAM read access time example which is based on SPICE-level transient simulation, and an SRAM static noise margin (SNM) example computed by a series of DC analyses of the circuit.

6.3.3.1 SRAM Read Access Time

Consider an SRAM cell with a read access time constraint. Recall from Sect. 6.1.2 that the read access time of a SRAM cell is the time between the word line going high and ΔBL reaching ΔBL_{min}.

Suppose the maximum acceptable read access time is t_f, then a direct application of (6.35) gives the boundary equation

$$h(\mathbf{p}; \mathbf{f_p}) \equiv h(\mathbf{p}; t_f) \equiv t(\mathbf{p})_{\Delta BL = \Delta BL_{\text{min}}} - t_f = 0, \tag{6.36}$$

where $t(\mathbf{p})_{\Delta BL = \Delta BL_{\text{min}}}$ denotes the time at which ΔBL reaches ΔBL_{min}.

Given a maximum read access time t_f, the voltage difference ΔBL at time $t = t_f$ can be calculated through performing a transient analysis on the SRAM circuit. If this voltage difference is smaller than the minimum voltage difference ΔBL_{min} which can be detected by the sense amplifier, the read failure will occur. Therefore, we obtain another equation defining the boundary manifold

$$h(\mathbf{p}; \mathbf{f_p}) \equiv h(\mathbf{p}; \Delta BL) \equiv \Delta BL(\mathbf{p})_{t=t_f} - \Delta BL_{\text{min}} = 0. \tag{6.37}$$

Note that (6.36) and (6.37) define the same boundary manifold, and that both of them are computed by performing a transient analysis. However, the formulation using (6.36) requires a transient simulation until the point when $\Delta BL = \Delta BL_{\text{min}}$, while the formulation using (6.37) requires a transient simulation from $t = 0$ to $t = t_f$.

Besides, ΔBL can be directly obtained from the simulation results, while $t_{\Delta BL=\Delta BL_{\min}}$ must be approximated in some way. Therefore, computing (6.37) is easier than (6.36), and we prefer the (6.37) formulation for this problem.

6.3.3.2 SRAM Static Noise Margin while Holding Data

The static noise margin of an SRAM cell (SNM) quantifies the amount of the voltage noise required at the internal nodes of the SRAM cell to flip the data stored. It is found graphically as the length of the side of the largest square which can fit between the voltage transfer curves (VTCs) of the two inverters. For example, as shown in Fig. 6.15, curves VTC1 and VTC2 represent the voltage transfer curves for the two inverters, respectively; and the square is the largest square that can be fitted between the two VTCs – the length of its side is the SNM.

Therefore, SNM is computed by two steps: (1) Perform a series of DC analysis on the two inverters in the SRAM cell, obtain the VTCs; (2) Draw the two VTCs together, with one of them inverted, and fit the largest square between the VTCs. The SNM is then computed to be the length of the side of this square.

If we denote SNM to be S and the minimum acceptable SNM to be $S_{\min}$, then the boundary equation is

$$h(\mathbf{p};\mathbf{f_p}) \equiv h(\mathbf{p};S) \equiv S(\mathbf{p})_{t=t_f} - S_{\min} = 0. \tag{6.38}$$

Again, we have a scalar equation to describe the boundary, and the computation of this scalar equation is via DC analysis and identification of the largest square that fits between the VTCs.

6.3.4 *Solving for the Boundary Using Line Search*

The main computational effort in YENSS is spent on solving for the boundary manifold. This includes step 3 (to find the intersection of the boundary and each

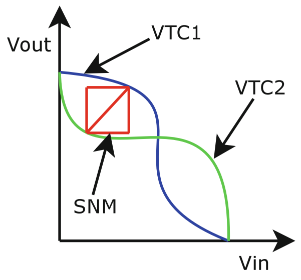


Fig. 6.15 Computing the static noise margin of an SRAM cell

parameter axis), and step 5 (to adaptively identify more points on the boundary). In this section, we discuss how a line search method is employed to solve for the boundary.

To start the YENSS algorithm, we need to first solve for the intersections of the boundary with each parameter axis. Note that it is possible that the boundary defined by the nonlinear scalar equation does not have any intersection with a parameter axis. However, in practice, parameter values are always bounded, hence the minimum/maximum points can be used for the intersection of the boundary with the parameter axis.

Solving for intersections of the boundary with each parameter axis is equivalent to solving the nonlinear equation (6.34) by fixing all the parameters but one (say p_i) at their nominal parameter values.

$$\begin{aligned} &h(\mathbf{p}, \mathbf{f_p}) = 0 \\ &p_j = p_{j\text{nom}}, \quad \forall j \neq i \end{aligned} \tag{6.39}$$

So the only unknown variable in the equation (6.34) is now a scalar p_i, rather than a vector $\mathbf{p}$ – we obtain a scalar equation with a scalar unknown variable. Therefore, we can apply any nonlinear solver, such as the secant method or the Newton–Raphson method, to find the intersections.

To use the Newton–Raphson method, however, we also need to calculate the Jacobian $\mathrm{d}h/\mathrm{d}\mathbf{p}$. Depending on the analysis used to compute $h(\mathbf{p}, \mathbf{f_p})$, we employ a corresponding sensitivity analysis to compute this quantity.

It is interesting to note that up to this stage, the computation is linear in the number of parameters. This implies that if the performance function is a linear or nearly linear function of the parameters then the computational expense of this method is linear in the number of parameters. Many practical problems have an approximately linear boundary surface, and therefore YENSS can be significantly faster than Monte-Carlo for these problems [9].

More generally, the earlier procedure to find the intersection points can be formulated as a line search method. Indeed, it is a special case of finding the intersections of the boundary manifold with any straight line – the line to be searched is the parameter axis in this case.

To force the solution of $h(\mathbf{p}, \mathbf{f_p}) = 0$ to lie on a line, we augment this equation by the equations defining the line. In a N_p-dimensional space, a straight line can be characterized by $N_p - 1$ linear equations. Equivalently, we can also use the N_p parametric equations

$$\mathbf{p} = \mathbf{p}_0 + s\mathbf{u} \tag{6.40}$$

as the line constraint. In (6.40), $\mathbf{p}_0$ is the coordinate of a point that is on the line, $\mathbf{u}$ the unit vector denoting the direction of the line, and s is a new scalar unknown variable introduced to parameterize the line.

For example, the parametric equation of p_1 axis can be written as:

$$\begin{aligned} p_1 &= s, \\ p_i &= 0, \quad 2 \le i \le N_p, \end{aligned} \tag{6.41}$$

where $\mathbf{p}_0 = [0, \ldots, 0]^T$, and $\mathbf{u} = [1, 0, \ldots, 0]^T$.

Before augmenting with line constraint equations, we have a single scalar equation $h(\mathbf{p}, \mathbf{f_p}) = 0$, but the number of parameters $\mathbf{p}$ is N_p. Therefore, the number of equations is less than the number of unknowns, and the equation system is under-determined.

After the performance constraint equation $h(\mathbf{p}, \mathbf{f_p}) = 0$ is augmented by the line constraint equations, we have one performance constraint equation and N_p parametric equations for the line, and we have N_p unknown variables $\mathbf{p}$ and one variable s introduced to parameterize the line. Therefore, the number of unknown variables and the number of equations are both $N_p + 1$, and the augmented equations are no longer under-determined.

Therefore, the Newton–Raphson method can be applied to this square system to find the intersection points. Or equivalently, we can formulate the scalar equation for the scalar unknown variable s as

$$h(\mathbf{p}, \mathbf{f_p}) = h(\mathbf{p}_0 + s\mathbf{u}, \mathbf{f_p}) = 0, \tag{6.42}$$

so that simpler and more robust one-dimensional nonlinear solvers, such as the bisection or secant methods, can be applied to search the line.

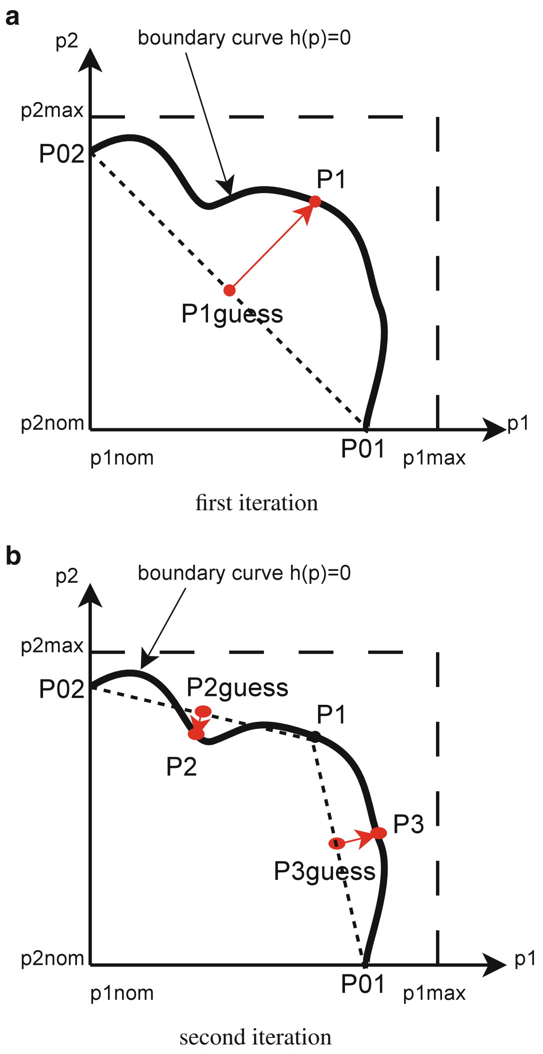
After the intersections with the parameter axes are found, the line search procedure is performed iteratively to find additional points on the boundary. After the points have been found, we can construct locally linear $(N_p - 1)$-dimensional hyperplanes from sets of N_p points. Stitching together all those hyperplanes gives a piecewise linear approximation of the boundary.

The key problem here is to ensure that the points are a good representation of the boundary manifold, at least from the standpoint of hypervolume computation. To achieve this, one desired property is that the points on the boundary evenly spread out – this gives a better global approximation of the boundary than the case where many points are clustered in a small region.

To try to align all the points evenly on the boundary surface, we construct a specific line to be searched with – it has two properties: (1) it goes through the centroid P_{guess} of the N_p points that constitute a locally linear hyperplane; (2) it is perpendicular to this locally linear hyperplane. Constructing this line and using P_{guess} as an initial guess or initial search point, we apply the line search method on the augmented system to find the intersection of this line with the real boundary. If the real boundary is almost linear, this line constraint will lead to a new point that is almost at the centroid of the boundary. In this case, the initial guess P_{guess} is also very close to the real solution, which implies that the nonlinear solver will converge in very few iterations.

For illustration, the results of two boundary refinement iterations in a 2-dimensional parameter space are shown in Fig. 6.16a, b, where the bold curve represents the

Fig. 6.16 Finding points on the boundary curve using line search (two iterations)



boundary curve defined by (6.35). In Fig. 6.16a, P_{01}, P_{02} are two intersections with parameter axes, calculated in the first step. Then $P_{1\text{guess}}$ is chosen to be the centroid of $P_{01}P_{02}$, and the line $P_{1\text{guess}}\,P_1$ which is perpendicular to $P_{01}P_{02}$ is constructed. The performance constraint equation $h(\mathbf{p}, \mathbf{f}_\text{p}) = 0$ is then augmented by the parametric equations for line $P_{1\text{guess}}\,P_1$. Using $P_{1\text{guess}}$ to be the initial guess for Newton–Raphson solver, it finally converges to P_1, and Newton iterations effectively search along the line $P_{1\text{guess}}\,P_1$.

In the second boundary refinement iteration (shown in Fig. 6.16b), $P_{2\text{guess}}$ and $P_{3\text{guess}}$ are selected to be the centroid of $P_{02}P_1$ and $P_{01}P_1$, respectively, and Newton–Raphson will follow the lines $P_{2\text{guess}}\,P_2$ and $P_{3\text{guess}}\,P_3$, converging to

P_2 and P_3, respectively. Also notice that the search distance for P_2 and P_3 is much less than that for P_1. This is another good feature of this method – as the boundary becomes more linear, the search distance decreases and therefore the computational cost is lessened.

Repeating this procedure, we will finally obtain enough points on the boundary curve needed for yield estimation. Importantly, it is straightforward to apply this procedure to high-dimensional problems.

6.3.4.1 Dealing with Multiple Performance Constraints

The line search method discussed earlier is directly applicable to the problem with one performance constraint (i.e., there is only one performance constraint equation $h(\mathbf{p}, \mathbf{f_p}) = 0$). When there are multiple performance constraints, however, we have to find the common interior of boundaries corresponding to N_p performance constraint equations $\mathbf{h}(\mathbf{p}, \mathbf{f_p}) = [h_1(\mathbf{p}, \mathbf{f_p}), \ldots, h_{N_p}(\mathbf{p}, \mathbf{f_p})] = 0$.

Therefore, to find the boundary of the intersection of acceptable regions, we must find the "first" point among the points the line intersects, with boundaries corresponding to different constraints. For example, in Fig. 6.17, suppose the dotted line is the line to be searched. On this line, there are three intersection points, with the three curves which represent three different constraints on the circuit performance. Among the three intersection points, the "first" intersection point is A.

To do this, the previous line search method needs to be adjusted. Suppose there are N_p performance constraints, defined by $h_i(\mathbf{p}) = 0$, $(i = 1, \ldots, N_p)$. The algorithm to find the samples on the interior of all the constraint boundaries is:

1. Mark all the constraints to be unsatisfied;
2. Loop until all the constraints are satisfied;
   (a) Pick up one unsatisfied constraint, calculate the intersection point using the line search method;
   (b) Test all the unsatisfied constraints;
   (c) If all the constraints are met, terminate;
   (d) Else, go to step 2a.

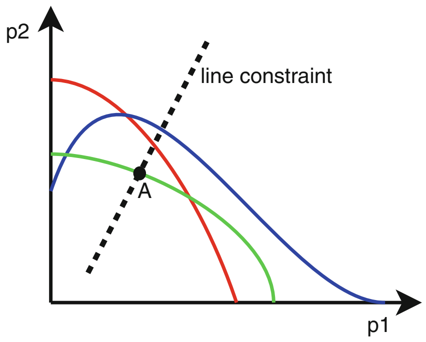


Fig. 6.17 Illustration of line search when there are multiple constraints

6.3.5 Solving for the Boundary Using Moore–Penrose Pseudo-Inverse Newton–Raphson

Another option is to use the Moore–Penrose pseudo-inverse Newton Raphson (MPNR) method [46] to solve the scalar equation (6.35) directly, without augmenting with additional linear equations. While the classical Newton–Raphson method can only be applied to square systems, MPNR can be used to solve under-determined equations.

The algorithm is similar to the classical Newton–Raphson method: to solve $f(x) = 0$, it starts with an initial guess x_0, and then updates x_i iteratively until the accuracy is within tolerance.

Its difference from classical Newton–Raphson is in the update step. In classical Newton–Raphson, the update step is

$$x_{i+1} = x_i - J(x_i)^{-1} f(x_i), \tag{6.43}$$

where $J(x_i)$ is the Jacobian matrix at $x = x_i$. However, in our problem, we have 1 scalar equation and N_p unknown variables; the size of the Jacobian matrix $J(x_i)$ is $1 \times N_p$ (not square), and hence its inverse does not exist.

Instead of using the inverse of $J(x_i)$, MPNR uses its Moore–Penrose pseudo-inverse

$$J(x_i)^{+} = J(x_i)^T \left[J(x_i) J(x_i)^T\right]^{-1} \tag{6.44}$$

in the update step. Therefore, the update step formula in MPNR is

$$x_{i+1} = x_i - J(x_i)^T \left[J(x_i) J(x_i)^T\right]^{-1} f(x_i). \tag{6.45}$$

Unlike line search, which has a unique solution, MPNR may end up with any solution to the nonlinear equation. However, it can be proved [46] that, under the right circumstances, MPNR will converge to a point on the solution curve that is closest to the initial guess. Therefore, if we choose the initial guess in the same way as in the line search method, MPNR will also generate samples that are well spaced on the boundary.

6.3.6 Calculating the Jacobian Matrix Using Sensitivity Analysis

In order to use the Newton–Raphson method in line search, or the MPNR method, the Jacobian matrix of the system must be calculated by performing a sensitivity analysis of the circuit. Given (6.35), we differentiate $h(\,\cdot)$ with respect to $\mathbf{p}$:

$$\frac{\mathrm{d}h}{\mathrm{d}\mathbf{p}} = \frac{\partial h}{\partial \mathbf{p}} + \frac{\partial h}{\partial \mathbf{f_p}} \frac{\partial \mathbf{f_p}}{\partial \mathbf{p}}. \tag{6.46}$$

In (6.46), $\partial h/\partial \mathbf{p}$ and $\partial h/\partial \mathbf{f_p}$ are available after the function $h(\mathbf{p}, \mathbf{f_p})$ is defined. However, we usually do not have an analytical form for the performance metrics $\mathbf{f_p}(\mathbf{p})$, the quantity $\partial \mathbf{f_p}/\partial \mathbf{p}$ remains unknown, and must be evaluated by performing a simulation.

Depending on the type of analysis used, the algorithm to compute the sensitivity $\partial \mathbf{f_p}/\partial \mathbf{p}$ varies. We now present derivations of sensitivity analysis based on transient simulation, which is extensively used in many applications.

6.3.6.1 Transient Simulation Based Sensitivity Evaluation

In many problems such as the SRAM read access failure problem, evaluation of the boundary equation is performed through a transient analysis of the circuit, i.e., by doing a numerical integration on the differential algebraic equations of the circuit

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q}(\mathbf{x}) + \mathbf{f}(\mathbf{x}) + \mathbf{b}(t) = 0, \tag{6.47}$$

where $\mathbf{x}$ are the unknowns (node voltages and branch currents), and $\mathbf{b}(t)$ are the inputs to the circuit.

Again taking the SRAM read access failure problem as an example, we have the boundary equation:

$$h(\mathbf{p}) \equiv \Delta BL - \Delta BL_{\min} = \mathbf{c}^T\mathbf{x}(t_f;\mathbf{p}) - \Delta BL_{\min} = 0, \tag{6.48}$$

where $\mathbf{c}^T$ is a vector which extracts the voltage difference $\mathbf{f_p} = \Delta BL$ from the nodal voltages $\mathbf{x}$.

Therefore, the derivatives of $\mathbf{f_p}(\mathbf{p})$ with respect to $\mathbf{p}$ is

$$\frac{\partial \mathbf{f_p}(\mathbf{p})}{\partial \mathbf{p}} = \mathbf{c}^T\frac{\partial \mathbf{x}}{\partial \mathbf{p}}. \tag{6.49}$$

In order to calculate $\partial \mathbf{x}/\partial \mathbf{p}$, we differentiate the circuit equations (6.47) with respect to the parameters $\mathbf{p}$:

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\left[\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q}(\mathbf{x}) + \mathbf{f}(\mathbf{x}) + \mathbf{b}(t)\right] = 0. \tag{6.50}$$

Explicitly, we obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{p}} + \frac{\partial \mathbf{q}}{\partial \mathbf{p}}\right] + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{p}} + \frac{\partial \mathbf{f}}{\partial \mathbf{p}} + \frac{\mathrm{d}\mathbf{b}}{\mathrm{d}\mathbf{p}} = 0, \tag{6.51}$$

where $\partial \mathbf{q}/\partial \mathbf{x}$ and $\partial \mathbf{f}/\partial \mathbf{x}$ are already calculated during the transient simulation, and $\partial \mathbf{q}/\partial \mathbf{p}$, $\partial \mathbf{f}/\partial \mathbf{p}$, $\mathrm{d}\mathbf{b}/\mathrm{d}\mathbf{p}$ can be computed.

Therefore, (6.51) is a set of differential equations with unknowns $\partial \mathbf{x}/\partial \mathbf{p}$, and hence can be solved using any numerical integration method, such as the Backward-Euler method or the trapezoidal method [52, 53].

Thus, the sensitivities of the nodal voltages and branch currents $\mathbf{x}$ with respect to varying parameters $\mathbf{p}$ can be calculated, following which $\partial \mathbf{h}/\partial \mathbf{p}$ can be calculated.

6.3.7 Adaptive Hypervolume Refinement and Error Estimation

6.3.7.1 Analytical Formula to Compute the Hypervolume

Given non-collinear $N + 1$ points in a N-dimensional space, we can construct a simplex. For example, three points in a 2-dimensional plane constitute a triangle. The area/volume/hypervolume of this simplex can be simply calculated by:

$$C_{\text{simplex}} = \frac{1}{N!} \begin{vmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N} & 1 \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{N+1,1} & x_{N+1,2} & \cdots & x_{N+1,N} & 1 \end{vmatrix}, \tag{6.52}$$

where $(x_{i,\,1}, \ldots, x_{i,\,N})$ are the coordinates of the ith point. Notice that this formula actually might lead to a "negative hypervolume" – it can be shown that the sign represents the "direction" of the simplex, and can be utilized in the volume refinement step, as discussed later.

6.3.7.2 Yield Hypervolume Calculation and Error Control

As the new points on the boundary manifold are found, we update the hypervolume estimate by adding/subtracting the hypervolume of the new simplexes we find. For example, two iterations of the refinement and hypervolume update procedure are shown in Fig. 6.18. S_0 is the original area estimate after the axis intersections have been found. After one iteration, P_1 is found, and S_1 (area of triangle $P_{01}P_{02}P_1$) is added to S_0 to refine the area estimate. After two iterations, P_2 and P_3 are found, and S_2 (area of triangle $P_{02}P_1P_2$) is subtracted and S_3 (area of triangle $P_{01}P_1P_3$) is added. So, after two level of refinements, the yield volume is $S_0 + S_1 - S_2 + S_3$. It is clear that after several iterations, we can get a fairly good estimate of the total area under the boundary curve.

Also note that the computational cost for calculating the determinant (6.52) is expensive ($O(N^3)$). However, since the points that constitute the new simplex share all but one points with the vertices of the existing simplex, we can reduce the computational cost to $O(N)$. For example, in Fig. 6.18 triangle $P_{01}P_{02}P_1$ and triangle $P_{01}P_1P_3$ share the edge $P_{01}P_1$. Therefore, the ratio of S_1 to S_3 is just the ratio of the distance from P_{02} to $P_{01}P_1$ and the distance from P_3 to $P_{01}P_1$, which can

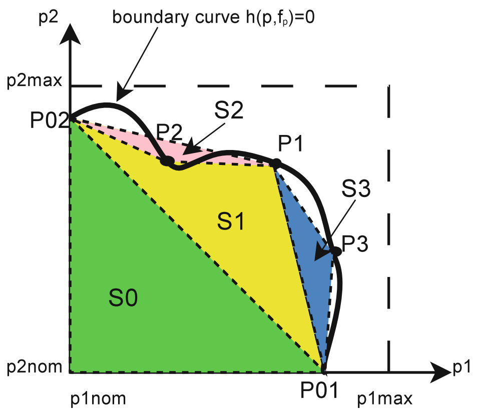


Fig. 6.18 Illustration of yield volume update scheme

be computed in $O(N)$ time since the normal vector is already known. Similarly, this applies to the computation of higher-dimensional hypervolumes.

The relative incremental hypervolume computed also provides an estimate of the error from the true yield. Based on the volume increment, YENSS adaptively decides whether to continue further iterations, depending on the accuracy desired. For example, in Fig. 6.18, if $S2$ is small enough (smaller than some predefined threshold), there is no need to find more points on curve $P_{02} - P_1$; but if $S3$ is still large, we can continue the curve-finding scheme to find more points on curve $P_1 - P_{01}$, until the designated accuracy is reached.

This automatic error control scheme helps to avoid redundant simulations that increase computational cost unnecessarily. If the boundary is a straight line, the error calculated by this scheme after one refinement is 0, and the algorithm terminates with perfect accuracy. This explains how this error estimation works and why YENSS is extremely efficient if the performance function varies linearly with parameter variations.

6.4 Examples and Comparisons

To illustrate how YENSS works, we first apply it on an illustrative multiple-constraint problem. Then we apply both YENSS and the Monte-Carlo method to the SRAM read access failure problem mentioned previously.

6.4.1 An Illustrative Example of YENSS

This example shows how YENSS works on a multiple-constraint problem. In this problem, we have two independent parameters p_1 and p_2, which are uniformly distributed on the interval [0, 0. 5]. There are three constraints on the parameters:

$$\begin{aligned} &2p_1 + p_2 - 1 \le 0, \\ &p_1 + 2p_2 - 1 \le 0, \\ &1.05^2 - (p_1 - 1)^2 - (p_2 - 1)^2 \le 0. \end{aligned} \tag{6.53}$$

We would like to compute the parametric yield.

Figure 6.19 shows the boundaries corresponding to the three constraints. From the previous sections, we know that the yield is the ratio of the area inside all the boundaries to the area of the box [0, 0. 5] × [0, 0. 5].

Applying YENSS, we first identify points on the interior of the three boundaries. As shown in Fig. 6.19, the dots are the points found by applying the line search method. After 6 refinements, 31 points are found that approximate the boundary. As can be seen, they approximate the boundary very well, and are almost evenly spaced on the boundary. The area inside boundaries is computed to be 0.1388, and therefore, we have the yield estimate to be $\frac{0.1388}{0.25} = 55.52\%$.

6.4.2 Application to SRAM Read Access Failure

We now apply YENSS and the Monte-Carlo method on the SRAM example mentioned previously. Considering the threshold voltages of $M1$ and $M2$ to be the varying parameters of interest, we obtain the scalar equation (6.34) for the boundary to be

$$h(V_{\text{th1}}, V_{\text{th2}}) \equiv (\mathbf{c}_{BL}^T - \mathbf{c}_{BL\_B}^T)\Phi(T_{\max}; V_{\text{th1}}, V_{\text{th2}}) - \Delta BL_{\min} = 0, \tag{6.54}$$

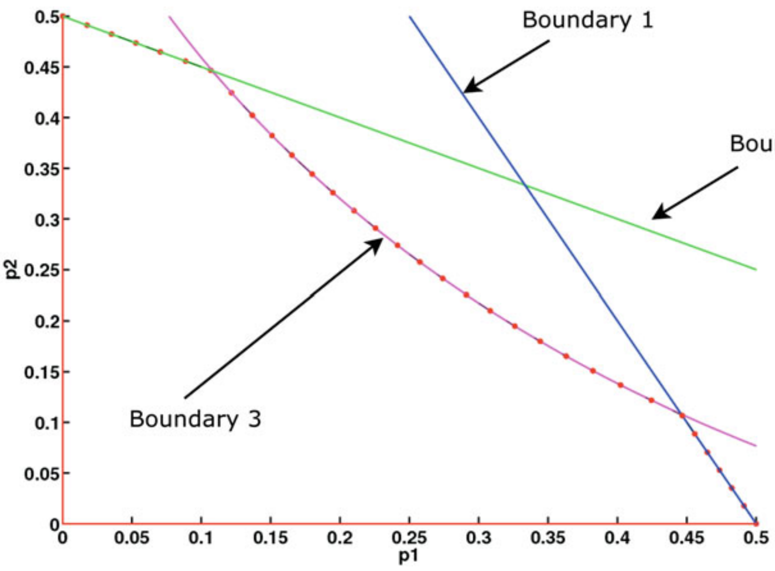


Fig. 6.19 Illustration of YENSS on a multiple-constraint problem

where: $\Phi(\cdot)$ is the state-transition function defining the node voltages and branch currents in the circuit, which is calculated by performing a transient simulation; $(\mathbf{c}_{BL}^T - \mathbf{c}_{BL\_B}^T)$ is the vector which selects the voltage difference between node BL and BL_B; and $T_{\max}$ is the specified maximum read access time, defined as the time required to produce the minimum voltage difference $\Delta BL_{\min}$ between BL and BL_B that can be detected by the sense amplifier (normally defined by $\Delta BL_{\min} \simeq 0.1Vdd$).

Assume that the threshold voltages are independent and are uniformly distributed on $[0.1V, 0.7V]$, and that $\Delta BL_{\min} = 168\text{mV}$ at $T_{\max} = 3.508$ ns. We first vary the threshold voltage of M1 (V_{th1}) and M2 (V_{th2}), and run simulations with different (V_{th1}, V_{th2}) pairs. Figure 6.20a depicts $h(V_{\text{th1}}, V_{\text{th2}})$ (i.e., $\Delta BL - \Delta BL_{\min}$) surface over ($V_{\text{th1}}$, V_{th2}) parameter space. From this surface generated by brute-force simulations, the boundary curve can be identified, visually as shown in Fig. 6.20b. There are two problems with the brute-force method: (1) the accuracy is limited by the step size on each parameter axis – each point generated in this method does not lie exactly on the boundary curve, and this is obviously observed in Fig. 6.20b; (2) the number of simulations can be too large to be affordable. Although some heuristics can be applied, (for example, one may safely skip simulating the area where V_{th1} and V_{th2} are both small), the computational cost is still expensive.

We then apply YENSS to find the boundary curve directly. Figure 6.21 shows the results of YENSS using different levels of refinements, as well as results from Monte-Carlo simulation. In Fig. 6.21a, only three levels of refinements are performed – seven points (marked on the curve) on the boundary curve are obtained. The boundary curve obtained is still coarse and jagged of this level of refinement. Figure 6.21b, and c shows the results of 4 and 5 levels of refinements, by which point a smooth boundary curve is obtained. The boundary found by YENSS is validated by Monte-Carlo simulation, as shown in Fig. 6.21d.

Detailed comparisons regarding accuracy, simulation time and speed-ups between YENSS and Monte-Carlo is shown in Table 6.1. To achieve an accuracy of 1% with 95% confidence, Monte-Carlo needs over 6,000 simulations. However, only three levels of refinement using YENSS, even though the curve is still jagged (shown in Fig. 6.21a), gives an accuracy of 1%, with 255 × speedup over the Monte-Carlo method.

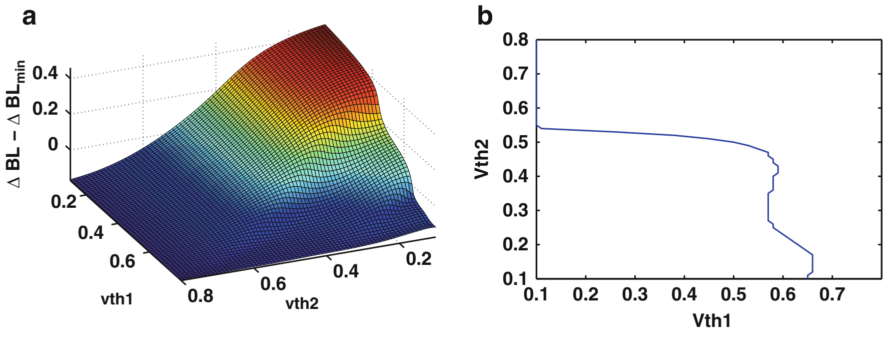


Fig. 6.20 Brute force simulation to get the boundary curve

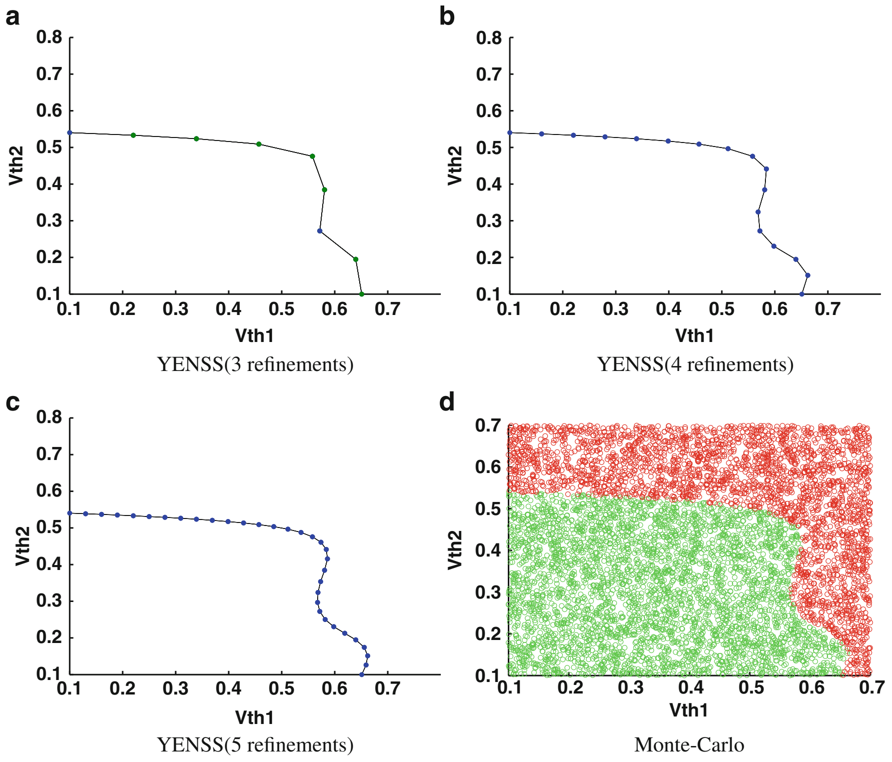


Fig. 6.21 Simulation results of YENSS and Monte-Carlo on SRAM read access time constraint

Table 6.1 Comparison of YENSS vs. Monte-Carlo for SRAM yield

Method	Accuracy	Yield	Number of transient simulations	Speedup
Monte-Carlo	10%	0.6173	80	
Monte-Carlo	1%	0.5730	6640	1 ×
YENSS				
3 levels	1%	0.5756	26	255 ×
4 levels	0.1%	0.5789	41	162 ×
5 levels	< 0.1%	0.5797	66	101 ×

6.5 Summary

In this chapter, we have formulated the yield estimation problem as a probabilistic hypervolume computation problem. We classified existing methods into two classes: statistical and deterministic/mixed methods, and have reviewed available algorithms. We have discussed the deterministic YENSS algorithm in detail, illustrated its application to an SRAM circuit, and provided comparisons against the classical Monte-Carlo method.

References

1. Nassif SR (2001) Modeling and analysis of manufacturing variations. In Proceedings of the IEEE conference on custom integrated circuits, 6–9 May 2001, pp 223–228
2. Papoulis A (1984) Probability, random variables, and stochastic processes. Mc-Graw Hill, New York
3. The International Technology Roadmap for Semiconductors (2008) http://www.itrs.net/
4. Maly W, Heineken H, Khare J, Nag PK (1996) Design for manufacturability in submicron domain. In Proceedings of the IEEE/ACM international conference on computer-aided design ICCAD-96. Digest of Technical Papers, 10–14 November 1996, pp 690–697
5. Gupta P, Kahng AB (2003) Manufacturing-aware physical design. In Proceedings of the ICCAD-2003 computer aided design international conference, 9–13 November 2003, pp 681–687
6. Heald R, Wang P (2004) Variability in sub-100nm sram designs. In Proceedings of the ICCAD-2004 computer aided design IEEE/ACM international conference, 7–11 November 2004, pp 347–352
7. Rabaey JM, Chandrakasan A, Nikolic B (2003) Digital integrated circuits, 2nd edn. (Printice Hall Electronics and Vlsi Series). Prentice Hall, Englewood Cliffs, NJ, USA
8. Agarwal K, Nassif S (2007) Characterizing process variation in nanometer cmos. In Proceedings of the 44th ACM/IEEE design automation conference DAC ’07, 4–8 June 2007, pp 396–399
9. Chenjie Gu, Roychowdhury J (2008) An efficient, fully nonlinear, variability-aware non-monte-carlo yield estimation procedure with applications to sram cells and ring oscillators. In Proceedings of the Asia and South Pacific design automation conference ASPDAC 2008, 21–24 March 2008, pp 754–761
10. Singhal K, Pinel J (1981) Statistical design centering and tolerancing using parametric sampling. Circuits Syst, IEEE Trans on 28(7):692–702
11. Wojciechowski JM, Vlach J (1993) Ellipsoidal method for design centering and yield estimation. Comput Aided Des Integr Circuits Syst, IEEE Trans on 12(10):1570–1579
12. Director S, Hachtel G (1977) The simplicial approximation approach to design centering. Circuits Syst, IEEE Trans on 24(7):363–372
13. Maly W, Director SW (1980) Dimension reduction procedure for the simplicial approximation approach to design centering. IEE Proc G Electronic Circuits Syst 127(6):255–259
14. Antreich K, Koblitz R (1982) Design centering by yield prediction. IEEE Trans Circuits and Syst 29(2):88–96
15. Seifi A, Ponnambalam K, Vlach J (1999) A unified approach to statistical design centering of integrated circuits with correlated parameters. IEEE Trans Circuits Syst I, Fundam Theory Appl 46(1):190–196
16. Low KK, Director SW (1989) A new methodology for the design centering of ic fabrication processes. In Proceedings of the IEEE International Conference on Computer-Aided Design ICCAD-89. Digest of Technical Papers, 5–9 November 1989, pp 194–197
17. Polak E, Sangiovanni-Vincentelli A (1979) Theoretical and computational aspects of the optimal design centering, tolerancing, and tuning problem. IEEE Trans Circuits Syst 26 (9):795–813
18. Brayton R, Director S, Hachtel G (1980) Yield maximization and worst-case design with arbitrary statistical distributions. Circuits Syst, IEEE Trans, 27(9):756–764
19. Chopra K, Shah S, Srivastava A, Blaauw D, Sylvester D (2005) Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation. In Proceedings of the ICCAD-2005 computer-aided design IEEE/ACM international conference, 6–10 November 2005, pp 1023–1028
20. Director SW, Feldmann P, Krishna K (1992) Optimization of parametric yield: a tutorial. In Proceedings of the custom integrated circuits conference the IEEE 1992, May 3–6, 1992, pp 3.1.1–3.1.8

21. Stehr G, Graeb H, Antreich K (2003) Performance trade-off analysis of analog circuits by normal-boundary intersection. In Proceedings of the Design Automation Conference, 2003, 2–6 June 2003, pp 958–963
22. Toumazou C, Moschytz GS, Gilbert B (eds) (2002) Trade-Offs in analog circuit design: the designer's companion. Kluwer Academic Publishers, Dordrecht (Hingham, MA)
23. Eeckelaert T, McConaghy T, Gielen G (2005) Efficient multiobjective synthesis of analog circuits using hierarchical pareto-optimal performance hypersurfaces. In Proceedings of the design, automation and test in Europe, 07–11 March 2005, pp 1070–1075
24. Mukhopadhyay S, Mahmoodi H, Roy K (2005) Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. Comput -Aided Des Integr Circuits Syst, IEEE Trans 24(12):1859–1880
25. Swidzinski JF, Chang K (2000) Nonlinear statistical modeling and yield estimation technique for use in Monte Carlo simulations [Microwave Devices and ICs]. Microwave Theory Tech, IEEE Trans 48(12):2316–2324
26. Liu JS (2002) Monte Carlo strategies in scientific computing. Springer, Berlin
27. Robert CP, Casella G (2005) Monte Carlo statistical methods (Springer texts in statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA
28. Kundert KS (1995) The designer's guide to spice and spectre. Kluwer Academic Publishers, Dordrecht (Hingham, MA)
29. Kundert K, White J, Sangiovanni-Vincentelli A (1990) Steady-state methods for simulating analog and microwave circuits. Kluwer Academic Publishers, Dordrecht (Hingham, MA)
30. Seevinck E, List FJ, Lohstroh J (1987) Static-noise margin analysis of MOS SRAM cells. IEEE J Solid-State Circuits 22(5):748–754
31. Devgan A, Kashyap C (2003) Block-based static timing analysis with uncertainty. In Proceedings of the ICCAD-2003 computer aided design international conference, 9–13 November 2003, pp 607–614
32. Chang H, Sapatnekar SS (2003) Statistical timing analysis considering spatial correlations using a single pert-like traversal. In ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design, Washington, DC, USA, November 2003. IEEE Computer Society, p 621
33. Kanj R, Joshi R, Nassif S (2006) Mixture importance sampling and its application to the analysis of sram designs in the presence of rare failure events. In Proceedings of the 43rd ACM/IEEE design automation conference, 24–28 July 2006, pp 69–72
34. Hastings WK (1970) Monte carlo sampling methods using markov chains and their applications. Biometrika 57(1):97–109
35. Keramat M, Kielbasa R (1997) A study of stratified sampling in variance reduction techniques for parametric yield estimation. In Proceedings of the IEEE international symposium on circuits and systems ISCAS '97, vol3, 9–12 June 1997, pp 1652–1655
36. Gallaher LJ (1973) A multidimensional monte carlo quadrature with adaptive stratified sampling. Commun ACM 16(1):49–50
37. Stein M (1987) Large sample properties of simulations using latin hypercube sampling. Technometrics 29(2):143–151
38. Hammersley JM, Morton KW (1956) A new Monte Carlo technique: antithetic variates. In Proceedings of the Cambridge Philosophical Society, vol52 of Proceedings of the Cambridge Philosophical Society, July 1956, pp 449–475
39. Director S, Hachtel G (1977) The simplicial approximation approach to design centering. Circuits Syst, IEEE Trans 24(7):363–372
40. Director S, Hachtel G, Vidigal L (1978) Computationally efficient yield estimation procedures based on simplicial approximation. Circuits Syst, IEEE Trans 25(3):121–130
41. Biernacki RM, Bandler JW, Song J, Zhang QJ (1989) Efficient quadratic approximation for statistical design. IEEE Trans Circuit Syst 36(11):1449–1454
42. Antreich KJ, Graeb HE, Wieser CU (1994) Circuit analysis and optimization driven by worst-case distances. Comput Aided Des Integr Circuits Syst IEEE Trans 13(1):57–71, Jan. 1994.

43. Heath MT (1996) Scientific computing: an introductory survey. McGraw-Hill Higher Education, New York
44. Abdel-Malek HL, Hassan AKSO (1991) The ellipsoidal technique for design centering and region approximation. Comput Aided Des 10(8):1006–1014
45. Srivastava S, Roychowdhury J (2007) Rapid estimation of the probability of SRAM failure due to MOS threshold variations. In Custom integrated circuits conference, 2007., Proceedings of the IEEE 2007, September 2007
46. Allgower EL, Georg K (1990) Numerical continuation methods. Springer-Verlag, New York
47. Nocedal J, Wright SJ (1999) Numerical optimization. Springer, New York
48. Li P (2006) Statistical sampling-based parametric analysis of power grids. Comput -Aided Des Integr Circuits Syst, IEEE Trans 25(12):2852–2867
49. Chang H, Zolotov V, Narayan S, Visweswariah C (2005) Parameterized block-based statistical timing analysis with non-Gaussian parameters, nonlinear delay functions. In Proceedings of the 42nd Design Automation Conference, 2005, 13–17 June 2005, pp 71–76
50. Michael C, Ismail MI (1993) Statistical modeling for computer-aided design of MOS VLSI circuits. Springer, Berlin
51. Golub GH, VanLoan CF (1996) Matrix computations (Johns Hopkins Studies in Mathematical Sciences). The Johns Hopkins University Press, Baltimore, MD, USA
52. Chua LO, Lin P-M (1975) Computer-aided analysis of electronic circuits : algorithms and computational techniques. Prentice-Hall, Englewood Cliffs, NJ
53. Gear CW (1971) Numerical initial value problems in ordinary differential equations. Prentice-Hall series in automatic computation. Prentice-Hall, Englewood Cliffs, NJ